vCon
Internet-Draft
Intended status: Informational
Expires: 10 May 2026

J. Miller
6 November 2025

vCon Zip Bundle
draft-miller-vcon-zip-bundle-00

Abstract

This document defines the vCon Zip Bundle (.vconz) file format for packaging one or more vCon conversation data containers with their associated media files into a single, self-contained ZIP archive. While vCons support external file references via HTTPS URLs with content hashes, these dependencies create availability and portability challenges. The vCon Zip Bundle addresses this through a standardized archive format that includes all referenced files, supports multiple vCons with automatic deduplication based on content hashes, preserves data integrity through hash verification, and enables offline processing. This specification maintains full compatibility with all vCon security forms (unsigned, signed, encrypted) as defined in the vCon core specification.

Table of Contents

1. Introduction

vCons support both inline content (base64-encoded in the JSON) and externally referenced files (via HTTPS URLs with content hashes). While external references enable efficient storage and network transfer, they create dependencies on external resources that may become unavailable over time. The vCon Zip Bundle (.vconz) format addresses this by:

1. *Self-containment*: All referenced files are included within the ZIP archive

2. *Multi-vCon support*: Multiple vCons can be bundled together with automatic file deduplication

3. *Integrity preservation*: Original content hashes are maintained for verification

4. *Platform independence*: Standard ZIP format supported across all platforms

5. *Simplicity*: Flat structure with hash-based file naming eliminates the need for mapping manifests

6. *Offline processing*: No network dependencies after bundle creation

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. File Structure

The vCon Zip Bundle (.vconz file) MUST follow this simplified structure designed for multi-vCon support and ease of use:

```
bundle.vconz
├── manifest.json                              # Bundle format identifiers
├── files/                                     # All media files (flat, deduplicated)
│   ├── sha512-GLy6IPa...UQ.wav                # Hash-named with file extensions
│   ├── sha512-Def456...XYZ.pdf
│   └── sha512-Transcript...ABC.json
├── vcons/                                     # All vCon JSON files
│   ├── 0195544a-b9b1-8ee4-b9a2-279e0d16bc46.json
│   └── 01955450-1234-5678-9abc-def012345678.json
└── extensions/                                # Future vCon extensions (optional)
    └── [extension-name]/
        ├── metadata.json
        └── files/
            └── sha256-ExtData...GHI.cbor
```

Figure 1: vCon Zip Bundle Directory Structure

2.1. Design Principles

Flat File Structure: All media files are stored in a single files/ directory, regardless of type (dialog, attachments, analysis). The vCon JSON already contains the semantic type information, eliminating the need for directory-based categorization.

Hash-Based Naming: Files are named by their content_hash value from the vCon, enabling:

* Direct lookup without manifest files

* Automatic deduplication across multiple vCons

* Integrity verification through filename

UUID-Based vCon Naming: Each vCon is stored as vcons/[uuid].json, making discovery trivial by scanning the directory.

No Metadata Folder: All necessary information is already in the vCon JSON files themselves. The bundle format relies on the vCon specification's built-in content_hash mechanism.

3. Core Files

3.1. manifest.json

The top-level manifest contains only essential bundle format identifiers:

```
{
  "format": "vcon-bundle",
  "version": "1.0"
}
```

Figure 2: Example manifest.json

Parameters:

* format: MUST be "vcon-bundle"

* version: Bundle format version (this specification defines version "1.0")

Bundle consumers discover vCons by scanning the vcons/ directory. No vCon enumeration is maintained in the manifest.

3.2. vCon JSON Files

Each vCon is stored in the vcons/ directory as [uuid].json:

* Filename MUST be the vCon's UUID followed by .json

* Content MUST be the complete, original vCon in any security form (unsigned, signed JWS, encrypted JWE)

* All external URL references and content_hash values MUST be preserved exactly as received

3.2.1. Security Form Handling

Unsigned vCons: Stored directly as JSON object

Signed vCons (JWS): The complete JWS structure MUST be preserved, including:

* JWS headers with signature algorithms and keys

* Base64url-encoded payload containing the actual vCon

* Signature verification data

Encrypted vCons (JWE): The complete JWE structure MUST be preserved, including:

* JWE headers with encryption algorithms and key information

* Encrypted payload (which may contain a signed vCon)

* All encryption metadata required for decryption

Bundle creators MUST NOT modify the vCon content, decrypt encrypted vCons, or remove signatures.

Bundle creators MAY extract embedded files from an unsigned or unencrypted vCon and replace them with external references. When doing so, the creator MUST upload the extracted content to an accessible HTTPS URL, compute the appropriate content_hash, and update the vCon JSON to include the url and content_hash fields while removing inline body and encoding fields.

3.3. File Lookup Mechanism

To resolve a file reference from a vCon:

1. Read the content_hash field from the vCon object (dialog, attachment, or analysis)

2. Extract the hash algorithm and value (e.g., "sha512-GLy6IPa...")

3. Look in files/ for files matching [content_hash] with any extension

4. The extension is provided for unzip/extraction friendliness and is determined by the file’s media type (see Section 4.2)

Example: If a dialog entry has "content_hash": "sha512-GLy6IPaIUM1...UQ" and "mediatype": "audio/wav", the file will be located at files/sha512-GLy6IPaIUM1...UQ.wav.

4. File Naming Conventions

4.1. Hash-Based Filenames

All externally referenced files MUST be stored using their content hash as the filename, ensuring:

* *Uniqueness*: Hash-based names prevent collisions

* *Integrity*: Filename directly corresponds to content verification

* *Deduplication*: Identical content (same hash) reuses the same file across multiple vCons

* *Direct lookup*: No manifest needed to map URLs to files

Files MUST be named using the following pattern:

[hash-algorithm]-[base64url-hash].[extension]

Implementations MUST support SHA-512 as defined in the vCon specification. Additional hash algorithms MAY be supported as specified in the vCon content_hash field.

Examples:

* sha512-GLy6IPaIUM1GqzZqfIPZlWjaDsNgNvZM0iCONNThnH0a75fhUM6cYzLZ5GynSURREvZwmOh54-2lRRieyj82UQ.wav

* sha256-Abc123DefGhi456JklMno789PqrStu012VwxYz345.mp4 (if content_hash specifies SHA-256)

* sha512-Def456UVW789XyzAbcDefGhi123JklMnoPqrStuVwxYz456AbcDefGhi789JklMno012PqrStuVwxYzA.pdf

4.2. Extension Determination

File extensions MUST be determined by the following priority:

1. *MIME type* from vCon mimetype field (preferred)
    * Use standard MIME type to extension mappings (e.g., "audio/wav" → ".wav")
2. *Content analysis* of the file header
    * Inspect magic bytes to determine file type
3. *Original URL extension* as fallback
    * Extract extension from the URL path if available
4. *Generic extension* (.bin) if undetermined

The extension enables operating systems and tools to handle files appropriately when the bundle is extracted.

4.3. Multi-Hash Support

When a vCon references files with multiple content hashes (as per vCon specification: "ContentHash" | "ContentHash[]"), implementations MUST:

1. *Primary hash*: Use the first hash algorithm in the array for filename generation

2. *Preferred hash*: If SHA-512 is present in the array, it SHOULD be used as primary regardless of position

3. *Validation*: Verify ALL provided hashes during bundle creation and extraction

Example: If content_hash is ["sha256-Abc...", "sha512-Def..."], the file SHOULD be named sha512-Def....ext.

5. vCon References and Relationships

5.1. Group Objects

vCons may reference other vCons through Group Objects (Section 4.6 of [I-D.ietf-vcon-vcon-core]). These references aggregate multiple vCons into a logical conversation.

When a vCon references another vCon via a Group Object:

* The referenced vCon SHOULD be included in the bundle as a separate file in vcons/[uuid].json

* The Group Object’s uuid field enables discovery of the referenced vCon

* If the Group Object includes a url with a content_hash, the bundler MAY include a copy of the URL-based reference in the bundle

* Bundle readers SHOULD check if the content_hash exists in the bundle’s files/ directory before attempting external URL resolution

5.2. Referenced vCon Discovery

To discover all vCons in a bundle and their relationships:

1. Scan vcons/ directory for all *.json files
2. For each vCon, check for group[] array entries with uuid fields
3. Verify that referenced vCons are present in the bundle (if intended)
4. Build relationship graph of vCons based on these references

6. Bundle Creation Process

6.1. vCon Security Form Handling

Bundle creators MUST handle different vCon security forms appropriately:

For Unsigned vCons:

1. MUST parse JSON directly to identify external references
2. MUST proceed with standard file resolution

For Signed vCons (JWS):

1. MUST preserve complete JWS structure in the output vCon file
2. SHOULD verify signature before processing
3. MUST parse base64url-decoded payload to identify external references
4. MUST resolve external files based on payload content

For Encrypted vCons (JWE):

1. MUST preserve complete JWE structure in the output vCon file
2. If decryption keys are available, bundle creator SHOULD decrypt to resolve external references
3. If decryption keys are unavailable, the bundle creator MUST include the encrypted vCon without resolving external references or including associated files

4. When keys are available, MUST parse decrypted content (which may itself be signed) to identify external references

5. MUST save original encrypted JWE structure after file resolution (not decrypted content)

6.2. External File Resolution

For each vCon being bundled:

1. *Parse vCon content*: MUST parse according to security form to identify all external references:
    * dialog[] entries with url and content_hash
    * attachments[] entries with url and content_hash
    * analysis[] entries with url and content_hash
    * group[] entries with url and content_hash (for referenced vCons)
2. *Validate URLs*: MUST ensure HTTPS protocol and validate URL accessibility
3. *Download files*: MUST download from HTTPS URLs with proper error handling and retry logic
4. *Verify content hashes*: MUST verify against downloaded content using ALL provided hash algorithms
    * MUST fail if any hash does not match
    * MUST support SHA-512 as primary algorithm
5. *Determine file extension*: MUST determine using priority defined in Section 4.2
6. *Generate filename*: MUST use pattern [primary-content-hash].[extension]
7. *Check for duplicates*: If file already exists in files/ with same hash, MUST skip (automatic deduplication)
8. *Store file*: MUST add to files/ directory if not already present

6.3. Multi-vCon Bundle Creation

When bundling multiple vCons:

1. *Collect all vCons*: MUST identify all vCons to be included in the bundle

2. *Resolve files for each vCon*: MUST follow procedures in Section 6.2 for each vCon

3. *Automatic deduplication*: Files with identical content_hash values MUST be stored only once

    * This is especially useful when multiple vCons reference the same recording, attachment, or analysis

4. *Store each vCon*: MUST store as vcons/[uuid].json while preserving original security form

5. *Handle vCon references*:

    * Bundle creator SHOULD include referenced vCons (via Group objects) when available

    * If including referenced vCons, MUST add them to the vcons/ directory

    * UUIDs in the vCon JSON enable discovery without additional manifests

6.4. Bundle Assembly

1. *Create ZIP structure*: MUST create the following directory structure:

```
manifest.json
files/
vcons/
extensions/ (if applicable)
```

2. *Write manifest.json*: MUST write with required format identifiers

3. *Add all files*: MUST add to files/ directory with hash-based names

4. *Add all vCons*: MUST add to vcons/ directory as [uuid].json

5. *Add extensions*: If vCons use extensions (Section 8), MUST include extension data

6. *Create ZIP archive*: MUST create with appropriate compression settings

6.5. Bundle Validation

Bundle creators MUST perform these validation steps:

1. *Hash verification*: MUST verify all content hashes match downloaded content (all algorithms)

2. *File completeness*: MUST ensure all external references from all vCons are resolved and bundled

3. *Security form integrity*: MUST verify original signatures/ encryption structures remain intact

4. *UUID uniqueness*: MUST verify no duplicate vCon UUIDs exist in vcons/ directory

5. *File accessibility*: SHOULD verify all files in files/ are referenced by at least one vCon

6. *Manifest validity*: MUST verify top-level manifest.json is valid JSON with all required fields

7. Bundle Extraction and Usage

7.1. Extraction Process

Bundle consumers MUST follow this extraction process:

1. *Extract ZIP* to temporary or permanent directory

2. *Validate bundle structure*:

    * manifest.json exists and is valid

    * vcons/ directory exists

    * files/ directory exists

3. *Load manifest.json* and verify:

    * format is "vcon-bundle"

   * version is supported (e.g., "1.0")

4. *Discover vCons*: Scan vcons/ directory for all *.json files

5. *For each vCon*:

   * Parse according to security form (unsigned/signed/encrypted)

   * Identify file references via content_hash fields

   * Verify files exist in files/ directory

   * Verify file integrity using content_hash values

6. *Build relationships*: Check for vCon references (group[], redacted) to understand bundle structure

7.2. File Resolution

To access a file referenced in a vCon:

1. Extract content_hash from the vCon object (e.g., "sha512-GLy6IPa...")

2. Optionally extract mediatype to determine expected extension

3. Look in files/ directory for [content_hash].[ext]

   * If extension is known, search for exact match

   * If extension is unknown, scan for any file starting with [content_hash].

4. Verify file hash matches content_hash value (all algorithms if multiple provided)

5. Use file content for processing

7.3. Security Form Processing

For Signed vCons:

* Bundle consumers SHOULD verify JWS signatures using appropriate keys

* Signature verification failure SHOULD result in processing warnings or errors

* Consumers MAY choose to process unsigned payloads with appropriate warnings

For Encrypted vCons:

* Bundle consumers MAY have appropriate decryption keys

* Decryption is optional and only possible if keys are available

* If decryption keys are unavailable, consumers MAY process the encrypted vCon metadata without accessing the encrypted payload

* Decrypted content may itself be signed and require signature verification

8. Extensibility

8.1. Future vCon Extensions

The extensions/ directory provides support for future vCon extensions that define custom parameters or file types beyond the core specification.

Directory Structure:

```
extensions/
└── [extension-name]/
    ├── metadata.json          # Extension metadata and schema
    └── files/                 # Extension-specific files (if needed)
        └── [hash-based-names] # Following same naming conventions
```

Figure 3: Extension Directory Structure

Extension Guidelines:

* Each extension MUST have its own subdirectory named after the extension

* Extension-specific files SHOULD follow same hash-based naming conventions

* Extension metadata MUST be stored in extensions/[name]/metadata.json

* Extensions SHOULD use files/ directory for standard media files (dialog, attachments, analysis)

* Extensions MAY use extensions/[name]/files/ for extension-specific file types

Extension Metadata Schema:

```
{
  "extension_name": "mimi-messages",
  "extension_version": "1.0",
  "vcon_version_compatibility": ["0.3.0"],
  "bundle_format_version": "1.0",
  "description": "MIMI protocol message support for vCon"
}
```

Figure 4: Example Extension Metadata

8.2. Bundle Format Versioning

* Bundle format version MUST be tracked in manifest.json
* This specification defines version "1.0" (multi-vCon with simplified structure)
* Forward/backward compatibility handled via version negotiation
* New versions MAY add directories or fields to manifest.json
* New versions MUST NOT break existing core structure
* Implementations SHOULD support at least one previous major version

9. Security Considerations

9.1. Content Verification

* All files MUST be verified against ALL provided content_hash values before inclusion
* Bundle creators MUST validate HTTPS certificate chains when downloading external files
* Hash algorithms MUST include SHA-512 as required by vCon specification
* Additional hash algorithms MAY be supported as specified in vCon content_hash arrays
* Bundle creators MUST fail bundle creation if any content hash verification fails

* Bundle consumers MUST verify file hashes during extraction to detect tampering

9.2. vCon Security Form Preservation

* *Signed vCons*: Bundle creators MUST preserve complete JWS structure and SHOULD verify signatures
* *Encrypted vCons*: Bundle creators MUST preserve complete JWE structure
* *Security downgrades*: Bundle creators MUST NOT convert signed/encrypted vCons to unsigned form
* *Key management*: Bundle creators are responsible for having appropriate keys for encrypted vCons
* *Signature verification*: Bundle consumers SHOULD verify JWS signatures before trusting vCon content

9.3. Privacy Protection

* Bundle creators MUST preserve any privacy controls from original vCons
* Redacted vCons MUST maintain redaction integrity in bundles
* Unredacted vCons referenced by redacted vCons SHOULD have access controls enforced
* Party identification information MUST be handled according to applicable privacy regulations
* Bundle-level encryption (ZIP encryption) MAY be used but does not replace vCon-level security

9.4. File Deduplication Security

Deduplication via content_hash is safe because:

* Same hash means identical content (cryptographic guarantee with SHA-512)
* No information leakage from file reuse across vCons
* Each vCon’s content_hash values are independently verifiable

Bundle consumers SHOULD verify that shared files have identical content_hash in all referencing vCons.

9.5. Threat Model Considerations

* *Tamper detection*: Content hashes provide integrity verification for individual files
* *Bundle integrity*: Complete bundle integrity depends on:
  - ZIP file integrity
  - Individual file hash verification
  - vCon signature verification (for signed vCons)
* *Key exposure*: Encrypted vCons protect against key exposure better than ZIP-level encryption
* *Metadata leakage*: File names (hashes) and vCon structure may reveal conversation patterns even if content is encrypted

9.6. Bundle-Level Security

* *ZIP encryption*: MAY be used for additional protection but MUST NOT replace vCon-level security
* *Compression*: SHOULD be applied judiciously to avoid side-channel analysis via compressed size
* *File permissions*: Bundle extractors SHOULD set appropriate file permissions on extracted content
* *Temporary files*: Bundle creators SHOULD securely delete temporary files containing sensitive content

10. IANA Considerations

10.1. Media Type Registration

This specification defines a new media type for vCon Zip Bundle (.vconz) files and requests IANA registration:

Type name: application

Subtype name: vcon+zip

Required parameters: None

Optional parameters:

* version: Bundle format version (default "1.0")

* vcon-version: Source vCon specification version

Encoding considerations: Binary (ZIP archive)

Security considerations: See Section 9

Interoperability considerations: Standard ZIP format with specific internal structure

Published specification: This document

Applications that use this media type: vCon processing tools, conversation analysis systems, conversation archives

Fragment identifier considerations: Not applicable

Additional information:

* *Magic number:* ZIP signature (0x504B0304) with manifest.json as first entry

* *File extensions:* .vconz

* *Macintosh file type code:* Not assigned

* *Uniform Type Identifier:* public.vcon-zip-bundle

Person & email address to contact: Jeremie Miller jeremie.miller@gmail.com (mailto:jeremie.miller@gmail.com)

Intended usage: COMMON

Restrictions on usage: None

Author: Jeremie Miller

Change controller: IETF

11. Implementation Guidelines

11.1. Required Features

Implementations MUST support:

* Multi-vCon bundling with automatic file deduplication
* All four vCon content arrays (parties, dialog, analysis, attachments)
* All three vCon security forms (unsigned, signed JWS, encrypted JWE)
* SHA-512 content hash verification as primary algorithm
* Hash-based file naming with extension determination
* Standard ZIP format with specified directory structure
* vCon discovery via vcons/ directory scanning
* File lookup via content_hash values in vCon JSON
* Group object reference handling

11.2. Recommended Features

Implementations SHOULD support:

* JWS signature verification for signed vCons
* JWE decryption for encrypted vCons (with appropriate keys)
* Additional hash algorithms (SHA-256) for broader compatibility
* Bundle validation and integrity checking tools
* Extension directory support for future vCon extensions
* Efficient handling of large media files (streaming)

11.3. Optional Features

Implementations MAY support:

* Compression optimization for specific media types
* Incremental bundle updates (adding/removing vCons)
* Bundle format version migration tools
* Custom extension handling beyond core specification

* ZIP-level encryption for additional security
* Automated re-publishing tools for bundle-to-external conversion
* Bundle analysis tools (relationship graphs, statistics)

11.4. Implementation Validation

Implementations SHOULD provide validation for:

* Bundle structure completeness (required directories and files)
* Content hash verification (all algorithms)
* vCon relationship integrity (group references)
* Security form preservation (signatures/encryption intact)
* UUID uniqueness across all vCons
* File reference completeness (all referenced files present)
* Extension compatibility

12. Examples

12.1. Single vCon Bundle (Unsigned)

```
simple-call.vconz
├── manifest.json                              # Format: vcon-bundle v1.0
├── files/
│   ├── sha512-GLy6IPa...UQ.wav                # Audio recording
│   └── sha512-Transcript...XYZ.json           # Generated transcript
└── vcons/
    └── 0195544a-b9b1-8ee4-b9a2-279e0d16bc46.json  # Unsigned vCon
```

Figure 5: Single vCon Bundle Structure

12.2. Multi-vCon Bundle with Shared Files

```
support-case-bundle.vconz
├── manifest.json
├── files/
│   ├── sha512-Call1Audio...ABC.wav          # First call recording
│   ├── sha512-Call2Audio...DEF.wav          # Second call recording
│   ├── sha512-SharedDoc...GHI.pdf           # Document referenced by both vCons (de
duplicated!)
│   ├── sha512-Transcript1...JKL.json        # First call transcript
│   └── sha512-Transcript2...MNO.json        # Second call transcript
└── vcons/
    ├── 0195544a-b9b1-8ee4-b9a2-279e0d16bc46.json  # First call vCon
    └── 0195544b-c2d3-4e5f-6a7b-8c9d0e1f2a3b.json  # Second call vCon (same case)
```

Figure 6: Multi-vCon Bundle with Deduplication

Note: sha512-SharedDoc...GHI.pdf is referenced by both vCons but stored only once due to automatic deduplication.

12.3. Signed vCon with Encrypted Attachment

```
secure-conference.vconz
├── manifest.json
├── files/
│   ├── sha512-VideoConf...PQR.mp4           # Video recording
│   ├── sha512-Slides...STU.pdf              # Presentation slides
│   └── sha512-ConfReport...VWX.json         # Conference summary
└── vcons/
    └── 0195544c-1234-5678-9abc-def012345678.json  # JWS-signed vCon
```

Figure 7: Signed vCon Bundle

12.4. Group of Related vCons

```
conversation-thread.vconz
├── manifest.json
├── files/
│   ├── sha512-EmailThread...ABC.json        # Email conversation
│   ├── sha512-ChatLog...DEF.json            # Chat messages
│   └── sha512-PhoneCall...GHI.wav           # Follow-up phone call
└── vcons/
    ├── 01955450-aaaa-1111-2222-333344445555.json  # Email vCon
    ├── 01955451-bbbb-3333-4444-555566667777.json  # Chat vCon
    ├── 01955452-cccc-5555-6666-777788889999.json  # Phone vCon
    └── 01955453-dddd-7777-8888-999900001111.json  # Aggregate vCon (has group[] refer
ences to others)
```

Figure 8: Group of Related vCons

12.5. Redacted vCon Bundle

```
redacted-support-call.vconz
├── manifest.json
├── files/
│   ├── sha512-RedactedAudio...ABC.wav        # PII redacted audio
│   ├── sha512-OriginalAudio...DEF.wav        # Original unredacted audio
│   └── sha512-RedactedTranscript...GHI.json  # Redacted transcript
└── vcons/
    ├── 01955460-aaaa-bbbb-cccc-ddddeeeeeeee.json  # Unredacted vCon (original)
    └── 01955461-ffff-gggg-hhhh-iiiijjjjkkkk.json  # Redacted vCon with PII removed
```

Figure 9: Redacted vCon Bundle

12.6. vCon with Extension

```
mimi-messages.vconz
├── manifest.json
├── files/
│   └── sha512-ChatMessages...ABC.json        # MIMI message content
├── vcons/
│   └── 01955470-1111-2222-3333-444455556666.json  # vCon with MIMI extension
└── extensions/
    └── mimi-messages/
        ├── metadata.json                     # MIMI extension metadata
        └── files/
            └── sha256-MsgMetadata...DEF.cbor  # MIMI-specific metadata
```

Figure 10: vCon with Extension

12.7. Minimal Bundle (Empty Arrays)

```
minimal.vconz
├── manifest.json
└── vcons/
    └── 01955480-aaaa-bbbb-cccc-ddddeeeeeeee.json  # vCon with empty dialog/analysis/a
ttachments arrays
```

Figure 11: Minimal vCon Bundle

Note: No files/ directory needed since there are no external references.

13. Error Handling and Edge Cases

13.1. Bundle Creation Errors

Implementations MUST handle these error conditions:

External File Resolution Errors:

* *Network failures*: SHOULD retry with exponential backoff, MUST fail after maximum attempts

* *Hash mismatches*: MUST fail bundle creation with detailed error message showing expected vs actual hash

* *Missing files*: MUST fail bundle creation unless explicitly configured to skip

* *Access denied (403/401)*: MUST fail with security error, MUST NOT retry

vCon Security Form Errors:

* *Invalid JWS signatures*: Bundle creators SHOULD warn but MAY continue

* *JWE decryption failures*: MUST fail bundle creation for encrypted vCons

* *Missing decryption keys*: MUST fail with clear error message about key requirements

* *Malformed JWS/JWE structures*: MUST fail with structural validation errors

Validation Errors:

* *Invalid vCon structure*: MUST fail with schema validation errors

* *Missing UUID*: MUST fail (UUID required for filename)

* *Duplicate UUIDs*: MUST fail or prompt user for conflict resolution

* *Broken vCon references*: WARN if group[] references non-existent vCons

13.2. Bundle Extraction Errors

Implementations MUST handle these extraction scenarios:

Bundle Integrity Errors:

* *Corrupted ZIP*: MUST fail with file corruption error

* *Missing manifest.json*: MUST fail

* *Missing vcons/ directory*: MUST fail

* *Hash verification failures*: MUST fail unless configured for warnings

Security Processing Errors:

* *JWS signature verification failures*: SHOULD warn, MAY continue based on policy

* *JWE decryption failures*: MUST fail with key-related error message

* *Downgrade attacks*: MUST detect and prevent security form downgrades

File Resolution Errors:

* *Missing files*: MUST fail or warn if file referenced in vCon is not in files/

* *Hash mismatches*: MUST fail if file content doesn’t match content_hash

* *Orphaned files*: MAY warn if files in files/ are not referenced by any vCon

13.3. Edge Case Handling

Empty vCon Arrays:

* Bundles with vCons containing empty dialog, analysis, or attachments arrays are valid

* No files/ directory is required if no vCons have external references

* Minimal bundle can be just manifest.json and vcons/[uuid].json

Large File Handling:

* Implementations SHOULD support streaming for large media files during bundle creation

* Bundle size limits MAY be implemented with clear error messages

* Memory-efficient processing SHOULD be used for multi-gigabyte files

* ZIP64 format SHOULD be used for bundles exceeding 4GB

Unicode and Encoding:

* All JSON files MUST use UTF-8 encoding

* vCon UUIDs in filenames MUST use valid filesystem characters (UUIDs are safe)

* Content_hash values use base64url encoding (filesystem-safe)

* File extensions MUST handle Unicode characters properly

Incomplete Dialog Types:

* Dialog entries with type "incomplete" or "transfer" have no media files (per vCon spec)

* These do not contribute files to the bundle

* Bundle remains valid with dialog entries that have no corresponding files

Inline Content:

* vCon objects with inline content (body/encoding) instead of external references (url/content_hash)

* No files added to files/ directory for inline content

* Content remains embedded in the vCon JSON

14. Migration from Legacy Formats

If a previous legacy format existed with per-vCon metadata folders:

Legacy Structure (hypothetical):

```
bundle.vconz
├── vcon.json
├── dialog/
├── attachments/
├── analysis/
└── metadata/
    ├── manifest.json
    ├── bundle-info.json
    └── relationships.json
```

Figure 12: Hypothetical Legacy Bundle Structure

Migration to Version 1.0:

1. Move vcon.json to vcons/[uuid].json

2. Move all files from dialog/, attachments/, analysis/ to flat files/ directory

3. Remove metadata/ folder (information redundant with vCon JSON)

4. Create minimal manifest.json with version 1.0

5. For multi-vCon migration, repeat for each vCon and deduplicate files by hash

15. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[I-D.ietf-vcon-vcon-core] Petrie, D., "The JSON format for vCon - Conversation Data Container", Work in Progress, Internet-Draft, draft-ietf-vcon-vcon-core-01, 15 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-vcon-vcon-core-01>.

Appendix A. Acknowledgments

The author would like to thank the vCon working group for their contributions and feedback on this specification. Special thanks to Thomas McCarthy-Howe for his work on the core vCon specification, which this bundle format builds upon.

Appendix B. Change Log

B.1. draft-miller-vcon-zip-bundle-00

* Initial version defining vCon Zip Bundle format in kramdown-rfc

* Multi-vCon support with automatic deduplication

* Simplified flat file structure with hash-based naming
* Support for all vCon security forms (unsigned, signed, encrypted)
* Complete IANA media type registration
* Comprehensive security considerations
* Implementation guidelines and examples

Author's Address

Jeremie Miller
Email: jeremie.miller@gmail.com
URI: https://bsky.app/profile/jeremie.com