Skill Bundle Attestation (SBA)

A Deterministic Identity and Attestation Framework for AI Agent Skill Bundles

RFC Version 1.0

Jason M. Lovell
jase.lovell@me.com

January 2026

Abstract

Skill Bundle Attestation (SBA) defines a deterministic identity and attestation framework for AI agent skill bundles. As AI agents increasingly rely on modular skills distributed as file bundles, the need for verifiable provenance and tamper detection becomes critical. SBA addresses this need through three key components: (1) a canonical bundle digest algorithm (`sba-directory-v1`) that produces reproducible identities independent of filesystem metadata; (2) attestation formats built on IN-TOTO Statement v1 with SBA-specific predicates for content, audit, and approval workflows; and (3) optional DSSE envelope signatures with SIGSTORE integration for identity-backed verification. This document specifies the normative behavior of SBA tooling, defines the threat model and security considerations, and provides implementation guidance for producers, auditors, and verifiers. SBA is designed to be minimal, auditable, and compatible with existing supply chain security standards.

Status of This Document

This document specifies a standards-track protocol for the AI agent skill bundle community. Distribution of this document is unlimited. Comments and suggestions should be directed to the author.

Contents

1. Introduction

AI agents increasingly rely on modular "skills"—self-contained bundles of code, prompts, and configuration that extend agent capabilities. These skill bundles are analogous to software packages in traditional software ecosystems, yet they lack the mature supply chain security tooling that protects conventional software distribution.

SBA (Skill Bundle Attestation) addresses this gap by providing:

- A **deterministic bundle digest** algorithm that produces the same identity regardless of the platform, filesystem, or extraction method used.
- A **standard attestation format** based on IN-TOTO Statement v1 that enables producers, auditors, and approvers to make verifiable claims about bundle contents.
- **Optional cryptographic signatures** using DSSE envelopes with support for SIGSTORE identity verification.

1.1 Motivation

Without deterministic identity and verifiable attestations, users of skill bundles cannot confidently answer critical questions:

1. Did the bundle change between the time it was built and the time it was deployed?
2. Is this the same bundle that was reviewed by a security team?
3. Can we detect if the bundle has been tampered with or replaced?
4. What claims has the producer made about the bundle's contents and capabilities?

These questions are fundamental to supply chain security. Traditional software ecosystems address them through package managers, code signing, and software bills of materials (SBOMs). SBA brings equivalent protections to agent skill bundles.

1.2 Scope

Ecosystem context: SBA is designed primarily for skill bundles following the Agent Skills pattern, where a `SKILL.md` manifest file describes the skill's metadata and capabilities. However, SBA is not coupled to any specific manifest format—the `SKILL.md` convention is used in examples throughout this document, but bundles **MAY** use alternative manifest structures. The digest algorithm operates on file contents regardless of manifest format.

This document specifies:

- The `sba-directory-v1` bundle digest algorithm
- The structure and semantics of SBA attestation predicates

- Verification rules for attestations and bundles
- Integration with DSSE signatures and SIGSTORE verification

The following are explicitly **out of scope**:

- Key management, identity provisioning, or PKI policy
- Secure distribution of bundles or attestations
- Runtime sandboxing or execution security
- Supply chain risk scoring or trust frameworks

1.3 Terminology

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHOULD**", "**SHOULD NOT**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in RFC 2119 [1].

Bundle A directory or archive containing a skill and its associated files.
Bundle Digest The canonical SHA-256 hash computed over bundle contents using the `sba-directory-v1` algorithm.
Archive Digest The SHA-256 hash of an archive file's raw bytes.
Attestation A JSON statement making verifiable claims about a bundle.
DSSE Dead Simple Signing Envelope—a format for signed attestations.
Predicate The typed payload within an IN-TOTO Statement describing specific claims.

1.4 Document Organization

Section 2 provides background on related standards and prior work. Section 3 describes the design principles and non-goals of SBA. Section 4 gives a high-level overview of the system architecture. Section 5 specifies the bundle digest algorithm in detail. Sections 6 and 7 describe the attestation format and signature support. Section 8 specifies verification rules. Section 9 presents the threat model and security analysis. Section 10 provides implementation guidance. Section 11 discusses integration with related standards. Appendices contain JSON schemas and test vectors.

2. Background and Related Work

2.1 Supply Chain Security Standards

The software supply chain security landscape has evolved significantly in recent years. Key standards and frameworks that inform SBA include:

SLSA (Supply-chain Levels for Software Artifacts) [2] defines a framework for achieving progressively stronger supply chain security guarantees. SBA aligns with SLSA principles by providing deterministic builds (reproducible digests) and provenance attestations.

in-toto [3] is an attestation framework that enables verifiable claims about software supply chain steps. SBA uses IN-TOTO Statement v1 as its attestation envelope format, ensuring compatibility with existing tooling and verification infrastructure.

DSSE (Dead Simple Signing Envelope) [4] provides a simple, secure format for signing attestations. SBA supports DSSE envelopes for optional signature protection.

2.2 Software Bill of Materials

Software Bills of Materials (SBOMs) enumerate the components within a software artifact. While SBA does not directly produce SBOMs, it provides complementary capabilities: SBA attests to the *identity* and *integrity* of a bundle, while SBOMs enumerate its *contents* and *dependencies*.

2.3 Code Signing and Verification

Traditional code signing relies on certificates and PKI infrastructure. SIGSTORE [5] offers an alternative model using identity-based signing with transparency logs. SBA supports both traditional key-based signatures and SIGSTORE identity verification.

2.4 Agent and Skill Systems

Modern AI agent frameworks often support extensibility through skill or tool plugins. Examples include Anthropic's Model Context Protocol (MCP), LangChain tools, and various agent frameworks. These systems typically lack standardized mechanisms for verifying skill provenance and integrity—a gap that SBA addresses.

3. Design Goals and Non-Goals

3.1 Design Principles

SBA is designed according to the following principles:

1. **Determinism**: Bundle digests **MUST** be reproducible across platforms, filesystems, and time. The same bundle contents **MUST** always produce the same digest.
2. **Minimalism**: The specification and reference implementation **SHOULD** be small enough to audit completely. Dependencies **SHOULD** be minimal.

3. **Compatibility**: SBA **MUST** build on existing standards (IN-TOTO, DSSE, SIGSTORE) rather than inventing new ones.
4. **Security by Default**: Safe handling of untrusted inputs (archives, paths) **MUST** be the default behavior.
5. **Separation of Concerns**: Content identity (digests) and signatures **MUST** be clearly separated. Unsigned attestations are still useful for integrity checking.

3.2 Non-Goals

The following are explicitly **not** goals of SBA:

- **Key Management**: SBA does not specify how signing keys are generated, distributed, rotated, or revoked. These are operational concerns left to deployers.
- **Runtime Security**: SBA does not sandbox skill execution or enforce capability restrictions at runtime. It provides attestations about *what* a bundle contains, not runtime enforcement of *what* it can do.
- **Trust Policy**: SBA does not define trust frameworks, reputation systems, or risk scoring. Consumers decide which attestations and signers to trust.
- **Distribution**: SBA does not specify how bundles or attestations are distributed. It is compatible with any distribution mechanism (registries, git, file transfer, etc.).

4. System Overview

4.1 Architecture

SBA consists of three primary components:

1. **Bundle Digest Algorithm** (`sba-directory-v1`): Computes a deterministic identity for a bundle based solely on file contents and normalized paths.
2. **Attestation Formats**: Structured claims about bundles expressed as IN-TOTO statements with SBA-specific predicates.
3. **Verification Tooling**: Validators that check attestation schemas, recompute bundle digests, and optionally verify signatures.

4.2 Actor Roles

SBA defines four actor roles:

Producer Builds a skill bundle and generates a content attestation (`sba-content-v1`).

Auditor Reviews a bundle (e.g., for security vulnerabilities) and issues an audit attestation (`sba-audit-v1`) referencing the content attestation.
Approver Authorizes a bundle for deployment by issuing an approval attestation (`sba-approval-v1`) referencing content and audit attestations.
Verifier Validates attestations against a bundle, checking schema conformance, digest consistency, and optionally signatures.

4.3 Workflow

A typical SBA workflow proceeds as follows:

1. Producer creates a skill bundle with a `SKILL.md` manifest.
2. Producer runs `sba attest content` to generate a content attestation.
3. (Optional) Producer signs the attestation with `-sign`.
4. Auditor reviews the bundle and generates an audit attestation.
5. Approver reviews content and audit attestations, then generates an approval attestation.
6. Consumer runs `sba verify` to validate the attestation chain against the bundle.

5. Bundle Identity Specification

The `sba-directory-v1` algorithm computes a deterministic digest over the contents of a bundle directory. This digest is independent of filesystem metadata (modification times, permissions, ownership) and is reproducible across platforms.

5.1 Overview

The algorithm proceeds in five phases:

1. **Enumeration**: Recursively walk the bundle directory, collecting regular files.
2. **Filtering**: Exclude paths matching the exclusion set.
3. **Normalization**: Validate and normalize each path.
4. **Entry Generation**: For each file, compute SHA-256 and format the entry.
5. **Digest Computation**: Sort entries and compute the final SHA-256 over the concatenation.

5.2 Path Normalization and Validation

For each candidate file path, the following rules apply:

- Paths **MUST** use forward slashes (`/`) as separators.
- Paths **MUST** be Unicode NFC normalized.

- Paths **MUST NOT** start with `/` (no absolute paths).
- Paths **MUST NOT** contain `..` or `.` segments.
- Paths **MUST NOT** contain NUL bytes (`\x00`).
- Paths **MUST NOT** contain backslashes (\).
- Path components **MUST NOT** be empty.
- Total path length **MUST** be $\leq$ 4096 characters.
- Individual path components **MUST** be $\leq$ 255 characters.

Paths violating any of these rules **MUST** be rejected with an error.

Additionally, verifiers **SHOULD** detect and reject case-insensitive path collisions (e.g., `File.txt` and `file.txt`) to ensure consistent behavior across filesystems.

5.3 Exclusion Patterns

S B A defines a minimal **required exclusion set** that **MUST** be applied by all implementations. This set contains only paths that are either (a) not part of bundle execution semantics, or (b) are S B A metadata that would create circular dependencies:

- **Version control**: `.git/` (directory only)
- **S B A metadata**: `.attestations/`, paths matching `*.sba.json`
- **OS artifacts**: `.DS_Store`, `Thumbs.db`

Security rationale: Directories like `node_modules/`, `.venv/`, and `__pycache__/` **MUST NOT** be excluded by default because they may contain executable code that affects bundle behavior. Silently excluding such directories would allow attackers to hide malicious code without affecting the bundle digest.

5.3.1 Custom Exclusions

Producers **MAY** declare additional exclusions in the `sba-content-v1` predicate via the `bundle.excludes` field:

```
{
  "bundle": {
    "excludes": ["__pycache__/", "*.pyc", "node_modules/"],
    ...
  }
}
```

When custom exclusions are declared:

- Verifiers **MUST** apply exactly the declared exclusion patterns when recomputing the digest.

- The exclusion list becomes part of the attested identity—changing exclusions changes the attestation.
- Verifiers **SHOULD** warn if exclusions include directories that typically contain executable code.

A path is excluded if it matches any pattern in the combined exclusion set (required + custom). Pattern matching uses glob semantics: `*` matches any characters except `/`, and trailing `/` indicates directory-only matching.

5.4 Entry Format

For each included file, compute the SHA-256 hash over the raw file bytes and format the entry as:

```
<path>\0sha256:<hex>\0<size>\n
```

Where:

- `<path>` is the normalized relative path
- `\0` is the NUL byte (0x00)
- `sha256:<hex>` is the lowercase hexadecimal SHA-256 digest
- `<size>` is the file size in bytes (decimal)
- `\n` is the newline character (0x0A)

Entries are UTF-8 encoded.

5.5 Sorting

Entries **MUST** be sorted by path using bytewise comparison of UTF-8 encoded bytes. This ensures deterministic ordering regardless of locale settings.

5.6 Final Digest

The bundle digest is the SHA-256 hash of the concatenated entry bytes. The digest is represented in the format `sha256:<hex>` where `<hex>` is the 64-character lowercase hexadecimal encoding.

5.7 Archive Mode

When operating on archive files (ZIP or tar):

1. The **archive digest** is the SHA-256 hash of the archive file bytes.
2. The archive is extracted using safe extraction helpers (see Section 9.3.1).

3. An optional `archiveRoot` parameter identifies the bundle root within the archive.
4. The **bundle digest** is computed over the extracted directory.

For archive bundles, the attestation subject digest **MUST** use the archive digest (not the bundle digest), as this is what consumers can verify against the distributed artifact.

6. Attestation Model

SBA attestations are IN-TOTO Statement v1 objects with SBA-specific predicate types.

6.1 in-toto Statement Structure

Every SBA attestation is an IN-TOTO Statement with the following structure:

```
{
  "_type": "https://in-toto.io/Statement/v1",
  "subject": [
    {
      "name": "<skill-name>",
      "digest": { "sha256": "<64-hex-chars>" }
    }
  ],
  "predicateType": "<sba-predicate-uri>",
  "predicate": { ... }
}
```

The `subject` array **MUST** contain exactly one entry representing the skill bundle. The `digest.sha256` value is:

- For directory bundles: the `sba-directory-v1` digest (without the `sha256:` prefix)
- For archive bundles: the archive file's SHA-256 hash

6.2 SBA Predicate Types

SBA defines three predicate types:

6.2.1 sba-content-v1

The content predicate describes bundle identity and metadata:

```
{
  "skill": {
    "name": "example-skill",
```

```
    "description": "An example skill bundle",
    "version": "1.0.0"
  },
  "bundle": {
    "digestAlgorithm": "sba-directory-v1",
    "digest": "sha256:<64-hex>",
    "entryCount": 5,
    "totalBytes": 12345,
    "bundleType": "directory"
  },
  "capabilities": { ... },
  "metadata": {
    "generatedAt": "2026-01-27T00:00:00Z",
    "generatorTool": "sba-attest",
    "generatorVersion": "0.1.0"
  }
}
```

6.2.2 sba-audit-v1

The audit predicate records security review results:

```
{
  "contentAttestation": { "sha256": "<digest-of-content>" },
  "auditor": { "name": "Security Team", "id": "..." },
  "result": "passed",
  "findings": [ ... ],
  "metadata": { "auditedAt": "2026-01-27T12:00:00Z" }
}
```

6.2.3 sba-approval-v1

The approval predicate authorizes deployment:

```
{
  "contentAttestation": { "sha256": "<digest-of-content>" },
  "auditAttestation": { "sha256": "<digest-of-audit>" },
  "approver": { "name": "Release Manager", "id": "..." },
  "decision": "approved",
  "constraints": { "environments": ["production"] },
  "metadata": { "approvedAt": "2026-01-27T14:00:00Z" }
}
```

6.3 Capability Declarations

The `capabilities` field in `sba-content-v1` allows producers to declare the bundle's requirements:

- **filesystem**: Read/write path patterns
- **network**: Outbound access and allowed domains
- **process**: Subprocess execution and interpreters
- **tools**: External tool/MCP access
- **secrets**: Required credentials
- **humanConfirmation**: Human-in-the-loop requirements

These declarations are *claims* by the producer, not enforced constraints. Verifiers and runtimes **MAY** use them for policy decisions.

6.4 Attestation Digest Function

When audit or approval predicates reference other attestations (via `contentAttestation.sha256` or `auditAttestation.sha256`), the digest **MUST** be computed as follows:

1. **Input**: The attestation artifact as distributed (the exact file bytes).
2. **Encoding**: The file **MUST** be valid UTF-8.
3. **Hash**: Compute SHA-256 over the raw file bytes.
4. **Output**: The 64-character lowercase hexadecimal encoding of the hash.

Normative rules:

- If the attestation is a DSSE envelope, hash the envelope JSON bytes (not the decoded payload).
- If the attestation is an unsigned statement, hash the statement JSON bytes.
- Whitespace differences *do* affect the digest—implementations **MUST NOT** normalize or reformat attestations before hashing.
- The digest is computed over what was received, not a reconstructed canonical form.

This "hash what you receive" approach avoids canonicalization ambiguity: the referenced attestation's digest is simply the SHA-256 of its file bytes as stored or transmitted.

7. DSSE and Signatures

SBA supports optional DSSE envelopes for tamper detection and authenticity verification.

7.1 Envelope Structure

A signed attestation is wrapped in a DSSE envelope:

```
{
  "payloadType": "application/vnd.in-toto+json",
  "payload": "<base64-encoded-statement>",
  "signatures": [
    {
      "keyid": "SHA256:<fingerprint>",
      "sig": "<base64-encoded-signature>"
    }
  ]
}
```

The `payload` is the base64 encoding of the UTF-8 encoded statement JSON bytes as produced by the signer. Verifiers **MUST** verify the signature over the exact bytes embedded in the envelope—no canonicalization or reformatting is performed.

Note: SBA does not require JSON canonicalization (such as RFC 8785 JCS) for DSSE payloads. The signature covers whatever bytes the signer placed in the `payload` field. This simplifies implementation and avoids canonicalization-related interoperability issues.

7.2 Signature Computation

Signatures are computed over the Pre-Authentication Encoding (PAE):

```
PAE(payloadType, payload) =
  "DSSEv1" + SP + len(payloadType) + SP + payloadType +
  SP + len(payload) + SP + payload
```

Where `SP` is a space character and `len()` returns the byte length as a decimal string.

7.3 Supported Algorithms

SBA supports the following signature algorithms:

- **ed25519**: Edwards-curve Digital Signature Algorithm
- **ecdsa-sha256**: Elliptic Curve DSA with SHA-256
- **rsa-pss-sha256**: RSA-PSS with SHA-256
- **rsa-pkcs1v15-sha256**: RSA PKCS#1 v1.5 with SHA-256

7.4 Sigstore Integration

For identity-based verification, SBA supports SIGSTORE bundles. Verifiers can constrain acceptable signatures by:

- **Identity**: The signer's email or OIDC subject
- **Issuer**: The OIDC identity provider (e.g., GitHub Actions)

This enables verification without managing public keys directly.

8. Verification Flow

Verification validates that an attestation correctly describes a bundle.

8.1 Verification Steps

A compliant verifier **MUST** perform the following steps:

1. **Parse**: Decode the attestation (or DSSE envelope).
2. **Schema Validation**: Validate the statement and predicate against their JSON schemas.
3. **Digest Recomputation**: Compute the bundle digest using `sba-directory-v1`.
4. **Digest Comparison**:
   - For directory bundles: verify `subject[0].digest.sha256` equals the computed bundle digest
   - For archive bundles: verify `subject[0].digest.sha256` equals the archive digest, and verify `predicate.bundle.digest` equals the computed bundle digest
5. **Metadata Consistency**: Verify `entryCount` and `totalBytes` match the actual bundle.
6. **Signature Verification** (if signed): Verify signatures using provided public keys or SIGSTORE constraints.
7. **Chain Verification** (for audit/approval): Verify referenced attestations exist and their digests match.

8.2 Verification Rules

VR-001 Verifiers **MUST** reject if the subject digest does not match the computed artifact digest.

VR-002 For directory bundles, verifiers **MUST** reject if the subject digest does not equal the `predicate.bundle.digest` (prefix removed).

VR-003 For archive bundles, verifiers **MUST** reject if `predicate.bundle.archiveDigest` does not equal the subject digest.

VR-004 Verifiers **MUST** reject if `entryCount` does not match the actual file count.
VR-005 Verifiers **MUST** reject if `totalBytes` does not match the actual total size.
VR-006 Verifiers **SHOULD** warn if `subject[0].name` does not match `predicate.skill.name`.

9. Threat Model and Security Analysis

9.1 Assets Under Protection

SBA protects the following assets:

- **Bundle Integrity**: The contents of a skill bundle (directory or archive)
- **Attestation Integrity**: The claims made about a bundle
- **Attestation Authenticity**: The identity of attestation signers
- **Verification Correctness**: The accuracy of verification results

9.2 Trust Boundaries

SBA defines the following trust boundaries:

- Bundle producer vs. bundle consumer
- Local filesystem vs. untrusted archive inputs
- Offline verification vs. SIGSTORE online verification

9.3 Threat Categories

9.3.1 T1: Archive Traversal and Path Injection

Threat: Malicious archives containing `../` traversal sequences or absolute paths could escape the extraction directory.

Mitigation: Safe extraction helpers reject traversal sequences, absolute paths, and symlinks. Path validation rejects unsafe paths before processing.

9.3.2 T2: Symlink and Non-Regular File Abuse

Threat: Symlinks or device files could be used to read sensitive files or cause unexpected behavior.

Mitigation: Bundle digests include only regular files. Symlinks and special files are excluded from enumeration.

9.3.3 T3: Case-Insensitive Path Collisions

Threat: Files like `File.txt` and `file.txt` may collide on case-insensitive filesystems, causing inconsistent verification.

Mitigation: Digest computation detects and rejects case-colliding paths.

9.3.4 T4: Attestation Tampering

Threat: An attacker modifies an attestation to match a malicious bundle.

Mitigation: DSSE signatures protect attestation integrity. SIGSTORE provides identity verification.

9.3.5 T5: Schema Bypass

Threat: Malformed statements accepted due to incomplete validation.

Mitigation: JSON Schema validation for all statement and predicate types.

9.3.6 T6: Digest Confusion

Threat: Substituting an archive digest where a directory digest is expected.

Mitigation: Separate fields (`digest`, `archiveDigest`) and explicit `bundleType` field.

9.3.7 T7: Resource Exhaustion

Threat: Zip bombs or extremely deep directory trees cause denial of service.

Mitigation: Callers **SHOULD** apply size and depth limits. Safe extraction includes basic sanity checks.

9.3.8 T8: Missing Dependencies

Threat: Verification without required crypto libraries silently skips signature checks.

Mitigation: Verification reports warnings for missing dependencies. `-require-signatures` enforces signature verification.

9.4 Residual Risks

The following risks are not fully mitigated by SBA:

- Key compromise or weak operational key handling
- Malicious code within an otherwise validly-signed bundle

- Trust in SIGSTORE infrastructure and identity providers
- Resource exhaustion from extremely large bundles

9.5 Operational Recommendations

1. Require signatures for production verification
2. Pin expected signer identities or key fingerprints
3. Enforce size and depth limits for untrusted archives
4. Run verification in sandboxed environments
5. Maintain audit logs of verification decisions

10. Implementation Guidance

10.1 Reference Implementation

The SBA reference implementation is intentionally minimal:

- `sba_digest.py`: Canonical digest algorithm
- `sba_attest.py`: Attestation generation
- `sba_verify.py`: Verification and schema validation
- `sba_crypto.py`: DSSE signing/verification helpers
- `sba_archive.py`: Safe ZIP/tar extraction
- `sba_zip.py`: Deterministic ZIP builder for test vectors

10.2 CLI Usage

```
# Compute directory digest
python3 sba.py digest path/to/skill

# Generate content attestation
python3 sba.py attest content path/to/skill \
    --output attestation.json

# Verify attestation
python3 sba.py verify attestation.json \
    --bundle path/to/skill

# Sign with DSSE
python3 sba.py attest content path/to/skill \
    --envelope --sign --private-key key.pem \
    --output signed.json
```

```
16
17 # Verify signatures
18 python3 sba.py verify signed.json \
19     --bundle path/to/skill \
20     --verify-signatures --public-key key.pub
```

10.3 Test Vectors

The `test-vectors/` directory contains canonical test cases:

- `tv-1-minimal`: Minimal single-file bundle
- `tv-2-complex`: Multi-file bundle with nested directories
- `tv-3-archive.zip`: Archive bundle with nested root

Each test vector includes expected digests and attestations for validation.

11. Interoperability

11.1 Relationship to in-toto

SBA uses IN-TOTO Statement v1 as its attestation format. This provides:

- Compatibility with IN-TOTO verification tooling
- Ability to combine SBA attestations with other IN-TOTO predicate types
- Future compatibility with IN-TOTO policy engines

11.2 Relationship to DSSE

SBA uses DSSE for signed attestations. This provides:

- Standard envelope format understood by supply chain tools
- Support for multiple signatures per attestation
- Compatibility with SIGSTORE signing workflows

11.3 Relationship to Sigstore

SBA integrates with SIGSTORE for identity-based verification:

- Keyless signing using OIDC identity
- Transparency log integration via Rekor
- Identity constraints (email, issuer) for verification

11.4 CI/CD Integration

SBA can be integrated into CI/CD pipelines:

- Generate attestations during build
- Sign with GitHub Actions OIDC tokens
- Verify before deployment
- Store attestations alongside bundles in registries

12. Performance and Scalability

12.1 Complexity Analysis

- **Digest computation**: $O(n)$ where n is total file bytes
- **Path enumeration**: $O(m \log m)$ where m is file count (dominated by sort)
- **Schema validation**: $O(s)$ where s is schema size
- **Signature verification**: Constant time per signature

12.2 Expected Performance

For typical skill bundles (10-100 files, <10MB total):

- Digest computation: <100ms
- Attestation generation: <200ms
- Verification: <300ms

12.3 Resource Limits

For very large bundles, implementers **SHOULD**:

- Stream file hashing rather than loading files into memory
- Apply configurable limits on file count and total size
- Consider parallel hashing for multi-core systems

13. Governance and Evolution

13.1 Versioning Strategy

SBA uses explicit versioning for all formats:

- Digest algorithm: `sba-directory-v1`
- Predicate types: `sba-content-v1`, `sba-audit-v1`, `sba-approval-v1`
- Schema URIs include version numbers

13.2 Schema Evolution

Schema changes follow these principles:

- **Additive changes**: New optional fields can be added without version bump
- **Breaking changes**: New required fields or semantic changes require new version
- **Deprecation**: Old versions supported for at least one major release

13.3 Backwards Compatibility

Verifiers **SHOULD** support multiple predicate versions simultaneously. Producers **SHOULD** generate attestations using the latest stable version.

14. Future Work

- **Build Provenance**: Predicate type for build environment and inputs
- **Dependency Manifests**: Integration with SBOM formats
- **Transparency Logs**: Attestation publication to Rekor or similar
- **Trust Profiles**: Configurable verification policies
- **Registry Integration**: Standard APIs for attestation storage/retrieval

15. Conclusion

Skill Bundle Attestation (SBA) provides a minimal, auditable framework for establishing verifiable identity and provenance for AI agent skill bundles. By building on established standards (in-toto, DSSE, Sigstore) and focusing on deterministic, platform-independent behavior, SBA enables the same supply chain security practices that protect traditional software to be applied to the emerging ecosystem of AI agent skills.

The specification prioritizes simplicity and security by default. Safe archive handling, case-collision detection, and clear separation of content identity from signatures reflect lessons learned from supply chain security incidents. The threat model and verification rules provide clear guidance for implementers.

SBA is designed to evolve. The versioning strategy and schema evolution principles ensure that improvements can be made while maintaining compatibility. Future work on build provenance, dependency manifests, and transparency log integration will further strengthen the security posture of skill bundle ecosystems.

References

References

[1] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, March 1997.

[2] Supply-chain Levels for Software Artifacts (SLSA), https://slsa.dev/, 2024.

[3] in-toto: A framework to secure the integrity of software supply chains, https://in-toto.io/, 2024.

[4] Dead Simple Signing Envelope (DSSE), https://github.com/secure-systems-lab/dsse, 2024.

[5] Sigstore: A new standard for signing, verifying and protecting software, https://www.sigstore.dev/, 2024.

[6] in-toto Attestation Framework Specification, https://github.com/in-toto/attestation, 2024.

[7] NIST, "Secure Hash Standard (SHS)," FIPS PUB 180-4, August 2015.

[8] Unicode Standard Annex #15: Unicode Normalization Forms, https://unicode.org/reports/tr15/, 2024.

[9] JSON Schema: A Media Type for Describing JSON Documents, https://json-schema.org/, 2024.

[10] A. Rundgren, B. Jordan, S. Erdtman, "JSON Canonicalization Scheme (JCS)," RFC 8785, June 2020.

A. JSON Schema: sba-statement-v1

The statement schema wraps SBA predicates in the IN-TOTO Statement v1 format. Note that `predicateType` uses a URI format string (not an enum) to allow future predicate types without schema revision.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://jlov7.github.io/sba/schemas/sba-statement-v1.json",
  "title": "SBA In-toto Statement",
  "type": "object",
  "required": ["_type", "subject", "predicateType", "predicate"],
  "properties": {
    "_type": { "const": "https://in-toto.io/Statement/v1" },
    "subject": {
      "type": "array",
      "minItems": 1, "maxItems": 1,
      "items": {
        "type": "object",
        "required": ["name", "digest"],
        "properties": {
          "name": { "type": "string", "maxLength": 128 },
          "digest": {
            "type": "object", "required": ["sha256"],
            "properties": { "sha256": { "pattern": "^[a-f0-9]{64}$" } }
          }
        }
      }
    },
    "predicateType": {
      "type": "string",
      "format": "uri",
      "description": "URI identifying the predicate schema"
    },
    "predicate": { "type": "object" }
  }
}
```

B. JSON Schema: sba-content-v1

```json
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://jlov7.github.io/sba/schemas/sba-content-v1.json",
  "title": "SBA Content Predicate v1",
  "type": "object",
  "required": ["skill", "bundle"],
  "properties": {
    "skill": {
      "type": "object",
      "required": ["name", "description"],
      "properties": {
        "name": { "type": "string", "maxLength": 128 },
        "description": { "type": "string", "maxLength": 1024 },
        "version": { "type": "string" }
      }
    },
    "bundle": {
      "type": "object",
      "required": ["digestAlgorithm", "digest", "entryCount", "totalBytes"],
      "properties": {
        "digestAlgorithm": { "const": "sba-directory-v1" },
        "digest": { "pattern": "^sha256:[a-f0-9]{64}$" },
        "archiveDigest": { "pattern": "^sha256:[a-f0-9]{64}$" },
        "bundleType": { "enum": ["directory", "archive"] },
        "excludes": { "type": "array", "items": { "type": "string" } },
        "entryCount": { "type": "integer", "minimum": 1 },
        "totalBytes": { "type": "integer", "minimum": 0 }
      }
    },
    "capabilities": { "type": "object" },
    "metadata": { "type": "object" }
  }
}
```

C. JSON Schema: sba-audit-v1

```json
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://jlov7.github.io/sba/schemas/sba-audit-v1.json",
  "title": "SBA Audit Predicate",
  "type": "object",
  "required": ["skill", "bundle", "audit"],
  "properties": {
    "skill": {
      "type": "object", "required": ["name"],
      "properties": { "name": { "type": "string" }, "version": { "type": "string" } }
    },
    "bundle": {
      "type": "object", "required": ["digest"],
      "properties": {
        "digest": { "pattern": "^sha256:[a-f0-9]{64}$" },
        "contentAttestationDigest": { "pattern": "^sha256:[a-f0-9]{64}$" }
      }
    },
    "audit": {
      "type": "object", "required": ["tool", "timestamp", "result"],
      "properties": {
        "tool": { "type": "object", "required": ["name", "version"] },
        "timestamp": { "type": "string", "format": "date-time" },
        "result": { "enum": ["PASS", "FAIL", "WARN", "SKIP"] }
      }
    },
    "findings": { "type": "array" },
    "sbom": { "type": "object" },
    "metadata": { "type": "object" }
  }
}
```

D. JSON Schema: sba-approval-v1

```json
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://jlov7.github.io/sba/schemas/sba-approval-v1.json",
  "title": "SBA Approval Predicate",
  "type": "object",
  "required": ["skill", "bundle", "approval"],
  "properties": {
    "skill": {
      "type": "object", "required": ["name"],
      "properties": { "name": { "type": "string" }, "version": { "type": "string" } }
    },
    "bundle": {
      "type": "object", "required": ["digest"],
      "properties": {
        "digest": { "pattern": "^sha256:[a-f0-9]{64}$" },
        "contentAttestationDigest": { "pattern": "^sha256:[a-f0-9]{64}$" },
        "auditAttestationDigest": { "pattern": "^sha256:[a-f0-9]{64}$" }
      }
    },
    "approval": {
      "type": "object", "required": ["decision", "timestamp", "scope"],
      "properties": {
        "decision": { "enum": ["APPROVED", "REJECTED", "CONDITIONAL", "REVOKED"] },
        "timestamp": { "type": "string", "format": "date-time" },
        "scope": { "enum": ["GLOBAL", "ORGANIZATION", "TEAM", "PROJECT", "REGISTRY"] }
      }
    },
    "approver": { "type": "object" },
    "conditions": { "type": "array" },
    "reviewedArtifacts": { "type": "array" },
    "metadata": { "type": "object" }
  }
}
```

E. Test Vectors

Complete schemas and test vectors are normative and maintained in the specification repository at https://github.com/jlov7/sba. The following excerpts provide reference values for implementer testing.

E.1 TV-1: Minimal Bundle

Bundle Contents:

```
tv-1-minimal/
  SKILL.md    (232 bytes)
```

SKILL.md (exact contents):

```
---
name: minimal-test-skill
description: A minimal skill bundle containing only SKILL.md for SBA
   ↪ test vector TV-1
version: 1.0.0
---

# Minimal Test Skill

This is a minimal skill for testing the SBA digest algorithm.

## Usage

This skill does nothing - it exists purely for digest testing.
```

Expected Bundle Digest:

```
sha256:1627201fc34e5fd7b076b6df18fdaa505848cebd480344b1ee881dbd39a3fa49
```

E.2 TV-2: Complex Bundle

Bundle Contents:

```
tv-2-complex/
  SKILL.md
  helper.py
  nested/
    deep/
      config.json
```

```
  resources/
    data/
      sample.txt
```

Expected Bundle Digest:

```
sha256:353102351f19e357f3da15f14020948157cb411afe59a064ae365b103dbf88ae
```

E.3 TV-3: Archive Bundle

Test vector 3 is distributed as `tv-3-archive.zip` with `archiveRoot="skill-bundle/"`. See the specification repository for complete test data.

F. Glossary

Attestation A signed or unsigned statement making claims about a software artifact.

Bundle A directory or archive containing skill code and configuration.

Bundle Digest The canonical SHA-256 hash of bundle contents per `sba-directory-v1`.

DSSE Dead Simple Signing Envelope—a format for cryptographically signing attestations.

in-toto An attestation framework for software supply chain integrity.

Predicate The typed payload within an IN-TOTO Statement.

Sigstore A project providing keyless signing and verification for software artifacts.

Skill A modular capability extension for an AI agent.