CalConnect TC

vObject — Integrity protection for vObject, vCard and iCalendar

Committee Draft Standard

Warning for drafts

This document is not a CalConnect Standard. It is distributed for review and comment, and is subject to change without notice and may not be referred to as a Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

The Calendaring and Scheduling Consortium, Inc. 2024

:2024

The Calendaring and Scheduling Consortium, Inc.

4390 Chaffin Lane
McKinleyville
California 95519
United States of America

copyright@calconnect.org
www.calconnect.org

Contents

:2024

Foreword

This document specifies an integrity checking mechanism and related properties for:

— vObject (I-D.calconnect-vobject-vformat)
— vCard version 4 (vCard v4) (RFC 6350); and
— iCalendar (Internet Calendaring and Scheduling Core Object Specification) (RFC 5545)

This work is produced by the CalConnect TC-VCARD and TC-CALENDAR committees.

1. Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

Internet-Draft draft-calconnect-vobject-vformat-00, RONALD HENRY TSE, PETER TAM, CYRUS DABOO and KENNETH MURCHISON. *The vObject Model and vFormat Syntax*. In: Internet-Draft. 2018. https://datatracker.ietf.org/doc/html/draft-calconnect-vobject-vformat-00. [viewed: December 9, 2024].

IETF RFC 2119, S. BRADNER. *Key words for use in RFCs to Indicate Requirement Levels*. In: BCP. 1997. RFC Publisher. https://www.rfc-editor.org/info/rfc2119. [viewed: December 9, 2024].

IETF RFC 5545, B. DESRUISSEAUX (ed.). *Internet Calendaring and Scheduling Core Object Specification (iCalendar)*. In: RFC. 2009. RFC Publisher. https://www.rfc-editor.org/info/rfc5545. [viewed: December 9, 2024].

IETF RFC 6321, C. DABOO, M. DOUGLASS and S. LEES. *xCal: The XML Format for iCalendar*. In: RFC. 2011. RFC Publisher. https://www.rfc-editor.org/info/rfc6321. [viewed: December 9, 2024].

IETF RFC 6350, S. PERREAULT. *vCard Format Specification*. In: RFC. 2011. RFC Publisher. https://www.rfc-editor.org/info/rfc6350. [viewed: December 9, 2024].

IETF RFC 6351, S. PERREAULT. *xCard: vCard XML Representation*. In: RFC. 2011. RFC Publisher. https://www.rfc-editor.org/info/rfc6351. [viewed: December 9, 2024].

IETF RFC 6352, C. DABOO. *CardDAV: vCard Extensions to Web Distributed Authoring and Versioning (WebDAV)*. In: RFC. 2011. RFC Publisher. https://www.rfc-editor.org/info/rfc6352. [viewed: December 9, 2024].

IETF RFC 7095, P. KEWISCH. *jCard: The JSON Format for vCard*. In: RFC. 2014. RFC Publisher. https://www.rfc-editor.org/info/rfc7095. [viewed: December 9, 2024].

IETF RFC 7265, P. KEWISCH, C. DABOO and M. DOUGLASS. *jCal: The JSON Format for iCalendar*. In: RFC. 2014. RFC Publisher. https://www.rfc-editor.org/info/rfc7265. [viewed: December 9, 2024].

IETF RFC 8126, M. COTTON, B. LEIBA and T. NARTEN. *Guidelines for Writing an IANA Considerations Section in RFCs*. In: BCP. 2017. RFC Publisher. https://www.rfc-editor.org/info/rfc8126. [viewed: December 9, 2024].

IETF RFC 8174, B. LEIBA. *Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words*. In: BCP. 2017. RFC Publisher. https://www.rfc-editor.org/info/rfc8174. [viewed: December 9, 2024].

2. Terms and definitions

For the purposes of this document, the following terms and definitions apply.

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 IETF RFC 2119 IETF RFC 8174 when, and only when, they appear in all capitals, as shown here.

The key words “**Private Use**”, “**Experimental Use**”, “ **Hierarchical Allocation**”, “**First Come First Served**”, “ **Expert Review**”, “**Specification Required**”, “**RFC Required**”, “ **IETF Review**”, “**Standards Action**” and “**IESG Approval**” in this document are to be interpreted as described in 4.

The definitions from Internet-Draft draft-calconnect-vobject-vformat-00 are inherited in this document unless explicitly overridden.

2.1. Definitions

Implementation Supported Checksum	An implementation is considered to support checksum calculation if it is able to calculate the checksum without external aid, i.e., it supports the parameters specified to calculate the checksum value.
Source Preferred Checksum Value (SPCV)	A CHECKSUM property that includes a PREF property parameter.
Receiver Preferred Checksum Value (RPCV)	The CHECKSUM property that uses the implementation's preferred checksum parameters.

3. Symbols And Abbreviations

3.1. Functions

These functions are **REQUIRED** and **MUST** be implemented for compliance to this document.

3.1.1. SORT

Sorts an list according to alphabetical order (A-Z).

3.1.2. LIST-TO-TEXT

This function returns a Unicode string (7) containing a string representation of a list of string values, each followed by a selected delimiter character.

```
LIST-TO-TEXT(list, delimiter) =
  value(list, 1) + delimiter +
  value(list, 2) + delimiter +
  ...
  value(list, last-element-position(list))
```

Figure 1

where: * + indicates concatenation; * `value(l, i)` is the i-th value in the list `l` in string representation; * `last-element-position(a)` returns the last element position of list `l`.

3.1.3. PREPHASH-PROPERTY-PARAMETER-KEY

This function returns a Unicode string (7) representation of the normalized property parameter key.

```
PREPHASH-PROPERTY-PARAMETER-KEY(parameter) = normalize(key(parameter))
```

Figure 2

where: * + indicates concatenation; * `key(parameter)` is the property parameter key; * `normalize(s)` is a function that normalizes the key s.

3.1.4. PREPHASH-PROPERTY-PARAMETER-VALUES

This function returns a Unicode string (7) representation of the normalized property parameter values.

```
PREPHASH-PROPERTY-PARAMETER-VALUES(parameter) =
  LIST-TO-TEXT(
```



```
    SORT(
      values(parameter, 1),
      values(parameter, 2),
      ...
    ),
    ";"
  )
```

Figure 3

where: * + indicates concatenation; * `values(parameter, i)` is the i-th property parameter value in `parameter`.

3.1.5. PREPHASH-PROPERTY-PARAMETER

Converts a property parameter into a string, with its key and values.

This function returns a Unicode string (7) containing a sequence of zero or more list values in string format, each followed by a ‘;’ character.

```
PREPHASH-PROPERTY-PARAMETER(parameter) =
  "{" +
    PREPHASH-PROPERTY-PARAMETER-KEY(property) + ":" +
    PREPHASH-PROPERTY-PARAMETER-VALUES(property) +
  "}"
```

Figure 4

where: * + indicates concatenation.

3.1.6. PREPHASH-PROPERTY-PARAMETERS

This function returns a Unicode string (7) representation of a set of property parameters.

We exclude the `VALUE` property parameter in this calculation (such as `VALUE=TEXT`) as this information is represented in PREPHASH-PROPERTY-VALUE-HASHA.

```
PREPHASH-PROPERTY-PARAMETERS(property) =
  "#" +
  LIST-TO-TEXT(
    SORT([
      PREPHASH-PROPERTY-PARAMETER(parameter(property, 1)),
      PREPHASH-PROPERTY-PARAMETER(parameter(property, 2)),
      ...
    ]),
    ";"
  )
```

Figure 5

where: * + indicates concatenation; * parameters(property, i) is the i-th parameter of `property`.

3.1.7. PREPHASH-PROPERTY-KEY

This function returns a Unicode string (7) representation of the normalized property key.

```
PREPHASH-PROPERTY-KEY(property) = normalize(key(property))
```

Figure 6

where: * + indicates concatenation; * key(property) is the property key; * normalize(s) is a function that normalizes the key s.

3.1.8. PREPHASH-PROPERTY-VALUE-HASHA

This function returns a Unicode string (7) representation of the normalized property value type. Since the property value type is represented here, we exclude the `VALUE` property parameter in PREPHASH-PROPERTY-PARAMETERS (such as `VALUE=TEXT`)

```
PREPHASH-PROPERTY-VALUE-HASHA(property) = UPCASE(type(property))
```

Figure 7

where: * + indicates concatenation; * type(property) is the property value type, if not explicitly provided, it should be filled in according to IETF RFC 6350; * normalize(s) is a function that normalizes the property value type s.

3.1.9. PREPHASH-PROPERTY-VALUES

This function returns a Unicode string (7) representation of the normalized property values.

Certain content types allow storing multiple values (as a list) in the same property line. For example, in the ADR and N properties, values are separated by the “;” delimiter, while in NICKNAME and CATEGORIES they are separated by the “,” delimiter 3.3.

```
PREPHASH-PROPERTY-VALUES(property) =
  LIST-TO-TEXT(
    SORT(
      values(property, 1),
      values(property, 2),
      ...
    ),
    ";"
  )
```

Figure 8

where: * + indicates concatenation; * values(property, i) is the i-th property value in `property`.

3.1.10. PREPHASH-PROPERTY

This function returns a Unicode string (7) representation of a single property.

```
PREPHASH-PROPERTY(property) =
  PREPHASH-PROPERTY-KEY(property) + ":" +
  PREPHASH-PROPERTY-VALUE-HASHA(property) + "/" +
  PREPHASH-PROPERTY-VALUES(property) + "?" +
  PREPHASH-PROPERTY-PARAMETERS(property)
```

Figure 9

where: * + indicates concatenation

3.1.11. HASH-PROPERTY

This function returns a Unicode string (7) representation of a single property.

```
HASH-PROPERTY-TO-TEXT(property) =
  PREPHASH-PROPERTY-KEY(property) + ":" +
```



```
  HASH(PREPHASH-PROPERTY(property)
```

Figure 10

where: * + indicates concatenation

3.1.12. HASH-AND-PREPHASH-PROPERTIES

This function returns a Unicode string (7) representation of a set of properties.

```
HASH-AND-PREPHASH-PROPERTIES(properties) =
  LIST-TO-TEXT(
    SORT([
      HASH-PROPERTY(property(properties, 1)),
      HASH-PROPERTY(property(properties, 2)),
      ...
    ]),
    CRLF
  )
```

Figure 11

where: * + indicates concatenation; * property(properties, i) is the i-th property of `properties`; * HASH(s) is selected cryptographic hash function applied to string `s`.

3.1.13. PREPHASH-COMPONENT-NAME

This function returns a Unicode string (7) representation of the normalized vObject name.

```
PREPHASH-COMPONENT-NAME(component) = normalize(name(component))
```

Figure 12

where: * name© is the component name of component `c`.

3.1.14. PREPHASH-COMPONENT

This function returns a Unicode string (7) representation of a vObject. The similarity of this representation with the vObject structure is intentional for readability purposes.

```
PREPHASH-COMPONENT(component) =
  "BEGIN:" + PREPHASH-COMPONENT-NAME(component) + ":CHECKSUM" + CRLF +
    HASH-AND-PREPHASH-PROPERTIES(properties(component)) + CRLF +
  "END:" + PREPHASH-COMPONENT-NAME(component) + ":CHECKSUM"
```

Figure 13

where: * + indicates concatenation; * properties© returns the properties of the component `c` in an list;

3.1.15. HASH-COMPONENT

This function returns a Unicode string (7) as the output of a selected cryptographic hash function applied on a vObject.

```
HASH-COMPONENT(component) = HASH(PREPHASH-COMPONENT(component))
```

Figure 14

3.1.16. HASH

This function returns the calculated hash of an input string and outputs the hash in string representation.

```
HASH(string) = generate-hash-function(
                 selected-hash-function,
                 selected-hash-parameters
               )(string)
```

Figure 15

where: * `generate-hash-function(a, p)` creates a new cryptographic hash function that uses the hash algorithm a with algorithm parameters p which takes a string input and generates the hash using a string output; * `selected-hash-function` is the selected cryptographic hash algorithm selected by the user (and/or CUA); * `selected-hash-parameters` are the selected parameters for the selected cryptographic hash function by the user (and/or CUA), and could be different per algorithm.

vObject — Integrity protection for vObject, vCard and iCalendar

4. TODOs

— Add CalDAV mechanisms and recommendations
— Fill in missing example hashes
— Fully replace normalization process with the vObject one, remove normalization process here

5. Introduction

The ubiquitous vCard and iCalendar standards, also known together as the “vObject” family of standards Internet-Draft draft-calconnect-vobject-vformat-00, powers digital contact exchanges, calendaring and scheduling on billions of devices today.

Integrity 2.1.2 is a key property of “information security” defined as the “preservation of confidentiality, integrity and availability of information” 2.33. When provided with a vObject, however, there is no inherent method to detect its own data integrity.

In reality, people are known to exchange vCard and iCalendar data through unreliable means, which could affect data integrity during its data lifecycle:

— transport of vObject data, such as over Internet mail IETF RFC 5322 and QR Codes ISO/IEC 18004;
— storage of vObject content, such as on disk, can be subject to silent corruption.

Previous standards were established in a time where integrity concerns were less widespread, and relied solely on data transport, application and storage integrity without considering on whether the content transmitted, processed or retrieved was as intended without modification or corruption.

This document specifically deals with information integrity in face of the following risks:

— vObjects on storage may face silent corruption;
— vObjects transmitted over networks or other channels may face network corruption that may go undetected by the underlying transport mechanism.

The standards subject to such risks include:

— vObject Internet-Draft draft-calconnect-vobject-vformat-00;
— vCard versions 2.1 [vCard21], 3 IETF RFC 2425 IETF RFC 2426 and 4 IETF RFC 6350;
— iCalendar IETF RFC 5545;
— Calendar Availability Extensions IETF RFC 7953;
— alternative formats for iCalendar and vCard, including xCal IETF RFC 6321, jCal IETF RFC 7265, xCard IETF RFC 6351, and jCard IETF RFC 7095.

This document provides:

— a stable mechanism to calculate vObject equivalence using cryptographic hash functions, valid across alternative representations, such as xCard/jCard and xCal/jCal;
— introduces a new property CHECKSUM to vObjects;
— usage of the CHECKSUM property on CardDAV IETF RFC 6352 and CalDAV IETF RFC 4791 systems;
— alternative representations of the CHECKSUM property for xCard IETF RFC 6351, jCard IETF RFC 7095, xCal IETF RFC 6321 and jCal IETF RFC 7265 representations of this property; and
— guidance to implementers on dealing with integrity concerns and the proper usage of CHECKSUM.

Organizations that implement information security management systems, such as ISO/IEC 27001, **MAY** find this document applicable to their own processes.

The decision to update the existing vCard version 4 IETF RFC 6350 and iCalendar IETF RFC 5545 standards were chosen to maintain maximum backwards compatibility.

This work is produced by the CalConnect TC-VCARD [CALCONNECT-VCARD] and TC-CALENDAR [CALCONNECT-CALENDAR] committees.

6. Properties

Property cardinalities are indicated in the same method as provided by IETF RFC 6350 based on ABNF 3.6.

6.1. CHECKSUM

These registration details for the CHECKSUM property adhere to rules specified in 10.2.1.

6.1.1. Namespace

Nil.

6.1.2. Property name

CHECKSUM

6.1.3. Purpose

Allows content integrity detection and verification against data corruption of a vObject.

6.1.4. Value type

A single text value.

6.1.5. Cardinality

*

6.1.6. Property parameters

HASHA, HASHP

6.1.7. Value

TEXT

6.1.8. Description

CHECKSUM is an **OPTIONAL** property of a vObject. There can be multiple CHECKSUM properties within the same vObject. vObject client implementations are **RECOMMENDED** to implement CHECKSUM for a basic level of integrity guarantee.

The CHECKSUM value used to compare the checksum of data should be selected in this way:

— the highest PREF value among all CHECKSUM properties; then
— the most applicable HASHA algorithm taking into account collision resistance and application support.

6.1.9. Format definition

ABNF:

```
CHECKSUM-param =  "VALUE=text"
CHECKSUM-param =  pid-param / pref-param / altid-param /
                  checksum-param-hasha / checksum-param-hashp /
                  iana-token

CHECKSUM-value = TEXT
  ; Value type and VALUE parameter MUST match.
```

Figure 16

6.1.10. Examples

```
CHECKSUM:
  ad58ca4f14b317dea48987f4991bdcd56fdf0f6a95049623f0fe5c4453d157e0
CHECKSUM;PREF=99:
  3ac0e03cccda6663ed32052749cc5c607d88e381f9cfcb795317bc39a57909e3
CHECKSUM;HASHA=sha224:
  22e92efac9d7b0e63695a9d960376ace1e69eb317e3d42c5c94f1401
```

Figure 17

7. Property Parameters

The CHECKSUM allowed property parameters of `PID`, `PREF`, `ALTID` have the same meaning as on other properties IETF RFC 6350.

7.1. PREF Property Parameter

The PREF property parameter indicates the preference of the vCard author on which CHECKSUM value to put most weight on.

Usage of this parameter is further explained in Clause 8.

7.2. HASHA Property Parameter

Registration details for the HASHA property parameter adhere to rules specified in 10.2.1

7.2.1. Namespace

Nil.

7.2.2. Parameter name

HASHA

7.2.3. Purpose

Specify the hash function used for the property value

7.2.4. Description

Possible values are defined in Clause 15.3.

The HASHA Property Parameter **MUST** not be applied on properties other than CHECKSUM unless specified.

New HASHA hash functions **MUST** be specified in a Standards Track RFC.

7.2.5. Format definition

ABNF:

```
hasha-param = "HASHA=" hasha-value *("," hasha-value)

hasha-value = "sha3-256" / iana-token / x-name
; This is further defined in <<checksum_functions>>
```

Figure 18

7.2.6. Examples:

```
CHECKSUM;HASHA=sha384:
  4055b176af753e251bc269007569c8f9633e6227a5f9727381cfba0bbb44a0c9
  25b8d31d72083d9cb4dc1da278f3a4e4
```

Figure 19

```
CHECKSUM;HASHA=streebog256:
  TODO
```

Figure 20

7.3. HASHP Property Parameter

Certain hash functions such as extendable output functions (XOFs) can be customized:

— SHAKE-128, SHAKE-256, cSHAKE-128, cSHAKE256, ParallelHash128, ParallelHash256 support customizable hash value length.
— cSHAKE-128, cSHAKE-256, support function name customization.
— cSHAKE-128, cSHAKE-256, ParallelHash128, ParallelHash256 support customizable bit strings.
— ParallelHash128, ParallelHash256 support customizable block sizes for parallel hashing.

Since each hash function may take different specifiers, each hash function identifier **MAY** specify its own set of HASHP specifiers in a particular order. The parameter value(s) entered **MUST** conform to the hash function's specification in a Standards Track RFC. An implementation **MUST** follow the value type interpretation specified for the hash function.

For example, in Clause 10.1.1, the cSHAKE-128 algorithm (with the identifier `cshake128`) takes `(L, N, S)` as input, where L is an integer to specify the output bit length, N is a text string representing the function name, S is a text string for customization purposes. When given a HASHP parameter value "512,address book,Orange", for the HASHA identifier `cshake128`, the implementation **MUST** recognize that L is the integer 512, N is the string "address book", and S is the string "Orange".

Registration details for the HASHP property parameter adhere to rules specified in 10.2.1

7.3.1. Namespace

Nil.

7.3.2. Parameter name

HASHP

7.3.3. Purpose

Describe hash function specifiers used for the property value.

7.3.4. Description

Provide specifiers for the HASHA hash function used to calculate the property value.

Possible values are defined in Clause 15.4.

The HASHP Property Parameter **MUST** not be applied on properties other than CHECKSUM unless specified.

7.3.5. Format definition

ABNF:

```
hashp-param = "HASHP=" hashp-value *("," hashp-value)

hashp-value = param-value
; This list of values must be specified in the exact order and value
  type defined in <<supported_table>>
```

Figure 21

Example(s):

```
CHECKSUM;HASHA=shake128;HASHP=512,"Directory Service Identifier":
  TODO
```

Figure 22

```
CHECKSUM;HASHA=parallelhash128;HASHP=64,512:
  TODO
```

Figure 23

8. Integrity Validation

8.1. Integrity In The vObject Life Cycle

Data integrity is important during storage and transmission of a vObject.

If an implementation stores vObjects directly on disk or in memory, it is **RECOMMENDED** that:

— Immediately prior to saving on target medium, a CHECKSUM is calculated and stored; and
— Immediately after retrieval from target medium, the included CHECKSUM is verified to ensure that it has not been corrupted.

An implementation that supports CHECKSUM **MUST** adhere to the following rules:

— If it supports importing of vObjects (including network import), it **MUST** verify the provided CHECKSUM property value immediately prior to import to ensure the vObject has not been damaged.
— If it supports exporting of vObject (including network export), it **MUST** insert at least one CHECKSUM property with corresponding checksum values to the vObject immediately prior to exporting, to ensure the recipient of the vObject can check against data integrity.

8.2. vObject Validity States

There are 3 validity states of a vObject:

Valid	This vObject is not corrupt.
Invalid	This vObject is corrupt.
Unable to determine	This vObject does not provide enough information to make a validity judgement.

8.3. Integrity Validity When Presented With A Single CHECKSUM Property

Given one CHECKSUM property, an implementation that supports the CHECKSUM property **SHOULD** reach the following conclusions about the vObject:

— Valid. The vObject is intact. Calculation by the implementation of the vObject's CHECKSUM property value was identical to the provided checksum value.
— Invalid. The vObject is corrupted. Calculation by the implementation of the vObject's CHECKSUM resulted in a different value as the provided checksum value.
— Unverified. The implementation is unable to determine data integrity of the vObject.
  — The vObject did not have a CHECKSUM property and therefore its data integrity cannot be verified.

- The vObject had a CHECKSUM property with a blank value and therefore its data integrity cannot be verified. This also signifies that the originator implementation was not able to calculate a CHECKSUM value.
- The vObject had a CHECKSUM property with a value but the current implementation does not support the chosen hash function, therefore its data integrity cannot be verified.

8.4. Integrity Validity When Presented With Multiple CHECKSUM Properties

If a vObject has more than one non-empty CHECKSUM property, an implementation should validate according to the rules below.

1) In the order of preference stated (PREF parameter value), validate all supported SPCV until one is verified.
   - If a vObject can be validated to any SPCV, it is deemed valid.
   - If all SPCVs are invalid, the vObject fails validation.
2) If a vObject does not have any SPCV, or the implementation does not support any SPCV, but contains a supported CHECKSUM property
   - If the CHECKSUM property value is valid, the vObject is deemed valid.
   - Otherwise, the vObject fails validation.

9. Method of CHECKSUM Value Calculation

The following method to calculate CHECKSUM is devised for these desired properties:

- Stable across alternative representation formats of the vCard and iCalendar, such as xCard/jCard.
- Allows comparison of equivalence of content rather than formatting. E.g., addition of new-lines within a vCard and order of listed properties do not affect the resulting checksum value.

For implementations that handle CHECKSUM, its calculation **MUST** be performed after all property updates including REV, which is often updated during save.

Steps to calculate CHECKSUM:

1) Calculate the hash value of the vObject
   a) Determine the need to add a new CHECKSUM property.
      - If there is no existing CHECKSUM property, add it as the last property of the vObject, with the selected cryptographic hash algorithm type and the selected hash parameters. Its value should be set to “” (empty string).
      - If there is an existing CHECKSUM property:
        - If its parameters are identical to the user’s current settings (or the CUA’s defaults), there is no need to add an extra CHECKSUM property. Set its value to “” (empty string).
        - Otherwise, add the extra CHECKSUM property as described above.
   b) Normalize the vObject in data model form (in accordance with Internet-Draft draft-calconnect-vobject-vformat-00)
   c) For each normalized property (including the newly added CHECKSUM property):
      i) For each normalized property parameter:
         A. For each normalized property parameter value:
            I. Obtain pre-hash string representation of the property parameter value
            II. Calculate hash value of the string representation of the property parameter value
         B. Obtain pre-hash string representation of the property parameter using hashes of its property parameter values
         C. Calculate hash value of the string representation of the property parameter
      ii) For each normalized property value:
         A. Obtain pre-hash string representation of the property value

   B. Calculate hash value of the string representation of the property value
   iii) Obtain pre-hash string representation of the property using hashes of its property values and property parameters
   iv) Calculate hash value of the string representation of the property
  d) Obtain pre-hash string representation of the vObject itself using hashes of its properties
  e) Calculate hash value of the string representation of the vObject.
2) This procedure is repeated to calculate the value for every CHECKSUM property (which may specify different cryptographic hash algorithms and parameters), with all CHECKSUM values set to "" (empty string) for calculation consistency.
   — If the implementation is unable to calculate the CHECKSUM due to unsupported or unrecognized parameters of a CHECKSUM property, assign the "" (empty string) as its value.
3) Enter the calculated CHECKSUM value for each CHECKSUM property.
4) The checksum calculation procedure is complete.

10. Cryptographic Hash Functions

The CHECKSUM value is calculated by a chosen cryptographic hash function specified in the HASHA property parameter. Certain hash functions accept customization specifiers, which can be specified in the HASHP property parameter.

10.1. Supported Hash Functions

CHECKSUM supports the following hash algorithms.

10.1.1. Hash Function Specifiers

CHECKSUM supported hash algorithms are listed in the following table.

— The CHECKSUM value contains the output of the hash function, which is usually stored in hexadecimal format as the `text` value type.
— The identifier from this table should be put as value of the property parameter HASHA.
— Algorithms with a "Variable" message digest size mean its length can be specified by a HASHP specifier.

Algorithms with no specifiers:

Table 1

Algorithm	Identifier	Message Digest Size (bits)	Description
SHA-2 SHA-224	sha224	224	IETF RFC 6234; NIST FIPS 180-4 fpd; ISO/IEC 10118-3 Dedicated Hash-Function 8 (SHA-224)
SHA-2 SHA-256	sha256	256	IETF RFC 6234; NIST FIPS 180-4 fpd; ISO/IEC 10118-3 Dedicated Hash-Function 4 (SHA-256)
SHA-2 SHA-384	sha384	384	IETF RFC 6234; NIST FIPS 180-4 fpd; ISO/IEC 10118-3 Dedicated Hash-Function 6 (SHA-384)
SHA-2 SHA-512	sha512	512	IETF RFC 6234; NIST FIPS 180-4 fpd; ISO/IEC 10118-3 Dedicated Hash-Function 5 (SHA-512)
SHA-2 SHA-512/224	sha512-224	224	NIST FIPS 180-4 fpd; ISO/IEC 10118-3 Dedicated Hash-Function 9 (SHA-512/224)
SHA-2 SHA-512/256	sha512-256	256	NIST FIPS 180-4 fpd; ISO/IEC 10118-3 Dedicated Hash-Function 10 (SHA-512/256)

Algorithm	Identifier	Message Digest Size (bits)	Description
WHIRLPOOL	whirlpool	512	[WHIRLPOOL]; ISO/IEC 10118-3 Dedicated Hash-Function 7 (WHIRLPOOL)
STREEBOG-256	streebog256	256	[STREEBOG] GOST R 34.11-2012; IETF RFC 6986; ISO/IEC 10118-3 Dedicated Hash-Function 12 (STREEBOG-256)
STREEBOG-512	streebog512	512	[STREEBOG] GOST R 34.11-2012; IETF RFC 6986; ISO/IEC 10118-3 Dedicated Hash-Function 11 (STREEBOG-512)
SHA3-224	sha3-224	224	NIST FIPS 202 fpd; ISO/IEC 10118-3 Dedicated Hash-Function 13 (SHA3-224)
SHA3-256	sha3-256	256	NIST FIPS 202 fpd; ISO/IEC 10118-3 Dedicated Hash-Function 14 (SHA3-256)
SHA3-384	sha3-384	384	NIST FIPS 202 fpd; ISO/IEC 10118-3 Dedicated Hash-Function 15 (SHA3-384)
SHA3-512	sha3-512	512	NIST FIPS 202 fpd; ISO/IEC 10118-3 Dedicated Hash-Function 16 (SHA3-512)
SM3	sm3	512	<<?I-D.shen-sm3-hash>>; [SM3]; ISO/IEC 10118-3 Dedicated Hash-Function 17 (SM3)
IANA registered hash algorithm	iana-token	iana-token	IANA
Vendor-specific hash algorithm	x-token	Vendor specific	Vendor specific

Algorithms with specifiers:

Table 2

Algorithm	Identifier	Message Digest Size (bits)	Specifier(s)	Description
SHAKE-128	shake128	Varys	L: integer (default: 256)	NIST FIPS 202 fpd
SHAKE-256	shake256	Varys	L: integer (default: 512)	NIST FIPS 202 fpd
cSHAKE-128	cshake128	Varys	L: integer (default: 256), N: text (default: “”), S: text (default: “”)	NIST SP 800-185 fpd
cSHAKE-256	cshake256	Varys	L: integer (default: 512), N: text (default: “”), S: text (default: “”)	NIST SP 800-185 fpd
ParallelHash-128	parallel128	Varys	B: integer (default: 64), L: integer (default: 256), S: text (default: “”)	NIST SP 800-185 fpd
ParallelHash-256	parallel256	Varys	B: integer (default: 64), L: integer (default: 256), S: text (default: “”)	NIST SP 800-185 fpd
IANA registered hash algorithm	iana-token	iana-token	iana-token	IANA
Vendor-specific hash algorithm	x-token	Vendor specific	Vendor specific	Vendor specific

10.1.1.1. Example

```
sha3-256('BEGIN:VCARD') = "f1fcbc9bddcd44b1e50db99a277bc868" +
                          "61736eb32cb30ef7e7a2c9ef95c05d50"
```

Figure 24

The default algorithm is `sha3-256`. An implementation that supports this document **MUST** support at least the `sha3-256` function.

10.1.2. The SHA-2 Hash Functions

Secure Hash Algorithm 2 (SHA-2) is a family of secure hash algorithms defined in NIST FIPS 180-4 fpd: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256.

— SHA-256 and SHA-512 are the two core hash functions that differ by process parameters, which produce a hash value of 256 and 512 bits respectively.
— SHA-224 is identical to SHA-256, except that different initial hash values are used, and the final hash value is truncated to 224 bits.
— SHA-384, SHA-512/224, SHA-512/256 are identical to SHA-512, except that different initial hash values are used, and the final hash value is truncated to 384, 224, 256 bits respectively. In particular, SHA-512/224 and SHA-512/256 use initial hash values generated by the “SHA-512/t IV Generation Function” given in NIST FIPS 180-4 fpd.

10.1.3. The WHIRLPOOL Hash Function

WHIRLPOOL is a hash function that operates on messages less than 2^256 bits in length, and produces a hash value of 512 bits [WHIRLPOOL].

It uses Merkle-Damgard strengthening and the Miyaguchi-Preneel hashing scheme with a dedicated 512-bit block cipher called “W” [WHIRLPOOL].

10.1.4. The SM3 Hash Function

SM3 is a hash function [SM3] for the use of electronic authentication service systems.

SM3 is an iterated hash function based on a Merkle-Damgard design, processes on 512-bit input message blocks with a 256-bit state, and produces a 256-bit hash value.

10.1.5. The SHA-3 Hash Functions

Secure Hash Algorithm-3 (SHA-3) is a family of hash functions defined in NIST FIPS 202 fpd consisting of:

— four cryptographic hash functions, SHA3-224, SHA3-256, SHA3-384, SHA3-512; and
— two extendable-output functions (XOFs), SHAKE128 and SHAKE256.

Each SHA-3 function is based on an instance of the KECCAK algorithm [KECCAK] which won the SHA-3 Cryptographic Hash Algorithm Competition NIST FIPS 202 fpd.

— SHA3-224, SHA3-256, SHA3-384, SHA3-512 produce a hash value output of 224, 256, 384 and 512 bits respectively.
— SHAKE128 and SHAKE256 are XOFs that produce output of arbitrary length, which can be specified using the “HASHP” property parameter.

Notes concerning SHA-3 based XOFs NIST FIPS 202 fpd:

— Output of a XOF can be considered as an infinite string, and the “HASHP” property parameter simply determines how many initial bits of the initial string to use.
— The SHAKE-256 and -128 functions, as long as at least 2x bits of their output is used, they have generic security strengths of 256 and 128 bits. However, using an excess of 64 or 32 bytes of their output respectively, does not increase their collision-resistance.

10.1.6. The STREEBOG Hash Functions

Streebog (or Stribog) is a family of two separate hash functions defined in the Russian standard GOST R 34.11-2012 [STREEBOG] where the functions differ in their output lengths, which are 256- and 512-bits respectively.

Streebog accepts message block sizes of 512-bits, and both functions only differ in the different IVs used other than the output length [STREEBOG].

10.1.7. The BLAKE2 Hash Functions

BLAKE2, described in [BLAKE2] and IETF RFC 7693, is a hash algorithm that comes in two flavors, BLAKE2b and BLAKE2s. It is the successor of BLAKE [BLAKE] which was a NIST SHA-3 competition finalist.

— BLAKE2b is optimized for 64-bit platforms and produces hash values of any size between 1 and 64 bytes
— BLAKE2s is optimized for 8- to 32-bit platforms and produces hash values of any size between 1 and 32 bytes

While BLAKE2 allows customizing parameters, including salt and a customization string, implementations that adhere to this specification should adopt BLAKE2 as defined in IETF RFC 7693.

10.1.8. The SHA-3 Extension Hash Functions

NIST SP 800-185 fpd defines a number of additional hash algorithms based on algorithms defined in NIST FIPS 202 fpd, including:

— cSHAKE-128, cSHAKE-256: customizable SHAKE functions, which take extra inputs of hash value length, a function name string, and a customization string;
— ParallelHash128, ParallelHash256: supports efficient hashing of very long strings by taking advantage of the parallelism available in modern processors based on SHAKE. These take the extra inputs of block size, hash value length and a customization string.

Both cSHAKE and ParallelHash are XOFs that can produce variable length output. The number suffix at their names mean the security strength bits of the algorithm.

10.2. Selection Considerations

10.2.1. Collision Resistance of Hash Function Families

For our purposes we specify the following strength families of hash algorithms. Hash functions placed in the higher bracket are considered “more resistant” in algorithm selection.

Table 3

Strength	Hash Function Identifier
1	sha224, sha256, sha384, sha512, sha512-224, sha512-256
2	whirlpool, streebog256, streebog512
3	blake2b256, blake2b384, blake2b512, blake2s224, blake2s256, sm3, shake128, shake256, sha3-224, sha3-256, sha3-384, sha3-512

10.2.2. Guidelines for Hash Function Selection

— Collision-resistance: higher bit length digests are generally preferable to lower bit length digests due to lower susceptibility to collisions.

— Performance: some hash functions are more computation intensive. Higher bit length digests generally require more computation to generate.
— History: a hash algorithm that has withstood cryptanalytic attacks provide better confidence than ones that have not been in widespread use.
— Availability and interoperability: certain hash algorithms, such as SHA-2 (IETF RFC 6234; NIST FIPS 180-4 fpd; ISO/IEC 10118-3 Dedicated Hash-Function 4 (SHA-256)), are more prevalently available on computing platforms.

Selection of the hash function should be based on a balance of collision resistance, performance, history and interoperability.

10.2.3. Hash Functions Considered Unsuitable

The following hash functions are specifically excluded due to stated reasons:

— RIPEMD-160 ISO/IEC 10118-3 Dedicated Hash-Function 1 and RIPEMD-128 ISO/IEC 10118-3 Dedicated Hash-Function 2, are specifically excluded as they do no longer provide a sufficient level of collision resistance, see 7.1 Note 2 8 Note 2. The RIPEMD optional extensions RIPEMD-256 and RIPEMD-320 [RIPEMD160] are also excluded as they are of the same security levels as RIPEMD-128 and RIPE-160 respectively.
— SHA-1 IETF RFC 3174 ISO/IEC 10118-3 Dedicated Hash-Function 3 is excluded as it does not provide a sufficient level of collision resistance, see 9.1 Note 2.
— CRC-32 ISO/IEC 13239 and in general CRC algorithms are excluded due to weak collision resistance.

11. Using CHECKSUM With Server Support

11.1. Usage of CHECKSUM in vCards on CardDAV servers

CardDAV servers are **RECOMMENDED** to calculate and provide an extra CHECKSUM property for al vCard retrieval requests in order to provide a base level of integrity guarantee.

The CHECKSUM property and its parameters are fully compatible with the CardDAV mechanism described in IETF RFC 6352.

11.1.1. Creating And Updating Address Object Resources

6.3.2 specifies how to create address object resources.

An implementation abiding to this specification **MUST** augment this process according to the following.

11.1.1.1. Client Implementations Should Transmit With CHECKSUM

— When a client issues a PUT to create an address object resource, a CHECKSUM property **SHOULD** be included in the request.
— The CHECKSUM property value **MAY** be empty if the client wishes the server to calculate the value according to the given HASHA and/or HASHP parameters.

11.1.2. Additional Server Semantics for PUT, COPY and MOVE

This specification creates an additional precondition and postcondition for the PUT, COPY, and MOVE methods when:

— A PUT operation requests an address object resource to be placed into an address book collection; and

— A COPY or MOVE operation requests an address object resource to be placed into (or out of) an address book collection.

11.1.2.1. Only Admit Valid vCard Data From Client

11.1.2.2. Additional Precondition

`(CARDDAV:valid-address-data-checksum)` The address object resource submitted in the PUT request, or targeted by a COPY or MOVE request, contains a CHECKSUM property:

— The address object resource's integrity **MUST** be valid as determined by methods of this specification.
— If the resource contains an empty CHECKSUM property value, the server **SHOULD** fill in the property value with its own calculation.
— The CHECKSUM property value **SHOULD** be stored by the server to enable data integrity verification.
— If the resource CHECKSUM is deemed invalid, the server **SHOULD** respond with a `409` (Conflict) status to indicate to the client so, hence the `<CARDDAV:valid-address-data-checksum>` condition is not met. In this case, the client may choose to empty the CHECKSUM property value for re-submission.

11.1.2.3. Resolve Discrepancy Between Server And Client vCard Data

Certain servers perform silent changes or cleanups of client provided vCard data when stored as address object resources, such as the order of property parameters or scrubbed values.

The resulting vCard data stored on the server (and when returned back to the client) may end up different than that of the client without its knowledge. It is therefore necessary for the client to be reported on such modifications.

11.1.2.4. Additional Postcondition

`(CARDDAV:resource-not-modified)`: The address object resource should not be modified by the server such that its original CHECKSUM value becomes invalid.

— After action execution, the server should re-calculate the CHECKSUM property value based on the retrieved address object resource.
— If the CHECKSUM property value is now different, the server **SHOULD** respond to client with the latest address object resource and the new CHECKSUM so that the client knows the resource has been changed by the server.

11.2. Usage of CHECKSUM with CalDAV

The CalDAV IETF RFC 4791 calendar access protocol allows clients and servers to exchange iCalendar data. iCalendar data is typically stored in calendar object resources on a CalDAV server.

A CalDAV server is **RECOMMENDED** to return iCalendar data with integrity protection.

11.2.1. Creating Calendar Resources

A CalDAV client typically updates the calendar object resource data via an HTTP PUT request, which requires sending the entire iCalendar object in the HTTP request body.

11.3. Usage of CHECKSUM with iTIP

iTIP IETF RFC 5546 defines how iCalendar data can be sent between calendar user agents to schedule calendar components between calendar users.

This specification is compatible with iTIP transfer of iCalendar data.

12. Alternative vObject Representations

12.1. xCard

The XML representation IETF RFC 6351 of the CHECKSUM property follows the example shown below. For this property, the value type **MUST** be set to “text” and parameter “type” **MUST** also be set to “text”.

```
<checksum>
  <parameters>
    <hasha>
      <text>sha224</text>
    </hasha>
    <pref>
      <integer>99</integer>
    </pref>
  </parameters>
  <text>22e92efac9d7b0e63695a9d960376ace1e69eb317e3d42c5c94f1401</text>
</checksum>
```

Figure 25

12.2. jCard

The JSON representation of the CHECKSUM property follows IETF RFC 7095 as the example shown below.

```
["checksum",
  { "hasha": "sha224", "pref": "99" },
  "text",
  "22e92efac9d7b0e63695a9d960376ace1e69eb317e3d42c5c94f1401"
]
```

Figure 26

13. Implementation Notes

13.1. vCard REV Update Guidelines For The CHECKSUM Property

Updating of the CHECKSUM property value should not affect the REV value of a vCard. However, if a CHECKSUM property is newly inserted, or its parameters changed (such as HASHA or HASHP), then the REV value should be updated according to IETF RFC 6350.

13.2. Calculating CHECKSUM From An xCard

Implementers **MUST** ignore individual parameter value types in xCard (6, Appendix A 4.1) during CHECKSUM value calculation to be compatible with vCard and jCard, as individual parameter value types are implicit (not explicitly represented) in both vCard and jCard properties.

13.3. Backwards Compatibility Concerns

If an implementation does not support the CHECKSUM property, it **MUST** ignore the CHECKSUM property entirely without providing it any value. If an incorrect value is provided, the receiving end of this vObject may falsely assume that the vObject is broken.

13.4. Unsupported Property Parameters

— If an implementation supports the CHECKSUM property but not certain parameters (e.g., a specified hash function), it **MUST** leave that property value empty as the insertion of the CHECKSUM property indicates the wish of the user to utilize it.
— If an implementation supports the CHECKSUM property, it **MUST** calculate the checksum values for every CHECKSUM property in the vObject.

13.5. Recommendations for Client User Applications

13.5.1. User Experience

— The CUA **SHOULD** honestly reflect checksum validation results to the user to allow further action from the user, e.g., to seek retransmission of the vObject.

13.5.2. Ongoing Improvements

— Cryptographic hash algorithms can break overtime. There will be a time when best practice designates a better one, CUA **SHOULD** take this in mind and promote best practice to update its security profile.

14. Security Considerations

— The function of the CHECKSUM property depends on the collision-free property of cryptographic hash functions. However, as time passes, today's recommended cryptographic hash functions may no longer be considered reliable in the future. Implementers **MUST** take this into account and update its security profile according to the latest best practice on cryptographic hash functions.
— The CHECKSUM property is not designed to protect against intentional and unauthorized modification. A malicious party with access to the vObject (such as a "man-in-the-middle attack" 3.3.5 4) could both modify the data and the CHECKSUM property at the same time and prevent detection.
— The CHECKSUM property is not designed to address data authenticity (2.8 2.1.3) concerns. A malicious party may send a vObject posing as another entity. This document does not protect against that situation.
— While many vObject properties can be used to transport URIs, the CHECKSUM property specifically does not allow setting a URI as its value due to extra security risks raised during the reference step to a URI (7). In any case, it is easy for an attacker to directly modify the CHECKSUM instead of modifying the results at a third-party URI, and therefore would not improve integrity protection of the vObject.
— Security considerations around vObject formats in the following documents **MUST** be adhered to:
  — vCard: IETF RFC 6350
  — iCalendar: IETF RFC 5545, IETF RFC 5789, IETF RFC 4791

15. IANA Considerations

15.1. Common vObject Registries

The IANA has created and will maintain the following registries under the heading "vObject Common Elements".

The registry policy is **Specification Required**; any newly proposed specification **MUST** be reviewed by the designated expert.

15.2. Registration Procedure For New Hash Functions And Hash Function Specifiers

This section defines the process for registering new or modified hash functions and hash function specifiers with IANA.

The IETF mailing lists for vObject (vobject@ietf.org), CardDAV (vcarddav@ietf.org) and CalDAV (vcaldav@ietf.org) **SHOULD** be used for public discussion of additional hash functions and hash function specifiers for the CHECKSUM property prior to registration.

The registration procedure specified in IETF RFC 6350 should be followed to register additional hash functions and hash function specifiers for vObjects.

15.3. vObject Hash Functions Registry

The registry policy is **Specification Required**; any newly proposed registration **MUST** be reviewed by the designated expert.

The registry **SHOULD** contain the following note:

```
Note: Experts are to verify that the proposed registration
*SHOULD* provide benefits for the wider vObject community,
and provides a publicly-available standard that can be implemented in
an interoperable way. Hash functions are preferrably approved by the CFRG
with references to IETF-published documents. The "Reference" value should
point to a document that details the implementation of this hash function
in the vObject context.
```

Figure 27

15.3.1. Registration Template for vObject Hash Functions

A Hash Function is defined by completing the following template.

Identifier — The identifier of the hash function.

Description — A short but clear description of the hash function, with any special notes about it.

Example(s) — One or more examples of input and output of the hash function.

15.3.2. Initial Registrations

The following table has been used to initialize the Hash Functions registry.

Table 4

Identifier	Description	Example(s)
sha224	SHA-2 SHA-224 Clause 10.1.2	Appendix A.2.1
sha256	SHA-2 SHA-256 Clause 10.1.2	Appendix A.2.2
sha384	SHA-2 SHA-384 Clause 10.1.2	Appendix A.2.3
sha512	SHA-2 SHA-512 Clause 10.1.2	Appendix A.2.4
sha512-224	SHA-2 SHA-512/224 Clause 10.1.2	Appendix A.2.5
sha512-256	SHA-2 SHA-512/256 Clause 10.1.2	Appendix A.2.6
whirlpool	WHIRLPOOL Clause 10.1.3	Appendix A.2.7
streebog256	GOST R 34.11-2012 256 bits Clause 10.1.6	Appendix A.2.8
streebog512	GOST R 34.11-2012 512 bits Clause 10.1.6	Appendix A.2.9
sha3-224	SHA-3-224 Clause 10.1.5	Appendix A.2.10

Identifier	Description	Example(s)
sha3-256	SHA-3-256 Clause 10.1.5	Appendix A.2.11
sha3-384	SHA-3-384 Clause 10.1.5	Appendix A.2.12
sha3-512	SHA-3-512 Clause 10.1.5	Appendix A.2.13
blake2b-256	BLAKE2b-256 Clause 10.1.7	Appendix A.2.15
blake2b-384	BLAKE2b-384 Clause 10.1.7	Appendix A.2.15.1
blake2b-512	BLAKE2b-512 Clause 10.1.7	Appendix A.2.16
blake2s-224	BLAKE2s-224 Clause 10.1.7	Appendix A.2.17
blake2s-256	BLAKE2s-256 Clause 10.1.7	Appendix A.2.18
sm3	OSCCA SM3 Clause 10.1.4	Appendix A.2.14
shake128	SHAKE-128 Clause 10.1.5	Appendix A.2.19
shake256	SHAKE-256 Clause 10.1.5	Appendix A.2.20
cshake128	cSHAKE-128 Clause 10.1.8	Appendix A.2.21
cshake256	cSHAKE-256 Clause 10.1.8	Appendix A.2.22
parallel128	ParallelHash128 Clause 10.1.8	Appendix A.2.23
parallel256	ParallelHash256 Clause 10.1.8	Appendix A.2.24

15.4. vObject Hash Function Specifier Registry

The registry policy is **Specification Required**; any newly proposed registration **MUST** be reviewed by the designated expert.

The registry **SHOULD** contain the following note:

```
Note: Experts are to verify that the proposed registration
*SHOULD* provide benefits for the wider vObject community,
and provides a publicly-available standard that can be implemented in
an interoperable way. Hash function specifiers are preferrably approved by the
CFRG
with references to IETF-published documents. The "Reference" value should
point to a document that details the implementation of this hash function
in the vObject context.
```

Figure 28

The “Specifier(s)” column in the registry **SHOULD** adhere to the following format:

ABNF:

```
specifier = specifier-tuple *("," specifier-tuple)

specifier-tuple = specifier-key ": " specifier-value-type +
                  "(default: " specifier-description ")"
specifier-key = text
specifier-value-type = value-type
specifier-description = text
```

Figure 29

15.4.1. Registration Template for vObject Hash Function Specifiers

A Hash Function Specifier is defined by completing the following template.

Identifier
: Identifier of the hash function that this specifier applies to.

Description
: A short but clear description of the hash function specifier.

Order
: In which position in the specifier list should this specifier be found.

Value Type The type of specifier value (e.g., text).

Example(s) One or more examples of input and output of the hash function.

15.4.2. Initial Registrations

The following table has been used to initialize the Hash Function Specifier registry.

Table 5

ID	Order	Description	Value Type	Example(s)
shake128	1	L: output bit length	integer	Appendix A.2.19
shake256	1	L: output bit length	integer	Appendix A.2.20
cshake128	1	L: output bit length	integer	Appendix A.2.21
cshake128	2	N: function-name	text	Appendix A.2.21
cshake128	3	S: customization string	text	Appendix A.2.21
cshake256	1	L: output bit length	integer	Appendix A.2.22
cshake256	2	N: function-name	text	Appendix A.2.22
cshake256	3	S: customization string	text	Appendix A.2.22
parallel128	1	B: block size in bytes	text	Appendix A.2.23
parallel128	2	L: output bit length	integer	Appendix A.2.23
parallel128	3	S: customization string	text	Appendix A.2.23
parallel256	1	B: block size in bytes	text	Appendix A.2.24
parallel256	2	L: output bit length	integer	Appendix A.2.24
parallel256	3	S: customization string	text	Appendix A.2.24

15.5. Property Registrations

This document defines the following new properties to be added to the registries defined in:

— vCard registry, 10.3.1
— iCalendar registry, 8.3.2

Table 6

Property	Status	Reference
CHECKSUM	Current	This document: Clause 6.1

15.6. Parameter Registrations

This document defines the following new property parameters to be added to the registries defined in:

— vCard registry, 10.3.2
— iCalendar registry, 8.3.3:

Table 7

Parameter	Status	Reference
HASHA	Current	This document: Clause 7.2
HASHP	Current	This document: Clause 7.3

15.6.1. Parameter Value Registrations

This document defines the following new parameter values to be added to the registries defined in:

— vCard registry, 10.3.4

— iCalendar registry, 8.3.4:

Table 8

Property	Parameter	Value	Reference
CHECKSUM	HASHA	sha224	This document: Clause 7.2
CHECKSUM	HASHA	sha256	This document: Clause 7.2
CHECKSUM	HASHA	sha384	This document: Clause 7.2
CHECKSUM	HASHA	sha512	This document: Clause 7.2
CHECKSUM	HASHA	sha512-224	This document: Clause 7.2
CHECKSUM	HASHA	sha512-256	This document: Clause 7.2
CHECKSUM	HASHA	whirlpool	This document: Clause 7.2
CHECKSUM	HASHA	streebog256	This document: Clause 7.2
CHECKSUM	HASHA	streebog512	This document: Clause 7.2
CHECKSUM	HASHA	sha3-224	This document: Clause 7.2
CHECKSUM	HASHA	sha3-256	This document: Clause 7.2
CHECKSUM	HASHA	sha3-384	This document: Clause 7.2
CHECKSUM	HASHA	sha3-512	This document: Clause 7.2
CHECKSUM	HASHA	sm3	This document: Clause 7.2
CHECKSUM	HASHA	blake2b256	This document: Clause 7.2
CHECKSUM	HASHA	blake2b384	This document: Clause 7.2
CHECKSUM	HASHA	blake2b512	This document: Clause 7.2
CHECKSUM	HASHA	blake2s224	This document: Clause 7.2
CHECKSUM	HASHA	blake2s256	This document: Clause 7.2
CHECKSUM	HASHA	shake128	This document: Clause 7.2
CHECKSUM	HASHA	shake256	This document: Clause 7.2
CHECKSUM	HASHA	cshake128	This document: Clause 7.2
CHECKSUM	HASHA	cshake256	This document: Clause 7.2
CHECKSUM	HASHA	parallel128	This document: Clause 7.2
CHECKSUM	HASHA	parallel256	This document: Clause 7.2

16. Acknowledgements

The authors wish to thank the following parties who helped this materialize and for their support of a better world.

— their families
— the CalConnect TC-VCARD committee
— members and the Board of Directors of CalConnect

This specification was developed by the CalConnect TC-VCARD committee.

Appendix A (normative) Examples

A.1. vCard CHECKSUM

A.1.1. Original vCard

```
BEGIN:VCARD
VERSION:4.0
KIND:individual
FN:Martin Van Buren
N:Van Buren;Martin;;;Hon.
TEL;VALUE=uri;PREF=1;HASHA="voice,home":tel:+1-888-888-8888;ext=8888
END:VCARD
```

Figure A.1

A.1.2. Setup

Location of the CHECKSUM property within the VCARD component does not matter as the method of calculation is agnostic with regards to line location of a property.

vCard extended with CHECKSUM property for CHECKSUM calculation at the last line, specifying the sha512 algorithm and value type `STRING`:

```
BEGIN:VCARD
VERSION:4.0
KIND:individual
FN:Martin Van Buren
N:Van Buren;Martin;;;Hon.
TEL;VALUE=uri;PREF=1;TYPE="voice,home":tel:+1-888-888-8888;ext=8888
CHECKSUM;VALUE=TEXT;HASHA=sha3-256:
END:VCARD
```

Figure A.2

A.1.3. Normalization: Properties

```
PREPHASH-PROPERTY("VERSION:4.0") =
  "VERSION:TEXT/[4.0]?#[]"

PREPHASH-PROPERTY("KIND:individual") =
  "KIND:TEXT/[individual]?#[]"

PREPHASH-PROPERTY("FN:Martin Van Buren") =
  "FN:TEXT/[Martin Van Buren]?#[]"

PREPHASH-PROPERTY("N:Van Buren;Martin;;;Hon.") =
  "N:TEXT/[Van Buren;Martin;;;Hon.]?#[]"

PREPHASH-PROPERTY("TEL;VALUE=uri;PREF=1;HASHA="voice,home":") =
  "TEL:URI/[tel:+1-888-888-8888;ext=8888]" +
  "?#[{PREF:[1]};{TYPE:[home;voice]}]"

PREPHASH-PROPERTY("CHECKSUM;VALUE=TEXT;HASHA=sha512:") =
  "CHECKSUM:TEXT/[]?#[{HASHA:[sha512]}]"
```

Figure A.3

A.1.4. Cryptographic Hashing: Properties

```
HASH("VERSION:TEXT/[4.0]?#[]") =
  "de2a19b21ce6dbbafd3feedebf7560966242d4af0bac8e380024135809729ba4"
HASH("KIND:TEXT/[individual]?#[]") =
  "25603f59dc07e045b470e3d773da10e2485c078c80f4a048c2e1cbeb678ab406"
HASH("FN:TEXT/[Martin Van Buren]?#[]") =
  "a9124e1bd40c8a2cb4031b4140629e2472046f837dddc379a257d5f6e7bceedd"
HASH("N:TEXT/[Van Buren;Martin;;;Hon.]?#[]") =
  "c11eadabeee1252502ddc6c085e5bd7fd48ae183f50399b953bb78a927172dc5"
HASH(
  "TEL:URI/[tel:+1-888-888-8888;ext=8888]" +
  "?#[{PREF:[1]};{HASHA:[home;voice]}]"
) = "dc22433d7cb2445dd9f083a1d998ee00e8f2f369f0e18ddb827f8135f0d7b30d"
HASH("CHECKSUM:TEXT/[]?#[{HASHA:[sha512]}]") =
  "65d32764ab8c9fcdd324f24409c65a45529f4a6df5cd070378463a177de04917"
```

Figure A.4

A.1.5. Normalization: Component

```
HASH-AND-PREPHASH-PROPERTIES(properties) = LIST-TO-TEXT(
  [
    "CHECKSUM:" +
      HASH("CHECKSUM:TEXT/[]?#[{HASHA:[sha512];VALUE:[TEXT]}]"),
    "FN:" +
      HASH("FN:TEXT/[Martin Van Buren]?#[{VALUE:[TEXT]}]"),
    "KIND:" +
      HASH("KIND:TEXT/[individual]?#[{VALUE:[TEXT]}]"),
    "N:" +
      HASH("N:TEXT/[Van Buren;Martin;;;Hon.]?#[{VALUE:[TEXT]}]"),
    "TEL:" +
      HASH(
        "TEL:URI/[tel:+1-888-888-8888;ext=8888]?" +
        "#[{PREF:[1]};{HASHA:[voice;home]};{VALUE:[TEXT]}]"
      ),
    "VERSION:" +
      HASH("VERSION:TEXT/[4.0]?#[{VALUE:[TEXT]}]")
  ],
  CRLF
)
```

Figure A.5

```
PREPHASH-COMPONENT(component) =
"BEGIN:VCARD:CHECKSUM
CHECKSUM:65d32764ab8c9fcdd324f24409c65a45529f4a6df5cd070378463a177de04917
FN:a9124e1bd40c8a2cb4031b4140629e2472046f837dddc379a257d5f6e7bceedd
KIND:25603f59dc07e045b470e3d773da10e2485c078c80f4a048c2e1cbeb678ab406
N:c11eadabeee1252502ddc6c085e5bd7fd48ae183f50399b953bb78a927172dc5
TEL:dc22433d7cb2445dd9f083a1d998ee00e8f2f369f0e18ddb827f8135f0d7b30d
VERSION:de2a19b21ce6dbbafd3feedebf7560966242d4af0bac8e380024135809729ba4
END:VCARD:CHECKSUM
"
```

Figure A.6

A.1.6. Cryptographic Hashing: Component

```
HASH-COMPONENT(component) =
```



```
  "212f3486f968df73dc9b9f909e8dfedae866135aeef2ceeaa3393675806960d1"
```

Figure A.7

A.1.7. Final Checksum

This is the final checksum of this component using the sha3-256 hash method.

The final vCard:

```
BEGIN:VCARD
VERSION:4.0
KIND:individual
FN:Martin Van Buren
N:Van Buren;Martin;;;Hon.
TEL;VALUE=uri;PREF=1;HASHA="voice,home":tel:+1-888-888-8888;ext=8888
CHECKSUM;VALUE=TEXT;HASHA=sha3-512:
  212f3486f968df73dc9b9f909e8dfedae866135aeef2ceeaa3393675806960d1
END:VCARD
```

Figure A.8

A.2. Hash Functions Registry Examples

A.2.1. SHA-2 SHA-224

```
input("BEGIN:VCARD") = "22e92efac9d7b0e63695a9d960376ace" +
                       "1e69eb317e3d42c5c94f1401"
```

Figure A.9

A.2.2. SHA-2 SHA-256

```
input("BEGIN:VCARD") = "99e3e442c1a5cbd115baa26d077c6bbb" +
                       "423310cd4990051d8974c3b2d581c3d4"
```

Figure A.10

A.2.3. SHA-2 SHA-384

```
input("BEGIN:VCARD") = "4055b176af753e251bc269007569c8f9" +
                       "633e6227a5f9727381cfba0bbb44a0c9" +
                       "25b8d31d72083d9cb4dc1da278f3a4e4"
```

Figure A.11

A.2.4. SHA-2 SHA-512

```
input("BEGIN:VCARD") = "a2d5b1339599039a7058d8446442f2cb" +
                       "341a149064eacb31fdc410e57e239849" +
                       "88efffc6f15842a6a6ae08fb4d791d2f" +
                       "9dd9dab4cf724f8e75b9fff2c21d3e1c"
```

Figure A.12

A.2.5. SHA-2 SHA-512/224

```
input("BEGIN:VCARD") = ""
```

Figure A.13

A.2.6. SHA-2 SHA-512/256

```
input("BEGIN:VCARD") = ""
```

Figure A.14

A.2.7. WHIRLPOOL (512-bit)

```
input("BEGIN:VCARD") = "6e9ca195e4e87afcc624fa88334088fb" +
                       "71038273b16cb1e47888072c03cfaf79" +
                       "29539375c5ff92fbd82b73924ed60b1d" +
                       "c9bb17bdb1bd2447cf2d3218a356736a"
```

Figure A.15

A.2.8. STREEBOG-256

```
input("BEGIN:VCARD") = ""
```

Figure A.16

A.2.9. STREEBOG-512

```
input("BEGIN:VCARD") = ""
```

Figure A.17

A.2.10. SHA-3-224

```
input("BEGIN:VCARD") = "630d7879cac76d221565dcc335bff595" +
                       "158b3496713910cc92166762"
```

Figure A.18

A.2.11. SHA-3-256

```
input("BEGIN:VCARD") = "f1fcbc9bddcd44b1e50db99a277bc868" +
                       "61736eb32cb30ef7e7a2c9ef95c05d50"
```

Figure A.19

A.2.12. SHA-3-384

```
input("BEGIN:VCARD") = "2d27f6dccb17bf6da9800386aae4a991" +
                       "cfdebc4f3a971f7d0e5264aa0c7b1394" +
                       "514c2eb5bd724f0702062935de9fd92d"
```

Figure A.20

A.2.13. SHA-3-512

```
input("BEGIN:VCARD") = "ceb5ab39356ce3440d99375a3098cfa5" +
                       "20db3d54a3c15184be9f19f6483165e7" +
                       "8769d4cf2e7f0976422ed4856122c957" +
                       "d22a3c4b922b733ccefc802eed753027"
```

Figure A.21

A.2.14. SM3 (256-bits)

```
input("BEGIN:VCARD") = ""
```

Figure A.22

A.2.15. BLAKE2b-256

```
input("BEGIN:VCARD") = ""
```

Figure A.23

A.2.15.1. BLAKE2b-384

```
input("BEGIN:VCARD") = ""
```

Figure A.24

A.2.16. BLAKE2b-512

```
input("BEGIN:VCARD") = ""
```

Figure A.25

A.2.17. BLAKE2s-224

```
input("BEGIN:VCARD") = ""
```

Figure A.26

A.2.18. BLAKE2s-256

```
input("BEGIN:VCARD") = ""
```

Figure A.27

A.2.19. SHAKE-128

```
input("BEGIN:VCARD") = ""
```

Figure A.28

A.2.20. SHAKE-256

```
input("BEGIN:VCARD") = ""
```

Figure A.29

A.2.21. cSHAKE-128

```
input("BEGIN:VCARD", L, N, S) = ""
```

Figure A.30

A.2.22. cSHAKE-256

```
input("BEGIN:VCARD", L, N, S) = ""
```

Figure A.31

A.2.23. ParallelHash128

```
input("BEGIN:VCARD", B, L, S) = ""
```

Figure A.32

A.2.24. ParallelHash256

```
input("BEGIN:VCARD", B, L, S) = ""
```

Figure A.33

Bibliography

[1] ISO/IEC 10118-3, International Organization for Standardization and International Electrotechnical Commission. *IT Security techniques — Hash-functions — Part 3: Dedicated hash-functions*. Fourth edition. Geneva. https://www.iso.org/standard/67116.html. [viewed: December 9, 2024].

[2] ISO/IEC 13239, International Organization for Standardization and International Electrotechnical Commission. *Information technology — Telecommunications and information exchange between systems — High-level data link control (HDLC) procedures*. Third edition. Geneva. https://www.iso.org/standard/37010.html. [viewed: December 9, 2024].

[3] ISO/IEC 18004, International Organization for Standardization and International Electrotechnical Commission. *Information technology — Automatic identification and data capture techniques — QR code bar code symbology specification*. Fourth edition. Geneva. https://www.iso.org/standard/83389.html. [viewed: December 9, 2024].

[4] ISO/IEC 27000, International Organization for Standardization and International Electrotechnical Commission. *Information technology — Security techniques — Information security management systems — Overview and vocabulary*. Fifth edition. Geneva. https://www.iso.org/standard/73906.html. [viewed: December 9, 2024].

[5] ISO/IEC 27001, International Organization for Standardization and International Electrotechnical Commission. *Information security, cybersecurity and privacy protection — Information security management systems — Requirements*. Third edition. Geneva. https://www.iso.org/standard/27001. [viewed: December 9, 2024].

[6] NIST FIPS 180-4 fpd, National Institute of Standards and Technology. *Secure Hash Standard (SHS)*. Revision 4. 2012. Gaithersburg, MD. https://csrc.nist.gov/pubs/fips/180-4/final. [viewed: December 9, 2024].

[7] NIST FIPS 202 fpd, National Institute of Standards and Technology. *SHA-3 Standard — Permutation-Based Hash and Extendable-Output Functions*. 2015. Gaithersburg, MD. https://csrc.nist.gov/pubs/fips/202/final. [viewed: December 9, 2024].

[8] NIST SP 800-185 fpd, JOHN M. KELSEY, SHU-JEN H. CHANG and RAY A. PERLNER. *SHA-3 Derived Functions — cSHAKE, KMAC, TupleHash, and ParallelHash*. 2016. Gaithersburg, MD. https://csrc.nist.gov/pubs/sp/800/185/final. [viewed: December 9, 2024].

[9] IETF RFC 2425, T. HOWES, M. SMITH and F. DAWSON. *A MIME Content-Type for Directory Information*. In: RFC. 1998. RFC Publisher. https://www.rfc-editor.org/info/rfc2425. [viewed: December 9, 2024].

[10] IETF RFC 2426, F. DAWSON and T. HOWES. *vCard MIME Directory Profile*. In: RFC. 1998. RFC Publisher. https://www.rfc-editor.org/info/rfc2426. [viewed: December 9, 2024].

[11] IETF RFC 3174, D. EASTLAKE 3RD and P. JONES. *US Secure Hash Algorithm 1 (SHA1)*. In: RFC. 2001. RFC Publisher. https://www.rfc-editor.org/info/rfc3174. [viewed: December 9, 2024].

[12] IETF RFC 3552, E. RESCORLA and B. KORVER. *Guidelines for Writing RFC Text on Security Considerations*. In: BCP. 2003. RFC Publisher. https://www.rfc-editor.org/info/rfc3552. [viewed: December 9, 2024].

[13] IETF RFC 3986, T. BERNERS-LEE, R. FIELDING and L. MASINTER. *Uniform Resource Identifier (URI): Generic Syntax*. In: STD. 2005. RFC Publisher. https://www.rfc-editor.org/info/rfc3986. [viewed: December 9, 2024].

[14] IETF RFC 4791, C. DABOO, B. DESRUISSEAUX and L. DUSSEAULT. *Calendaring Extensions to WebDAV (CalDAV)*. In: RFC. 2007. RFC Publisher. https://www.rfc-editor.org/info/rfc4791. [viewed: December 9, 2024].

[15] IETF RFC 4949, R. SHIREY. *Internet Security Glossary, Version 2*. In: FYI. 2007. RFC Publisher. https://www.rfc-editor.org/info/rfc4949. [viewed: December 9, 2024].

[16] IETF RFC 5234, P. OVERELL. *Augmented BNF for Syntax Specifications: ABNF*. In: STD. 2008. RFC Publisher. https://www.rfc-editor.org/info/rfc5234. [viewed: December 9, 2024].

[17] IETF RFC 5322, P. RESNICK (ed.). *Internet Message Format*. In: RFC. 2008. RFC Publisher. https://www.rfc-editor.org/info/rfc5322. [viewed: December 9, 2024].

[18] IETF RFC 5546, C. DABOO (ed.). *iCalendar Transport-Independent Interoperability Protocol (iTIP)*. In: RFC. 2009. RFC Publisher. https://www.rfc-editor.org/info/rfc5546. [viewed: December 9, 2024].

[19] IETF RFC 5789, L. DUSSEAULT and J. SNELL. *PATCH Method for HTTP*. In: RFC. 2010. RFC Publisher. https://www.rfc-editor.org/info/rfc5789. [viewed: December 9, 2024].

[20] IETF RFC 6234, D. EASTLAKE 3RD and T. HANSEN. *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*. In: RFC. 2011. RFC Publisher. https://www.rfc-editor.org/info/rfc6234. [viewed: December 9, 2024].

[21] IETF RFC 6986, A. DEGTYAREV. *GOST R 34.11-2012: Hash Function*. In: RFC. 2013. RFC Publisher. https://www.rfc-editor.org/info/rfc6986. [viewed: December 9, 2024].

[22] IETF RFC 7253, T. KROVETZ and P. ROGAWAY. *The OCB Authenticated-Encryption Algorithm*. In: RFC. 2014. RFC Publisher. https://www.rfc-editor.org/info/rfc7253. [viewed: December 9, 2024].

[23] IETF RFC 7693, J-P. AUMASSON. *The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)*. In: RFC. 2015. RFC Publisher. https://www.rfc-editor.org/info/rfc7693. [viewed: December 9, 2024].

[24] IETF RFC 7953, C. DABOO and M. DOUGLASS. *Calendar Availability*. In: RFC. 2016. RFC Publisher. https://www.rfc-editor.org/info/rfc7953. [viewed: December 9, 2024].

[25] IETF RFC 8259, T. BRAY (ed.). *The JavaScript Object Notation (JSON) Data Interchange Format*. In: STD. 2017. RFC Publisher. https://www.rfc-editor.org/info/rfc8259. [viewed: December 9, 2024].

