Technische Dokumentation

Ausgestellt von: DACTA Core

Datum: 13.11.2026 04:29 UTC

Inhalt

Aggiorna il sommario: tasto destro → Aggiorna campo (F9).

Versione dei formati: passaporto-evento/1, passaporto-prova/1, passaporto-chiavi/1. Una modifica alla forma canonica, all'algoritmo d'impronta o alla struttura dell'albero crea una versione nuova; le versioni precedenti restano documentate e verificabili.

1. Documento dell'evento (passaporto-evento/1)

Ogni evento di filiera è un documento JSON costruito dalle colonne dell'evento. Campi:

Campo	Contenuto
formato	passaporto-evento/1
prodotto, lotto	identificativo persistente del prodotto e, se c'è, del lotto
sequenza	numero progressivo dell'evento nella catena del prodotto (da 1)
precedente	impronta dell'evento precedente dello stesso prodotto (null per il primo)
emittente	azienda che registra: id, ragione_sociale, piva
tipo	tipo di evento del catalogo (produzione, assemblaggio, collaudo, spedizione, passaggio_proprieta, rettifica…)
epcis	corrispondenza GS1 EPCIS 2.0: type, action, bizStep, disposition
quando, fuso	istante dell'evento in UTC (ISO 8601) e scostamento del fuso locale
dove	luogo: nome, id
passaggio	per i passaggi di proprietà: cedente e cessionario; altrimenti null
rettifica_di	per le rettifiche: impronta dell'evento rettificato; altrimenti null
visibilita	pubblico o riservato
nota, attributi	testo libero e attributi propri del tipo di evento
riservati	impegno sui dati riservati: keccak256(canonico({"dati": …, "sale": …})), oppure null. I dati riservati non stanno nel documento pubblico
allegati	per ogni file: nome, tipo, dimensione, sha256, visibilita
sale	32 byte casuali per evento (esadecimale)

Sono esclusi i campi generati dopo il calcolo dell'impronta o che ne dipendono: impronta, firma, stato, lotto d'ancoraggio, prova, identificativi interni dell'evento, data di registrazione.

2. Forma canonica e impronta

- Forma canonica: RFC 8785 (JSON Canonicalization Scheme). Chiavi ordinate per unità di codice UTF-16, nessuno spazio, numeri come in ECMAScript, stringhe con le sole sequenze di escape obbligatorie. Gli interi fuori dall'intervallo ±(2^53−1) sono rifiutati.
- La stringa canonica è salvata così com'è: è quella che si verifica.
- Impronta: keccak256 dei byte UTF-8 della stringa canonica (Keccak-256 originale, quello delle catene EVM; non SHA3-256).
- Catena: il documento contiene precedente, quindi ogni impronta dipende da tutte le precedenti dello stesso prodotto.

3. Firma (attestazione)

- Una chiave per azienda. Algoritmo ecdsa-p256-sha256: ECDSA su curva P-256, messaggio = i 32 byte dell'impronta dell'evento, hash SHA-256; firma in formato r‖s (64 byte).
- L'attestazione è un'entità separata dall'evento: emittente, chiave usata, tipo di firma, firma.
- Registro delle chiavi pubbliche (passaporto-chiavi/1): per ogni chiave identificativo (keccak256 della chiave pubblica in formato DER SubjectPublicKeyInfo), azienda, algoritmo, chiave pubblica (DER in base64), validità (valida_dal, valida_al), stato (attiva, ruotata, revocata). Le chiavi ruotate restano nel registro.

Verifica con OpenSSL (la firma va prima convertita da r‖s a DER):

base64 -d pubblica.b64 > pubblica.der

openssl pkey -pubin -inform DER -in pubblica.der -out pubblica.pem

openssl dgst -sha256 -verify pubblica.pem -signature firma.der impronta.bin

4. Lotto d'ancoraggio e albero di Merkle

- Foglia di un evento: keccak256(keccak256(impronta ‖ keccak256(firma))). È la «foglia standard» di due valori bytes32 (abi.encode(bytes32, bytes32)): la foglia lega l'evento alla sua firma.
- Foglia del registro delle chiavi: keccak256(keccak256(keccak256("passaporto-chiavi/1") ‖ impronta_del_registro)). In ogni lotto entra anche l'impronta del registro delle chiavi in quel momento: chi verifica controlla la chiave della firma contro un registro ancorato, non contro la piattaforma.
- Albero: foglie ordinate in modo crescente; nodo = keccak256(min(a,b) ‖ max(a,b)) (coppie ordinate); disposizione ad albero binario completo. È la costruzione della libreria @openzeppelin/merkle-tree (StandardMerkleTree.of(valori,

["bytes32","bytes32"])) e le prove sono accettate da MerkleProof.verify di OpenZeppelin. I vettori di prova sono stati confrontati con quella libreria: radice, foglie e prove identiche.
- Prova d'inclusione: l'elenco dei fratelli dal basso verso l'alto. Si risale combinando a coppie ordinate fino alla radice.
- Una transazione per lotto. In catena vanno solo la radice e l'identificativo del lotto.

5. Contratto di registro

RegistroRadici (Solidity 0.8.26, sorgente e ABI nel pacchetto del prodotto):

function ancora(bytes32 radice, bytes32 lotto) external

function ancorataIl(address mittente, bytes32 radice) external view returns (uint64)

event RadiceAncorata(address indexed mittente, bytes32 indexed radice, bytes32 indexed lotto, uint64 quando)

Non ha proprietario, non ha funzioni amministrative, non è aggiornabile (nessun proxy), non riceve fondi. Ogni radice è legata all'indirizzo che l'ha registrata: la verifica chiede al contratto ancorataIl(mittente della piattaforma, radice).

Rete	Catena	Contratto	Mittente (portafoglio operativo)	Esploratore
Polygon Amoy (rete di prova)	80002	0x9a28fEa6e92f1F0A9d7DeeF30377fEDB0cD01FD9	0x56C2Ba3837100b2227B77a3F08EA0FC1e2565C92	https://amoy.polygonscan.com/address/0x9a28fEa6e92f1F0A9d7DeeF30377fEDB0cD01FD9

6. Pacchetto di prova (passaporto-prova/1)

Si scarica da ogni evento del passaporto («Scarica evento e prova») o da GET /e/{id}/prova.json:

{"formato": "passaporto-prova/1",

"evento": {"canonico": "…", "impronta": "0x…", "oscurato": false},

"attestazione": {"tipo_firma": "ecdsa-p256-sha256", "firma": "0x…", "chiave": "0x…", "pubblica": "…"},

"prova": {"foglia": "0x…", "fratelli": ["0x…"], "radice": "0x…"},

"registro_chiavi": {"documento": "…", "impronta": "0x…", "fratelli": ["0x…"]},

"ancoraggio": {"rete": "…", "catena_id": 80002, "contratto": "0x…", "mittente": "0x…", "transazione": "0x…", "blocco": 0, "lotto_in_catena": "0x…"}}

7. Verificare a mano

1. keccak256(UTF-8(evento.canonico)) deve dare evento.impronta.

2. La firma attestazione.firma deve essere valida sull'impronta con attestazione.pubblica; keccak256(DER della chiave) deve dare attestazione.chiave.

3. foglia = keccak256(keccak256(impronta ‖ keccak256(firma))) deve essere prova.foglia.

4. Partendo dalla foglia, per ogni fratello: h = keccak256(min(h, fratello) ‖ max(h, fratello)). Il risultato deve essere prova.radice.

5. Stesso procedimento per il registro delle chiavi: da keccak256(keccak256(etichetta ‖ registro_chiavi.impronta)) con registro_chiavi.fratelli si deve arrivare alla stessa radice; la chiave della firma deve comparire nel documento del registro, intestata all'emittente dell'evento.

6. Sulla rete pubblica: ancorataIl(mittente, radice) sul contratto deve dare un istante diverso da zero; in alternativa si apre la transazione sull'esploratore e si legge l'evento RadiceAncorata.

Strumenti pronti: verificatore in un solo file HTML (https://dactacore.com/verificatore), script a riga di comando senza dipendenze (verifica.py), vettori di prova (https://dactacore.com/verificatore/vettori.json).

8. Stati di un evento

registrato (firmato, non ancora in un lotto) → in_ancoraggio (lotto chiuso, transazione da inviare o in attesa di conferme) → confermato (numero di conferme configurato) → finalizzato (blocco definitivo). Lo stato non è scritto sull'evento: si legge dal lotto che lo contiene. Se la catena si riorganizza il lotto torna in ancoraggio e viene ritrasmesso.

9. Solo in aggiunta

Le tabelle di eventi, attestazioni, foglie dei lotti e registro delle operazioni rifiutano UPDATE, DELETE e TRUNCATE a livello di database (trigger), per qualunque utente applicativo. Unica eccezione dichiarata: l'oscuramento del contenuto di un evento per il diritto alla cancellazione, che svuota i campi di contenuto e lascia impronta, sale, catena e firma.

10. API per i sistemi aziendali

Base: https://dactacore.com/api/v1 · descrizione OpenAPI: https://dactacore.com/api/v1/openapi.json · autenticazione: Authorization: Bearer <token dell'azienda>.

Metodo e percorso	Cosa fa
GET /api/v1/prodotti	elenco dei prodotti dell'azienda
POST /api/v1/prodotti	crea un prodotto (nome, modello, categoria, gtin, descrizione, dati_pubblici, dati_riservati)
POST /api/v1/prodotti/{pid}/lotti	crea un lotto o un numero di serie (codice, seriale, quantita)
POST /api/v1/eventi	registra un evento (prodotto, lotto, tipo, quando, fuso, luogo, nota, attributi, riservati, visibilita, allegati, cessionario_id, rettifica_di)
GET /api/v1/eventi/{id}	l'evento con lo stato dell'ancoraggio
GET /api/v1/eventi/{id}/prova	il pacchetto di prova

- Idempotenza: intestazione Idempotency-Key; la stessa richiesta ripetuta torna lo stesso risultato.
- Errori: {"errore": {"codice": "…", "messaggio": "…", "campo": "…"}} con stato 400, 401, 403, 404, 422, 503.
- Webhook alla conferma dell'ancoraggio: POST all'indirizzo dell'azienda con corpo JSON {"tipo": "ancoraggio.confermato", "lotto", "numero", "radice", "rete", "catena_id", "contratto", "transazione", "blocco", "eventi": [{"id", "impronta"}]}. Firma nell'intestazione X-Passport-Signature: t=<istante>,v1=<HMAC-SHA256 esadecimale di "<istante>.<corpo>" col segreto dell'azienda>. Ritentativi con attesa crescente; dopo otto tentativi la consegna è «fallita» e resta visibile.

Esempio:

curl -X POST https://dactacore.com/api/v1/eventi -H "Authorization: Bearer ppk_…" -H "Idempotency-Key: ddt-2026-000123" \

  -H "Content-Type: application/json" \

  -d '{"prodotto": "7K2M9Q4X8TD3A", "tipo": "spedizione", "quando": "2026-11-12T08:30:00Z", "fuso": "+01:00", "luogo": "Magazzino", "attributi": {"colli": 2}}'

Dokument erstellt am 13.11.2026 04:29 UTC