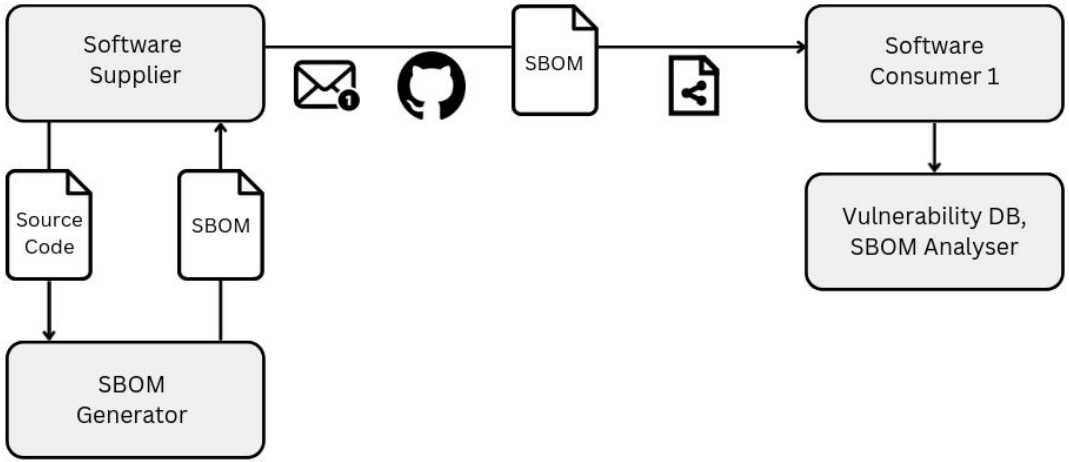
zkSBOM: Privacy-Preserving SBOM Sharing with Zero-Knowledge Sets

Tom Sorger
sorger@kth.se

SBOM Sharing

- Sharing SBOMs increases transparency,
  but uncontrolled disclosure can expose attack surfaces to malicious actors
- SBOMs can act as a blueprint for attackers
- Interest in balancing openness with controlled disclosure

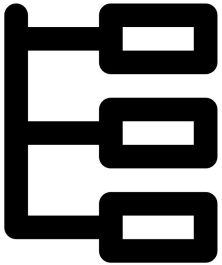


zkSBOM: Privacy-Preserving SBOM Sharing with Zero-Knowledge Sets

Tom Sorger
sorger@kth.se

Structure of the Presentation

- Background
- Problem
- zkSBOM
- Evaluation
- Limitations
- Related Work
- Conclusion



Background

- Software Bill of Materials (SBOM)
- Zero-Knowledge Sets (ZKS)
  - Sparse Merkle Trees (SMT)

SBOM

- Software Bill of Materials
- Inventory of all components contained in a software artifact
- Standards
  - CycloneDX
  - SPDX
  - SWID Tags

```
{
    "metadata": {
        "authors": [
            {
                "name": "Name",
                "email": "Email"
            }
        ],
        "component": {
            "author": "Author",
            "name": "Example SBOM",
            "version": "0.1.0"
        }
    },
    "components": [
        {
            "type": "library",
            "name": "openssl",
            "version": "0.10.1",
            "purl": "pkg:cargo/openssl@0.10.1"
        }
    ]
}
```

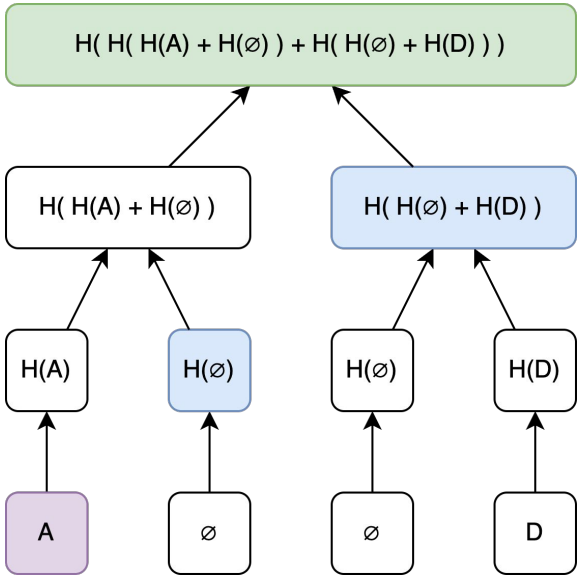
Zero-Knowledge Sets (ZKS)

- **Principle**
  - Prover commits to finite set S using a Commitment
  - Prover can prove for any queried element x, whether $x \in S$ or $x \notin S$
  - Proofs don't reveal any additional information about S, including its size

- For understanding: **SMT**

Sparse Merkle Trees (SMT)

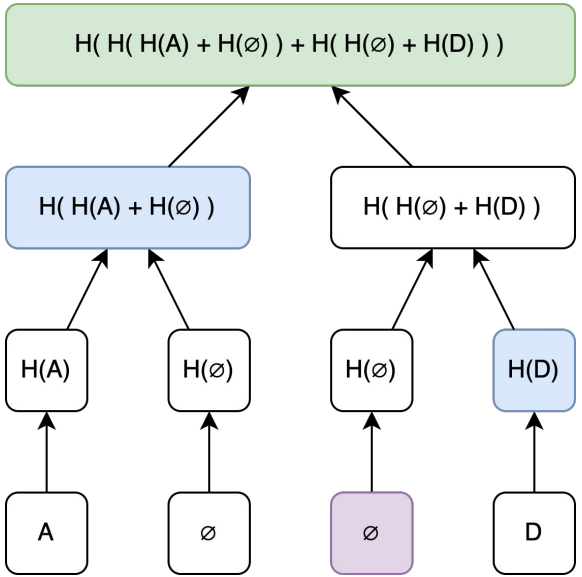
- **Structure** [1]
  - Binary tree with fixed depth (e.g., 256)
  - Root represents the entire tree (also called: commitment)
  - Empty leaves contain a default hash (e.g., `0x00...0`)
  - Insertion works by creating a key-value pair
  - Parent nodes store the hash of their child nodes
- **Insertion**
  - `Value=A; Key=H(A)=00`
  - `Value=D; Key=H(D)=11`
- **Inclusion Proof**
  - Calculate Expected Key
  - Identify and Return Required Nodes



[1] F. Haider, "Compact sparse merkle trees," Cryptology ePrint Archive, Paper 2018/955, 2018.

Sparse Merkle Trees (SMT)

- **Structure** [1]
  - Binary tree with fixed depth (e.g., 256)
  - Root represents the entire tree (also called: commitment)
  - Empty leaves contain a default hash (e.g., `0x00...0`)
  - Insertion works by creating a key-value pair
  - Parent nodes store the hash of their child nodes
- **Insertion**
  - `Value=A; Key=H(A)=00`
  - `Value=D; Key=H(D)=11`
- **Inclusion Proof**
  - Calculate Expected Key
  - Identify and Return Required Nodes
- **Non-Inclusion Proof**
  - Calculate Expected Key
  - Identify and Return Required Nodes

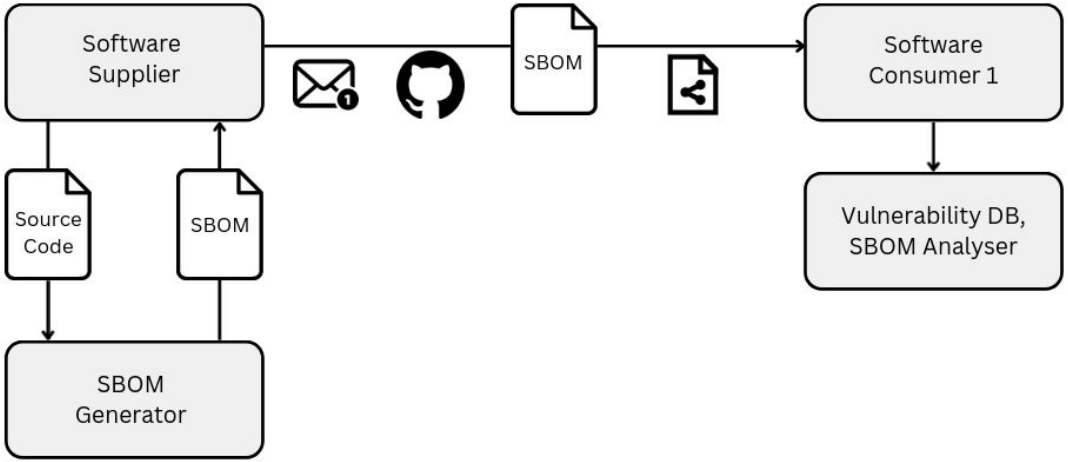


[1] F. Haider, "Compact sparse merkle trees," Cryptology ePrint Archive, Paper 2018/955, 2018.

Zero-Knowledge Sets (ZKS)

- **Principle**
  - Prover commits to finite set S using a Commitment
  - Prover can prove for any queried element x, whether $x \in S$ or $x \notin S$
  - Proofs don't reveal any additional information about S, including its size
- For easier understanding: **SMT**
- *Simplification*:
  ZKS uses structures like SMT under-the-hood + some more

Current SBOM Sharing



- Email
- Documentation
- File Sharing

[2] W. Enck, Y. Acar, M. Cukier, A. Kapravelos, C. Kästner, and L. Williams, "S3C2 Summit 2023-06: Government Secure Supply Chain Summit," 2023.
[3] G. Tystahl, Y. Acar, M. Cukier, W. Enck, C. Kastner, A. Kapravelos, D. Wermke, and L. Williams, "S3c2 summit 2024-03: Industry secure supply chain summit," 2024.
[4] N. Zahan, Y. Acar, M. Cukier, W. Enck, C. Kästner, A. Kapravelos, D. Wermke, and L. Williams, "S3c2 summit 2023-11: Industry secure supply chain summit," 2024.
[5] Cyber Security Agency of Singapore and OWASP Foundation, "Advisory on software bill of materials and real-time vulnerability monitoring for open-source software and third-party dependencies," Cyber Security Agency of Singapore, Tech. Rep., February 2025.
[6] Framing Working Group, "Sharing and exchanging sboms," National Telecommunications and Information Administration (NTIA), Tech. Rep., February 2021.
[7] Cybersecurity Division, "Guidance on introduction of software bill of materials (sbom) for software management," Ministry of Economy, Trade and Industry (METI), Japan, Tech. Rep., August 2024, version 2.0.
[8] Ministry of Electronics and Information Technology, Government of India, "Technical guidelines for software bill of materials (sbom)," Indian Computer Emergency Response Team (CERT-In), Tech. Rep., October 2024, version 1.0.

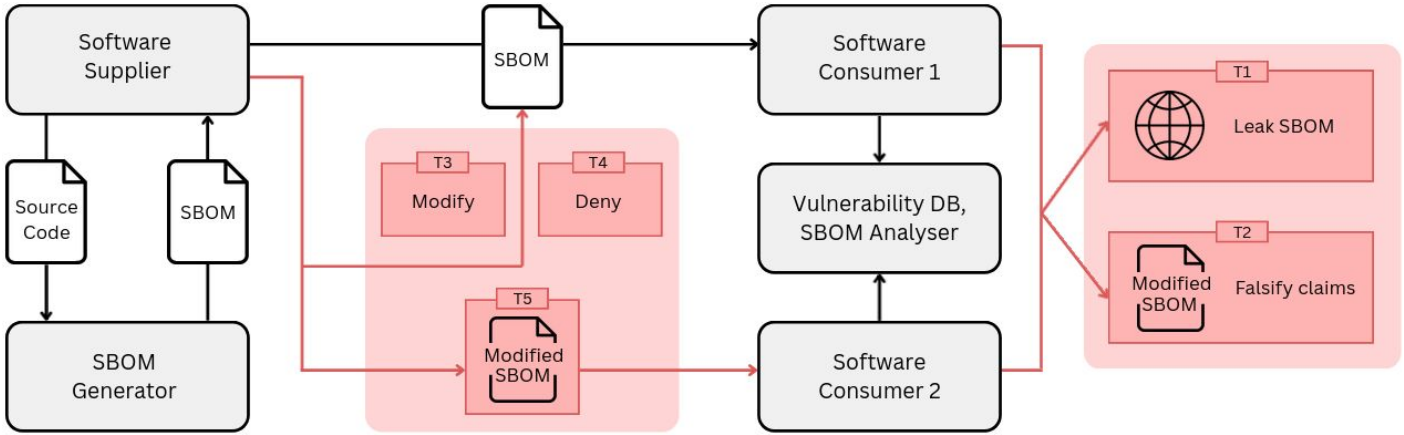
Threat Model

Software Supplier

- **T3**: Modify SBOM after sharing
- **T4**: Deny SBOM after sharing
- **T5**: Share different SBOMs to consumer

Software Consumer

- **T1**: Leak SBOM
- **T2**: Falsify Claims



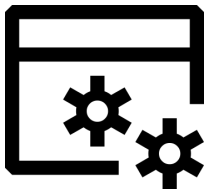
Security Goals

- **G1: Confidentiality**
  - Sensitive SBOM data is accessible only to authorized entities
  - Possibility to revoke access after business relationship ended
- **G2: Integrity**
  - SBOMs remain unaltered after release
  - Otherwise, SBOM modifications must be detectable
- **G3: Non-Repudiation**
  - Software supplier cannot retroactively deny authorship of published SBOM
- **G4: Non-Equivocation**
  - All consumers receive the same SBOM

zkSBOM – Overview

zkSBOM Operator

- SBOM Upload
- Proof Generation



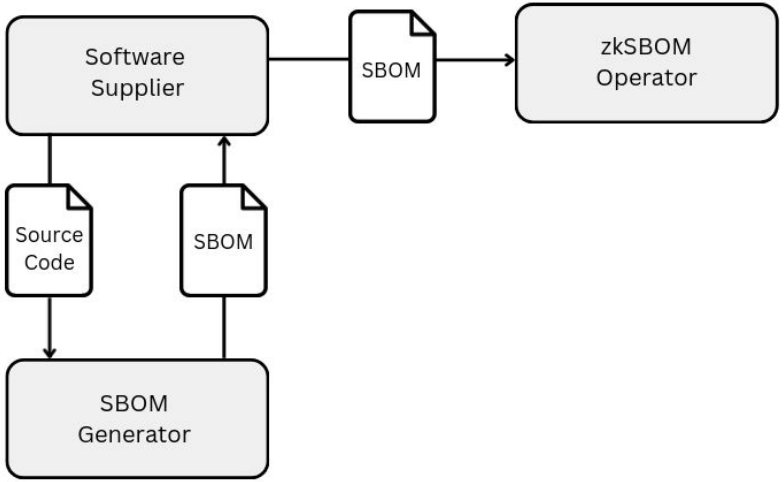
zkSBOM Verifier

- Local Proof Verification

SBOM Generation and Submission

Software supplier:

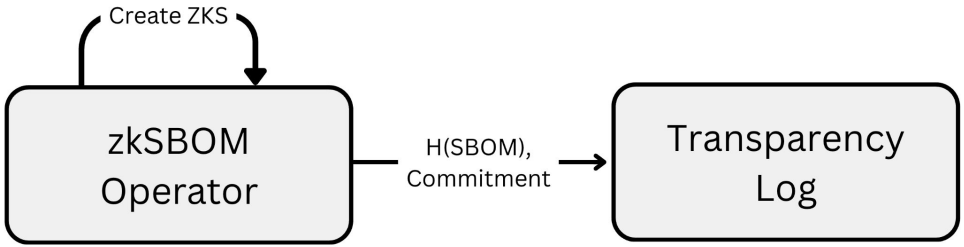
1. Completes development
2. Uses SBOM generator to generate SBOM
3. Reviews SBOM
4. Submits SBOM to zkSBOM operator



Commitment Creation

zkSBOM operator:

1. Extracts dependencies into a datastore
2. Uses datastore + random seed to create ZKS
3. Publishes H(SBOM) + commitment to transparency log
4. Returns commitment + random seed to software supplier



Commitment Verification and Publication

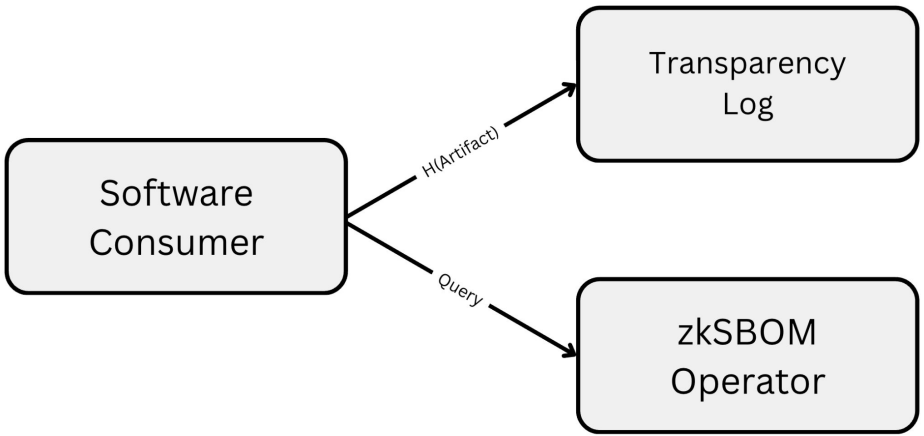
Software supplier:

1. Verifies returned commitment
2. Signs the commitment
3. Publishes following to transparency log
   a. Commitment
   b. Signed commitment
   c. H(Artifact)
   d. H(SBOM)
4. Delivers software artifact to the software consumer

Software Consumer Query

Software consumer:

1. Queries transparency log using H(Artifact)
2. Verifies signature of the software supplier
3. Requests at zkSBOM operator whether artifact is vulnerable



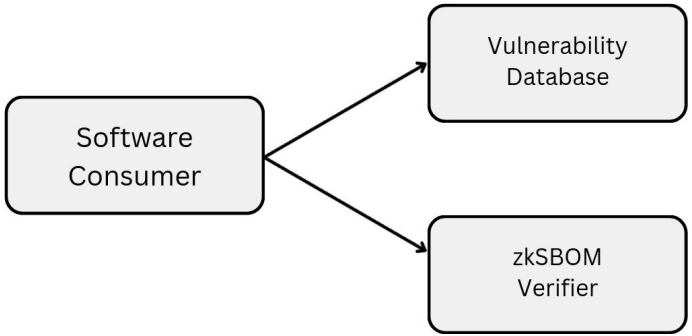
Proof Generation

zkSBOM operator:

1. Receives query
2. Queries vulnerability database for vulnerable components
3. Checks whether at least one vulnerable component is present
4. Creates either
   a. *Inclusion Proof*: for all vulnerable components present in SBOM
   b. *Non-Inclusion Proof*: for all vulnerable components
5. Returns proof to software consumer

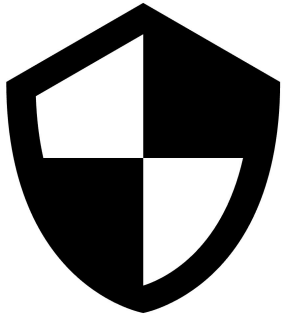
Proof Verification

1. Software consumer verifies
   a. All proved components correspond to the requested vulnerability identifier
   b. Received proof is cryptographically correct



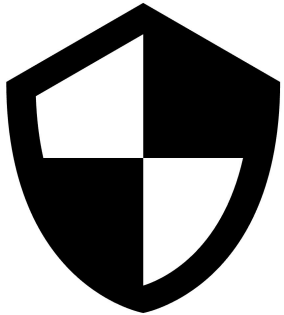
Security Analysis

- Software Consumer
- Software Supplier
- zkSBOM Operator



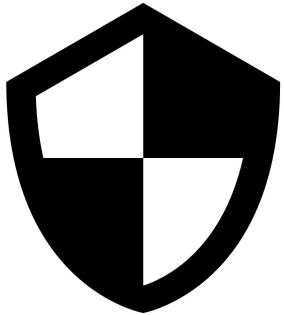
Security Analysis – Software Consumer

- **Confidentiality** (*T1*: Software consumer leaks full SBOM)
  - Consumer doesn't have full SBOM access
  - Raises question: *How much does a (non-)inclusion proof additionally leak in context of public packaging ecosystems?*
- **Integrity** (*T2*: Software consumer falsifies claims)
  - Cannot influence zkSBOM operator to generate false proofs
  - Cannot create convincing proofs himself



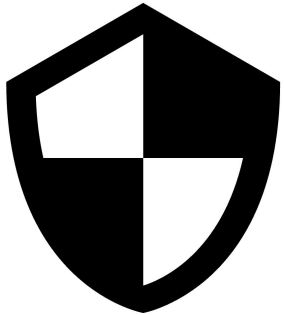
Security Analysis – Software Supplier

- **Integrity** (*T3*: Software supplier modifies SBOM after sharing)
  - Commitment prevents supplier from altering underlying dataset
  - Cannot influence zkSBOM operator to generate false proofs
  - Cannot create convincing proofs himself
- **Non-Repudiation** (*T4*: Software supplier denies SBOM after sharing)
  - Publishes signed commitment to transparency log
- **Non-Equivocation** (*T5*: Software supplier shares different SBOMs to consumer)
  - Publishes commitment to transparency log



Security Analysis – zkSBOM Operator

- **Integrity** (*T6*: Proving false statements about the software supplier's SBOM)
  - Cannot create false proofs without changing the commitment



Implementation

- **zkSBOM:**
  - https://github.com/chains-project/zkSBOM
- **Programming Languages**
  - Rust
  - C++
- **Supported Cryptographic Methods**
  - Merkle Trees: binary-merkle-tree [9]
  - Sparse Merkle Trees (SMT): sparse-merkle-tree [10]
  - Merkle Patricia Tries (MPT): trie-db [11]
  - Zero-Knowledge Sets (ZKS): oZKS [12]
- **Database**
  - SQLite
- **Vulnerability Database**
  - GitHub Advisory Database

[9] https://crates.io/crates/binary-merkle-tree
[10] https://crates.io/crates/sparse-merkle-tree
[11] https://crates.io/crates/trie-db
[12] https://github.com/microsoft/oZKS

Evaluation

- **Feasibility**
  Can zkSBOM correctly generate and verify (non-)inclusion proofs for real-world software artifacts across diverse package ecosystems?

- **Performance**
  Does zkSBOM perform practically w.r.t. time and memory usage during SBOM upload, proof construction, and proof verification?

Evaluation – End-To-End Feasibility

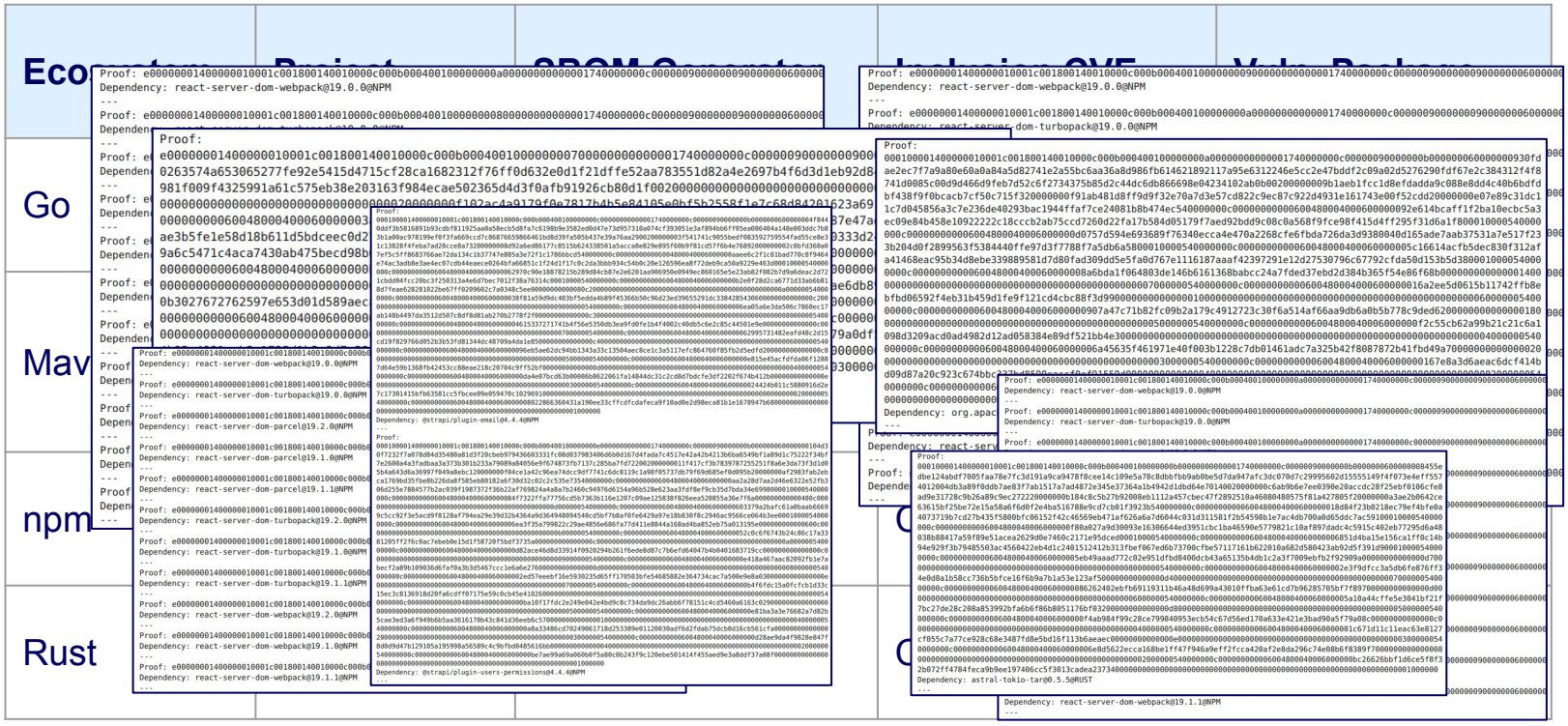
- End-to-End
  - SBOM Generation
  - SBOM Upload at zkSBOM Operator
  - Consumer Vulnerability Query
  - ZK Proof Generation
  - Proof Verification with zkSBOM Verifier
- Package Ecosystems
  - Maven
  - npm
  - Go
  - Rust

Evaluation – End-To-End Feasibility

Ecosystem	Project	SBOM Generator	Inclusion CVE	Vuln. Package
Go	Kubernetes	cyclonedx-gomod	CVE-2024-21626	runc@1.1.10
Maven	Apache Druid	Maven CycloneDX plugin	CVE-2021-44228	log4j-core@2.8.2
npm	Strapi	cyclonedx-npm	CVE-2023-22621	@strapi/plugin-email@4.4.4
Rust	uv	cargo-cyclonedx	CVE-2025-62518	astral-tokio-tar @0.5.5

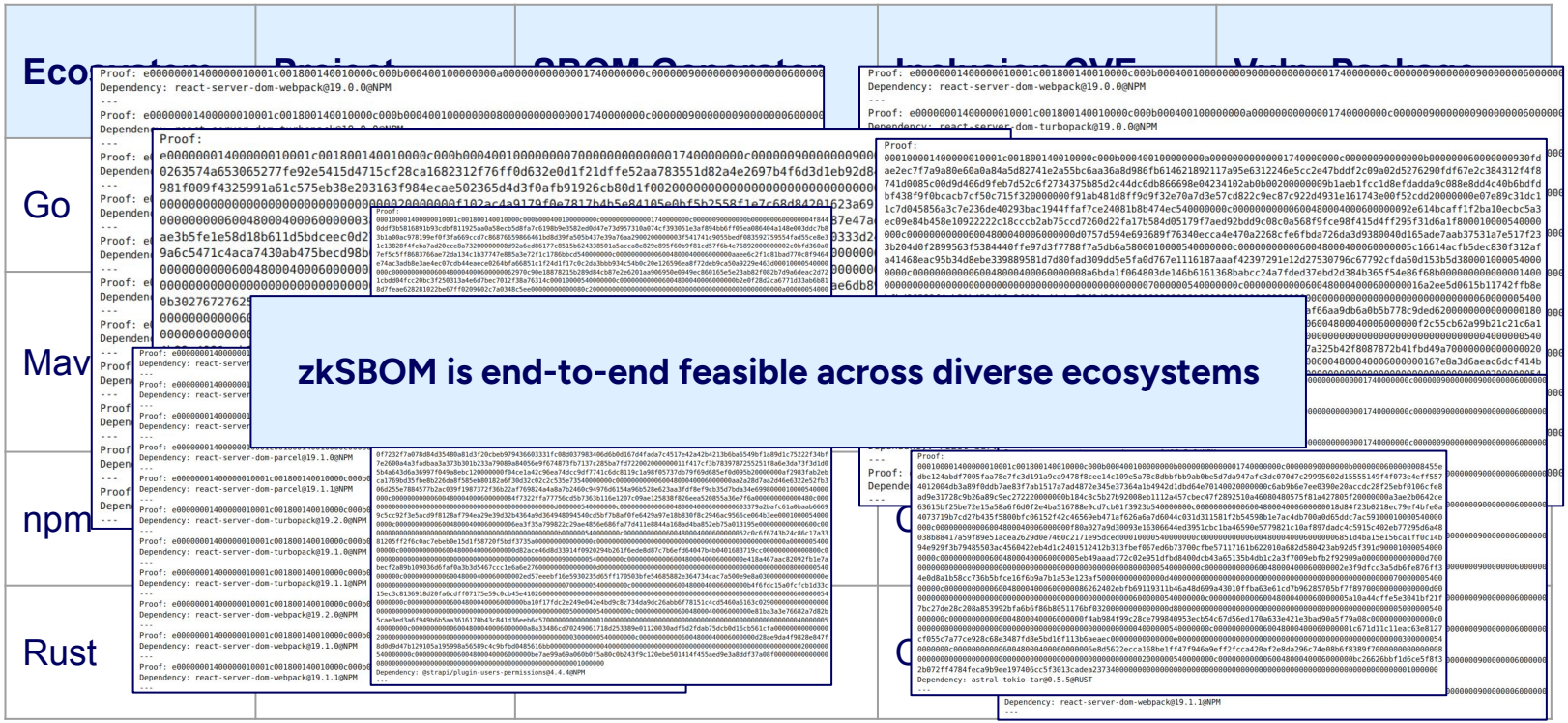
Evaluation – End-To-End Feasibility

- Background
- Problem
- zkSBOM
- Evaluation
- Limitations
- Related Work
- Conclusion



Evaluation – End-To-End Feasibility

- Background
- Problem
- zkSBOM
- Evaluation
- Limitations
- Related Work
- Conclusion



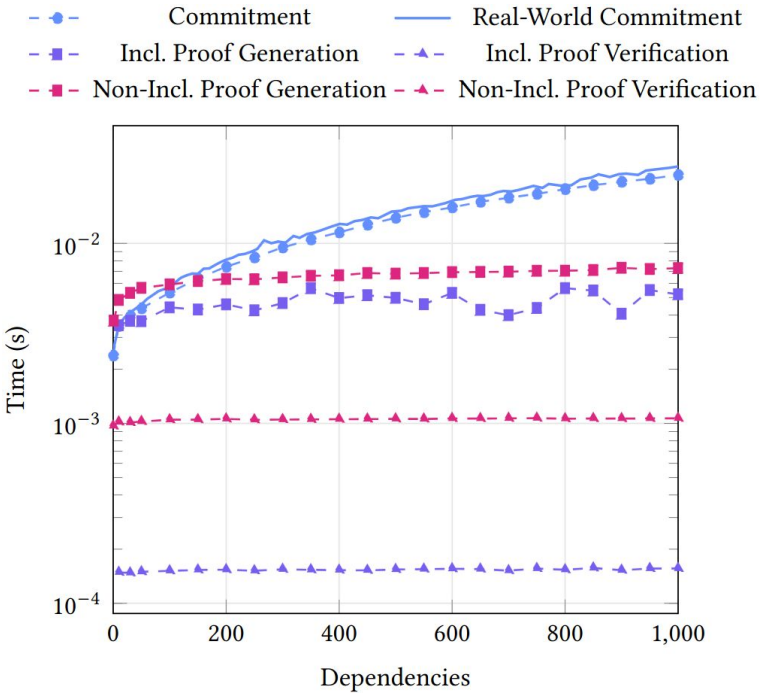
Evaluation – Performance

- **Runtime**
  - Commitment Computation
  - (Non-)Inclusion Proof Generation
  - (Non-)Inclusion Proof Verification
- **Storage**
  - Database Sizes
  - (Non-)Inclusion Proof Sizes
- **Dataset**
  - Synthetic SBOMs
    - Up to 10,000 Components
  - Real-World SBOMs [13]
    - Up to 1,000 Components
    - Only Commitment Computation and Database Sizes

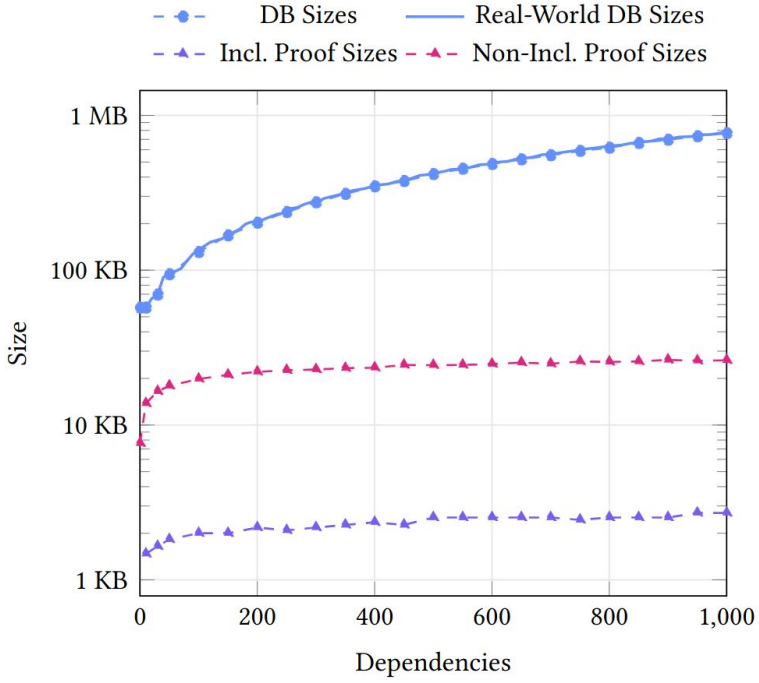
[13] L. Soeiro, T. Robert, and S. Zacchiroli, "Wild SBOMs: a Large-scale Dataset of Software Bills of Materials from Public Code", 2025.

Evaluation – Performance

Runtime

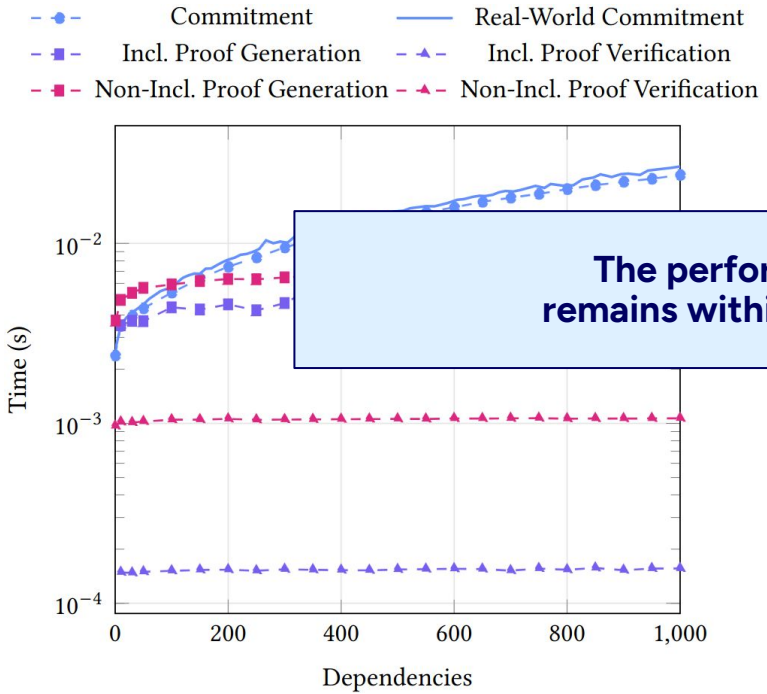


Storage

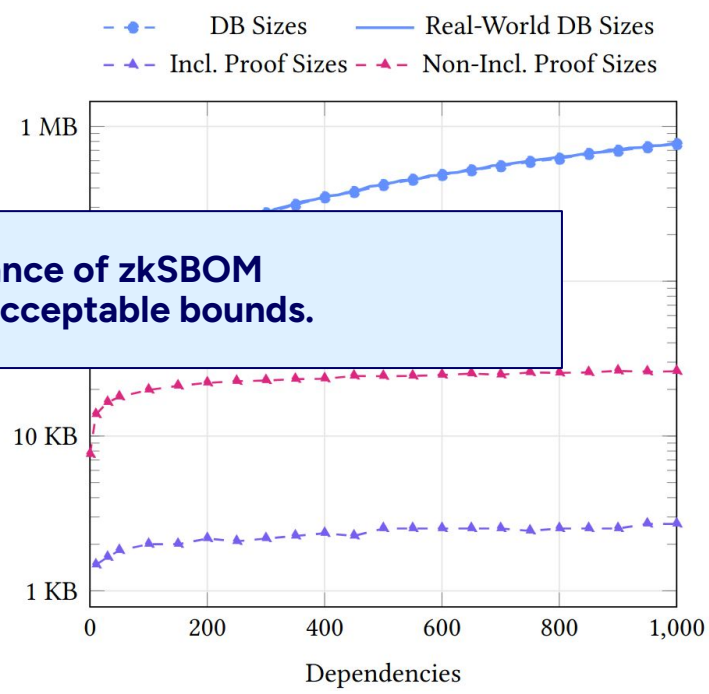


Evaluation – Performance

Runtime



Storage



The performance of zkSBOM remains within acceptable bounds.

Limitations

- **SBOM Correctness**
  - Software Supplier can omit components from the SBOM before uploading it to zkSBOM
- **Data Accuracy**
  - Doesn't address SBOM accuracy → Faulty SBOM results in faulty proofs
  - Possible mismatches with Vulnerability Database
- **SBOM Corrections**
  - Uploaded SBOM cannot be corrected
  - Even if the SBOM contains an honest mistake
- **Positive Queries**
  - Doesn't lend itself to positive queries (*"Are all components vetted by an auditor?"*)
  - Inclusion Proofs would leak all components

Related Work

- **Privacy-Preserving SBOMs**
  - Petra [14]
  - VeriSBOM [15]

- **Zero-Knowledge Software Supply Chain Applications**
  - Cheesecloth [16]

[14] E. Abu Ishgair, C. Okafor, M. S. Melara, and S. Torres-Arias, "Trustworthy and Confidential SBOM Exchange", 2025.
[15] G. Castiglione, S. Ebrahimi, and N. Khakpour, "VeriSBOM: Secure and Verifiable SBOM Sharing Via Zero-Knowledge Proofs", 2026.
[16] S. Cuéllar, B. Harris, J. Parker, S. Pernsteiner, and E. Tromer, "Cheesecloth: Zero-Knowledge Proofs of Real World Vulnerabilities", 2023.

Conclusion

- **Presented a privacy-preserving SBOM sharing system based on zero-knowledge sets**
- **Formalized Threat Model**
- **Conducted a security analysis of our system**
- **Implemented a prototype**
- **Show**
  - Feasibility
  - Computational efficiency

Thank you for Listening!

Questions


KTH
VETENSKAP
OCH KONST