

TCPP Node

Wie man einen TCPP-Node betreibt und bedient: schemagesteuerte Anfragen zusammenstellen, signierte Frames mit anderen Nodes austauschen, auf Eingegangenes antworten und Identitäten, Peers und Records verwalten.

Schema 1.3.0 27 Paare aus Anfrage und Antwort, ein gemeinsamer Umschlag.	ES384 Jeder Frame ist ein kompakter JWS über ECDSA P-384.	Eine Binärdatei Anwendung, Schemas und Beispiele einkompiliert. Nichts zu installieren.
--	---	---

ZUR SPRACHE

Die Oberfläche der Anwendung ist englisch. Bedienelemente stehen deshalb auch hier unter ihrem englischen Namen, genau so, wie sie auf dem Bildschirm und in den Abbildungen heißen.

INHALT

01	Was ein Node ist	09	Peers und Verifikation
02	Erste Schritte	10	Identität
03	Das Fenster	11	Log
04	Compose	12	Arbeitsabläufe
05	Die Record-Bibliothek	13	Kommandozeile und Dateien
06	Validierung	14	HTTP-API
07	Posteingang und Antworten	15	Die TCPP-Nachricht
08	Sent	16	Anhang

■ Was ein Node ist

Ein TCPP-Node ist eine **einzelne ausführbare Datei**. Sie starten sie, und sie tut zweierlei zugleich: Sie liefert eine schemagesteuerte Web-Anwendung an Ihren Browser aus, und sie nimmt TCPP-Frames anderer Nodes über TLS entgegen. Alles, was sie dafür braucht, ist in die Binärdatei einkompiliert: die JSON-Schemas der Version 1.3.0, die mitgelieferten Beispiel-Nutzlasten und die Browser-Anwendung selbst. Es muss nichts weiter installiert werden, und sie greift auf nichts im Internet zu.

Die Arbeitseinheit ist ein **Frame**: eine TCPP-Nachricht, vom sendenden Node als kompaktes JWS (ein JWT) signiert und an einen Peer geschickt. Der Node hält jeden gesendeten und jeden empfangenen Frame in einem Log neben der Binärdatei fest, an das nur angehängt wird, und zwar mit Nutzlast, rohem Token, Absender und dem Ergebnis der Signaturprüfung.

Was Sie damit tun können

- Jeden der 27 TCPP-Anfragetypen in einem Formular ausfüllen, das aus dem Schema erzeugt wird.
- Ihn signieren, an einen anderen Node senden und dort live ankommen sehen.
- Eine eingegangene Anfrage über ein Antwortformular beantworten, das aus ihr vorbefüllt ist.
- Zurechtgelegte Nutzlasten als benannte *Records* aufbewahren und später erneut senden.
- Die öffentlichen Schlüssel der Peers hinterlegen und jeden gespeicherten Frame dagegen prüfen lassen.
- Die vollständige Historie nachlesen: Nutzlast, Token, JOSE-Header, Transportergebnis.

Eine einzige Binärdatei deckt zudem jede Konstellation ab, die bei der Arbeit mit TCPP anfällt. Der Node kann:

- Frames mit sich selbst austauschen, sodass eine einzelne Instanz für einen vollständigen Umlauf genügt. Ohne `--peer` ist die eigene Adresse das voreingestellte Ziel.
- Frames mit weiteren Nodes auf demselben Rechner austauschen, von denen jeder auf einem eigenen Port läuft.

- Frames mit Nodes auf anderen Hosts über das Netz austauschen. Kommt nichts an, ist die Firewall das Erste, was zu prüfen ist.
- Die mitgelieferten Beispiel-Nutzlasten laden sowie eigene Testdaten sichern und wieder laden.
- Ein eigenes Schlüsselpaar erzeugen und die öffentlichen Schlüssel der Peers direkt bei diesen abholen, was in einer etablierten PKI seinen vollen Nutzen entfaltet.
- Als Gegenstelle dienen, während Sie eine eigene TCPP-Implementierung entwickeln und testen.

■ Erste Schritte

Was Sie bekommen haben

Eine ausführbare Datei, sonst nichts. Es gibt kein Installationsprogramm, keine Laufzeitumgebung, die vorher eingerichtet werden müsste, und keine Konfigurationsdatei zu schreiben: Legen Sie die Datei in ein eigenes Verzeichnis und starten Sie sie. Der Download ist für ein Betriebssystem und einen Prozessor gebaut; nehmen Sie also die Datei, die zu Ihrem Rechner passt:

DATEI	LÄUFT AUF
<code>tcpp-node-windows-amd64.exe</code>	Windows, Intel- oder AMD-Prozessor (der Normalfall)
<code>tcpp-node-windows-arm64.exe</code>	Windows auf ARM
<code>tcpp-node-linux-amd64</code>	Linux, Intel oder AMD
<code>tcpp-node-linux-arm64</code>	Linux auf ARM, ein Raspberry Pi oder ein ARM-Server
<code>tcpp-node-darwin-amd64</code>	macOS, Intel
<code>tcpp-node-darwin-arm64</code>	macOS, Apple Silicon (M1 und neuer)

Es ist dasselbe Programm; nur der Maschinencode unterscheidet sich. Jede Datei steht für sich, ohne Bibliotheken danebenzulegen und ohne etwas auszupacken, und keine von ihnen greift ins Internet. Benennen Sie die heruntergeladene Datei in `tcpp-node` um (unter Windows `tcpp-node.exe`); so wird sie in diesem Handbuch durchgehend geschrieben.

Geben Sie ihr ein Verzeichnis, das Sie wiederfinden. Alles, was der Node anlegt (seinen Schlüssel, sein Zertifikat, sein Frame-Log, seine Records), schreibt er neben die ausführbare Datei; das Verzeichnis *ist* also der Node.

Starten

Ohne Argumente gestartet, lauscht er auf Port 8443.

WINDOWS

In der PowerShell oder in `cmd`, aus dem Verzeichnis des Nodes heraus. Ein Doppelklick funktioniert auch, nur schließt sich das Fenster dann zusammen mit

dem Node, und Sie sehen nie, woran er gescheitert ist.

```
.\tcpp-node.exe
```

MACOS UND LINUX

Eine heruntergeladene Datei ist noch nicht ausführbar, also einmal markieren:

```
chmod +x tcpp-node  
./tcpp-node
```

DER DOWNLOAD IST NICHT SIGNIERT

Er trägt kein Code-Signing-Zertifikat, deshalb protestiert jedes System einmal, und nur beim ersten Start:

- **Windows.** SmartScreen meldet einen unbekannten Herausgeber: *Weitere Informationen* → *Trotzdem ausführen*. Weigert es sich ganz, ist die Datei als aus dem Internet stammend markiert: *Rechtsklick* → *Eigenschaften* → *Zulassen*, dann *OK*.
- **macOS.** Gatekeeper blockiert sie: Ordner öffnen, *Rechtsklick auf die Datei* → *Öffnen* und bestätigen, oder die Markierung mit `xattr -d com.apple.quarantine tcpp-node` entfernen.
- **Linux.** Nichts protestiert; `chmod +x` genügt.

Von da an ist es überall dasselbe Programm. Der erste Start erzeugt eine Identität und gibt aus, was er gleich ausliefern wird:

```
2026/09/03 18:03:48 created C:\Users\User\tcpp-lab\a\tcpp-key.pem
2026/09/03 18:03:48 created C:\Users\User\tcpp-lab\a\tcpp-cert.pem
2026/09/03 18:03:48 TCPP node CSU / P:NODE-9BED6853
2026/09/03 18:03:48 listening https://localhost:8443/
2026/09/03 18:03:48 advertising localhost:8443
2026/09/03 18:03:48 signing ES384, kid 9bed6853
2026/09/03 18:03:48 identity C:\Users\User\tcpp-lab\a
2026/09/03 18:03:48 frames 0 replayed from tcpp-frames.jsonl
2026/09/03 18:03:48 schemas 27 endpoints
2026/09/03 18:03:48 records 63 in C:\Users\User\tcpp-lab\a\records
2026/09/03 18:03:48 target localhost:8444
2026/09/03 18:03:48 The certificate is self-signed, so the browser will
```

Öffnen Sie **https://localhost:8443/**. Das Zertifikat, das der Node sich selbst ausgestellt hat, ist selbstsigniert, deshalb unterbricht der Browser einmal mit einer Sicherheitswarnung; gehen Sie daran vorbei (in Chrome: *Erweitert* → *Weiter zu...*). Für diesen Host kommt die Warnung nicht wieder, bis der Node ein neues Zertifikat bekommt.

DER NODE HAT KEINE ANMELDUNG

Nichts an der HTTP-Schnittstelle ist authentifiziert. Wer den Port erreicht, kann das Frame-Log lesen, Records speichern, Peers ändern und Frames als dieser Node senden. Betreiben Sie ihn in einem Netz, dem Sie trauen (einem Laptop, einem Laborsegment, einem privaten VPN), nicht unter einer öffentlichen Adresse.

Was auf der Platte entsteht

Der gesamte Zustand liegt neben der Binärdatei (oder in `--data-dir`). Ein Node ist ein Verzeichnis, das man kopieren kann.

DATEI	INHALT	RECHTE
tcpp-key.pem	privater ECDSA-P-384-Schlüssel, PKCS#8; das Geheimnis	0600
tcpp-cert.pem	selbstsigniertes Zertifikat zum selben Schlüssel, 10 Jahre gültig	0644
tcpp-node.json	TCPP-Typ und -Id dieses Nodes	0644

DATEI	INHALT	RECHTE
tcpp-frames.jsonl	jeder gesendete und empfangene Frame, ein JSON-Objekt pro Zeile	0644
tcpp-peers.json	bekannte Peers und ihre öffentlichen Schlüssel	0644
records/	die Record-Bibliothek, <schema>/<name>.json	0755

Der Kurztest mit einem Node

Für den Rundlauf brauchen Sie keinen zweiten Node. Ohne `--peer` ist das Standardziel `localhost:<Port>`, dieser Node selbst; ein Frame, den er signiert, landet also in seinem eigenen Posteingang. Öffnen Sie [Compose](#), wählen Sie `ping`, drücken Sie [Load...](#) → `example` und dann [Sign & Send](#). Er erscheint sofort unter [Inbox](#).

Das Zwei-Node-Labor dieses Handbuchs

Jede Abbildung in diesem Handbuch stammt aus zwei Nodes auf einem Rechner: **A**, eine Charge Switch Unit auf Port 8443, und **B**, ein Trusted Management Host auf Port 8444. Jeder hat sein eigenes Verzeichnis und damit seinen eigenen Schlüssel, sein eigenes Log und seine eigenen Records.

```
# macOS und Linux: zwei Verzeichnisse, in jedem eine Kopie der Binärdatei
mkdir a b
cp tcpp-node a/ && cp tcpp-node b/

# Terminal 1
a/tcpp-node --port 8443 --node-type CSU --peer localhost:8444
# Terminal 2
b/tcpp-node --port 8444 --node-type TMH --peer localhost:8443
```

```
# PowerShell
mkdir a, b
copy tcpp-node.exe a\; copy tcpp-node.exe b\

# Fenster 1
a\tcpp-node.exe --port 8443 --node-type CSU --peer localhost:8444
# Fenster 2
b\tcpp-node.exe --port 8444 --node-type TMH --peer localhost:8443
```

Öffnen Sie <https://localhost:8443/> und <https://localhost:8444/> in zwei Browser-Tabs, dann sehen Sie beide Seiten eines Austauschs gleichzeitig.

■ Das Fenster

The screenshot displays the TCP Node interface with the following components:

- Header:** TCP Node schema 1.3.0, CSU / P:NODE-FDB7CC0A, :8443 | ES384 · fdb7cc0a, LIVE
- Compose Panel (Left):** A sidebar with a 'Compose' button and a list of actions including archive, charge, display, email, error, key, measurement, module, parameters, status, tariffs, trigger, and verification. The 'ping' action is currently selected.
- REQUEST SCHEMAS (Middle):** A section for defining request schemas. It includes a 'filter...' input and a 'Load...' button. The 'ping_request' schema is shown with fields for url, head, record_type, record_direction, record_uds, record_pagination, record_version, origin_type, origin_id, target_type, target_id, context_type, context_id, and error_code.
- OUTGOING REQUEST (Right):** A section for viewing outgoing requests. It includes a 'valid' button and a 'Copy JSON' button. The JSON output is displayed, showing the request details and the 'ping' action.

ABBILDUNG 01 · Die ganze Anwendung. Die Kopfzeile benennt den Node, die Leiste links wechselt die Ansicht, und die drei Spalten gehören zu **Compose**.

Kopfzeile

Der Chip ganz oben benennt immer den Node, den Sie gerade vor sich haben, was wichtig ist, sobald zwei Tabs offen sind:

- **CSU / P:NODE-9BED6853**: TCP-Modultyp und -Id des Nodes.
- **:8443**: der Port, von dem diese Seite ausgeliefert wurde.
- **ES384 · 9bed6853**: der Signaturalgorithmus und die kurze Schlüssel-Id, die in jedem signierten Token auftaucht.

Rechts bedeutet **LIVE**, dass der Browser am Ereignisstrom des Nodes hängt und neue Frames von selbst erscheinen. Bricht die Verbindung ab, wird daraus **OFFLINE**.

Navigationsleiste

ANSICHT	WOZU SIE DIENT
Compose	Eine Anfrage aus einem Schema zusammenstellen und senden.
Inbox	Hier eingegangene Frames. Das rote Abzeichen zählt die ungelesenen.
Sent	Von diesem Node gesendete Frames, mit ihrem Transportergebnis.
Peers	Bekannte Nodes, ihre Adressen und öffentlichen Schlüssel.
Identity	Der eigene Schlüssel, der Fingerabdruck und die Dateipfade dieses Nodes.
Log	Alles, gesendet wie empfangen, in dichten Zeilen.

Am Fuß der Leiste steht, wie viele Frames der Node zurzeit hält. Kurze Bestätigungen (*sent to localhost:8444, peer saved, loaded ping_request/example*) erscheinen für einen Moment als blaue Einblendung unten rechts.

Hell und dunkel

Die Anwendung folgt dem Farbschema Ihres Betriebssystems. Sonst ändert sich nichts.

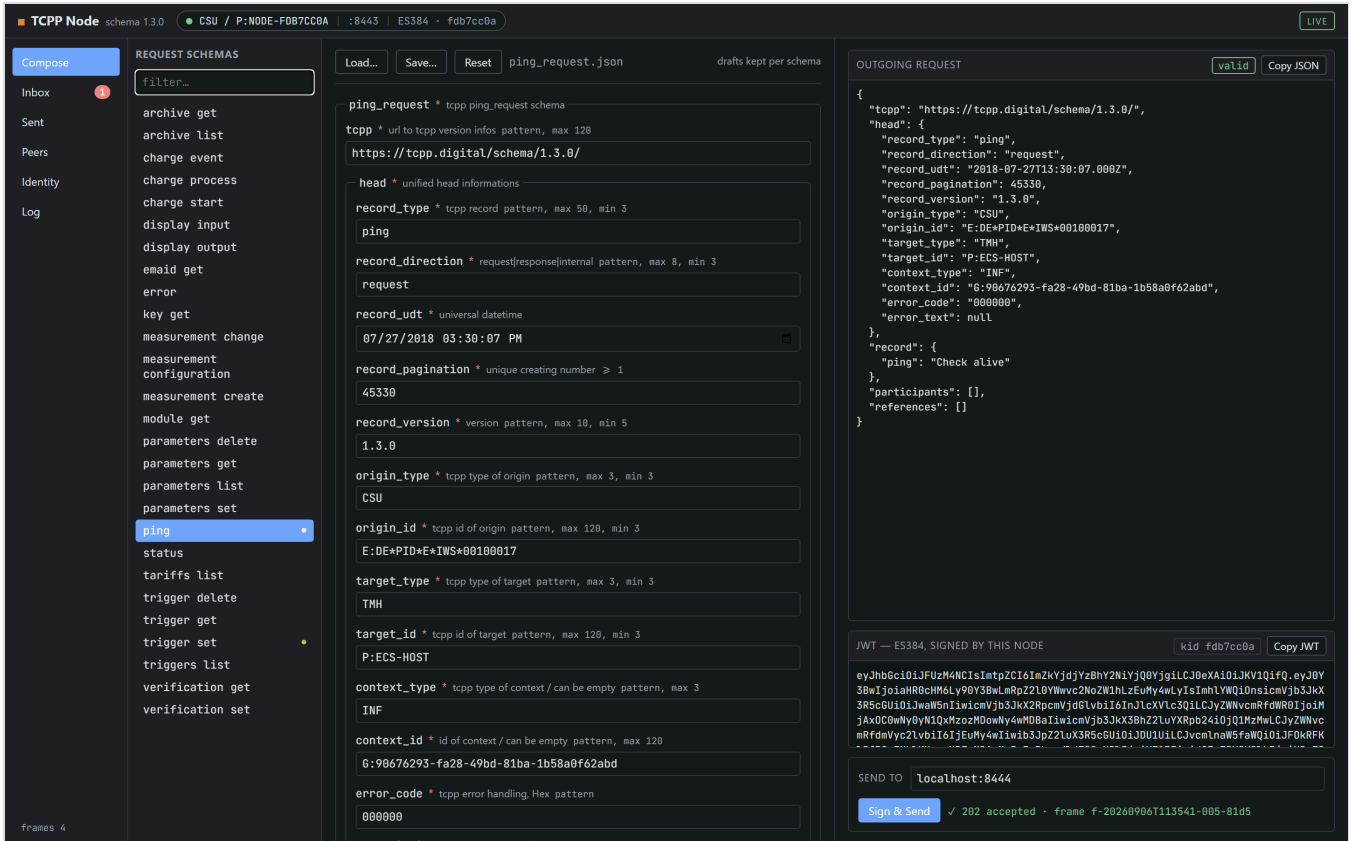


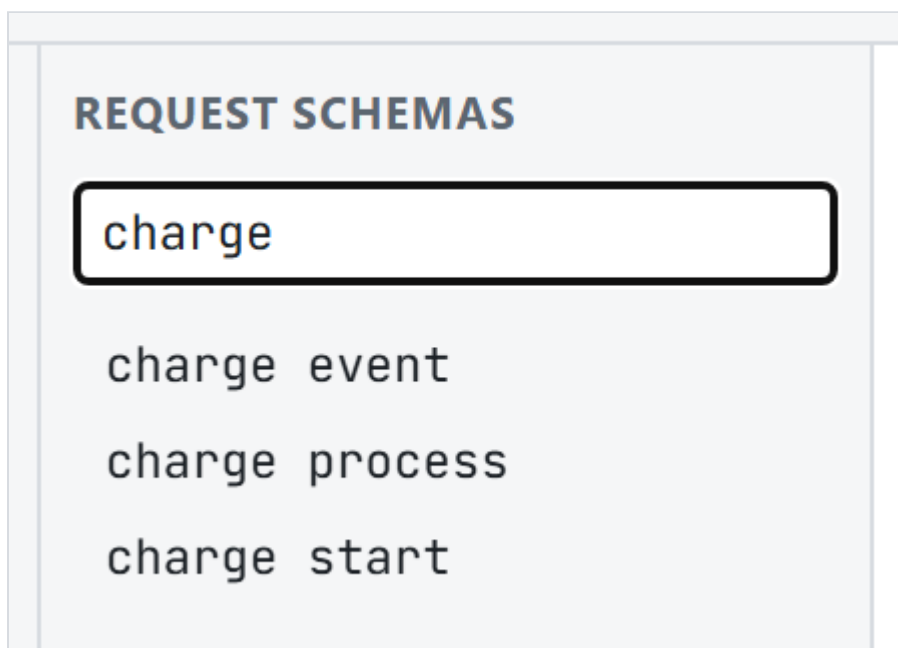
ABBILDUNG 02 · Derselbe Bildschirm, mit dem System auf dunkel gestellt.

■ Compose

Compose besteht aus drei Spalten: links die Anfrage-Schemas, in der Mitte ein aus dem gewählten Schema erzeugtes Formular und rechts das, was tatsächlich hinausgeht.

Ein Schema wählen

Die linke Spalte listet alle 27 Anfragetypen. Tippen Sie in **filter...**, um die Liste einzugrenzen; gefiltert wird nach dem Schemanamen, **charge** lässt also die drei Charge-Endpunkte übrig.



REQUEST SCHEMAS

charge

charge event

charge process

charge start

ABBILDUNG 03 • Die gefilterte Schema-Liste. Ein kleiner bernsteinfarbener Punkt markiert ein Schema, zu dem ein ungesendeter Entwurf existiert; das gewählte Schema ist hervorgehoben.

Das erzeugte Formular

Die mittlere Spalte wird aus dem JSON-Schema gebaut, nicht von Hand geschrieben, und passt deshalb immer zu dem, was der Node annimmt. Jedes Feld zeigt seinen Namen in Festbreitenschrift, ein rotes *, wenn es Pflicht ist, die Beschreibung aus dem Schema und eine knappe Zusammenfassung seiner Einschränkungen (`pattern`, `max 120`, `min 3`, `≥ 1`).

IM SCHEMA STEHT	SIE BEKOMMEN
<code>enum</code>	ein Auswahlfeld, mit einem Eintrag <code>(none)</code> , wenn das Feld entfallen oder null sein darf
<code>format: date</code>	eine Datumsauswahl
<code>format: date-time</code>	eine Datums- und Zeitauswahl, ausgelesen als ISO-Zeitstempel
String, großes <code>maxLength</code>	ein mehrzeiliges Textfeld
String	ein Textfeld; fahren Sie darüber, um das geforderte Muster zu sehen
Zahl, Ganzzahl	ein Zahlenfeld, das <code>minimum</code> und <code>multipleOf</code> mitführt
Boolean	ein Ankreuzfeld
Objekt	eine umrandete Gruppe; eine nullbare bekommt einen Schalter <i>set to null</i> , der ihren Inhalt ausblendet
Array	nummerierte Element-Karten mit <code>Remove</code> und darunter eine Schaltfläche <code>+ Add ...</code>
Objekt ohne Eigenschaften	ein rohes JSON-Textfeld, beim Tippen geprüft

record

trigger_check_sign * CLOCK.TIME[CHECK.HARDWARE]...

pattern, max 50, min 5

CLOCK.TIME

trigger_steps f.e. 'CLOCK.TIME': 5 = Every 5 Minutes

15

subject_id optional id of legal_subject pattern, max 120

module_id optional id of module pattern, max 120

☒ pause_on_breakNot try while break/deactivate/shutdown/etc...

☐ value_was_changedOnly send if value was changed

value_min if value number/string/boolean arrived a minimum

value_max if value number/string/boolean arrived a maximum

value_exact if value number/string/boolean arrived a exact value

+ Add trigger_formula

trigger_actions * List of actions start

#1

Remove

record

action_sign * ACTION.REQUEST[... pattern, max 50, min 5

ACTION.REQUEST

action_id * id of action pattern, max 120

G:eb735807-d849-4d98-b8d1-e36f7ae62164

target_record_type tcp record pattern, max 50

status

target_type request to target type pattern, max 3

CPD

ABBILDUNG 04 · `trigger_set`, heruntergescrollt bis `record`: Ankreuzfelder, ein nummeriertes Array-Element mit `Remove` und `+ Add trigger_formula`, um ein weiteres anzuhängen.

Was hinausgeht

Die rechte Spalte spiegelt das Formular, während Sie tippen. **Outgoing request** ist genau das JSON, das signiert wird, und das Abzeichen daneben sagt `valid` oder wie viele Beanstandungen das Schema gefunden hat. Der Bereich **JWT** bleibt bis

zum Senden leer: Signiert wird im Node, der private Schlüssel gelangt also nie in den Browser. Nach dem Senden steht dort das kompakte Token, und [Copy JSON](#) / [Copy JWT](#) legen das eine oder das andere in die Zwischenablage.

Send to nimmt ein `host:port`. Anfangs steht dort das Standardziel des Nodes (`-peer`), und die bekannten Peers werden als Vorschläge angeboten; an einen reinen Hostnamen oder eine IP wird der eigene Port angehängt. Drücken Sie [Sign & Send](#), und die Statuszeile daneben meldet das Ergebnis.

ENTWÜRFE

Jedes Schema behält seinen eigenen Entwurf, solange der Tab offen ist; Sie können also zu einem anderen Schema springen und zu dem zurückkommen, was Sie gerade tippten. **Reset** verwirft den Entwurf und baut das Formular leer neu auf. Entwürfe überleben kein Neuladen der Seite; was bleiben soll, speichern Sie als Record.

■ Die Record-Bibliothek

Ein Record ist eine im Node gespeicherte Nutzlast, unter

`records/<schema>/<name>.json`. Beim ersten Start schreibt der Node jedes in

der Binärdatei mitgelieferte Beispiel in dieses Verzeichnis; von da an gehört die

Bibliothek Ihnen. **Load...** und **Save...** liegen in der Werkzeugleiste des Compose-

wie des Antwortformulars.

The screenshot shows the TCPP Node interface. On the left, the 'Compose' panel has a sidebar with 'REQUEST SCHEMAS' and a list of actions like 'archive get', 'archive list', etc. The 'ping' action is selected. The main panel shows 'ping_request - records' with a table of records. The table has columns for 'example', 'shipped', '2026-09-06 11:34', and 'x'. The 'saved' record is highlighted. The right panel shows the 'OUTGOING REQUEST' JSON, which is a complex object with fields like 'tcpp', 'head', 'record', and 'context'. The 'record' field contains a 'ping' action with a 'Check alive' message. The bottom right panel shows the 'SEND TO' field with 'localhost:8444' and a 'Sign & Send' button.

ABBILDUNG 06 • **Load...** listet die für dieses Schema gespeicherten Records. Ein

shipped-Record kam mit der Binärdatei, ein **saved**-Record ist Ihrer. **x** löscht einen,

nach Rückfrage.

The screenshot shows the TCPP Node interface. On the left, the 'REQUEST SCHEMAS' sidebar lists various request types, with 'ping' selected. The main area displays a form for creating a 'ping' request. A 'Save...' button has been clicked, opening a dialog box asking for a name. The dialog shows 'lab ping' as the suggested name. The form fields include: record_type (tcp record pattern), record_direction (request|response|internal pattern), record_udt (universal datetime), record_pagination (unique creating number), record_version (version pattern), origin_type (tcp type of origin pattern), origin_id (tcp id of origin pattern), target_type (tcp type of target pattern), target_id (tcp id of target pattern), context_type (tcp type of context / can be empty pattern), context_id (id of context / can be empty pattern), and error_code (tcp error handling. Hex pattern). The right sidebar shows the 'OUTGOING REQUEST' JSON, which is a valid JSON object. Below the JSON, there is a 'Sign & Send' button and a 'not sent yet' status.

ABBILDUNG 07 • **Save...** fragt nach einem Namen. Es schlägt den zuletzt geladenen vor, warnt vor dem Überschreiben und lehnt Namen außerhalb von **Buchstaben, Ziffern, Leerzeichen, Punkt, Unterstrich, Bindestrich** ab.

Das Laden eines Records ersetzt den Formularinhalt; das Speichern legt ab, was das Formular gerade enthält. Die Bytes bleiben **exakt** erhalten, Schlüsselreihenfolge eingeschlossen, denn genau diese Bytes werden signiert, wenn der Record das nächste Mal geladen und gesendet wird.

DIE MITGELIEFERTEN BEISPIELE ZURÜCKHOLEN

Angelegt wird je Schema-Verzeichnis: ein bereits vorhandenes bleibt unangetastet, ein gelöschter Record bleibt also gelöscht. Löschen Sie **records/<schema>/** oder **records/** ganz, und die mitgelieferten Beispiele sind beim nächsten Start wieder da. Die eingebetteten Beispiele sind unveränderlich und stets unter **/api/example/...** verfügbar; das Bearbeiten eines Records kann ihnen nichts anhaben.

■ Validierung

Das Formular wird beim Tippen gegen das Schema geprüft. Ein Feld, das durchfällt, bekommt einen roten Rahmen und darunter die Begründung, jede Gruppe mit einem fehlerhaften Feld wird umrandet, und das Abzeichen über dem JSON-Bereich zählt, was noch offen ist. Ein Klick darauf springt zum ersten beanstandeten Feld.

The screenshot shows the TCPP Node interface with a schema editor on the left and a form on the right. The 'REQUEST SCHEMAS' panel on the left lists various actions, with 'ping' selected. The main form area displays the 'ping_request' schema. A red border highlights the 'not-an-id' field, which contains the text 'does not match pattern ^https?://(www|)?[-a-zA-Z0-9@%._+~#]{1,255}[/a-zA-Z0-9]{1,63}/b(?:[-a-zA-Z0-9@%._+~#&V=]*)\$'. Below the form, the 'OUTGOING REQUEST' section shows a JSON object with fields like 'tcp', 'head', 'record', 'context', and 'error_code'. A red box with '1 issue' is visible above the JSON, indicating a validation error. The 'Sign & Send' button is active, and the 'SEND TO' field is set to 'localhost:8444'.

ABBILDUNG 08 · Ein Feld verfehlt sein Muster. Die Meldung unter dem Feld ist die tatsächliche Regel aus dem Schema, und das Abzeichen liest sich **1 issue**.

Das Feld, in dem Sie gerade tippen, wird nie markiert; die Markierung erfolgt, wenn Sie es verlassen, damit das Formular während der Arbeit nicht rot aufblitzt.

BEANSTANDUNGEN VERHINDERN NIE DAS SENDEN

Sign & Send bleibt aktiv, wie viele Beanstandungen es auch gibt. Einen absichtlich fehlerhaften Record zu senden ist ein legitimer Weg zu prüfen, was ein Peer damit macht. Über das, was angenommen wird, entscheidet der Peer, nicht der Absender.

Geprüft werden `type`, `required`, `properties`, `items`, `minItems`, `minLength`, `maxLength`, `pattern`, `minimum`, `maximum`, `multipleOf`, `enum`,

`format: date`, `format: date-time` und lokale `$ref`. Alles andere im Schema wird ignoriert, statt es zu erraten.

■ Posteingang und Antworten

Alles, was ein anderer Node hierher sendet, landet in **Inbox**, auch Frames, die sich nicht auswerten ließen; sie werden festgehalten statt verworfen. Neuankömmlinge erscheinen ohne Neuladen und lösen eine Einblendung aus.

TCPP Node schema 1.3.0

TMH / P:NODE-DA7083FF | :8444 | ES384 · da7083ff

Compose

Inbox 3

Sent

Peers

Identity

Log

Received from any

any 1 peer 3

CSU-node localhost:8443 3 · 3 new

Frames 3 frames

Mark all read

ping REQUEST 13:35:41

from CSU-node signature ✓

measurement_create REQUEST 13:35:23

from CSU-node signature ✓

charge_start REQUEST 13:35:23

from CSU-node signature ✓

ABBILDUNG 09 · Die linke Spalte des Posteingangs. Der obere Bereich filtert nach Absender, der untere listet die Frames, neueste zuerst, mit **Mark all read** rechts.

Eine Frame-Zeile lesen

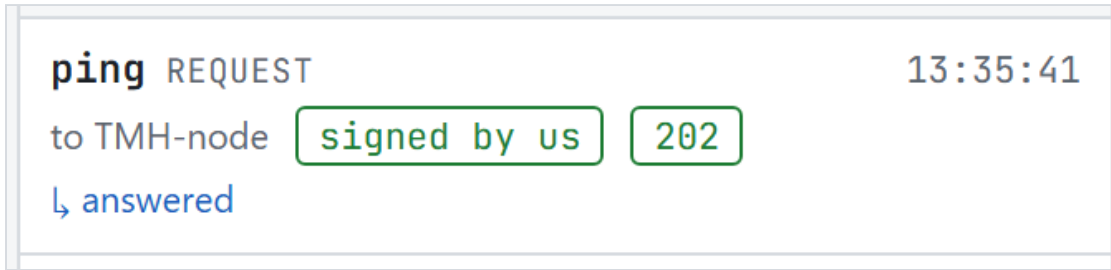


ABBILDUNG 10 · Ein Eintrag. Oben: der Record-Typ, seine Richtung und die Uhrzeit.
Mitte: der Peer und die Zustands-Badges. Unten: wie er zu einem anderen Frame gehört.

Die dritte Zeile erscheint nur, wenn der Frame Teil eines Paares ist:

- `↳ answered`: eine Ihrer Anfragen, die inzwischen beantwortet wurde.
- `↳ awaiting response`: eine Anfrage, die noch keine Antwort hat.
- `↳ answers an earlier request`: dieser Frame ist die Antwort auf eine.

Frame-Detail

Ein ausgewählter Frame öffnet sich rechts und gilt damit als gelesen.

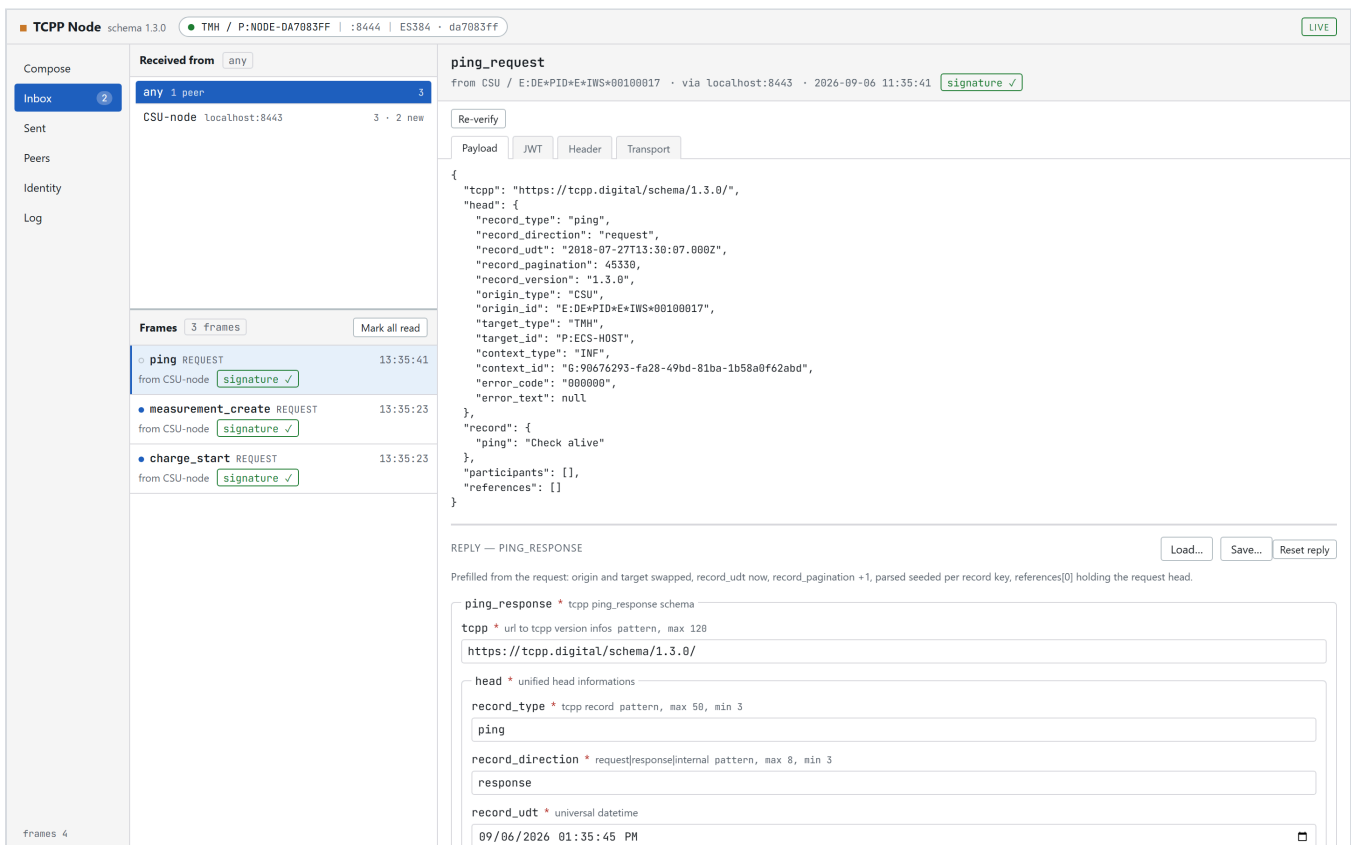


ABBILDUNG 11 · Eine empfangene Anfrage. Die Zeile unter dem Titel nennt den Absender, über welche Adresse sie ankam, wann, und wie ihre Signatur ausfiel.

Vier Reiter enthalten alles, was der Node aufbewahrt hat:

REITER	ZEIGT
Payload	die dekodierte TCPP-Nachricht, formatiert.
JWT	eine einzeilige Zusammenfassung und das rohe kompakte Token, wie es ankam.
Header	den dekodierten JOSE-Header: alg, kid, typ.
Transport	Frame-Id, Richtung, Peer, Prüfzustand, Zustellergesult, Zuordnungen.

```
ping_request  
from CSU / E:DEPID+E:IWS+00100017 · via localhost:8443 · 2026-09-06 11:35:41 signature ✓  
  
Re-verify  
  
Payload JWT Header Transport  
  
alg ES384 · typ JWT · kid fdb7cc0accbb44b8 · 96-byte signature  
  
eyJhbGciOiJIJFZlM4NCIsImptZCI6ImlkYkdjYzBhY2NiYWQybjgiLCJOeXAiOiJKV1QiFC5eyJB0Y3BwLWJoiaHR0cHM6Ly90Y3BwLmRpZDZlOWVwc2NoZWlhZSUmY4WyLiIsImhlYWQioNsnCnVjb3JkX3R5cGU0IiwjaWsnaSwicmVjb3JkX2RpcmcVjdGlvbiI6InJlcXVlc3QilClJCjYZWNvcmrFdWR0IjoimJAxDCEWnyNyN1QxMzozMdowNy4wMDBBaIwicmVjb3JkX3BhZDZlYXRpb24iodjQMzwLClJyZWNVcmrfdmVyc2lubiI6IjEuMy4wIiwib3JpZDZlUX3R5cGU0IjDUUiUlClJvcmlnaW5faWQioiJFokRFklBJRCpFKKlUYuyMDewMDAxNyIsInRhcmdldfF9oeXBFIjoive1TIIiwidGFyZDV0ZXlkIjoieUDPqFMtSE9TVClSiImVnbmRleHRRfdHlwZSI6IkRoIriSiImVnbmRleHRRfaWQioiJHOjkwnjc2MjksZWZhMgtndLIzCE04MWJhLTFiNTThMGY2MmFiczISImVycm9yX2NvZGUoIiImDAADMAiLClcnJvc190ZXh0IjpudWxsfsSwicmVjb3JkIjpw7InBpmcciOijDaGVjayBhbGl2ZSJ9LClJCjwYXJ0aWNPcgFudHMiOlTdLClCYZWZlcmVuY2VziJpbXX0.EtpX WswsVVdWNbn-y n95lgugQHMMwu2EaiwgJgqnrxW6R2uaCDBrCiVRt2TrkpzxapoGxiMhmMcfgubQEouUTi9hvBe962mf66LqJVetUACENCvDBLIwfIWif2Wirxfx1IFmqss
```

ABBILDUNG 12 • Der JWT-Reiter. Das Token ist exakt das, was geprüft wurde; auf dem Weg herein wurde nichts neu kodiert.

Re-verify prüft die gespeicherte Signatur erneut gegen den Schlüssel, den der Peer jetzt hat, und meldet das Ergebnis als Einblendung. Es ist die Schaltfläche, die man drückt, nachdem man den Schlüssel eines Peers hinzugefügt oder geändert hat.

Antworten

Ist der geöffnete Frame eine *Anfrage* und hat der Schemasatz eine passende Antwort, wird unter den Reitern ein Antwortformular angehängt, bereits aus der Anfrage ausgefüllt:

- Herkunft und Ziel vertauscht, denn Sie antworten dem Absender;
- `record_u dt` auf jetzt gesetzt und `record_pagination` um eins erhöht;
- `error_code` auf `000000` zurückgesetzt, statt den der Anfrage zu wiederholen;
- `parsed` mit einem Eintrag je Record-Schlüssel vorbelegt;
- `references[0]` mit dem `head` der Anfrage gefüllt, was beiden Nodes erlaubt, die zwei Frames einander zuzuordnen.

The screenshot shows the TCP Node interface. On the left, there's a sidebar with 'Compose', 'Inbox' (2), 'Sent', 'Peers', 'Identity', and 'Log'. The main area is divided into two panes. The left pane shows 'Received from' with a filter 'any' and a list of messages: 'ping REQUEST' (13:35:41), 'measurement_create REQUEST' (13:35:23), and 'charge_start REQUEST' (13:35:23). The right pane shows a 'REPLY' form for 'PING_RESPONSE'. The form is pre-filled with data from the request and includes fields for 'tcp', 'head', 'record_type', 'record_direction', 'record_udt', 'record_pagination', 'record_version', 'origin_type', 'origin_id', 'target_type', 'target_id', 'context_type', 'context_id', and 'error_code'. Buttons for 'Load...', 'Save...', and 'Reset reply' are at the top right of the form.

ABBILDUNG 13 · Der Antwortblock. Beachten Sie die vertauschten `origin_type` / `target_type` und die erhöhte `record_pagination`. [Load...](#) und [Save...](#) beziehen sich hier auf das *Response*-Schema.

Die Antwortadresse ist vorgelegt und änderbar. Hat der Absender eine Rückadresse mitgeteilt, wird Ihnen das gesagt; musste der Node sie aus der Verbindung erschließen, werden Sie gebeten, sie zu prüfen. [Sign & Send response](#) signiert und stellt zu, [Reset reply](#) befüllt das Formular erneut aus der Anfrage.

Eine begonnene Antwort bleibt je Frame erhalten, solange der Tab offen ist; zu einem anderen Frame und zurück zu wechseln verliert sie also nicht. Ist der geöffnete Frame selbst eine *Antwort*, gibt es kein Formular, nur einen Hinweis auf die Anfrage, die er beantwortet.

Sent

Sent ist dieselbe Anordnung für ausgehende Frames. Ungelesen- Markierungen gibt es hier nicht; worauf es ankommt, ist, ob die Zustellung geklappt hat.

The screenshot shows the TCPP Node interface. On the left, a sidebar lists 'Compose', 'Inbox', 'Sent', 'Peers', 'Identity', and 'Log'. The 'Sent' tab is active, showing a list of outgoing frames. The main panel displays a detailed view of a 'ping_request' frame. The frame is sent to 'any 2 peers' and includes a 'payload' section with a JSON object. The JSON object contains metadata and a 'record' section. The 'record' section includes a 'ping' event with a 'check alive' status. The frame is signed by 'us' and has a status of '202' (accepted).

TCPP Node schema 1.3.0 | CSU / P: NODE-FDB7CC8A | :8443 | ES384 - fdb7cc8a | LIVE

Compose
Inbox
Sent
Peers
Identity
Log

Sent to any 2 peers 4

TMH-node localhost:8444 3
localhost:9099 unknown node 1

Frames 4 frames

ping REQUEST 13:35:41
to TMH-node signed by us 202
↳ answered

charge_event REQUEST 13:35:23
to localhost:9099 signed by us send failed
↳ awaiting response

measurement_create REQUEST 13:35:23
to TMH-node signed by us 202
↳ awaiting response

charge_start REQUEST 13:35:23
to TMH-node signed by us 202
↳ awaiting response

ping_request
to CSU / E: DE*PID+E*IWS*00100017 · via localhost:8444 · 2026-09-06 11:35:41 signed by us 202 accepted

Payload JWT Header Transport

```
{
  "tcpp": "https://tcpp.digital/schema/1.3.0/",
  "head": {
    "record_type": "ping",
    "record_direction": "request",
    "record_uat": "2018-07-27T13:30:07.000Z",
    "record_pagination": 45330,
    "record_version": "1.3.0",
    "origin_type": "CSU",
    "origin_id": "E:DE*PID+E*IWS*00100017",
    "target_type": "TMH",
    "target_id": "P:ECS-HOST",
    "context_type": "INF",
    "context_id": "G:90676293-fa28-49bd-81ba-1b58a0f62abd",
    "error_code": "000000",
    "error_text": null
  },
  "record": {
    "ping": "Check alive"
  },
  "participants": [],
  "references": []
}
```

frames 6

ABBILDUNG 14 · Ausgehende Frames. **signed by us** bei allen, **202**, wo der Peer sie angenommen hat, und **send failed**, wo niemand zuhörte.

Ein Frame wird festgehalten, ob er zugestellt wurde oder nicht; ein gescheitertes Senden hinterlässt also einen vollständigen Eintrag, Token eingeschlossen, und der Grund steht im Reiter **Transport**. Sobald die Antwort eintrifft, bekommt die Anfrage ihre Zeile **↳ answered**.

■ Peers und Verifikation

Ein Peer ist ein Node, mit dem Sie Frames ausgetauscht haben, oder einer, den Sie selbst eingetragen haben: ein Name, ein `host:port` und wahlweise sein öffentlicher Schlüssel. Peers erscheinen von selbst, sobald zum ersten Mal ein Frame in die eine oder andere Richtung läuft, aber **nie mit Schlüssel**. Den Schlüssel zu hinterlegen ist immer eine bewusste Handlung.

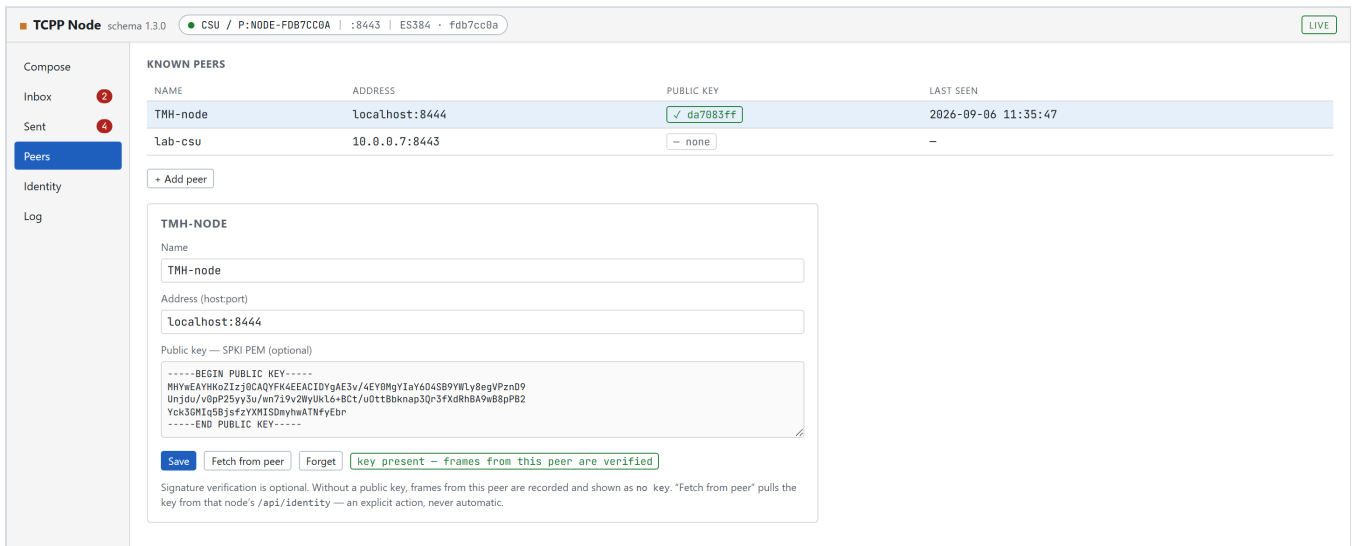


ABBILDUNG 15 • Zwei Peers: einer, dessen Schlüssel geholt wurde, und einer, der nie kontaktiert wurde. Eine ausgewählte Zeile öffnet sich in der Karte darunter.

Einem Peer seinen Schlüssel geben

- 01 Wählen Sie den Peer in der Tabelle, oder drücken Sie **+ Add peer** und tragen Sie Name und Adresse ein.
- 02 Fügen Sie entweder sein SPKI-PEM in das Schlüsselfeld ein, oder drücken Sie **Fetch from peer**, um es von dessen `/api/identity` zu holen. Der Peer muss gespeichert sein, bevor von ihm geholt werden kann.
- 03 Drücken Sie **Save**. Jeder gespeicherte Frame dieses Peers wird sofort erneut geprüft, und das Abzeichen unter dem Feld wird grün.

Das einzufügende PEM finden Sie auf dem Bildschirm **Identity** des anderen Nodes hinter **Copy PEM**. **Forget** entfernt den Peer; die von ihm gesendeten Frames bleiben im Log und fallen auf `no key` zurück.

Was die Badges bedeuten

BADGE	BEDEUTUNG
<code>signature ✓</code>	Das Token verifiziert gegen den für diesen Peer gespeicherten Schlüssel.
<code>signature ✗</code>	Tut es nicht. Der Frame wurde trotzdem festgehalten; sehen Sie ihn sich an.
<code>no key</code>	Für diesen Peer ist kein Schlüssel hinterlegt, es wurde also nichts geprüft.
<code>unverified</code>	Nicht geprüft, etwa ein Frame, dessen Token sich nicht auswerten ließ.
<code>signed by us</code>	Dieser Node hat ihn signiert. Steht an allem unter <code>Sent</code> .

VERIFIZIEREN IST FREIWILLIG, SIGNIEREN NICHT

Ein Frame von einem Peer ohne Schlüssel wird festgehalten und angezeigt, nicht abgewiesen; Sie können ein ganzes Labor betreiben, ohne einen einzigen Schlüssel zu hinterlegen. Signiert wird dagegen immer: Jeder Frame, den dieser Node sendet, ist ein ES384-JWS.

■ Identität

The screenshot shows the 'Identity' tab of the TCPP Node interface. The top bar indicates the node is 'LIVE' and provides details: CSU / P:NODE-FDB7CC0A, port 8443, ES384 algorithm, and node ID fdb7cc9a. The left sidebar contains navigation links: Compose, Inbox (2), Sent (4), Peers, Identity (selected), and Log.

THIS NODE

Node	CSU / P:NODE-FDB7CC0A
Advertised	localhost:8443 - sent as X-TCPP-From so peers can reply
Algorithm	ES384 (ECDSA P-384 / SHA-384)
Key id	fdb7cc0accbb44b8
Fingerprint	47:3b:43:39:3a:91:23:15:25:fc:c9:d7:31:3a:a7:12:67:d4:74:27:2f:57:8e:78:52:0b:c3:4b:43:68:fa:5e
Certificate	self-signed, valid to 2036-09-03
Started	2026-09-06 11:35:17

PUBLIC KEY — GIVE THIS TO PEERS THAT WANT TO VERIFY YOU

```
-----BEGIN PUBLIC KEY-----
MHYwEAYHkoZizj0CAQYFK4EEACIDYgAE9c5w1QaICjX+L4Vlna7wVUFJ0mUuZpW
fRx5rT2pTYd9T+VPdR5FoaLHWA0JabohN5LF8XgtRUu2Ynq3GJGyynS8r+prxB
aoL15ZLN04HyLOvNBfnzgsCKVU0L/CZ
-----END PUBLIC KEY-----
```

FILES NEXT TO THE BINARY

Private key	C:\Users\User\tcpp-lab\tcpp-key.pem
Certificate	C:\Users\User\tcpp-lab\tcpp-cert.pem
Node name	C:\Users\User\tcpp-lab\tcpp-node.json
Frame log	C:\Users\User\tcpp-lab\tcpp-frames.jsonl
Peers	C:\Users\User\tcpp-lab\tcpp-peers.json
Records	C:\Users\User\tcpp-lab\tcpp-records

Drop your own tcpp-key.pem and tcpp-cert.pem next to the binary and restart to use them — the same pair signs JWTs and serves TLS.

ABBILDUNG 16 · Alles, was dieser Node ist, und wo er es aufbewahrt.

Node

Der TCPP-Modultyp und die Id, als die dieser Node signiert und spricht.

Advertised

Die Adresse, die Peers als **X-TCPP-From** mitgeteilt wird, damit sie wissen, wohin sie antworten sollen. Mit **--advertise** zu ändern, wenn Peers Sie unter einem anderen Namen erreichen.

Algorithm

Immer ES384: ECDSA auf P-384 mit SHA-384.

Key id

Aus dem öffentlichen Schlüssel abgeleitet. Ihre ersten acht Zeichen stehen im JOSE-Header jedes Tokens; daran erkennt man auf einen Blick, welcher Schlüssel einen Frame signiert hat.

Fingerprint

SHA-256 über das Zertifikat, zum Vergleichen auf anderem Weg.

Certificate

Selbstsigniert, zehn Jahre, beim ersten Start erzeugt.

Die mittlere Karte enthält den öffentlichen Schlüssel für jeden, der diesen Node verifizieren möchte; [Copy PEM](#) legt ihn in die Zwischenablage. Die untere Karte listet die sechs Pfade aus [Erste Schritte](#), aufgelöst.

DEN EIGENEN SCHLÜSSEL MITBRINGEN

Dasselbe Paar signiert JWTs und terminiert TLS; die eigene Identität zu verwenden heißt deshalb schlicht, `tcpp-key.pem` und `tcpp-cert.pem` neben die Binärdatei zu legen und neu zu starten. Ein Schlüssel ohne Zertifikat bekommt ein frisches, aus ihm abgeleitetes Zertifikat. Ein Zertifikat, das nicht zum Schlüssel passt, ist ein Startfehler statt einer stillen Unstimmigkeit, und ein Zertifikat ohne seinen Schlüssel wird rundheraus abgelehnt, denn ein Zertifikat kann nichts signieren.

Log

TCPP Node schema 1.3.0 | CSU / P: NODE-FDB7CC0A | :8443 | ES384 - fdb7cc9a | LIVE

Compose

Inbox 2

Sent 4

Peers

Identity

Log

FRAME LOG

all directions | all peers | search record type, id... | Refresh | 6 of 6 frames

2026-09-06 11:35:47	RECV	localhost:8444	ping response	signature ✓
2026-09-06 11:35:41	SENT	localhost:8444	ping request	202
2026-09-06 11:35:23	SENT	localhost:9099	change_event request	ERR
2026-09-06 11:35:23	RECV	localhost:8444	status request	signature ✓
2026-09-06 11:35:23	SENT	localhost:8444	measurement_create request	202
2026-09-06 11:35:23	SENT	localhost:8444	change_start request	202

appended to C:\Users\User\tcpp-lab\atcpp-frames.jsonl - one JSON object per line, tailed here

ABBILDUNG 17 · Beide Richtungen an einem Ort. Die Zeile unten nennt die Datei, aus der diese Zeilen gelesen werden.

Jede Zeile besteht aus einem Zeitstempel, **SENT** oder **RECV**, der Peer-Adresse, Record-Typ und Richtung sowie einer Zustandsspalte: dem HTTP-Status oder **ERR** bei ausgehenden Frames, dem Prüfergebnis bei eingehenden. Filtern Sie nach Richtung oder Peer, suchen Sie über Record-Typ, Herkunft, Ziel und Frame-Id, und drücken Sie **Refresh**, um die Liste neu zu holen.

Die Datei dahinter

`tcp-frames.jsonl` wird nur angehängt: ein JSON-Objekt pro Zeile, nie umgeschrieben, mit dem Prüfzustand *wie er in jenem Moment war*. Beim Start spielt der Node die letzten `--log-tail` Zeilen wieder ein (standardmäßig 500), damit Zuordnung und Historie einen Neustart überleben, und prüft die Signaturen erneut gegen die Schlüssel, die Sie jetzt haben. Nach einem Neustart ist nichts ungelesen. Zeilen, die sich nicht auswerten lassen, werden mit einer Warnung übersprungen, statt den Node anzuhalten.

■ Arbeitsabläufe

A. Die erste Anfrage senden

- 01 Öffnen Sie **Compose** und wählen Sie **ping** in der Schema-Liste.
- 02 Drücken Sie **Load...** und wählen Sie **example**. Das Formular füllt sich, und das Abzeichen wird **valid**.
- 03 Ändern Sie, was Sie mögen; **origin_id** und **target_id** sind die üblichen ersten Stationen.
- 04 Prüfen Sie **Send to**. Dort steht das Standardziel; die Auswahlliste bietet bekannte Peers an.
- 05 Drücken Sie **Sign & Send**. **✓ 202 accepted** heißt, der Peer hat ihn genommen.
- 06 Öffnen Sie **Sent**, um den Frame, sein Token und sein Transportergebnis zu sehen.

B. Eine eingegangene Anfrage beantworten

- 01 Öffnen Sie auf dem empfangenden Node **Inbox**. Ungelesene Frames tragen einen gefüllten Punkt, und das Abzeichen in der Leiste zählt sie.
- 02 Wählen Sie den Frame. Er gilt damit als gelesen und öffnet sich rechts.
- 03 Lesen Sie ihn unter **Payload**; wer ihn signiert hat, sagt Ihnen das Badge in der Kopfzeile.
- 04 Scrollen Sie zu **Reply** hinunter. Die Antwort ist aus der Anfrage vorbefüllt; füllen Sie die Teile aus, die Ihre Antwort tragen.
- 05 Bestätigen Sie die Rückadresse und drücken Sie **Sign & Send response**.
- 06 Auf dem ersten Node kommt die Antwort im Posteingang an, und seine Anfrage liest sich nun **↳ answered**.

C. Aus `no key` ein `signature ✓` machen

- 01 Öffnen Sie `Peers` und wählen Sie den Peer, oder legen Sie ihn mit seinem `host:port` an.
- 02 Geben Sie ihm einen Namen, den Sie in den Frame-Listen wiedererkennen, und drücken Sie `Save`.
- 03 Drücken Sie `Fetch from peer` (oder fügen Sie das PEM vom Bildschirm `Identity` jenes Nodes ein) und drücken Sie erneut `Save`.
- 04 Zurück in `Inbox`: Jeder Frame, den dieser Peer je gesendet hat, liest sich nun `signature ✓`. Die Prüfung wird über das ganze gespeicherte Log neu gerechnet, nicht nur über Neuankömmlinge.

D. Eine Nutzlast aufheben, die Sie wieder brauchen

- 01 Stellen Sie die Anfrage zusammen, bis sie sagt, was sie sagen soll.
- 02 Drücken Sie `Save...`, geben Sie einen Namen (`lab ping`, `ac-start-berlin`) und drücken Sie `Save`.
- 03 Sie erscheint unter `Load...` als `saved` markiert, und auf der Platte unter `records/<schema>/<name>.json`.
- 04 Records sind gewöhnliche Dateien. Kopieren Sie das Verzeichnis auf einen anderen Node, und die Bibliothek zieht mit.

E. Prüfen, was ein Peer mit einer fehlerhaften Nachricht macht

- 01 Laden Sie einen Record und machen Sie ein Feld absichtlich kaputt: eine Id, die ihr Muster verfehlt, ein geleertes Pflichtfeld.
- 02 Das Feld wird rot, und das Abzeichen zählt die Beanstandung. Senden Sie trotzdem.

- 03 Auf dem empfangenden Node wird der Frame wie jeder andere festgehalten; was dessen Betreiber sieht, ist genau das, was auch Ihre Peer-Implementierung sähe.
- 04 Ein nicht auswertbares Token wird ebenfalls festgehalten, als `unverified` markiert und mit `400` statt `202` beantwortet.

F. Einen Node umziehen oder ihm eine neue Identität geben

- 01 Zum Umziehen halten Sie den Node an und kopieren sein ganzes Verzeichnis. Schlüssel, Zertifikat, Name, Log, Peers und Records reisen zusammen, er kommt also als derselbe Node zurück.
- 02 Für eine neue Identität legen Sie Ihre `tcpp-key.pem` und `tcpp-cert.pem` hinein und starten neu.
- 03 Damit eine frische erzeugt wird, löschen Sie beide Dateien und starten neu; die Node-Id wird aus dem neuen Schlüssel abgeleitet.
- 04 Was daraus geworden ist, zeigt `./tcpp-node --print-identity`; geben Sie das ausgegebene PEM den Peers, die Sie verifizieren.

DER SCHLÜSSEL IST DER NODE

`tcpp-key.pem` ist ein Geheimnis und nicht wiederherstellbar. Ein neuer Schlüssel bedeutet eine neue Schlüssel-Id und eine neue Node-Id, und jeder Peer, der Sie verifiziert, braucht den neuen öffentlichen Schlüssel.

■ Kommandozeile und Dateien

Flags

Es gibt keine Unterbefehle und keine Umgebungsvariablen; die Flags und die Dateien auf der Platte sind die gesamte Konfiguration. Sowohl `-flag` als auch `--flag` funktioniert.

FLAG	STANDARD	BEDEUTUNG
<code>--port</code>	<code>8443</code>	TLS-Port, auf dem gelauscht wird.
<code>--data-dir</code>	neben der Binärdatei	Wo Schlüssel, Zertifikat, Log, Peers und Records liegen.
<code>--advertise</code>	<code>localhost:<Port></code>	Adresse, an die Peers antworten sollen, gesendet als <code>X-TCPP-From</code> .
<code>--peer</code>	<code>localhost:<Port></code>	Standardziel, im Browser vorbelegt. Der Standard ist dieser Node selbst.
<code>--node-type</code>	<code>CSU</code> , oder der gespeicherte Wert	Drei Großbuchstaben. Nach dem ersten Start gespeichert.
<code>--node-id</code>	aus dem Schlüssel abgeleitet	3–120 Zeichen. Nach dem ersten Start gespeichert.
<code>--hosts</code>	keine	Weitere Hostnamen oder IPs für das Zertifikat, kommasetrennt.
<code>--log-tail</code>	<code>500</code>	Beim Start aus dem Log wiedereingespielte Frames.
<code>--insecure-peers</code>	<code>true</code>	Selbstsignierte Zertifikate der Peers annehmen. Mit <code>--insecure-peers=false</code> abschalten.
<code>--print-identity</code>	<code>false</code>	Identität anlegen oder laden, ausgeben und beenden, ohne zu lauschen.

Das Zertifikat, das ein Node erzeugt, deckt immer `localhost`, `127.0.0.1`, `::1` und den Hostnamen des Rechners ab; alles Weitere, worüber Peers ihn erreichen sollen, gehört in `--hosts`.

EIN NODE JE VERZEICHNIS

Zwei Kopien der Binärdatei, im *selben* Verzeichnis gestartet, teilen sich einen Schlüssel, ein Frame-Log und eine Record-Bibliothek, und die zweite bekäme den Port nicht. Geben Sie jedem Node ein eigenes Verzeichnis oder ein eigenes `--data-dir`.

--print-identity

```
$ ./tcpp-node --print-identity
node      CSU / P:NODE-74734B2A
alg       ES384
kid       74734b2a3dc9c8f9
fingerprint 48:81:2b:67:07:83:91:d7:ce:59:20:9a:83:0c:07:fa:ac:0f:da:f0:6a:da:02:6c
key       C:\Users\User\tcpp-lab\a\tcpp-key.pem
cert      C:\Users\User\tcpp-lab\a\tcpp-cert.pem (valid to 2036-08-31T16:33:05Z)

-----BEGIN PUBLIC KEY-----
MHYwEAYHKoZIzj0CAQYFK4EEACIDYgAEnfz9/7TBzZv7NCAdmkGv+KzYUVnfVeum
amuA/gGEdCStd2Tm7X1JJALLWPYN7dJ9LzNIX4qNPUIMiRyOKZkif28ZvL6iuRTr
+voCuitlwh+WBuGNBFiCSvFQcFbfjDpy
-----END PUBLIC KEY-----
```

Fehlt eine Identität, wird sie dabei angelegt: ein bequemer Weg, einen Node vor seinem ersten Start einzurichten.

Laufen und Anhalten

Der Node läuft im Vordergrund und schreibt seine Meldungen nach Standardfehler. Strg+C (oder `SIGTERM`) fährt ihn binnen fünf Sekunden geordnet herunter. Er endet mit `0` nach sauberem Herunterfahren und mit `1`, nachdem er `tcppnode: <Grund>` ausgegeben hat, sei es ein bereits belegter Port, ein Datenverzeichnis, in das er nicht schreiben kann, oder ein Zertifikat, das nicht zu seinem Schlüssel passt.

Was zu sichern ist

`tcpp-key.pem` und `tcpp-cert.pem` sind die Identität und lassen sich nicht wiederherstellen. `tcpp-node.json`, `tcpp-peers.json` und `records/` sind

erhaltenswerte Konfiguration. `tcpp-frames.jsonl` ist Historie: sie wächst endlos und darf rotiert werden, wenn sie lästig wird; der Node liest immer nur ihr Ende.

■ HTTP-API

Die Browser-Anwendung ist ein gewöhnlicher Client der HTTP-Schnittstelle des Nodes; alles, was sie tut, lässt sich also skripten. Alles davon wird über TLS auf dem einen Port des Nodes ausgeliefert.

KEINE AUTHENTIFIZIERUNG

Keine dieser Routen verlangt Zugangsdaten, und es gibt keine Origin-Prüfung. Wer den Node erreicht, hat ihn vollständig in der Hand.

Node zu Node

ROUTE	BEDEUTUNG
POST /tcpp	Der Rumpf ist ein kompaktes JWS, <code>Content-Type: application/jwt</code> , bis 1 MB. Senden Sie <code>X-TCPP-From</code> , damit der Empfänger antworten kann, und <code>X-TCPP-Node</code> , um sich zu benennen. Antwortet <code>202</code> mit <code>{"received":"<Frame-Id>", "node":"<Bezeichnung>"}</code> , oder <code>400</code> , wenn sich das Token nicht lesen lässt; der Frame wird dann trotzdem festgehalten.

Browser

ROUTE	LIEFERT
GET /	Die Web-Anwendung, eingebettet in die Binärdatei.
GET /schema/1.3.0/{name}	Ein eingebettetes Schema, als <code>application/schema+json</code> Endung <code>.json</code> wird akzeptiert.
GET /api/identity	Node-Typ und -Id, Bezeichnung, angekündigte Adresse, Standard-Peer, öffentliches Schlüssel-PEM, <code>kid</code> , Fingerabdruck, Zertifikatsablauf, Dateipfad, Startzeit, Frame- und Ungeräte-Zähler.
GET /api/catalog	Jeden Endpunkt mit den Namen seiner Request- und Response-Schemas und seinen Beispielvarianten.
GET /api/example/{base}/{variant}	Eine mitgelieferte Beispiel-Nutzlast. Ohne Variante bedeutet <code>example</code> .

ROUTE	LIEFERT
POST /api/send	Signiert und stellt zu. Rumpf <code>{"peer": "host:port", "message": "...", "in_reply_to": "<Frame-Id>"}</code> . Immer 200, mit dem festgehaltenen <code>frame</code> , dem <code>jwt</code> und einem <code>transport</code> Objekt; <code>error</code> steht dabei, wenn die Zustellung scheiterte.
GET /api/frames	Frame-Liste samt Zählern je Peer und Gesamtzahlen. Filter <code>direction=sent received</code> , <code>?peer=host:port</code> , <code>?limit=...</code> .
GET /api/frames/{id}	Einen Frame vollständig: Nutzlast, Token, dekodierter JOSE Header und, wo es eines gibt, das Antwortschema.
POST /api/frames/{id}/read POST /api/frames/read	Einen Frame oder alle als gelesen markieren.
POST /api/frames/{id}/verify	Eine Signatur erneut prüfen. Ein optionaler Rumpf <code>{"public_key_pem": "..."} </code> prüft gegen einen von Ihnen gelieferten Schlüssel statt gegen den gespeicherten.
GET /api/peers	Bekannte Peers mit Name, Adresse, Schlüsselzustand, <code>known</code> Node-Id, zuletzt gesehen.
PUT /api/peers	Anlegen oder ändern: <code>{"name", "address", "public_key_pem", "node_id", "remember_key"}</code> Beim Speichern wird jeder gespeicherte Frame dieses Peer geprüft.
DELETE /api/peers/{name}	Einen Peer vergessen.
POST /api/peers/{name}/fetch-key	Die <code>/api/identity</code> des Peers lesen und seinen öffentlichen Schlüssel speichern.
GET /api/records/{schema}	Die für ein Schema gespeicherten Records, jeder mit <code>ship_id</code> , <code>modified</code> , <code>size</code> .
GET /api/records/{schema}/{name}	Einen Record, Byte für Byte wie gespeichert.
PUT /api/records/{schema}/{name}	Einen Record speichern. Der Rumpf ist die rohe JSON-Nutzlast, max. 1 MB.
DELETE /api/records/{schema}/{name}	Einen Record entfernen.

ROUTE	LIEFERT
GET /api/events	Server-Sent Events: <code>frame.received</code> , <code>frame.sent</code> , <code>frame.updated</code> .

Einen Node aus der Shell steuern

Das Zertifikat des Nodes ist selbstsigniert, einem Client muss man also sagen, dass er es annehmen soll.

```
# die eigene Identität des Nodes
curl -k https://localhost:8443/api/identity

# einen gespeicherten Record an einen Peer senden
curl -k -X POST https://localhost:8443/api/send \
  -H 'content-type: application/json' \
  -d '{"peer":"localhost:8444","message":$(curl -sk https://localhost:8443/ap

# was angekommen ist
curl -k 'https://localhost:8443/api/frames?direction=received&limit=10'
```

```
# PowerShell
$node = 'https://localhost:8443'
$msg = Invoke-RestMethod "$node/api/records/ping_request/example" -SkipCertificate
Invoke-RestMethod "$node/api/send" -Method Post -SkipCertificateCheck `
  -ContentType 'application/json' `
  -Body (@{ peer = 'localhost:8444'; message = $msg } | ConvertTo-Json -Depth 20)
```

NUTZLAST-BYTES WERDEN NIE NEU KODIERT

`/api/send` signiert die `message` genau so, wie Sie sie übergeben, Schlüsselreihenfolge eingeschlossen, und ein Empfänger liest sie wortwörtlich zurück. Das ist Absicht: Eine Signatur deckt Bytes ab, nicht ein abstraktes Objekt; nichts darf eine Nachricht zwischen Signieren und Prüfen neu serialisieren.

■ Die TCPP-Nachricht

TCPP ist das Protokoll, das diese Schemas definieren: ein JSON-Umschlag, der zwischen Modulen der Ladeinfrastruktur ausgetauscht wird. Die Version dieses Handbuchs ist **1.3.0**, und jede Nachricht nennt sie in ihrem ersten Feld.

```
{
  "tcpp": "https://tcpp.digital/schema/1.3.0/",
  "head": {
    "record_type": "ping",
    "record_direction": "request",          request | response | internal
    "record_udt": "2018-07-27T13:30:07.000Z",
    "record_pagination": 45330,             zählt aufwärts, von 1 bis endlos
    "record_version": "1.3.0",
    "origin_type": "CSU",                   drei Buchstaben, das sendende Modul
    "origin_id": "E:DE*PID*E*IWS*00100017",
    "target_type": "TMH",
    "target_id": "P:ECS-HOST",
    "context_type": "INF",                  der Vorgang, nicht das Modul
    "context_id": "G:90676293-fa28-49bd-81ba-1b58a0f62abd",
    "error_code": "000000"                 hexadezimal; 000000 = kein Fehler
  },
  "record": { "ping": "Check alive" },     Form hängt vom record_type ab
  "parsed": { },                           nur Antworten: ein Fehlercode je Element
  "participants": [ ],                     steuert die ausgehende Kommunikation
  "references": [ ]                       eine Antwort trägt hier den head der A
}
```

Ids tragen ihren Namensraum

Jede Id beginnt mit einem zwei Zeichen langen Präfix, das sagt, welche Art von Bezeichner folgt.

PRÄFIX	BEDEUTUNG	PRÄFIX	BEDEUTUNG
E:	EVSEID / EMAID	G:	GUID
D:	Domänen-Namensraum	H:	Hash
P:	privater Namensraum	N:	numerische Paginierung
I:	intern	C:	Vertrags- / Tarif-Id

PRÄFIX	BEDEUTUNG	PRÄFIX	BEDEUTUNG
V:	VID	X:	undefiniert

Ein Node, der sich seine Id selbst erzeugt, verwendet den privaten Namensraum; deshalb heißen die Labor-Nodes `P:NODE-9BED6853`.

Modultypen

`origin_type` und `target_type` sind drei Großbuchstaben, die die Art des Moduls benennen: `TMH` Trusted Management Host, `TEH` Trusted Exchange Host, `TAH` Trusted Archive Host, `MSP` eMobility Service Provider, `CP0` Charge Point Operator, `CSU` Charge Switch Unit, `EMU` Energy Management Unit und so fort. Seinen eigenen kündigt ein Node mit `--node-type` an.

Wie eine Anfrage und ihre Antwort zueinanderfinden

Zu jeder Anfrage gehört genau eine Antwort. Der Antwortende kopiert den vollständigen `head` der Anfrage nach `references[0]`, und beide Nodes berechnen daraus denselben Schlüssel: Record-Typ, Richtung, Zeitstempel, Paginierung, Herkunft und Ziel. So können die Frame-Listen auf beiden Seiten `↳ answered` und `↳ answers an earlier request` sagen, ohne dass ein Node eine Sitzung führen müsste. Antworten Sie aus dem Posteingang, übergibt der Node zusätzlich die Frame-Id ausdrücklich, damit die Zuordnung nicht davon abhängt, dass die Zeitstempel unverseht bleiben.

Signieren

Ein ausgehender Frame ist ein kompaktes JWS über die Bytes der Nutzlast, mit dem Header `{"alg":"ES384","kid":"...","typ":"JWT"}`. Zwei Einzelheiten sind es wert, gekannt zu werden, weil interoperable Implementierungen üblicherweise genau dort scheitern:

- **R||S in fester Breite.** JWS will die ECDSA-Signatur als zwei 48-Byte- Hälften (IEEE P1363), genau das, was die Web Crypto des Browsers erzeugt. Gos `ecdsa.SignASN1` erzeugt stattdessen DER und darf nicht verwendet werden; etwa jede 256. Signatur hat eine kurze Koordinate, die links aufgefüllt werden muss.

- **Die Nutzlast wird so signiert, wie sie übergeben wurde.** `JSON.stringify` erhält die Einfügereihenfolge der Schlüssel, während `Gos` `encoding/json` die Schlüssel von Maps sortiert; eine Nachricht wird zwischen Signieren und Prüfen deshalb nie neu serialisiert.

■ Anhang

Die 27 Endpunkte

Jeder hat ein Request- und ein Response-Schema, ausgeliefert unter

`/schema/1.3.0/<base>_request.json` und `..._response.json`. Manche bringen mehr als ein Beispiel mit, und jedes Beispiel wird zu einem Record, den Sie laden können.

ENDPUNKT	WEITERE BEISPIELVARIANTEN
<code>archive_get</code> , <code>archive_list</code>	keine
<code>charge_event</code>	Request: <code>example_with_charge_end</code>
<code>charge_process</code> , <code>charge_start</code>	keine
<code>display_input</code> , <code>display_output</code>	keine
<code>email_get</code>	Request: <code>example_via_vehicle</code>
<code>error</code>	Request: <code>example_hardware_internal</code> , <code>example_hardware_request</code>
<code>key_get</code>	keine
<code>measurement_change</code> , <code>measurement_configuration</code> , <code>measurement_create</code>	keine
<code>module_get</code>	keine
<code>parameters_delete</code> , <code>parameters_get</code> , <code>parameters_set</code>	keine
<code>parameters_list</code>	Response: <code>example_measurement</code>
<code>ping</code>	Response: <code>example_with_participants</code> , <code>example_with_reference_jws</code>
<code>status</code>	keine

ENDPUNKT	WEITERE BEISPIELVARIANTEN
<code>tariffs_list</code>	Request: <code>example_cp_in_location</code> , <code>example_cp_search</code>
<code>trigger_delete</code> , <code>trigger_get</code> , <code>trigger_set</code> , <code>triggers_list</code>	keine
<code>verification_get</code> , <code>verification_set</code>	keine

Glossar

Frame

Eine TCPP-Nachricht, wie dieser Node sie gesehen hat: Token, dekodierte Nutzlast, Richtung, Peer, Prüfzustand, Transportergebnis und Zuordnungen.

Peer

Ein anderer Node: ein Name, eine Adresse und wahlweise ein öffentlicher Schlüssel.

Record

Eine gespeicherte Nutzlast in der Bibliothek des Nodes, unter

`records/<schema>/<name>.json`.

Entwurf

Ungesendeter Formularinhalt, im Browser-Tab gehalten. Je Schema im Compose-Formular, je Frame im Antwortformular.

Ziel

Das `host:port`, an das ein Frame gesendet wird.

kid

Schlüssel-Id, aus dem öffentlichen Schlüssel abgeleitet und im Header jedes Tokens mitgeführt.

JWS / JWT

Die kompakte signierte Form, in der ein Frame reist. Hier immer ES384.

Über die Abbildungen

Jede Abbildung in diesem Handbuch stammt aus einem laufenden Zwei-Node-Labor, keine ist gezeichnet; was Sie hier sehen, ist also, was die Anwendung tut. Aufgenommen werden sie von einem Skript, das neben der Quelle des Handbuchs

liegt und einen Browser ohne Anzeige durch dieselben Schritte führt, die ein Bediener geht; sie werden neu aufgenommen, wann immer sich die Oberfläche ändert. Die Anwendung ist englisch, deshalb zeigen auch die Abbildungen dieser Ausgabe die englische Oberfläche.

tcpp.digital Deutsches Benutzerhandbuch · Schema 1.3.0

Abbildungen aus einem Zwei-Node-Labor (CSU auf 8443, TMH auf 8444), aufgenommen am 3. September 2026. Teile dieser Dokumente wurden mit KI-Unterstützung erstellt.