Independent Stream
Internet-Draft
Intended status: Informational
Expires: 20 April 2026

C.G. Grothoff
E.B. Benoist
B.P. Potuzhnyi
Bern University of Applied Sciences
F.D. Dold
Taler Systems AG

17 October 2025

The ’donau’ URI scheme for validation of Donau donation statements. draft-grothoff-donau-00

Abstract

This document defines the ’donau’ Uniform Resource Identifier (URI) scheme for triggering interactions with a validator for Donau donation statements.

This URI scheme allows applications to trigger interactions with a Donau validator. A Donau validator is typically run by a tax authority to validate tax records from citizens that made donations to a charity that supports the Donau protocol. The Donau validator will receive ’donau’ URIs representing the sum of donations a taxpayer made to recognized charities over a year. Donors would submit ’donau’ URLs (or QR codes with ’donau’ URLs) to tax authorities to have their donations recognized by the tax authority as tax-deductable expenditures. The application logic to verify the validity of the donation is triggered by ’donau’ URIs. The validator application would then typically confirm to the tax official the validity of the signature encoded in the URI and show the total amount donated as well as the taxpayer identification number and the year of the donation. Multiple URIs could be submitted per donor, and the application can correctly determine which submissions are cummulative and which ones are redundant.

This specification only covers the syntax of the ’donau’ URI scheme and excludes details on the protocol(s) that would allow taxpayers to donate to recognized charities to obtain these suitable signed donation statements. While a privacy-preserving protocol to obtain such statements exists within the context of the GNU Taler protocol suite, other protocols could be developed in the future and still yield compatible ’donau’ URIs as the URI scheme is reasonably generic.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

This Internet-Draft will expire on 20 April 2026.

Copyright Notice

Table of Contents

1. Introduction 3
  1.1. Objective 3
  1.2. Requirements Language 3
2. Syntax of a 'donau' URI 3
3. Semantics 4
4. Examples 6
5. Verification of a Donau URI 6
6. Access the Donau server public keys 6
7. Get donation statement and signature 9
8. Signature and total are available 10
9. Signature of the donation statement 11
10. Signature verification for a donation statement 12
11. Base 32 representation of binary data 12
12. Appendix. Test Vector: Verify from URI 13
13. References 15
  13.1. Normative References 15
  13.2. Informational References 16
Authors' Addresses 16

1. Introduction

This document defines the ’donau’ Uniform Resource Identifier (URI) [RFC3986] scheme for triggering interactions with Donau validators.

1.1. Objective

A ’donau’ URI always instructs a Donau validator to perform the validation of a Donau donation statement. A ’donau’ URI consists of the reference to the authority that signed the statement, an identifier for the specific taxpayer, the year of the donation, a salt and optional parameters. Optional parameters include the amount donated by the taxpayer and the signature of the Donau donation statement.

1.2. Requirements Language

The keywords "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. Syntax of a ’donau’ URI

This document uses the Augmented Backus-Naur Form (ABNF) of [RFC5234].

```
; Scheme and high-level structure
scheme    = "donau://" / "DONAU://" / "donau+http://" / "DONAU+HTTP://"
donau-URI = scheme base "/" year "/" taxid-enc "/" salt
            [ "?" [ "total=" amount "&sig=" algo ":" signature ] ]

; The base is the HTTP(S) origin plus optional path segments that
; identify the Donau REST API deployment. Everything before the last
; three path segments (year, taxid-enc, salt) is part of the base.
base      = 1*( ALPHA / DIGIT / "-" / "." / ":" / "/" / "_" )

; Path parameters
year      = DIGIT DIGIT DIGIT DIGIT

; The taxid-enc is the percent-encoding (RFC 3986) of the UTF-8
; taxpayer identifier string. See semantics for decoding rules.
taxid-enc = 1*( unreserved / pct-encoded )
pct-encoded = "%" HEXDIG HEXDIG
unreserved  = ALPHA / DIGIT / "-" / "." / "_" / "~"

; Salt remains an opaque string for the URI and is interpreted by
; the Donau system. (See semantics.)
salt      = 1*( DIGIT )

; Query parameters and signature encoding
amount    = currency ":" unit [ "." fraction ]
currency  = 1*ALPHA
unit      = 1*( DIGIT )
fraction  = 1*( DIGIT )
algo      = "ED25519"
signature = *( ALPHA / DIGIT )
```

3. Semantics

The base of a Donau URI refers to the base URL of the Donau REST API. It consists of the network location (host name or IP address) and MAY include additional path segments (for example, https://admin.ch/taxes/donau/) mapping to the Donau deployment behind reverse proxies. Validators MUST NOT assume a specific domain naming scheme (such as donau.example) and MUST accept bases with or without additional path components.

The year of a Donau URI referes to the year when the donations have been done.

The taxid of a Donau URI referes to the taxid of the person that did the donations and asks for their Donau donation statement to be verified. The format of the taxid is specific for each tax

authority. The taxid is conveyed in the URI as a single path segment using percent-encoding per [RFC3986]. Implementations MUST decode taxid-enc by applying percent-decoding to obtain the exact UTF-8 bytes of the tax identifier before any processing. In particular, the character ’/’ MUST be percent-encoded as %2F if present in a tax identifier.

The salt of a Donau URI referes to the information used by the Donau-wallet application to generate the Donau donation statement. This salt is specific for each wallet, allowing a taxpayer to have many Donau donation statements for the same year that should be treated as cummulative by the validator application. The salt can also be used to obfuscate the taxpayer ID in the donation protocol.

The total follows the syntax from [RFC8905] and represents the total amount donated in this year by the given taxpayer with the given salt. Thus, given multiple ’donau’ URIs with the same salt, the maximum amount should be used, while the amounts from ’donau’ URIs with different salts should be added.

Parsing note: A Donau validator MUST parse a Donau URI by taking the last three path segments as {year}/{taxid-enc}/{salt}, and treat all preceding path segments (plus the authority) as the base. This ensures that bases with additional path components are unambiguous even when tax identifiers contain reserved characters that are percent-encoded.

Finally, algo specifies the specific signature algorithm used; for now, only ED25519 (see [RFC8032]) is supported. The signature using the specified algorithm must be made over the information above (see Section 9) and MUST be encoded using the Base 32 U Crockford encoding scheme Section 11.

The default operation of applications that invoke a URI with the Donau scheme MUST be to launch a Donau validator (if available). If no Donau validator is available, an application SHOULD show a QR code with the content of the URI. If multiple Donau validators are registered, the user SHOULD be able to choose which application to launch. This allows users with multiple validators to choose which validator to perform the operation with.

An application SHOULD allow dereferencing a "donau://" URI even if the action of that URI is not registered in the "Donau URI Actions" sub-registry.

Donau-validators seeing a "donau://" URI MUST use HTTP over TLS when talking to the respective network service. Donau-validators seeing a "donau+http://" URI MUST use HTTP without TLS when talking to the

respective network service. Donau-validators SHOULD support "donau+http://"-URIs only when run in developer or debug mode. Validators would contact the base to obtain the public key used for the signature. The base origin SHOULD also be shown to the user to indicate which authority issued the proof of donation. Alternatively, specific validation apps MAY only accept 'donau' URLs from a specific set of hard-coded authorities and simply display an error message when given 'donau' URLs from other authorities.

4. Examples

```
; Minimal example with a simple base and simple taxid
donau://example.com/2025/7560001010000/1234?total=EUR:15&sig=ED25519:H9PM3BW3P8MEKB34
GZ0G1F7JSNVX7B8AHXRFFMS37QZM7TXZ5MWPXTEDZZGN1QRB1AFPKNCFXJB39NJHP3BAFGCZSCXHEYPHA1YJY28

; Example with percent-encoded taxid containing '/'
donau://donau.test.taler.net/2025/123%2F456%2F789/1234?total=TESTKUDOS:1&sig=ED25519:
B14WGS43FFPEB8JMSR6W1H8M6KH9AV33JFH376R6PM2MNH4GR24FP1C93C4ZPDG21W5WY4SASZQ4CRS427F4WJZJF
ZMQ5Y4HZNXGY30

; Example with a base URL that includes path components
donau://admin.ch/taxes/donau/2025/7560001010000/1234?total=EUR:15&sig=ED25519:H9PM3BW
3P8MEKB34GZ0G1F7JSNVX7B8AHXRFFMS37QZM7TXZ5MWPXTEDZZGN1QRB1AFPKNCFXJB39NJHP3BAFGCZSCXHEYPH
A1YJY28
```

5. Verification of a Donau URI

The verification requires the Donau verification application to test if a given taxpayer has got the right to be granted a tax reduction. For doing this, the Donau verification application must verify that the Donau-URI corresponds to a valid Donau donation statement. The Donau verification application MUST verify the validity of the donation statement. There are two possibilities: signature and total are available, or at least one is not available.

If signature and total are available, then the verification application will verify that the signature is valid for the server key (see Section 6 to get the key).

If both are missing, the application MUST download them from the get donation statement endpoint Section 7.

6. Access the Donau server public keys

The verification app MUST have a signing public key corresponding to the base for the given year. The key MAY be retrieved from a cache. If no key for this base and this year is available in the cache, the verification app MUST download the key from the base. The Donau server MUST provide the key at the /keys entrypoint. If the key is not available, no verification can take place. The verification app SHOULD cache downloaded signing public keys for the current year.

* Endpoint: /keys

* Method: GET

* Syntax: base/keys

* Syntax response: defined hereunder

* Contact: N/A

* References: [this.I-D]

Each of the "signkeys" is valid between "stamp_start" and "stamp_expire" and the public "key" returned is encoded using Base 32 U Crockford encoding Section 11.

```
   ; Core JSON tokens (simplified; assumes standard JSON for strings, numbers, booleans,
null, arrays, objects)
   JSONString      = DQUOTE *(%x20-21 / %x23-5B / %x5D-10FFFF / "\" EscapedChar) DQUOTE
   UCrockford = *( ALPHA32 / "-" )
   ALPHA32 = DIGIT / LETTER
   DIGIT   = %x30-39                          ; 0-9
   LETTER  = %x41-48 / %x4A-4E / %x50-5A / %x61-68 / %x6A-6E / %x70-7A
   JSONUCrockfordString      = DQUOTE UCrockford DQUOTE
   EscapedChar    = %x22 / %x5C / %x2F / %x62 / %x66 / %x6E / %x72 / %x74 /
                    "u" 4HEXDIG
   JSONNumber     = [ "-" ] 1DIGIT *( DIGIT ) [ "." 1DIGIT *( DIGIT ) ]
                    [ ( "e" / "E" ) [ "+" / "-" ] 1DIGIT *( DIGIT ) ]
   JSONBool       = "true" / "false"
   JSONNull       = "null"
   JSONArray      = "[" [ JSONException *( "," JSONException ) ] "]"
   JSONObject     = "{" [ JSONMember *( "," JSONMember ) ] "}"
   JSONMember     = JSONString ":" JSONException
   JSONException  = JSONString / JSONNumber / JSONObject / JSONArray / JSONBool / JSONNul
l

   ; Specific DonauKeysResponse structure
   DonauKeysResponse = "{"
                         JSONMemberSep
                       "}"
   JSONMemberSep  = version , domain , base_url , currency , currency_fraction_digits ,
                    donation_units , signkeys

   version       = DQUOTE "version" DQUOTE ":" JSONString
   domain        = DQUOTE "domain"  DQUOTE ":" JSONString
   base_url      = DQUOTE "base_url" DQUOTE ":" JSONString
   currency      = DQUOTE "currency" DQUOTE ":" JSONString
   currency_fraction_digits = DQUOTE "currency_fraction_digits" DQUOTE ":" JSONNumber
   EDDSAPubKey   = JSONUCrockfordString
   SignKey       = DQUOTE "key" DQUOTE ":" EDDSAPubKey
   donation_units = DQUOTE "donation_units" DQUOTE ":" "[" DonationUnitKeyGroup *( "," Do
nationUnitKeyGroup ) "]"
   signkeys       = DQUOTE "signkeys" DQUOTE ":" "[" SignKey *( "," SignKey ) "]"

   DonationUnitKeyGroup = "{" cipher_field "," value_field "," donation_units_array "}"
   cipher_field           = DQUOTE "cipher" DQUOTE ":" ( DQUOTE "RSA" DQUOTE / DQUOTE "CS
" DQUOTE )
   value_field            = DQUOTE "value"  DQUOTE ":" JSONNumber
   donation_units_array   = DQUOTE "donation_units" DQUOTE ":" "[" DUNK_UNIT_KEY *( "," D
UNK_UNIT_KEY ) "]"

   ; donation unit entries differ depending on cipher
   DUNK_UNIT_KEY = "{" base_fields "," key_specific "}"
   base_fields   = DQUOTE "year"  DQUOTE ":" JSONNumber [ "," DQUOTE "lost" DQUOTE ":" JS
ONBool ]
   key_specific  = ( DQUOTE "rsa_pub" DQUOTE ":" JSONObject ) /
                   ( DQUOTE "cs_pub"  DQUOTE ":" JSONString )

   SignKey = "{" DQUOTE "key" DQUOTE ":" JSONObject "," DQUOTE "year" DQUOTE ":" JSONNumb
er "}"
```

Example, a response to a GET request to the /key/ endpoint.

```
 {"signkeys":
   [ "stamp_start":{"ts_s":42},
     "stamp_expire":{"ts_s":43},
     "key":"XXXXXXXXXXXXXXXX"
   ]
 }
```

7. Get donation statement and signature

If the Donau-URI does not contain the total or the signature, the verification app MUST download them from the /donation-statement/ endpoint of the base.
The verification app will compute the donor hash H using SHA-512 [RFC6234] over the exact UTF-8 bytes of the taxpayer ID string, followed by a single NUL byte (0x00), followed by the salt string, followed by a final NUL byte (0x00): H = SHA-512(taxid || 0x00 || salt || 0x00). Here, taxid is the UTF-8 string obtained by percent-decoding the taxid-enc path segment. This produces the hash-donor-id. The verification app will contact the base in the endpoint /donation-statement with the year and the hash-donor-id. The hash-donor-id must be encoded using Base 32 U Crockford encoding Section 11.

* Endpoint: /donation-statement

* Method: GET

* Syntax: base/donation-statement/{year}/{hash-donor-id}

* Contact: N/A

* References: [this.I-D]

Servers implementing the donation-statement endpoint MUST respect the following syntax; all three fields (total-field, sig-field, pub-field) MUST be included. The amount is a string formed first of the currency (in capital letters) then ":" and then the value (can be an integer or a decimal number).

```
   json-object     = "{" ws field-list ws "}"

   field-list      = field *(ws "," ws field)
   field           = total-field / sig-field / pub-field

   ; Allow any order of the three fields
   total-field     = DQUOTE "total" DQUOTE ws ":" ws DQUOTE currency ":" value DQUOTE
   sig-field       = DQUOTE "donation_statement_sig" DQUOTE ws ":" ws DQUOTE signature DQ
UOTE
   pub-field       = DQUOTE "donau_pub" DQUOTE ws ":" ws DQUOTE signature DQUOTE  ; same
format as signature

   currency        = 1*(%x41-5A)   ; A-Z uppercase letters
   value           = int / decimal
   int             = ["-"] 1*DIGIT
   decimal         = ["-"] 1*DIGIT "." 1*DIGIT

   signature       = 1*(ALPHA / DIGIT / "=")

   ; Common tokens
   ws              = *WSP
   DQUOTE          = %x22  ; "
   WSP             = %x20 / %x09
   DIGIT           = %x30-39
   ALPHA           = %x41-5A / %x61-7A
```

Example of an element of USD 100.00 :

```
{
   total: "USD:100",
   donation_statement_sig: "SIGNATURE",
   donau_pub: "EDDSA_PUBLIC_KEY"
}
```

8. Signature and total are available

The verification app MUST verify that the signature corresponds to the claimed values. If the information is not in the URI, the verification app MUST download the total and the signature from the base using the GET /donation-statement/ endpoint Section 7. Otherwise, it SHOULD use the value given in the URI. The verification of the signature is done using EdDSA as specified in [RFC8032]. The verification MAY be done with the following instructions. The signed data is the concatenation of the following fields in network byte order (big-endian):

* size (4 bytes): 32-bit unsigned integer with the total length of the signed data in bytes.

* purpose (4 bytes): 32-bit unsigned integer with the constant value 1500.

* total value (8 bytes): integer part as an unsigned 64-bit integer.

* total fraction (4 bytes): fractional part as an unsigned 32-bit integer in units of 1/100000000.

* currency (12 bytes): ASCII, left-aligned and zero-padded to 12 bytes.

* donor hash H (64 bytes): H = SHA-512(UTF-8(taxid) || 0x00 || UTF-8(salt) || 0x00).

* year (4 bytes): 32-bit unsigned integer (the donation year).

9. Signature of the donation statement

The server signing the donation statement MUST use the Sign(d,message) procedure implemented as defined in [RFC8032].

0

Figure 1: The binary representation of the data to sign

The signature over the public key covers a 32-bit pseudo header conceptually prefixed to the EXPIRATION and BDATA fields. The wire format is illustrated in Figure 2.

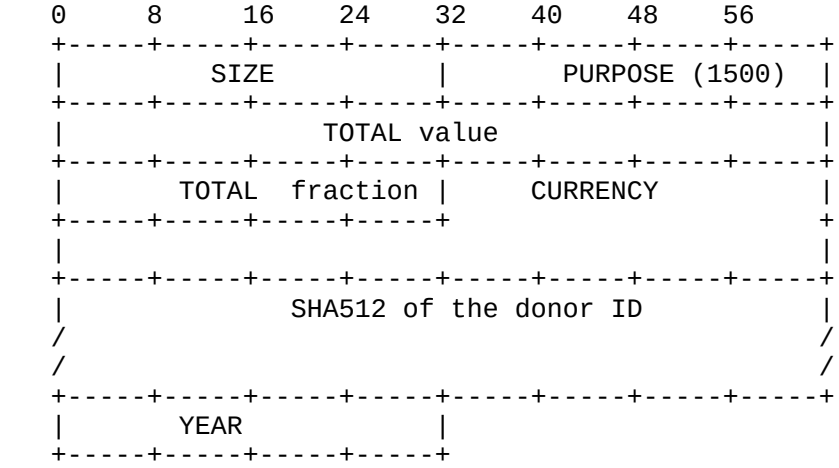

Figure 2: The Wire Format Used for Creating the Signature of the RRBLOCK

SIZE: A 32-bit value containing the length of the signed data in bytes in network byte order.

PURPOSE: A 32-bit signature purpose flag in network byte order. The value of this field MUST be 1500. It defines the context in which the signature is created so that it cannot be reused in other parts of the protocol that might include possible future extensions. The value of this field corresponds to an entry in the GANA "GNUnet Signature Purposes" registry [GANA].

TOTAL value: The integer part of the total value. Unsigned integer containing the integer part of the value for the total. It is represented on 64 bits value (in big endian),

TOTAL fraction: The fractional part of the total value is represented by an unsigned integer on 32 bits (also in big endian notation). It represents the fractional part of the value in 1/100000000th of the base unit.

TOTAL currency A string representing the currency. ASCII, left-aligned and zero-padded to exactly 12 bytes.

HASH: The SHA512 hash code for the donor ID.

YEAR: The year on a 32 bit unsigned integer in big endian.

10. Signature verification for a donation statement

The signature verification step is taking the data presented in the figure Figure 1 as input. Signature MUST be verified using the EdDSA scheme described in [RFC8032] the procedure is Verify(zk,message,signature). Public key and signature are given encoded in Base 32 U Crockford MUST first be decoded to get a binary out of it.

The Sign(d,message) and Verify(zk,message,signature) procedures MUST be implemented as defined in [RFC8032].

11. Base 32 representation of binary data

All binary data MUST be encoded to be transmitted. For encoding, one MUST use the Base32 U Crockford encoding. This is a variation of the base32 encoding [RFC4648]. This encoding is presented in details in the appendix of the RFC for GNU Name System [RFC9498]

The encoding works similarly to the standard, but uses another Base 32 Alphabet. The new alphabet is given in the Table Figure 3, below.

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	0	9	9	18	J	27	V
1	1	10	A	19	K	28	W
2	2	11	B	20	M	29	X
3	3	12	C	21	N	30	Y
4	4	13	D	22	P	31	Z
5	5	14	E	23	Q		
6	6	15	F	24	R	(pad)	=
7	7	16	G	25	S		
8	8	17	H	26	T		

Figure 3: The Base 32 Encoding Alphabet.

To prevent optical character reading (OCR) problems, a system decoding binary data encoded with base 32 U Crockford MUST accept the codes presented in the Table Figure 4, below. System reading a U MUST evaluate it as a V.

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	0	9	9	18	J|j	27	V|v|u|U
1	1|i|I|l|L	10	A|a	19	K|k	28	W|w
2	2	11	B|b	20	M|m	29	X|x
3	3	12	C|c	21	N|n	30	Y|y
4	4	13	D|d	22	P|p	31	Z|z
5	5	14	E|e	23	Q|q		
6	6	15	F|f	24	R|r	(pad)	=
7	7	16	G|g	25	S|s		
8	8	17	H|h	26	T|t		

Figure 4: The Base 32 Decoding Alphabet.

12. Appendix. Test Vector: Verify from URI

This appendix shows how to verify a donation statement starting from a received URI. Public key is fetched from the Donau base.

```
Example URI:
donau://donau.test.taler.net/2025/123%2F456%2F789/AWNFDRFT0WX45W4Y32A9DJA03S1EF66GFQZ9EV5
EF9JTHWZ37WR0?total=TESTKUDOS:1&sig=ED25519:B14WGS43FFPEB8JMSR6W1H8M6KH9AV33JFH376R6PM2MN
H4GR24FP1C93C4ZPDG21W5WY4SASZQ4CRS427F4WJZJFZMQ5Y4HZNXGY30

Extracted fields:
- base: donau.test.taler.net
- year: 2025
- taxid: 123%2F456%2F789
- taxid (decoded): 123/456/789
- salt: AWNFDRFT0WX45W4Y32A9DJA03S1EF66GFQZ9EV5EF9JTHWZ37WR0
- total: TESTKUDOS:1
- signature: ED25519:B14WGS43FFPEB8JMSR6W1H8M6KH9AV33JFH376R6PM2MNH4GR24FP1C93C4ZPDG21W5W
Y4SASZQ4CRS427F4WJZJFZMQ5Y4HZNXGY30

Fetched public key for year 2025 from https://donau.test.taler.net/keys:
- pub: 2FRN2CAK9DMDWE157W6HY97RAVSP0ZCCC08X9N6JD2MK7413XXZG
```

Verification steps:

1. Input (URI):
   donau://donau.test.taler.net/2025/123%2F456%2F789/AWNFDRFT0WX45W
   4Y32A9DJA03S1EF66GFQZ9EV5EF9JTHWZ37WR0?total=TESTKUDOS:1&sig=ED2
   5519:B14WGS43FFPEB8JMSR6W1H8M6KH9AV33JFH376R6PM2MNH4GR24FP1C93C4
   ZPDG21W5WY4SASZQ4CRS427F4WJZJFZMQ5Y4HZNXGY30

2. Parse fields:
   - base: donau.test.taler.net
   - year: 2025 (4 digits)
   - taxid-enc: 123%2F456%2F789
   - taxid (UTF-8, percent-decoded, no trimming): 123/456/789
   - salt (ASCII):
   AWNFDRFT0WX45W4Y32A9DJA03S1EF66GFQZ9EV5EF9JTHWZ37WR0
   - total (amount string): TESTKUDOS:1
   - sig (Crockford Base32, algorithm ED25519): B14WGS...XGY30

3. Public signing key (Crockford Base32) (from /keys):
   2FRN2CAK9DMDWE157W6HY97RAVSP0ZCCC08X9N6JD2MK7413XXZG

4. Donor hash H = SHA-512(UTF-8(taxid) || 0x00 || UTF-8(salt) ||
   0x00):
   - length: 64 bytes
   - hex:
   4aaa1e16fc5be44842b863b1f17da39296ca7b3529a720e11aba9c8bd729f7a1
   e2bb0b9a39c02d271da5dd15aea66ce95be78bcaf380de19a0bdbcd8a7938f1b
   - Crockford Base32 (for HTTP endpoints):
   9AN1W5QWBFJ4GGNRCERZ2ZD3JABCMYSN56KJ1R8TQAE8QNS9YYGY5ERBK8WW0B97
   3PJXT5DEMSPEJPZ7HF5F706Y36GBVF6RMY9RY6R

5. Amount encoding (TALER_AmountNBO):
   - currency (12 bytes): "TESTKUDOS" then 0x00 x 3
   - value (uint64 BE): 1 -> 0x0000000000000001
   - fraction (uint32 BE): 0 -> 0x00000000

6. Signed message M layout (network byte order, total 100 bytes):
   - [0000..0003] 4 bytes size: 0x00000064
   - [0004..0007] 4 bytes purpose: 0x000005DC (1500)
   - [0008..0015] 8 bytes amount.value: 0x0000000000000001
   - [0016..0019] 4 bytes amount.fraction: 0x00000000
   - [0020..0031] 12 bytes amount.currency: 54 45 53 54 4b 55 44 4f 53 00 00 00 ("TESTKUDOS\x00\x00\x00")
   - [0032..0095] 64 bytes i.hash: donor hash H
   - [0096..0099] 4 bytes year: 0x000007E9 (2025)

7. Message M (hex, 100 bytes):
   00000064 000005dc 0000000000000001 00000000
   544553544b55444f53000000 4aaa1e16fc5be44842b863b1f17da39296ca7b3
   529a720e11aba9c8bd729f7a1e2bb0b9a39c02d271da5dd15aea66ce95be78bc
   af380de19a0bdbcd8a7938f1b 000007e9

8. Signature (ED25519, Crockford Base32 -> bytes):
   - length: 64 bytes
   - R (first 32 bytes, hex):
   5849c864837bece5a254ce0dc0c51434e2956c6393e2339b06b5054ac490c088
   - S (last 32 bytes, hex):
   fb05891b09fb36020f0bcf132acfee46632411de4e4bf27fe972f891fd7b0f0c

9. Public key (Ed25519, Crockford Base32 -> bytes):
   - length: 32 bytes
   - hex:
   13f15131534b68de38253f0d1f24f856f3607d8c6011d4d4d268a9339023ef7f

10. Verification (expected): call crypto_sign_verify_detached(sig, M, 100, pubkey). The result MUST indicate a valid signature for this vector.

11. Negative check (optional): flip one bit of M or sig and repeat; verification MUST fail.

13. References

13.1. 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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[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>.

[RFC8905] Dold, F. and C. Grothoff, "The 'payto' URI Scheme for Payments", RFC 8905, DOI 10.17487/RFC8905, October 2020, <https://www.rfc-editor.org/info/rfc8905>.

[RFC9498] Schanzenbach, M., Grothoff, C., and B. Fix, "The GNU Name System", RFC 9498, DOI 10.17487/RFC9498, November 2023, <https://www.rfc-editor.org/info/rfc9498>.

13.2. Informational References

[GANA] GNUnet e.V., "GNUnet Assigned Numbers Authority (GANA)", April 2020, <https://gana.gnunet.org/>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

Authors' Addresses

Christian Grothoff
Bern University of Applied Sciences
H旦heweg 80
CH-2501 Biel/Bienne
Switzerland

Email: christian.grothoff@bfh.ch

Emmanuel Benoist
Bern University of Applied Sciences
H旦heweg 80
CH-2501 Biel/Bienne
Switzerland
Email: emmanuel.benoist@bfh.ch

Bohdan Potuzhnyi
Bern University of Applied Sciences
H旦heweg 80
CH-2501 Biel/Bienne
Switzerland
Email: bohdan.potuzhnyi@bfh.ch

Florian Dold
Taler Systems AG
7, rue de Mondorf
L-5421 Erpeldange
Luxembourg
Email: dold@taler.net