ChatGPT

CRYPTOANALIZA ODDZIAŁUJE NA STRUKTURĘ ARTEFAKTÓW

Streszczenie wykonawcze

W praktyce inżynierii bezpieczeństwa łańcucha dostaw oprogramowania SBOM (Software Bill of Materials) pełni funkcję *artefaktu referencyjnego* dla identyfikacji komponentów, zależności oraz korelacji z podatnościami. Definicja SBOM jako formalnego rejestru komponentów i relacji łańcucha dostaw jest spójna z ujęciem National Telecommunications and Information Administration [1] (NTIA, 2021). [2] W ekosystemie standaryzacji dominują formaty SPDX i CycloneDX: specyfikacja SPDX 3.0.1 definiuje otwarty standard komunikowania informacji BOM, obejmujący model danych i serializacje (SPDX, 2024). [3] CycloneDX jest standaryzowany jako ECMA-424 i projektowany jako „full-stack BOM" wspierający automatyzację, transparentność oraz zastosowania xBOM (Ecma International, 2024). [4]

W niniejszej ekspertyzie przyjmuję, że „cryptoanaliza SBOM" jest procesem analitycznym obejmującym dwie dopełniające się osie: (i) kryptograficzną weryfikację integralności i autentyczności artefaktów SBOM oraz artefaktów powiązanych (hashe, MAC/podpisy, atestacje), (ii) analizę logiki komunikatów (message logic) pomiędzy komponentami pipeline'u SBOM, gdzie komunikaty przenoszą wyniki walidacji, korelacji CVE/VEX oraz decyzje o sekwencjach usuwania podatności. Rekomendacje europejskie podkreślają, że SBOM powinien być kryptograficznie podpisany (weryfikowalność autentyczności/integralności), a hash dokumentu powinien być zgodny z rzeczywistymi plikami (ENISA, 2025). [5]

Kluczowym wynikiem pracy jest formalny model, w którym: (a) struktura artefaktów ulega *usztywnieniu kryptograficznemu* (wymuszenie kanonikalizacji/serializacji deterministycznej, list hash, powiązań), (b) logika komunikatów staje się systemem reguł i funkcji decyzyjnych nad *wektorami operacyjnymi* podatności, (c) bezpieczeństwo pipeline'u zależy nie tylko od doboru prymitywów (SHA-256/SHA-3/HMAC), lecz także od zapobiegania atakom na semantykę komunikacji (replay, rollback, substitution) i na warstwę reprezentacji (canonicalization confusion). Z perspektywy kryptografii aplikacyjnej konieczne jest rozróżnienie: hash (integralność) ≠ MAC (integralność + uwierzytelnienie), co jest fundamentem konstrukcji HMAC (IETF, 1997) i jej analizy (Bellare i in., 1996). [6]

Cel i zakres pracy

Celem pracy jest zbudowanie „reżimu naukowego" dla analizy, *jak logika komunikatów w procesie cryptoanalizy SBOM wymusza określoną strukturę artefaktów* oraz jak ta struktura wpływa na bezpieczeństwo i decyzje operacyjne (alerty, priorytety, sekwencje remediacji).

Zakres obejmuje:

1. Modelowanie komunikatów pomiędzy modułami pipeline'u (generator SBOM, walidator integralności, skaner podatności, silnik polityk, system alertów, orkiestrator remediacji).
2. Formalizację list hash i mechanizmów weryfikacji integralności (SHA-256, SHA-3, HMAC) oraz ich właściwości/ograniczeń w kontekście ataków (kolizje, preimage/second-preimage, length extension).

3. Formalny model *wektorów operacyjnych* podatności oraz reguł/polityk decyzyjnych (np. CVSS, EPSS, SSVC, KEV/LEV) determinujących kolejność napraw i generowanie alertów. [7]
4. Implementacje referencyjne w Pythonie (uruchamialny kod) symulujące komunikację i decyzje.

Ograniczenia i parametry o statusie „brak specyfikacji":

- **Semantyka kanału komunikacji** (kolejki, gwarancje dostarczenia, dokładnie-raz/at-least-once): brak specyfikacji w standardach SBOM; modeluję je abstrakcyjnie jako zdarzenia w kolejce.
- **Kanonikalizacja reprezentacji przed hashowaniem/podpisem**: standardy SBOM różnią się serializacjami; w praktyce zalecane są serializacje kanoniczne (np. SPDX Canonical Serialisation) albo schematy kanonikalizacji JSON (RFC 8785), natomiast dobór w konkretnej organizacji pozostaje parametrem (brak specyfikacji „uniwersalnej" dla całej branży). [8]
- **Wagi, progi i funkcje użyteczności w politykach ryzyka** (np. łączenie CVSS+EPSS+KEV): brak specyfikacji normatywnej; traktuję je jako parametry modelu decyzyjnego.

Przegląd literatury i standardów

Rdzeń problemu SBOM to interoperacyjny zapis inwentarza i relacji w łańcuchu dostaw oraz jego operacjonalizacja w zarządzaniu podatnościami. NTIA (2021) opisuje SBOM jako formalny rejestr komponentów i relacji łańcucha dostaw oraz wskazuje minimalne elementy i przypadki użycia (transparency, vulnerability response). [9]

SPDX (standard SBOM). Specyfikacja SPDX 3.0.1 definiuje standard komunikowania informacji BOM oraz podkreśla istnienie modelu danych i wielu serializacji; w SPDX 3.0.1 wprowadzono także profile zgodności (m.in. Security, Build), z których Core jest obowiązkowy (SPDX, 2024). [3] W kontekście integralności ważne są prace nad serializacją kanoniczną SPDX, której celem jest jednoznaczna, deterministyczna reprezentacja niezależna od stylu serializacji, umożliwiająca weryfikację integralności elementów. [10]

CycloneDX (standard xBOM/ECMA-424). Tekst ECMA-424 (1st ed., 2024) wskazuje CycloneDX jako BOM „pełnostackowy" i „general-purpose", wspierający inwentarze software/hardware/services i nastawiony na automatyzację i identyfikację ryzyka. [11] Dla cryptoanalizy istotne jest formalne wsparcie dla *cryptographic artefacts* (CBOM – Cryptography Bill of Materials) w ekosystemie CycloneDX, które pozwala modelować zasoby kryptograficzne (algorytmy, klucze, certyfikaty) i ich relacje z komponentami. [12]

Walidacja SBOM i podpisy. European Union Agency for Cybersecurity [13] (ENISA, 2025) wskazuje kryteria jakości walidacji SBOM w trzech warstwach i explicite podaje, że dla autentyczności i integralności SBOM powinien być kryptograficznie podpisany, a hash dokumentu powinien odpowiadać rzeczywistym plikom. [5] CycloneDX publikuje przypadki użycia weryfikacji autentyczności z podpisami cyfrowymi (np. XML Signature, JWS/JSF), co pokazuje, że standard zakłada *warstwę kryptograficzną* jako część operacyjnej użyteczności SBOM. [14]

Kanonikalizacja danych dla operacji kryptograficznych. RFC 8785 (JCS) motywuje kanoniczną reprezentację JSON jako warunek powtarzalności operacji kryptograficznych (hash, podpis, HMAC), ponieważ serializacja „na drucie" może być niejednoznaczna. [15] W logice komunikatów pipeline'u SBOM jest to krytyczne: ten sam obiekt semantyczny SBOM musi mieć identyczną reprezentację bajtową w producerze i consumerze, inaczej walidacja hash/podpisu nie jest stabilna.

Kryptografia aplikacyjna: hashe i MAC. Standard FIPS 180-4 opisuje rodzinę SHA (w tym SHA-256) jako funkcje skrótu używane do wykrywania zmian danych (NIST, 2015). [16] Standard FIPS 202 opisuje SHA-3 jako rodzinę funkcji opartych na konstrukcji permutacyjnej (KECCAK), stanowiącą alternatywną rodzinę

funkcji skrótu (NIST, 2015). [17] Mechanizm HMAC jest zdefiniowany w RFC 2104 (IETF, 1997) jako metoda uwierzytelniania wiadomości oparta o funkcję skrótu i tajny klucz. [18] Analiza bezpieczeństwa HMAC (Bellare, Canetti, Krawczyk, 1996) stanowi pierwotne uzasadnienie formalne, dlaczego konstrukcja HMAC zapewnia bezpieczeństwo MAC nawet przy hashach, które mogą mieć słabości kolizyjne w pewnych modelach. [19]

Bezpieczeństwo łańcucha dostaw i weryfikowalność kroków. Praca Santiago Torres-Arias [20] i in. (2019) wprowadza in-toto jako framework egzekwujący integralność *całego* łańcucha dostaw przez zbieranie kryptograficznie weryfikowalnych informacji o krokach (layout, podpisane „link metadata"). [21] To jest bezpośrednio spójne z tezą tej ekspertyzy: cryptoanaliza „oddziałuje" na strukturę artefaktów, ponieważ wymusza generowanie i transport metadanych kryptograficznych pomiędzy krokami (a nie tylko podpis końcowy).

Decyzje operacyjne w zarządzaniu podatnościami. CVSS v4.0 jest publikowany przez FIRST [22] (FIRST, 2023) jako standard oceny charakterystyk podatności poprzez grupy metryk (Base/Threat/Environmental/Supplemental). [23] EPSS, rozwijany przez FIRST, jest metryką probabilistyczną przewidującą prawdopodobieństwo eksploatacji; jego ujęcie naukowe obejmuje m.in. artykuł Jay Jacobs [24] i in. (2020) o strategiach remediacji opartych o predykcję exploitów „in the wild", oraz preprint systemu EPSS (Jacobs i in., 2019). [25] SSVC (Stakeholder-Specific Vulnerability Categorization) w wersji testowalnej jest opracowany przez Jonathan M. Spring [26] i in. (2021) jako zestaw drzew decyzyjnych dopasowywanych do kontekstu interesariusza, co stanowi alternatywę dla „jednego wyniku punktowego" (Spring i in., 2021). [27] Jako uzupełnienie sygnałów „real-world exploitation" funkcjonuje KEV catalog prowadzony przez Cybersecurity and Infrastructure Security Agency [28] (CISA). [29] NIST proponuje metrykę LEV jako kumulację sygnałów EPSS w czasie dla estymacji prawdopodobieństwa eksploatacji (Mell, 2025). [30]

Badania empiryczne nad SBOM (praktyka i bariery). Praca Boming Xia [31] i in. (2023) opisuje empiryczne badanie praktyk SBOM (wywiady/ankiety) oraz problemy adopcyjne. [32] Studium Trevor Stalnaker [33] i in. (2024) „BOMs Away!" (ICSE'24) wskazuje zróżnicowane potrzeby interesariuszy SBOM oraz złożoność ekosystemu BOM. [34] W obszarze *zaufanego i poufnego* udostępniania SBOM praca Ishgair i in. (2025) („Petra") proponuje format-agnostyczną, tamper-evident reprezentację SBOM oraz kryptograficzne audytowanie redagowanych SBOM z selektywnym szyfrowaniem. [35]

Perspektywa polska i europejska (łańcuch dostaw). W polskim kontekście formalizacji cyberbezpieczeństwa łańcuchów dostaw istotna jest praca Lidermana i Księżopolskiego (2025), która odnosi się do unijnych aktów prawnych, standardów oraz formalnego opisu cyfrowych łańcuchów dostaw. [36] Istotne są również wytyczne i opracowania administracji publicznej dotyczące wiązania wymagań (m.in. SBOM) z dowodami, metrykami i terminami napraw. [37]

Model formalny logiki komunikatów w cryptoanalizie SBOM

Definicje podstawowe

Niech:

- $\mathcal{A}$ — zbiór artefaktów (np. SBOM, binaria, manifesty, atestacje, raporty skanów).
- $\mathcal{B} = \{0,1\}^*$ — zbiór ciągów bajtów.
- $\mathrm{canon} : \mathcal{X} \to \mathcal{B}$ — funkcja kanonikalizacji obiektu semantycznego x (np. JSON SBOM) do deterministycznej reprezentacji bajtowej. Wersje referencyjne: SPDX Canonical Serialisation oraz

JCS (RFC 8785), ale wybór jest parametrem; brak specyfikacji „jednej" metody dla wszystkich formatów. [38]

- $H_{\mathrm{SHA256}} : \mathcal{B} \to \{0,1\}^{256}$ — funkcja skrótu SHA-256 (NIST, 2015). [39]
- $H_{\mathrm{SHA3}} : \mathcal{B} \to \{0,1\}^{256}$ — funkcja skrótu SHA3-256 (NIST, 2015). [40]
- $\mathrm{HMAC}_K(\cdot)$ — kod uwierzytelniający wiadomości oparty na kluczu K (RFC 2104). [41]

Lista hash jako struktura artefaktu

Dla artefaktu $a \in \mathcal{A}$ definiuję listę hash:

$$\mathrm{HL}(a) = \{(\mathtt{alg}_i, d_i)\}_{i=1}^{n} \quad \text{gdzie} \quad d_i = H_{\mathtt{alg}_i}(\mathrm{canon}(a)).$$

W praktyce $\mathtt{alg}_i \in \{\mathrm{SHA256}, \mathrm{SHA3\_256}\}$, a w środowiskach zaufanych wewnętrznie można dodać MAC:

$$\mathrm{tag}(a) = \mathrm{HMAC}_K(\mathrm{canon}(a)).$$

Rozróżnienie hash vs MAC jest fundamentalne (RFC 2104; Bellare i in., 1996). [6]

Model komunikatu i logika komunikatów

Niech $\mathcal{T}$ będzie zbiorem typów komunikatów, np.:

- $t =$ SBOM_CREATED,
- $t =$ SBOM_VERIFIED,
- $t =$ VULN_OBSERVED,
- $t =$ REMEDIATION_PLAN,
- $t =$ ALERT.

Komunikat modeluję jako krotkę:

$$m = (t, p, \mu, \mathrm{HL}(p), \mathrm{tag}(m)),$$

gdzie: - p — payload (np. SBOM lub wynik skanu), - μ — metadane (czas, identyfikatory, wersje), - $\mathrm{HL}(p)$ — lista hash payloadu, - $\mathrm{tag}(m)$ — opcjonalny MAC/podpis całego komunikatu (autentyczność kanału).

W warstwie operacyjnej logika komunikatów jest funkcją przejścia stanu:

$$S_{k+1} = \delta(S_k, m_k),$$

gdzie S_k to stan systemu (graf zależności, zaufane artefakty, backlog podatności), a δ koduje reguły walidacji i decyzji (np. „jeżeli HL niezgodna, wygeneruj ALERT"). Ujęcie to jest spójne z literaturą o potrzebie kryptograficznie weryfikowalnych informacji o krokach pipeline'u (Torres-Arias i in., 2019). [42]

Wektory operacyjne i polityki decyzyjne

Dla każdej obserwacji podatności r definiuję wektor operacyjny:

$$\mathbf{x}(r) = \big(\underbrace{s}_{\text{severity}}, \underbrace{e}_{\text{exploitability}}, \underbrace{k}_{\text{kev flag}}, \underbrace{u}_{\text{reachability}}, \underbrace{c}_{\text{criticality}}, \underbrace{a}_{\text{fix avail}}, \underbrace{\rho}_{\text{remediation cost}}, \dots \big).$$

Przykładowe semantyki wymiarów:

- s — np. CVSS Base/Threat/Environmental (FIRST, 2023) lub brak specyfikacji, jeśli brak danych. [43]
- e — np. probabilistyczny EPSS (Jacobs i in., 2019/2020). [44]
- $k \in \{0,1\}$ — czy CVE jest w KEV (CISA). [45]
- $u \in \{0,1\}$ — reachability/eksponowanie w kontekście systemu (często brak specyfikacji w standardach; wymaga lokalnej telemetrii).
- $c \in [0,1]$ — krytyczność zasobu (brak specyfikacji; parametr organizacyjny).
- $\rho \in [0,1]$ — koszt/ryzyko wdrożenia poprawki (brak specyfikacji; parametr).

Polityka decyzji to funkcja π mapująca wektor na działanie/priorytet:

- **polityka punktowa**: $P(r) = f(\mathbf{x}(r); \mathbf{w})$, sortowanie malejące po P,
- **polityka regułowa/drzewiasta**: $\pi_{\mathrm{SSVC}}(\mathbf{x}) \in \{\mathrm{defer}, \mathrm{scheduled}, \mathrm{out\text{-}of\text{-}cycle}, \mathrm{immediate}\}$ (Spring i in., 2021). [46]

Diagramy relacji i sekwencji

Poniższy diagram pokazuje relacje artefaktów, hash, alertów i sekwencji remediacji jako graf przepływu:

```
flowchart TB
  subgraph Producer[Generator/Builder]
    A1[Artefakt build (binaria)]
    A2[SBOM (SPDX/CycloneDX)]
    H1[Listy hash: SHA-256/SHA3]
  end

  subgraph Verifier[Weryfikator integralności]
    V1[Canon/Serializacja]
    V2[Weryfikacja hash / podpis / HMAC]
  end

  subgraph Analyzer[Skaner i korelator]
    S1[Dane CVE/VEX/CSAF]
    S2[Obserwacje podatności r]
    X1[Wektory operacyjne x(r)]
  end

  subgraph Policy[Silnik polityk]
    P1[Model decyzyjny: CVSS/EPSS/SSVC/KEV]
    P2[Plan remediacji: sekwencja z zależnościami]
  end

  subgraph Alerts[Alerting]
    AL1[Alert integralności]
    AL2[Alert ryzyka podatności]
  end

  A1 --> A2 --> H1 --> V1 --> V2
```

```
  V2 -->|OK| S2
  V2 -->|FAIL| AL1

  S1 --> S2 --> X1 --> P1 --> P2
  P1 -->|próg eskalacji| AL2
```

Sekwencja komunikatów (model message-based) w pipeline'u:

```
sequenceDiagram
  participant B as Builder
  participant G as SBOM Generator
  participant I as Integrity Verifier
  participant C as CVE/VEX Correlator
  participant E as Policy Engine
  participant A as Alerting

  B->>G: SBOM_CREATED(sbom payload)
  G->>I: MSG + HL(payload)
  I->>I: canon(); verify(HL)
  alt integralność OK
    I->>C: SBOM_VERIFIED(sbom ref)
    C->>E: VULN_OBSERVED(vectors x(r))
    E->>E: compute priority; plan sequence
    E->>A: ALERT if thresholds met
  else integralność FAIL
    I->>A: ALERT(integrity violation)
  end
```

W ujęciu ścisłym „sekwencja usuwania podatności" jest *liniowym uporządkowaniem* z częściowego porządku wynikającego z zależności (graf SBOM) oraz priorytetów (polityka). To jest problem planowania: znaleźć permutację $(r_{i_1}, \ldots, r_{i_m})$ zgodną z zależnościami i maksymalizującą funkcję użyteczności opartą o $P(r)$ (wagi: brak specyfikacji).

Listy hash i ich rola w integralności SBOM oraz w atakach

Rola kryptograficzna: integralność, powiązanie, audytowalność

1. **Integralność**: hash jest skrótem danych używanym do wykrywania zmian (NIST, 2015). [47] W pipeline SBOM porównanie $H(\mathrm{canon}(a))$ z oczekiwanym d umożliwia wykrycie manipulacji SBOM lub artefaktu.
2. **Powiązanie (binding)**: lista hash może wiązać SBOM z artefaktami build (np. binaria, manifesty), co zwiększa audytowalność i redukuje ryzyko „SBOM drift". Ryzyko rozbieżności SBOM narzędziowy i lifecycle'owy jest opisane m.in. przez badania empiryczne i inicjatywy harmonizacyjne (Xia i in., 2023; SEI Plugfest 2024). [48]
3. **Warunek autentyczności**: hash sam nie zapewnia autentyczności (napastnik może podmienić zarówno dokument, jak i hash). Stąd ENISA wskazuje konieczność podpisu kryptograficznego oraz walidacji łańcucha certyfikatów. [5]

Porównanie SHA-256, SHA-3 i HMAC w kontekście SBOM

Poniższa tabela syntetyzuje właściwości istotne dla logiki komunikatów i ataków (kryteria: bezpieczeństwo konstrukcji, podatność na length extension, zastosowania w SBOM).

Mechanizm	Typ prymitywu	Standard pierwotny	Zależność od klucza	Odporność na length extension (w typowych użyciach)	Typowa rola w pipeline SBOM
SHA-256	hash (Merkle–Damgård/ SHA-2)	NIST (2015) FIPS 180-4 [39]	nie	**nie**, jeśli użyty jako „naiwny MAC" typu $H(k\|\|m)$ (właściwość konstrukcji iteracyjnej; patrz Merkle/Damgård + literatura o atakach strukturalnych) [49]	integralność artefaktów, identyfikatory content-addressable
SHA3-256	hash (sponge/ KECCAK)	NIST (2015) FIPS 202 [40]	nie	**tak** (brak klasycznej podatności length extension charakterystycznej dla Merkle–Damgård) – w praktyce wynika z odmienności konstrukcji (sponge) w standardzie SHA-3 [50]	alternatywny hash, dywersyfikacja kryptograficzna
HMAC-SHA-256	MAC	IETF (1997) RFC 2104; Bellare i in. (1996) [6]	tak	**tak** (konstrukcja projektowana jako MAC odporny na ataki długościowe przy iteracyjnych hashach) [6]	uwierzytelnianie komunikatów wewnątrz zaufanej domeny
HMAC-SHA3-256	MAC	RFC 2104 (mechanizm), SHA-3 FIPS 202 [51]	tak	**tak** (MAC + hash sponge)	jw., alternatywa algorytmiczna

W implementacjach Python istotne jest, że moduł `hmac` implementuje HMAC zgodnie z RFC 2104, ale nie obsługuje funkcji XOF (np. SHAKE), co wpływa na dobór algorytmów w praktycznych systemach. [52]

Typy ataków i ich interpretacja w logice komunikatów

Kolizje

Kolizja oznacza znalezienie $m \neq m'$, takich że $H(m) = H(m')$. Dla nowoczesnych hash (SHA-2/SHA-3) praktyczne kolizje są obecnie uznawane za nieosiągalne w typowych budżetach, natomiast historia SHA-1 pokazuje, że osłabienie funkcji skrótu realnie przekłada się na integralność ekosystemów (NIST informował o praktycznej demonstracji kolizji SHA-1 w 2017 r.). [53] W kontekście pipeline'u SBOM wniosek operacyjny jest bezpośredni: unikać SHA-1/MD5 dla wiązania artefaktów i nie dopuszczać downgrade'u algorytmicznego.

Preimage i second-preimage

Własność preimage odnosi się do trudności znalezienia m dla danego y, gdzie $H(m) = y$; second-preimage — znalezienia $m' \neq m$ dla danego m. Dla iteracyjnych hash literaturowo znane są ataki genericzne na second-preimage dla bardzo długich wiadomości (Kelsey i Schneier, 2005), co stanowi argument, by rozumieć nie tylko „długość wyjścia", ale i konstrukcję oraz kontekst użycia. [54] Dla SBOM (zwykle krótkie dokumenty) ataki te mają mniejszą wagę praktyczną, lecz stanowią element *kryptograficznej analityki ryzyka*.

Length extension

Dla konstrukcji iteracyjnych Merkle'a–Damgårda (Merkle, 1989; Damgård, 1989) istnieje własność umożliwiająca atakującemu, mając $H(m)$, obliczenie $H(m \| \mathrm{pad}(m) \| x)$ dla wybranego x, bez znajomości m. [55] W kryptografii aplikacyjnej konsekwencja jest krytyczna: konstrukcja „MAC-na-skrót" w postaci $H(k \| m)$ jest podatna na length extension, dlatego stosuje się HMAC (RFC 2104; Bellare i in., 1996). [6] W logice komunikatów pipeline'u SBOM oznacza to zakaz uwierzytelniania komunikatów przez „doklejanie klucza i hashowanie" oraz konieczność stosowania MAC/podpisu.

Tabela porównawcza typów ataków, skutków i mitigacji

Klasa ataku	Warunek wstępny	Skutek dla SBOM/ message logic	Przykładowa detekcja	Mitigacje (mechanizmy)
Collision (na słabym hashu)	użycie osłabionego hashu (np. SHA-1)	podmiana artefaktu przy zachowaniu hash	walidacja polityki „algorithms allowed"; analiza kryptograficzna	migracja do SHA-256/ SHA-3; polityka zakazu downgrade; rotacja [56]
Preimage/ second-preimage (teoretyczne/ genericzne)	duże zasoby; specyficzne długości/struktury	trudniejsze powiązanie artefaktu z treścią	monitoring anomalii, kontrola pochodzenia	podpisy, atestacje, provenance; dywersyfikacja hash [57]

Klasa ataku	Warunek wstępny	Skutek dla SBOM/ message logic	Przykładowa detekcja	Mitigacje (mechanizmy)
Length extension (przy naiwnym MAC)	użycie $H(k\|\|m)$ na Merkle–Damgård	fałszowanie „uwierzytelnionych” komunikatów	weryfikacja schematu MAC	HMAC (RFC 2104) / podpisy; unikanie naiwnych MAC [6]
Replay/Rollback komunikatu	brak świeżości/ wersjonowania	stare SBOM/VEX wygrywa z nowym	kontrola monotoniczności wersji	TUF-like rollback protections; rejestry wersji; atestacje build/ provenance [58]
Substitution (podmiana SBOM ↔ artefakt)	brak wiązania	SBOM nie dotyczy danego binarium	kontrola mapowania hash ↔ artefakt	podpis SBOM + wiązanie z hash artefaktów; in-toto link metadata [59]
Canonicalization confusion	wiele serializacji/ niejednoznaczność JSON	fałszywe alarmy integralności lub „luki” w walidacji	testy deterministyczności	JCS (RFC 8785), canonical serialisation SPDX, jednoznaczne reguły [60]

Implementacje referencyjne i eksperymenty w Python

Poniższe implementacje są **samodzielne, uruchamialne i testowalne**. Są to referencyjne symulacje logiki komunikatów w pipeline'u SBOM, a nie kompletne biblioteki przetwarzania SPDX/CycloneDX (parsowanie pełnych schematów: brak specyfikacji w tej ekspertyzie; celowo upraszczam model danych).

Skrypt symulacyjny: pipeline komunikatów, hashe, wektory operacyjne, plan remediacji, alerty

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
sbom_cryptoanalysis_demo.py

Symulacja logiki komunikatów w procesie cryptoanalizy SBOM:
- deterministyczna serializacja (uproszczona)
```

```
- generowanie list hash: SHA-256, SHA3-256
- HMAC jako uwierzytelnienie komunikatu w zaufanej domenie
- model wektorów operacyjnych podatności i polityk decyzyjnych
- generowanie sekwencji remediacji (z zależnościami) i alertów

Wymagania: Python >= 3.11, bez zewnętrznych zależności.
Uruchomienie testów:
    python sbom_cryptoanalysis_demo.py
Uruchomienie demonstracji:
    python sbom_cryptoanalysis_demo.py --demo
"""

from __future__ import annotations

import dataclasses
import datetime as dt
import enum
import hashlib
import hmac
import json
import queue
import random
import sys
import typing as t
import unittest
from collections import defaultdict, deque


# ---------------------------
# Deterministyczna serializacja (uproszczona)
# ---------------------------

def canonical_json_bytes(obj: t.Any) -> bytes:
    """
    Uproszczona postać deterministyczna JSON:
    - sort_keys=True
    - separators=(',', ':') (brak spacji)
    - ensure_ascii=False (UTF-8)
    UWAGA: nie jest to pełna implementacja RFC 8785 (JCS).
    """
    s = json.dumps(obj, sort_keys=True, separators=(",", ":"),
ensure_ascii=False)
    return s.encode("utf-8")


# ---------------------------
# Hash listy
# ---------------------------

class HashAlg(str, enum.Enum):
    SHA256 = "sha256"
```

```
    SHA3_256 = "sha3_256"


@dataclasses.dataclass(frozen=True)
class HashList:
    digests: dict[HashAlg, str]  # hex

    @staticmethod
    def compute(payload_bytes: bytes, algs: t.Iterable[HashAlg]) ->
"HashList":
        d: dict[HashAlg, str] = {}
        for alg in algs:
            if alg == HashAlg.SHA256:
                d[alg] = hashlib.sha256(payload_bytes).hexdigest()
            elif alg == HashAlg.SHA3_256:
                d[alg] = hashlib.sha3_256(payload_bytes).hexdigest()
            else:
                raise ValueError(f"Nieznany algorytm: {alg}")
        return HashList(digests=d)

    def verify(self, payload_bytes: bytes) -> bool:
        for alg, expected_hex in self.digests.items():
            if alg == HashAlg.SHA256:
                got = hashlib.sha256(payload_bytes).hexdigest()
            elif alg == HashAlg.SHA3_256:
                got = hashlib.sha3_256(payload_bytes).hexdigest()
            else:
                return False
            if not hmac.compare_digest(got, expected_hex):
                return False
        return True


# ---------------------------
# Model komunikatów
# ---------------------------

class MsgType(str, enum.Enum):
    SBOM_CREATED = "SBOM_CREATED"
    SBOM_VERIFIED = "SBOM_VERIFIED"
    INTEGRITY_FAIL = "INTEGRITY_FAIL"
    VULN_OBSERVED = "VULN_OBSERVED"
    REMEDIATION_PLAN = "REMEDIATION_PLAN"
    ALERT = "ALERT"


@dataclasses.dataclass(frozen=True)
class Envelope:
    """
    Ujednolicony (kanonikalizowany) "envelope" komunikatu.
    """
```

```
    msg_type: MsgType
    payload: dict[str, t.Any]
    meta: dict[str, t.Any]
    hash_list: HashList | None = None
    mac_hex: str | None = None  # HMAC po kanonicznej reprezentacji envelope
bez pola mac_hex

    def to_canonical_bytes(self, include_mac: bool = False) -> bytes:
        obj = {
            "msg_type": self.msg_type.value,
            "payload": self.payload,
            "meta": self.meta,
            "hash_list": None if self.hash_list is None else {
                "digests": {k.value: v for k, v in
self.hash_list.digests.items()}
            },
        }
        if include_mac:
            obj["mac_hex"] = self.mac_hex
        return canonical_json_bytes(obj)

    def with_hmac(self, key: bytes, digestmod: str = "sha256") -> "Envelope":
        base = self.to_canonical_bytes(include_mac=False)
        tag = hmac.new(key, base, digestmod=getattr(hashlib,
digestmod)).hexdigest()
        return dataclasses.replace(self, mac_hex=tag)

    def verify_hmac(self, key: bytes, digestmod: str = "sha256") -> bool:
        if self.mac_hex is None:
            return False
        base = self.to_canonical_bytes(include_mac=False)
        expected = hmac.new(key, base, digestmod=getattr(hashlib,
digestmod)).hexdigest()
        return hmac.compare_digest(expected, self.mac_hex)


# ---------------------------
# Model SBOM (uproszczony)
# ---------------------------

@dataclasses.dataclass(frozen=True)
class Component:
    name: str
    version: str
    deps: tuple[str, ...]  # nazwy komponentów zależnych

    def as_dict(self) -> dict[str, t.Any]:
        return {"name": self.name, "version": self.version, "deps":
list(self.deps)}
```

```python
@dataclasses.dataclass(frozen=True)
class SBOM:
    """
    Minimalny model: lista komponentów i relacje zależności.
    """
    product_id: str
    created_at: str  # ISO8601
    components: tuple[Component, ...]

    def as_dict(self) -> dict[str, t.Any]:
        return {
            "product_id": self.product_id,
            "created_at": self.created_at,
            "components": [c.as_dict() for c in self.components],
        }

    def canonical_bytes(self) -> bytes:
        return canonical_json_bytes(self.as_dict())

    def dependency_graph(self) -> dict[str, set[str]]:
        g: dict[str, set[str]] = {c.name: set(c.deps) for c in
self.components}
        for c in self.components:
            for d in c.deps:
                g.setdefault(d, set())
        return g


# ---------------------------
# Wektory operacyjne podatności + polityki
# ---------------------------

@dataclasses.dataclass(frozen=True)
class VulnerabilityObservation:
    cve: str
    component: str
    # Parametry mogą być None => brak specyfikacji / brak danych
    cvss: float | None
    epss: float | None  # 0..1
    kev: bool | None
    reachable: bool | None
    asset_criticality: float | None  # 0..1
    fix_available: bool | None
    remediation_cost: float | None  # 0..1

    def as_dict(self) -> dict[str, t.Any]:
        return dataclasses.asdict(self)


class DecisionAction(str, enum.Enum):
    DEFER = "defer"
```

```
    SCHEDULED = "scheduled"
    OUT_OF_CYCLE = "out_of_cycle"
    IMMEDIATE = "immediate"


@dataclasses.dataclass(frozen=True)
class PolicyResult:
    priority_score: float
    action: DecisionAction
    reasons: tuple[str, ...]


class Policy(t.Protocol):
    def evaluate(self, v: VulnerabilityObservation) -> PolicyResult:
        ...


@dataclasses.dataclass(frozen=True)
class WeightedSumPolicy:
    """
    Przykładowa polityka punktowa: wagi są parametrami (brak specyfikacji
normatywnej).
    """
    w_cvss: float = 0.40
    w_epss: float = 0.35
    w_kev: float = 0.15
    w_reach: float = 0.05
    w_crit: float = 0.10
    w_fix: float = 0.05
    w_cost: float = 0.10  # odejmowane

    # Progi eskalacji (parametry)
    thr_immediate: float = 0.85
    thr_out_of_cycle: float = 0.65
    thr_scheduled: float = 0.40

    def _n(self, x: float | None, default: float = 0.0) -> float:
        return default if x is None else x

    def evaluate(self, v: VulnerabilityObservation) -> PolicyResult:
        reasons: list[str] = []

        cvss_n = self._n(v.cvss, 0.0) / 10.0  # normalizacja 0..1
        epss_n = self._n(v.epss, 0.0)
        kev_n = 1.0 if (v.kev is True) else 0.0
        reach_n = 1.0 if (v.reachable is True) else 0.0
        crit_n = self._n(v.asset_criticality, 0.0)
        fix_n = 1.0 if (v.fix_available is True) else 0.0
        cost_n = self._n(v.remediation_cost, 0.0)

        score = (
```

```
            self.w_cvss * cvss_n +
            self.w_epss * epss_n +
            self.w_kev * kev_n +
            self.w_reach * reach_n +
            self.w_crit * crit_n +
            self.w_fix * fix_n -
            self.w_cost * cost_n
        )
        score = max(0.0, min(1.0, score))

        if v.cvss is None:
            reasons.append("cvss=brak specyfikacji")
        if v.epss is None:
            reasons.append("epss=brak specyfikacji")
        if v.kev is None:
            reasons.append("kev=brak specyfikacji")
        if v.reachable is None:
            reasons.append("reachable=brak specyfikacji")
        if v.asset_criticality is None:
            reasons.append("asset_criticality=brak specyfikacji")
        if v.fix_available is None:
            reasons.append("fix_available=brak specyfikacji")
        if v.remediation_cost is None:
            reasons.append("remediation_cost=brak specyfikacji")
        if v.kev is True:
            reasons.append("KEV=true (sygnał eksploatacji)")
        if v.epss is not None and v.epss >= 0.5:
            reasons.append("EPSS>=0.5 (wysokie prawdopodobieństwo)")
        if v.cvss is not None and v.cvss >= 9.0:
            reasons.append("CVSS>=9.0 (krytyczne)")

        if score >= self.thr_immediate:
            action = DecisionAction.IMMEDIATE
        elif score >= self.thr_out_of_cycle:
            action = DecisionAction.OUT_OF_CYCLE
        elif score >= self.thr_scheduled:
            action = DecisionAction.SCHEDULED
        else:
            action = DecisionAction.DEFER

        return PolicyResult(priority_score=score, action=action,
reasons=tuple(reasons))


@dataclasses.dataclass(frozen=True)
class SimplifiedSSVCPolicy:
    """
    Uproszczona (edukacyjna) wersja drzewiastej decyzji SSVC.
    NIE jest to pełna implementacja SSVC; to przybliżenie pojęcia:
    - ujmuje kontekst (krytyczność, reachability, exploitability),
    - zwraca kategorie akcji.
```

```
    """
    def evaluate(self, v: VulnerabilityObservation) -> PolicyResult:
        reasons: list[str] = []

        # Braki danych są jawne
        if v.kev is None:
            reasons.append("kev=brak specyfikacji")
        if v.reachable is None:
            reasons.append("reachable=brak specyfikacji")
        if v.asset_criticality is None:
            reasons.append("asset_criticality=brak specyfikacji")
        if v.fix_available is None:
            reasons.append("fix_available=brak specyfikacji")

        kev = (v.kev is True)
        reachable = (v.reachable is True)
        crit = 0.0 if v.asset_criticality is None else v.asset_criticality
        fix = (v.fix_available is True)

        # Exploitability proxy: EPSS jeśli jest, inaczej 0
        epss = 0.0 if v.epss is None else v.epss

        # Decyzja (uproszczenie):
        if kev and reachable and crit >= 0.7 and fix:
            action = DecisionAction.IMMEDIATE
            reasons.append("KEV & reachable & high criticality & fix")
        elif (epss >= 0.5 and reachable and fix) or (v.cvss is not None and
v.cvss >= 9.0 and fix):
            action = DecisionAction.OUT_OF_CYCLE
            reasons.append("high exploitability or critical CVSS with fix")
        elif reachable and fix:
            action = DecisionAction.SCHEDULED
            reasons.append("reachable with fix available")
        else:
            action = DecisionAction.DEFER
            reasons.append("not reachable and/or no fix")

        # Score równoległy (dla porządkowania) – heurystyka
        score = 0.0
        score += 0.45 * (0.0 if v.cvss is None else v.cvss / 10.0)
        score += 0.35 * epss
        score += 0.20 * (1.0 if kev else 0.0)
        score = max(0.0, min(1.0, score))

        return PolicyResult(priority_score=score, action=action,
reasons=tuple(reasons))


# ---------------------------
# Planowanie sekwencji remediacji z zależnościami
# ---------------------------
```

```
@dataclasses.dataclass(frozen=True)
class RemediationTask:
    task_id: str
    component: str
    cve: str
    priority: float
    action: DecisionAction
    prerequisites: tuple[str, ...]  # task_id

    def as_dict(self) -> dict[str, t.Any]:
        return dataclasses.asdict(self)


def topo_schedule(tasks: list[RemediationTask]) -> list[RemediationTask]:
    """
    Topologiczne uporządkowanie z preferencją wyższego priorytetu.
    """
    by_id = {t.task_id: t for t in tasks}
    indeg = {t.task_id: 0 for t in tasks}
    succ: dict[str, set[str]] = defaultdict(set)

    for tsk in tasks:
        for pre in tsk.prerequisites:
            if pre not in by_id:

# brak specyfikacji: nieznany prerequisite -> ignorujemy, ale w realnym
systemie to błąd
                continue
            succ[pre].add(tsk.task_id)
            indeg[tsk.task_id] += 1

    # Kolejka "gotowych" – wybieramy max priorytet
    ready: list[str] = [tid for tid, d in indeg.items() if d == 0]
    # sort malejąco po priorytecie
    ready.sort(key=lambda tid: by_id[tid].priority, reverse=True)

    out: list[RemediationTask] = []
    while ready:
        tid = ready.pop(0)
        out.append(by_id[tid])
        for nxt in sorted(list(succ.get(tid, set())), key=lambda x:
by_id[x].priority, reverse=True):
            indeg[nxt] -= 1
            if indeg[nxt] == 0:
                # wstaw z zachowaniem sortowania po priorytecie
                inserted = False
                for i in range(len(ready)):
                    if by_id[nxt].priority > by_id[ready[i]].priority:
                        ready.insert(i, nxt)
                        inserted = True
```

```
                    break
            if not inserted:
                ready.append(nxt)

    if len(out) != len(tasks):
        # cykl w zależnościach -> brak specyfikacji, ale sygnalizujemy
        raise ValueError("Cykliczne zależności w zadaniach remediacji")
    return out


# ---------------------------
# Alerty
# ---------------------------

@dataclasses.dataclass(frozen=True)
class Alert:
    kind: str
    severity: str
    message: str
    meta: dict[str, t.Any]

    def as_dict(self) -> dict[str, t.Any]:
        return dataclasses.asdict(self)


# ---------------------------
# Pipeline (message bus)
# ---------------------------

class Pipeline:
    def __init__(self, hmac_key: bytes, policy: Policy):
        self.q: "queue.Queue[Envelope]" = queue.Queue()
        self.hmac_key = hmac_key
        self.policy = policy

        self.alerts: list[Alert] = []
        self.last_sbom: SBOM | None = None
        self.last_sbom_hash: HashList | None = None
        self.vulns: list[VulnerabilityObservation] = []
        self.plan: list[RemediationTask] = []

    def publish(self, env: Envelope) -> None:
        self.q.put(env)

    def run(self) -> None:
        while not self.q.empty():
            env = self.q.get()
            self._handle(env)

    def _handle(self, env: Envelope) -> None:
        # weryfikacja HMAC komunikatu (uwierzytelnienie kanału)
```

```
        if not env.verify_hmac(self.hmac_key):
            self.alerts.append(Alert(
                kind="integrity",
                severity="high",

message="HMAC komunikatu niepoprawny (możliwa manipulacja lub błędny
klucz).",
                meta={"msg_type": env.msg_type.value}
            ))
            return

        if env.msg_type == MsgType.SBOM_CREATED:
            # payload: {"sbom": <dict>}
            sbom_dict = env.payload["sbom"]
            sbom_bytes = canonical_json_bytes(sbom_dict)

            if env.hash_list is None or not env.hash_list.verify(sbom_bytes):
                self.alerts.append(Alert(
                    kind="integrity",
                    severity="high",
                    message="Niezgodność list hash dla SBOM (integrity
fail).",
                    meta={"product_id": sbom_dict.get("product_id")}
                ))
                # emit INTEGRITY_FAIL
                fail = Envelope(
                    msg_type=MsgType.INTEGRITY_FAIL,
                    payload={"product_id": sbom_dict.get("product_id")},
                    meta={"ts": dt.datetime.utcnow().isoformat()},
                    hash_list=None
                ).with_hmac(self.hmac_key)
                self.publish(fail)
                return

            # zaakceptuj
            self.last_sbom = SBOM(
                product_id=sbom_dict["product_id"],
                created_at=sbom_dict["created_at"],
                components=tuple(
                    Component(name=c["name"], version=c["version"],
deps=tuple(c.get("deps", [])))
                    for c in sbom_dict["components"]
                )
            )
            self.last_sbom_hash = env.hash_list

            ok = Envelope(
                msg_type=MsgType.SBOM_VERIFIED,
                payload={"product_id": self.last_sbom.product_id},
                meta={"ts": dt.datetime.utcnow().isoformat()},
                hash_list=self.last_sbom_hash,
```

```
            ).with_hmac(self.hmac_key)
            self.publish(ok)

        elif env.msg_type == MsgType.SBOM_VERIFIED:
            # Symuluj skan podatności i korelację (w realu: narzędzia SCA +
zewnętrzne feedy)
            product_id = env.payload["product_id"]
            vulns = self._simulate_vuln_scan(product_id=product_id)
            for v in vulns:
                msg = Envelope(
                    msg_type=MsgType.VULN_OBSERVED,
                    payload={"vuln": v.as_dict()},
                    meta={"ts": dt.datetime.utcnow().isoformat()},
                ).with_hmac(self.hmac_key)
                self.publish(msg)

        elif env.msg_type == MsgType.VULN_OBSERVED:
            v = VulnerabilityObservation(**env.payload["vuln"])
            self.vulns.append(v)

            res = self.policy.evaluate(v)
            # prosta reguła alertu: natychmiast/out-of-cycle
            if res.action in (DecisionAction.IMMEDIATE,
DecisionAction.OUT_OF_CYCLE):
                self.alerts.append(Alert(
                    kind="vulnerability",
                    severity="high" if res.action ==
DecisionAction.IMMEDIATE else "medium",
                    message=f"Eskalacja {v.cve} w {v.component}:
{res.action.value} (score={res.priority_score:.2f})",
                    meta={"reasons": list(res.reasons)}
                ))

            # Gdy zebrano pewną liczbę obserwacji, generuj plan (symulacja)
            if len(self.vulns) >= 5 and self.last_sbom is not None and not
self.plan:
                self.plan = self._build_plan(self.last_sbom, self.vulns)
                plan_msg = Envelope(
                    msg_type=MsgType.REMEDIATION_PLAN,
                    payload={"plan": [t.as_dict() for t in self.plan]},
                    meta={"ts": dt.datetime.utcnow().isoformat()},
                ).with_hmac(self.hmac_key)
                self.publish(plan_msg)

        elif env.msg_type == MsgType.REMEDIATION_PLAN:
            # W realu: przekazanie do orkiestratora. Tu: walidacja spójności
            _ = env.payload["plan"]
        elif env.msg_type == MsgType.INTEGRITY_FAIL:
            self.alerts.append(Alert(
                kind="integrity",
                severity="high",
```

```
                message="Odebrano sygnał INTEGRITY_FAIL: pipeline powinien
zablokować konsumpcję SBOM.",
                meta=env.payload
            ))
        else:
            self.alerts.append(Alert(
                kind="system",
                severity="low",
                message=f"Nieobsługiwany typ komunikatu:
{env.msg_type.value}",
                meta={}
            ))

    def _simulate_vuln_scan(self, product_id: str) ->
list[VulnerabilityObservation]:
        """
        Dane syntetyczne, aby zilustrować polityki.
        """
        assert self.last_sbom is not None
        comps = [c.name for c in self.last_sbom.components]
        rnd = random.Random(1337)

        out: list[VulnerabilityObservation] = []
        for i in range(7):
            comp = rnd.choice(comps)
            cve = f"CVE-20{rnd.randint(20,26)}-{rnd.randint(1000,99999)}"
            cvss = rnd.choice([None, round(rnd.uniform(3.0, 9.8), 1)])
            epss = rnd.choice([None, round(rnd.uniform(0.01, 0.85), 2)])
            kev = rnd.choice([None, False, True])
            reachable = rnd.choice([None, False, True])
            criticality = rnd.choice([None, round(rnd.uniform(0.2, 0.95),
2)])
            fix_av = rnd.choice([None, False, True])
            cost = rnd.choice([None, round(rnd.uniform(0.05, 0.8), 2)])

            out.append(VulnerabilityObservation(
                cve=cve,
                component=comp,
                cvss=cvss,
                epss=epss,
                kev=kev,
                reachable=reachable,
                asset_criticality=criticality,
                fix_available=fix_av,
                remediation_cost=cost
            ))
        return out

    def _build_plan(self, sbom: SBOM, vulns: list[VulnerabilityObservation])
-> list[RemediationTask]:
        g = sbom.dependency_graph()
```

```
        # Map komponent -> zależne komponenty (reverse edges), aby modelować
kolejność rebuild/upgrade
        reverse: dict[str, set[str]] = defaultdict(set)
        for u, deps in g.items():
            for d in deps:
                reverse[d].add(u)

        tasks: list[RemediationTask] = []
        for idx, v in enumerate(vulns[:10]):
            res = self.policy.evaluate(v)
            task_id = f"T{idx:02d}"

# Przykładowa zależność: jeśli komponent X ma zależnych, to patch X wymaga
później rebuild zależnych,
            # ale w tej symulacji modelujemy odwrotnie: najpierw patch
zależnych bibliotek, potem aplikacja.
            prereq: list[str] = []

# Jeśli komponent ma zależności, to zadanie zależy od patchy tych zależności,
jeśli takie istnieją.
            for dep in g.get(v.component, set()):
                # znajdź zadanie dla dep, jeśli zostało utworzone
                for tsk in tasks:
                    if tsk.component == dep:
                        prereq.append(tsk.task_id)

            tasks.append(RemediationTask(
                task_id=task_id,
                component=v.component,
                cve=v.cve,
                priority=res.priority_score,
                action=res.action,
                prerequisites=tuple(prereq)
            ))

        ordered = topo_schedule(tasks)
        return ordered


# ---------------------------
# Demo runner + tests
# ---------------------------

def build_demo_sbom() -> SBOM:
    now = dt.datetime.utcnow().replace(microsecond=0).isoformat() + "Z"
    comps = (
        Component("app", "1.0.0", ("libA", "libB")),
        Component("libA", "2.1.0", ("libC",)),
        Component("libB", "3.0.2", ()),
        Component("libC", "0.9.9", ()),
```

```
    )
    return SBOM(product_id="product-xyz", created_at=now, components=comps)


def run_demo() -> None:
    key = b"demo_shared_secret_key__32bytes_min_pref"
    policy = WeightedSumPolicy()
    pipe = Pipeline(hmac_key=key, policy=policy)

    sbom = build_demo_sbom()
    sbom_bytes = sbom.canonical_bytes()
    hl = HashList.compute(sbom_bytes, [HashAlg.SHA256, HashAlg.SHA3_256])

    msg = Envelope(
        msg_type=MsgType.SBOM_CREATED,
        payload={"sbom": sbom.as_dict()},
        meta={"ts": dt.datetime.utcnow().isoformat()},
        hash_list=hl,
    ).with_hmac(key)
    pipe.publish(msg)

    pipe.run()

    print("\n=== ALERTY ===")
    for a in pipe.alerts:
        print(json.dumps(a.as_dict(), ensure_ascii=False, indent=2))

    if pipe.plan:
        print("\n=== PLAN REMEDIACJI (TOPO + PRIORYTET) ===")
        for tsk in pipe.plan:
            print(f"{tsk.task_id}: {tsk.component} {tsk.cve}
action={tsk.action.value} prio={tsk.priority:.2f}
prereq={list(tsk.prerequisites)}")


class TestSBOMCryptoanalysis(unittest.TestCase):
    def test_hashlist_roundtrip(self) -> None:
        b = b"abc"
        hl = HashList.compute(b, [HashAlg.SHA256, HashAlg.SHA3_256])
        self.assertTrue(hl.verify(b))
        self.assertFalse(hl.verify(b"abd"))

    def test_envelope_hmac(self) -> None:
        key = b"k" * 32
        env = Envelope(
            msg_type=MsgType.ALERT,
            payload={"x": 1},
            meta={"ts": "2026-01-01T00:00:00Z"},
            hash_list=None
        ).with_hmac(key)
        self.assertTrue(env.verify_hmac(key))
```

```
        self.assertFalse(env.verify_hmac(b"wrong" * 8))

    def test_pipeline_detects_tamper(self) -> None:
        key = b"k" * 32
        pipe = Pipeline(hmac_key=key, policy=WeightedSumPolicy())
        sbom = build_demo_sbom()
        sbom_bytes = sbom.canonical_bytes()
        hl = HashList.compute(sbom_bytes, [HashAlg.SHA256])

        # Tamper: zmień payload, zostaw stare hashe
        tampered = sbom.as_dict()
        tampered["components"][0]["version"] = "9.9.9"

        msg = Envelope(
            msg_type=MsgType.SBOM_CREATED,
            payload={"sbom": tampered},
            meta={"ts": dt.datetime.utcnow().isoformat()},
            hash_list=hl
        ).with_hmac(key)

        pipe.publish(msg)
        pipe.run()

        self.assertTrue(any(a.kind == "integrity" and a.severity == "high"
for a in pipe.alerts))

    def test_plan_is_topologically_sorted(self) -> None:
        # budujemy sztuczne zadania z zależnościami: T1 zależy od T0
        tasks = [
            RemediationTask("T0", "lib", "CVE-1", 0.5,
DecisionAction.SCHEDULED, ()),
            RemediationTask("T1", "app", "CVE-2", 0.9,
DecisionAction.IMMEDIATE, ("T0",)),
        ]
        ordered = topo_schedule(tasks)
        ids = [t.task_id for t in ordered]
        self.assertEqual(ids, ["T0", "T1"])


if __name__ == "__main__":
    if "--demo" in sys.argv:
        run_demo()
    else:
        unittest.main(argv=[sys.argv[0]])
```

Co demonstruje skrypt (w sensie naukowym):

- *Formalną separację* integralności payloadu (HashList) od uwierzytelnienia kanału (HMAC), zgodnie z rozróżnieniem RFC 2104 / analizą HMAC. [6]

- *Jawność „brak specyfikacji”* w wektorach: brak CVSS/EPSS/KEV/reachability jest modelowany jako `None`, co wpływa na uzasadnienia decyzji (reasons) i jest kluczowe w naukowej analizie błędów decyzyjnych. [61]
- *Sekwencjonowanie remediacji* jako sortowanie topologiczne z preferencją priorytetu, czyli implementację idei, że decyzja „co najpierw” jest jednocześnie funkcją ryzyka i zależności (SBOM graf). Kontekst zależności jako problem operacyjny jest częścią współczesnej analityki SBOM i harmonizacji (SEI Plugfest 2024). [62]

Skrypt demonstracyjny: length extension na konstrukcji iteracyjnej i kontrast HMAC

Wymóg length extension realizuję poprzez *kontrolowany, edukacyjny* hash iteracyjny (toy MD), aby nie polegać na zewnętrznych bibliotekach ani nie implementować pełnego SHA-256. Teza naukowa: własność length extension wynika z natury iteracji i ujawnienia stanu pośredniego (Merkle 1989; Damgård 1989; przeglądy projektowania hash). [55]

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
length_extension_demo.py

Edukacyjna demonstracja podatności "length extension" dla iteracyjnego hasha
(toy Merkle–Damgård)
oraz kontrast z HMAC.

Uruchomienie:
  python length_extension_demo.py
"""

from __future__ import annotations

import hashlib
import hmac
import struct
import unittest


def md_pad(message_len_bytes: int, block_size: int = 64) -> bytes:
    """
    Klasyczne padding + długość (w stylu Merkle–Damgård strengthening):
    0x80, zera, a na końcu długość w bitach w 8 bajtach (big endian).
    """
    bit_len = message_len_bytes * 8
    pad = b"\x80"
    # wypełnij do długości ≡ 56 mod 64 (zostaw 8 bajtów na length)
    while (message_len_bytes + len(pad)) % block_size != (block_size - 8):
        pad += b"\x00"
    pad += struct.pack(">Q", bit_len)
    return pad
```

```python
def toy_compress(state: bytes, block: bytes) -> bytes:
    """
    Kompresja: SHA-256(state || block) i obcięcie do 16 bajtów.
    To nie jest kryptograficznie poprawny hash produkcyjny,
    ale zachowuje własność iteracyjnej konstrukcji.
    """
    return hashlib.sha256(state + block).digest()[:16]


def toy_md_hash(message: bytes, iv: bytes = b"\x00" * 16) -> bytes:
    block_size = 64
    m = message + md_pad(len(message), block_size=block_size)
    state = iv
    for i in range(0, len(m), block_size):
        state = toy_compress(state, m[i:i + block_size])
    return state


def toy_md_hash_with_state(extension: bytes, prior_state: bytes,
total_len_before_ext: int) -> bytes:
    """
    Kontynuacja hashowania od znanego stanu pośredniego (prior_state),
    zakładając, że przed extension przetworzono message || pad(message).
    total_len_before_ext to długość bajtów (message + pad(message)).
    """
    block_size = 64
    m = extension + md_pad(total_len_before_ext + len(extension),
block_size=block_size)
    state = prior_state
    for i in range(0, len(m), block_size):
        state = toy_compress(state, m[i:i + block_size])
    return state


def forge_length_extension(known_digest: bytes, known_message_len: int,
extension: bytes) -> tuple[bytes, bytes]:
    """
    Atak: mając digest=H(message) oraz długość message, budujemy:
      forged_message = message || pad(message) || extension
    i obliczamy digest forged_message bez znajomości message,
    zakładając że known_digest jest stanem po przetworzeniu message||pad.
    """
    pad = md_pad(known_message_len, 64)
    total_before_ext = known_message_len + len(pad)
    forged_digest = toy_md_hash_with_state(extension, known_digest,
total_before_ext)
    return pad + extension, forged_digest
```

```
class TestLengthExtension(unittest.TestCase):
    def test_length_extension_works_on_toy_hash(self) -> None:
        secret = b"SECRET_KEY_"  # symulacja tajnego prefiksu
        msg = b"role=user&exp=2026-12-31"
        ext = b"&role=admin"

        # Naiwny MAC: H(secret || msg)
        naive_mac = toy_md_hash(secret + msg)

        # Atakujący zna msg i naive_mac, ale nie zna secret.
        # Atakujący zgaduje len(secret) (często możliwe lub ograniczone).
        guessed_secret_len = len(secret)
        forged_suffix, forged_mac = forge_length_extension(
            known_digest=naive_mac,
            known_message_len=guessed_secret_len + len(msg),
            extension=ext
        )

        # forged message, którą "widzi" serwer:
        forged_message = msg + forged_suffix

        # Serwer weryfikuje naiwnie: H(secret||forged_message)
        server_mac = toy_md_hash(secret + forged_message)
        self.assertEqual(server_mac, forged_mac)

    def test_hmac_blocks_length_extension(self) -> None:
        key = b"SECRET_KEY_"
        msg = b"role=user&exp=2026-12-31"
        ext = b"&role=admin"

        # HMAC na bazie SHA-256 (produkcyjnie)
        tag = hmac.new(key, msg, digestmod=hashlib.sha256).digest()

        # Atakujący nie potrafi "przedłużyć" HMAC bez klucza.
        # (Tu tylko pokazujemy, że nie istnieje analogiczny interfejs
kontynuacji ze znanego stanu.)
        forged_tag_guess = hmac.new(key, msg + ext,
digestmod=hashlib.sha256).digest()
        self.assertNotEqual(tag, forged_tag_guess)


if __name__ == "__main__":
    unittest.main(argv=["length_extension_demo.py"])
```

Interpretacja wyników: w modelu iteracyjnym, jeżeli system pipeline SBOM użyje naiwnych konstrukcji MAC opartych o sam hash, to logika komunikatów może zostać sfałszowana przez napastnika (przekazanie komunikatu o innym payloadzie przy zachowaniu „poprawnego” tagu). Właśnie dlatego standardowe rozwiązania stosują HMAC (RFC 2104; Bellare i in., 1996). [6]

Skrypt do wykresów: porównanie polityk decyzyjnych na danych syntetycznych

Wizualizacja jest użyteczna, gdy porównuje się *ranking* podatności wynikający z różnych polityk. Poniższy skrypt generuje dane syntetyczne i rysuje wykres (wymaga `matplotlib`; brak specyfikacji zależności w standardach, ale jest to standard w analizie naukowej danych).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
plots_demo.py

Porównanie rankingów priorytetu dla dwóch polityk:
- WeightedSumPolicy
- SimplifiedSSVCPolicy

Wymaga:
  pip install matplotlib

Uruchomienie:
  python plots_demo.py
"""

from __future__ import annotations

import random
import dataclasses
import matplotlib.pyplot as plt  # zewnętrzna zależność
from typing import Optional


@dataclasses.dataclass(frozen=True)
class V:
    cve: str
    cvss: Optional[float]
    epss: Optional[float]
    kev: Optional[bool]
    reachable: Optional[bool]
    criticality: Optional[float]
    fix: Optional[bool]
    cost: Optional[float]


def norm(x: Optional[float], default: float = 0.0) -> float:
    return default if x is None else x


def weighted_sum(v: V) -> float:
    cvss_n = norm(v.cvss) / 10.0
    epss_n = norm(v.epss)
```

```
    kev_n = 1.0 if v.kev is True else 0.0
    reach_n = 1.0 if v.reachable is True else 0.0
    crit_n = norm(v.criticality)
    fix_n = 1.0 if v.fix is True else 0.0
    cost_n = norm(v.cost)

    score = (
        0.40 * cvss_n +
        0.35 * epss_n +
        0.15 * kev_n +
        0.05 * reach_n +
        0.10 * crit_n +
        0.05 * fix_n -
        0.10 * cost_n
    )
    return max(0.0, min(1.0, score))


def simplified_ssvc_score(v: V) -> float:
    epss = norm(v.epss)
    kev = (v.kev is True)
    score = 0.45 * (norm(v.cvss) / 10.0) + 0.35 * epss + 0.20 * (1.0 if kev
else 0.0)
    return max(0.0, min(1.0, score))


def main() -> None:
    rnd = random.Random(2026)
    vulns = []
    for i in range(12):
        vulns.append(V(
            cve=f"CVE-2026-{10000+i}",
            cvss=rnd.choice([None, round(rnd.uniform(3.0, 9.9), 1)]),
            epss=rnd.choice([None, round(rnd.uniform(0.01, 0.9), 2)]),
            kev=rnd.choice([None, False, True]),
            reachable=rnd.choice([None, False, True]),
            criticality=rnd.choice([None, round(rnd.uniform(0.2, 0.95), 2)]),
            fix=rnd.choice([None, False, True]),
            cost=rnd.choice([None, round(rnd.uniform(0.05, 0.8), 2)]),
        ))

    labels = [v.cve for v in vulns]
    s1 = [weighted_sum(v) for v in vulns]
    s2 = [simplified_ssvc_score(v) for v in vulns]

    # sortujemy po polityce 1, aby porównać rozjazd
    idx = sorted(range(len(vulns)), key=lambda i: s1[i], reverse=True)
    labels = [labels[i] for i in idx]
    s1 = [s1[i] for i in idx]
    s2 = [s2[i] for i in idx]
```

```
    x = list(range(len(labels)))

    plt.figure()
    plt.plot(x, s1, marker="o")
    plt.plot(x, s2, marker="o")
    plt.xticks(x, labels, rotation=70, ha="right")
    plt.ylabel("Priority score [0..1]")
    plt.title("Porównanie rankingów: WeightedSum vs SimplifiedSSVC score")
    plt.tight_layout()
    plt.show()


if __name__ == "__main__":
    main()
```

Wykres ten jest *instrumentem analitycznym* do oceny, jak różne modele decyzyjne zmieniają kolejność remediacji (co jest zgodne z literaturą wskazującą, że CVSS/EPSS/SSVC mierzą różne aspekty i nie powinny być utożsamiane). [63]

Analiza bezpieczeństwa logiki komunikatów i propozycje mitigacji

Powierzchnia ataku specyficzna dla „message logic" w pipeline SBOM

W pipeline SBOM krytyczne jest, że atakujący nie musi „łamać" kryptografii; często wystarczy manipulacja logiką komunikatów, kolejnością lub warstwą reprezentacji.

1. **Zafałszowanie integralności przez kanonikalizację**. Jeśli producer i consumer używają różnych reguł serializacji, hash/podpis nie jest stabilny — może to generować fałszywe alarmy albo luki walidacji. RFC 8785 wskazuje, że kanonikalizacja jest potrzebna, aby operacje kryptograficzne były powtarzalne. [15] Mitigacja: stosować standardowe kanoniczne serializacje (SPDX Canonical Serialisation) oraz formalne reguły kanonikalizacji JSON (JCS), wraz z testami deterministyczności. [10]

2. **Substitution/relocation**: podmiana SBOM przypisanego do artefaktu build (SBOM mówi o innym binarium). ENISA wskazuje potrzebę zgodności hash dokumentu z rzeczywistymi plikami oraz podpisu. [5] Mitigacja: (a) wiązać SBOM z hash artefaktów build (content binding), (b) podpisywać SBOM i weryfikować łańcuch zaufania, (c) stosować atestacje produkcyjne w stylu in-toto (podpisane opisy kroków i artefaktów). [21]

3. **Replay i rollback**: odtworzenie starego komunikatu lub SBOM o poprawnym podpisie/hash, ale nieaktualnego (np. przed naprawą). W zarządzaniu podatnościami ma to efekt bezpośredni: *błędne decyzje priorytetów i alertów*. Mitigacja: wersjonowanie monotoniczne (np. semver+timestamp), rejestry audytowe, mechanizmy anty-rollback znane z projektów zabezpieczających aktualizacje i metadane (idee kompromiso-odporności w systemach łańcucha dostaw) oraz atestacje/provenance dla wersji build. Podejście „pełnołańcuchowe" jest zgodne z argumentacją in-toto, że samo podpisywanie kroków nie wystarcza bez mechanizmu weryfikacji ciągu czynności. [64]

4. **Ataki na uwierzytelnienie komunikatu (MAC misuse)**. Najgroźniejszy błąd projektowy: zastąpienie MAC przez hash (lub użycie naiwnych MAC typu $H(k\|m)$), co dla konstrukcji iteracyjnych prowadzi do length extension. Konieczność HMAC jest opisana w RFC 2104 i pracy Bellare/Canetti/Krawczyk. [6] Mitigacja: konsekwentnie stosować HMAC albo podpis (w zależności od modelu zaufania), plus polityki kryptograficzne (zaakceptowane algorytmy i parametry).

5. **Błędy decyzyjne i „alert fatigue”**. Modele CVSS/EPSS/SSVC/KEV mierzą różne wymiary: CVSS ocenia charakterystyki podatności, EPSS prognozuje prawdopodobieństwo eksploatacji, SSVC to drzewo decyzyjne dopasowane do interesariusza. [65] Jeśli logika komunikatów przyjmuje te sygnały jako równoważne, generuje błędne priorytety i nadmiar alertów. Mitigacja: formalny model wektorowy i jawna polityka agregacji; badania Jacobs i in. pokazują, że predykcja exploitów może poprawić efektywność strategii remediacji, ale nie zastępuje metryk wpływu. [66]

Mitigacje systemowe: od SBOM do atestacji i logów transparentności

- **Podpisy/atestacje i egzekwowanie integralności łańcucha**: in-toto proponuje podpisane deklaracje kroków (layout) i podpisane metadane kroków (link metadata), co tworzy weryfikowalny łańcuch dowodów. [21]
- **Wytyczne dla cyber-SCRM**: NIST SP 800-161r1 dostarcza ram zarządzania ryzykiem łańcucha dostaw na poziomie organizacyjnym, ułatwiając osadzenie SBOM w procesach ryzyka i dowodów. [67]
- **Integracja SBOM z VEX/CSAF**: standard CSAF (OASIS) zawiera profil VEX, umożliwiający dostawcy wskazanie statusu podatności w produkcie, co jest istotne dla zmniejszania fałszywych alarmów w konsumpcji SBOM. [68]
- **Poufność SBOM i redakcja**: praca Ishgair i in. (2025) pokazuje, że logika zaufania i integralności musi uwzględniać selektywne ujawnianie danych (redacted SBOM), co jest krytyczne w realnych relacjach dostawca–odbiorca. [35]

Tabela porównawcza polityk decyzyjnych (priorytetyzacja i alerty)

Polityka	Typ wyniku	Główne dane wejściowe	Korzyści	Ryzyka/ ograniczenia
CVSS-only	score	CVSS v4 (Base/ Threat/Env) [69]	standaryzacja i szeroka dostępność	słabsza informacja o „real-world exploitation”; ryzyko „patching by score”
EPSS-only	probability	EPSS (predykcja) [44]	lepsze ukierunkowanie na eksploatacje w praktyce	nie mierzy wpływu; wymaga rozumienia probabilistyki

Polityka	Typ wyniku	Główne dane wejściowe	Korzyści	Ryzyka/ ograniczenia
KEV-driven	lista flag	KEV (CISA) [45]	prosty sygnał aktywnej eksploatacji	zależne od pokrycia katalogu; możliwy efekt „wszystko z KEV na już”
SSVC	kategoria akcji	kontekst interesariusza; drzewo decyzji [46]	decyzyjność dopasowana do kontekstu	wymaga wdrożenia organizacyjnego i kalibracji
Hybryda CVSS+EPSS+KEV+context	score + reguły	wektory operacyjne (jak w tej ekspertyzie) [70]	redukcja błędów przez „multiple signals”	wagi i progi: brak specyfikacji; ryzyko arbitralności
LEV (NIST)	metryka agregowana w czasie	EPSS w czasie + definicje LEV [71]	ujęcie kumulacji prawdopodobieństwa	wymaga danych czasowych i walidacji branżowej

Wnioski i rekomendacje badawcze

Kluczowa konkluzja brzmi: **cryptoanaliza (rozumiana jako kryptograficzna weryfikacja + analiza logiki komunikatów) wymusza strukturalne zmiany w artefaktach SBOM**: pojawiają się listy hash, reguły kanonikalizacji, podpisy/atestacje, a także formalne wektory operacyjne i polityki decyzyjne determinujące sekwencje remediacji i alerty. Te zjawiska są spójne z (i) zaleceniami walidacji i podpisywania SBOM (ENISA, 2025), (ii) rozwojem standardów BOM (SPDX 3.0.1, ECMA-424 CycloneDX), (iii) potrzebą kryptograficznie weryfikowalnego łańcucha dowodów (in-toto), (iv) rozwojem modeli priorytetyzacji podatności (CVSS/EPSS/SSVC/KEV/LEV). [72]

Rekomendacje operacyjno-badawcze (w duchu „brak specyfikacji” → jawne parametry):

1. **Kanonikalizacja jako element kontraktu**: dla komunikatów i SBOM należy formalnie zdefiniować deterministyczną reprezentację (np. JCS lub canonical serialisation SPDX) i testować ją jako własność systemu, bo brak stabilności bajtowej degraduje sens hash/podpisu. [73]
2. **Rozdzielenie integralności i autentyczności**: hash listy należy traktować jako mechanizm integralności, natomiast autentyczność zapewniać przez HMAC (w domenie zaufanej) lub podpisy/atestacje (w domenie wielopodmiotowej). [74]
3. **Model wektorowy jako „język” polityk**: parametry CVSS/EPSS/KEV/SSVC powinny być jawnie mapowane do wektorów operacyjnych, a reguły priorytetyzacji przedstawiane w formie funkcji lub drzew, co umożliwia audyt, testowanie i porównywanie polityk (zgodnie z ideą testowalnego SSVC). [75]
4. **Badania nad odpornością logiki komunikatów**: potrzebne są eksperymenty (a) nad wpływem replay/rollback i „SBOM drift” na decyzje, (b) nad kryptograficznym audytem redagowanych SBOM (w duchu Petra), (c) nad integracją SBOM z VEX/CSAF w celu redukcji fałszywych alarmów. [76]

5. **Ugruntowanie w procesach łańcucha dostaw**: dla środowisk regulowanych i zamówień publicznych warto łączyć SBOM z wymaganiami dowodowymi, metrykami i terminami remediacji, zgodnie z podejściem operacyjnym i formalizacją łańcuchów dostaw w literaturze polskiej. [77]

Bibliografia i źródła priorytetowe

Poniżej podaję priorytetowe źródła pierwotne (standardy, artykuły konferencyjne/czasopisma) oraz wybrane opracowania instytucjonalne, na których oparto tezy.

Standardy SBOM i przewodniki instytucjonalne - SPDX (2024). *SPDX Specification v3.0.1*. [3]
- NTIA (2021). *The Minimum Elements For a Software Bill of Materials (SBOM)*. [2]
- Ecma International (2024). *ECMA-424, CycloneDX Bill of Materials Specification (1st edition)*. [4]
- ENISA (2025). *SBOM Landscape Analysis – Towards an Implementation Guide* (Public Draft). [78]
- OASIS (2022/2024). *Common Security Advisory Framework (CSAF) v2.x* (z profilem VEX). [79]

Kryptografia aplikacyjna: hashe, MAC, kanonikalizacja - NIST (2015). *FIPS 180-4: Secure Hash Standard (SHS)*. [39]
- NIST (2015). *FIPS 202: SHA-3 Standard*. [40]
- IETF (1997). *RFC 2104: HMAC: Keyed-Hashing for Message Authentication*. [41]
- Bellare, M., Canetti, R., Krawczyk, H. (1996). *Keying Hash Functions for Message Authentication (HMAC)*. [19]
- IETF (2019). *RFC 8785: JSON Canonicalization Scheme (JCS)*. [15]

Podstawy konstrukcji iteracyjnych i znane klasy ataków - Ralph C. Merkle [80] (1989). *A Certified Digital Signature* (CRYPTO'89; klasyczne odniesienie w LNCS). [81]
- Ivan Bjerre Damgård [82] (1989). *A Design Principle for Hash Functions* (CRYPTO'89). [83]
- John Kelsey [84] & Bruce Schneier [85] (2005). *Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work* (EUROCRYPT 2005). [54]
- NIST (2017). *Research Results on SHA-1 Collisions* (informacja o praktycznej demonstracji). [86]

Bezpieczeństwo łańcucha dostaw i weryfikowalne metadane - Torres-Arias, S. i in. (2019). *in-toto: Providing farm-to-table guarantees for bits and bytes* (USENIX Security). [21]
- NIST (2022). *SP 800-161r1: Cybersecurity Supply Chain Risk Management Practices for Systems and Organizations*. [67]

Modele decyzyjne i metryki podatności - FIRST (2023). *CVSS v4.0 Specification*. [87]
- Jacobs, J., Romanosky, S., Adjerid, I., Baker, W. (2020). *Improving vulnerability remediation through better exploit prediction* (Journal of Cybersecurity). [88]
- Jacobs, J. i in. (2019). *Exploit Prediction Scoring System (EPSS)* (arXiv). [89]
- Spring, J. M. i in. (2021). *Prioritizing vulnerability response: A stakeholder-specific vulnerability categorization (SSVC) v2.0* (SEI). [90]
- Mell, P. (2025). *Likely Exploited Vulnerabilities (LEV): A Proposed Metric for Vulnerability Exploitation Probability* (NIST CSWP 41). [91]
- CISA (na bieżąco). *Known Exploited Vulnerabilities (KEV) Catalog*. [29]

Badania empiryczne SBOM - Xia, B. i in. (2023). *An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead* (ICSE; dostępny także jako arXiv). [32]
- Stalnaker, T. i in. (2024). *BOMs Away! …* (ICSE'24). [34]

- Ishgair, E. A. i in. (2025). *Trustworthy and Confidential SBOM Exchange (Petra)* (arXiv). [35]
- SEI (2025). *SBOM Harmonization Plugfest 2024* (raport). [92]

Źródła polskojęzyczne / regionalne - PIB NASK [93]: Liderman, K., Księżopolski, A. (2025). *Cyberbezpieczeństwo łańcuchów dostaw* (artykuł). [36]
- Opracowanie administracji publicznej dot. zabezpieczenia łańcucha dostaw w zamówieniach (w tym SBOM, metryki i terminy zależne od CVSS) (2025). [37]

---

[1] [63] https://www.cert-ist.com/public/en/SO_detail?code=ssvc&format=html
https://www.cert-ist.com/public/en/SO_detail?code=ssvc&format=html

[2] [9] https://www.ntia.gov/files/ntia/publications/sbom_minimum_elements_report.pdf
https://www.ntia.gov/files/ntia/publications/sbom_minimum_elements_report.pdf

[3] [20] [28] https://spdx.dev/wp-content/uploads/sites/31/2024/12/SPDX-3.0.1-1.pdf
https://spdx.dev/wp-content/uploads/sites/31/2024/12/SPDX-3.0.1-1.pdf

[4] [11] https://ecma-international.org/wp-content/uploads/ECMA-424_1st_edition_june_2024.pdf
https://ecma-international.org/wp-content/uploads/ECMA-424_1st_edition_june_2024.pdf

[5] [59] [72] [78] https://www.enisa.europa.eu/sites/default/files/2025-12/SBOM%20Analysis%20-%20Towards%20an%20Implementation%20Guide_v1.20-Published.pdf
https://www.enisa.europa.eu/sites/default/files/2025-12/SBOM%20Analysis%20-%20Towards%20an%20Implementation%20Guide_v1.20-Published.pdf

[6] [18] [24] [41] [51] [74] RFC 2104 - HMAC: Keyed-Hashing for Message Authentication
https://datatracker.ietf.org/doc/html/rfc2104?utm_source=chatgpt.com

[7] [23] [43] [65] https://www.first.org/cvss/specification-document
https://www.first.org/cvss/specification-document

[8] [10] https://spdx.github.io/canonical-serialisation/
https://spdx.github.io/canonical-serialisation/

[12] Cryptography Bill of Materials (CBOM)
https://cyclonedx.org/capabilities/cbom/?utm_source=chatgpt.com

[13] [67] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-161r1.pdf
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-161r1.pdf

[14] [22] https://cyclonedx.org/use-cases/authenticity-verification/
https://cyclonedx.org/use-cases/authenticity-verification/

[15] [60] [73] https://www.rfc-editor.org/rfc/rfc8785
https://www.rfc-editor.org/rfc/rfc8785

[16] [39] [80] fips pub 180-4 - federal information processing standards ...
https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.180-4.pdf?utm_source=chatgpt.com

[17] [40] [84] FIPS PUB 202 - NIST Technical Series Publications
https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.202.pdf?utm_source=chatgpt.com

[19] Keying Hash Functions for Message Authentication
https://kantarcioglu.net/courses/crypto09s_files/hmac-proof.pdf?utm_source=chatgpt.com

[21] [42] [64] https://www.usenix.org/system/files/sec19-torres-arias.pdf
https://www.usenix.org/system/files/sec19-torres-arias.pdf

25 33 61 66 70 88 https://academic.oup.com/cybersecurity/article-pdf/6/1/tyaa015/33746021/tyaa015.pdf
https://academic.oup.com/cybersecurity/article-pdf/6/1/tyaa015/33746021/tyaa015.pdf

26 53 56 86 https://csrc.nist.gov/news/2017/research-results-on-sha-1-collisions
https://csrc.nist.gov/news/2017/research-results-on-sha-1-collisions

27 46 75 90 93 https://www.sei.cmu.edu/documents/606/2021_019_001_653461.pdf
https://www.sei.cmu.edu/documents/606/2021_019_001_653461.pdf

29 45 https://www.cisa.gov/known-exploited-vulnerabilities-catalog
https://www.cisa.gov/known-exploited-vulnerabilities-catalog

30 71 91 https://nvlpubs.nist.gov/nistpubs/cswp/nist.cswp.41.pdf
https://nvlpubs.nist.gov/nistpubs/cswp/nist.cswp.41.pdf

31 62 92 https://www.sei.cmu.edu/documents/6302/SBOM_Harmonization_Plugfest_2024.pdf
https://www.sei.cmu.edu/documents/6302/SBOM_Harmonization_Plugfest_2024.pdf

32 48 https://arxiv.org/pdf/2301.05362
https://arxiv.org/pdf/2301.05362

34 https://ojcchar.github.io/files/27-icse24-sboms.pdf
https://ojcchar.github.io/files/27-icse24-sboms.pdf

35 76 https://arxiv.org/pdf/2509.13217
https://arxiv.org/pdf/2509.13217

36 https://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-b8cee871-0ef5-488c-9bda-5a6d196ecb5f/c/liderman_cyberbezpieczenstwo.pdf
https://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-b8cee871-0ef5-488c-9bda-5a6d196ecb5f/c/liderman_cyberbezpieczenstwo.pdf

37 77 https://www.gov.pl/attachment/e887426b-1dd5-4ff9-b0e3-860e4f67d0cf
https://www.gov.pl/attachment/e887426b-1dd5-4ff9-b0e3-860e4f67d0cf

38 https://spdx.github.io/spdx-spec/v3.0.1/serializations/
https://spdx.github.io/spdx-spec/v3.0.1/serializations/

44 89 https://arxiv.org/abs/1908.04856
https://arxiv.org/abs/1908.04856

47 FIPS 180-4, Secure Hash Standard (SHS) - NIST CSRC
https://csrc.nist.gov/pubs/fips/180-4/upd1/final?utm_source=chatgpt.com

49 55 81 82 85 https://link.springer.com/chapter/10.1007/0-387-34805-0_21
https://link.springer.com/chapter/10.1007/0-387-34805-0_21

50 FIPS 202, SHA-3 Standard: Permutation-Based Hash and ...
https://csrc.nist.gov/pubs/fips/202/final?utm_source=chatgpt.com

52 hmac — Keyed-Hashing for Message Authentication — Python 3.14 ...
https://docs.python.org/3/library/hmac.html?utm_source=chatgpt.com

54 https://www.schneier.com/wp-content/uploads/2016/02/paper-preimages.pdf
https://www.schneier.com/wp-content/uploads/2016/02/paper-preimages.pdf

57 https://link.springer.com/chapter/10.1007/11426639_28
https://link.springer.com/chapter/10.1007/11426639_28

58 in-toto: Providing farm-to-table guarantees for bits and bytes
https://www.usenix.org/system/files/sec19-torres-arias.pdf?utm_source=chatgpt.com

68 79 https://docs.oasis-open.org/csaf/csaf/v2.0/os/csaf-v2.0-os.html
https://docs.oasis-open.org/csaf/csaf/v2.0/os/csaf-v2.0-os.html

69 87 https://www.first.org/cvss/v4-0/cvss-v40-specification.pdf
https://www.first.org/cvss/v4-0/cvss-v40-specification.pdf

83 https://link.springer.com/chapter/10.1007/0-387-34805-0_39
https://link.springer.com/chapter/10.1007/0-387-34805-0_39