TAXIDENT

Server Database Design Document

Schema v3.1

February 2026

Zero-Knowledge Encrypted Storage • Anonymous WebAuthn Auth • Per-Record Encryption • Advisor Delegation

Table of Contents

Table of Contents — update in Word/LibreOffice to populate

1. Overview

1.1 Purpose

Taxident is a zero-knowledge encrypted (ZKE) SaaS application for deterministic multi-jurisdiction tax residency analysis. The server stores encrypted user data, authentication credentials, advisor delegation grants, and public reference data. The server never sees plaintext user data.

Following the Proton/Tuta model, each user record is stored as an individually encrypted row. The server retains plaintext structural metadata (record dates, vault types, period labels) to support filtering, pagination, access-control enforcement, and sync cursor management. Encrypted payloads are opaque to the server.

1.2 Design Principles

#	Principle	Description
P1	Zero knowledge	Server never sees plaintext user data.
P2	No PII server-side	No email or username. Identity = WebAuthn credential public key.
P3	Per-record encryption	Each record is one encrypted row (Tuta/Proton model). Replaces monolithic blob-per-vault.
P4	Scoped key hierarchy	Master seed → scope keys per data category → quarterly period keys via HKDF.
P5	Advisor delegation	Period keys wrapped with advisor public keys. Server enforces grant boundaries on plaintext metadata.
P6	Public reference data	Jurisdictions, rulesets, rules, treaties are plaintext published law.
P7	Referential integrity	Lookup tables enforce consistency via FK. Enums use CHECK constraints.

1.3 Schema Version History

Version	Key Changes
v1	Monolithic encrypted blobs per account. Single vault per scope.
v2	Period-scoped vaults, advisor_accounts and advisor_grants tables, vault_types lookup table.
v3	Per-record encryption via encrypted_records table. Added record_history for versioning. Removed vault_chunks.
v3.1	Added countries lookup table (ISO 3166-1). Added crypto_algorithms lookup table. FK migrations from free-text to lookup references. Column renames to _id suffix convention (encryption_algo → encryption_algo_id, wrapping_algo → wrapping_algo_id). Added 'countries' to reference_data_versions CHECK constraint.

1.4 Target Platform

Cloudflare D1 (SQLite-compatible). D1 provides edge-distributed SQLite with read replication and low-latency access. The schema uses SQLite-compatible types, constraints, and index syntax throughout. All query patterns are single-tenant (scoped by account_id), minimising write contention.

2. Key Derivation Model

2.1 Derivation Hierarchy

The following tree shows the full key derivation hierarchy from the master seed:

```
master_seed (BIP39 24-word mnemonic recovery)
→ HKDF("auth") → auth_key_seed (WebAuthn binding)
→ HKDF("scope:presence") → presence_scope_key
→ HKDF(scope_key, "2025-Q1") → period_key_2025_Q1
→ HKDF(scope_key, "2025-Q2") → period_key_2025_Q2
→ HKDF("scope:status_modifiers") → status_modifiers_scope_key
→ ...
→ HKDF("scope:settings") → settings_scope_key (no period derivation)
```

Each vault_type maps to a scope. Temporal scopes derive period keys via HKDF with the period_label as info. Non-temporal scopes (identity, settings) use the scope master key directly.

2.2 Design Decision: Quarterly Granularity

Design Decision: Quarterly Granularity
Quarterly period granularity was chosen over monthly, weekly, or per-entry alternatives. This yields 4 grants per year for advisor delegation, with a maximum overshare of approximately 2 months of data at period boundaries.
The period_label field is an opaque string (e.g. '2025-Q1'). The server does not parse this value. This design allows the client to change granularity in future versions (e.g. monthly) without server-side schema changes.

2.3 Encryption Algorithms

The following algorithms are registered in the crypto_algorithms lookup table:

Algorithm	Type	WebCrypto Native	Notes
AES-256-GCM	symmetric	Yes	WebCrypto native. Strict nonce non-reuse. 96-bit nonce.
XChaCha20-Poly1305	symmetric	No (libsodium)	Safe random nonces. 192-bit nonce.
X25519 + XSalsa20-Poly1305	wrapping	No (libsodium)	NaCl crypto_box. Key wrapping for advisor delegation.
ECDH P-256 + AES-256-GCM	wrapping	Yes	WebCrypto native key agreement + encryption.
HKDF-SHA-256	kdf	Yes	Key derivation. Scope and period key derivation.

2.4 Design Decision: AES-256-GCM Default

Design Decision: AES-256-GCM as Default Encryption
AES-256-GCM is the default encryption algorithm because it is hardware-accelerated via the WebCrypto API, available in all modern browsers without external dependencies.
XChaCha20-Poly1305 is supported as an optional alternative for users who prefer it, but requires the libsodium library.
P-256 ECDH is preferred over X25519 for key agreement for the same reason: native WebCrypto availability without library dependencies.

3. Table Catalogue

3.0 Summary

The server schema contains 19 tables organised into 10 functional groups:

Group	Table	Purpose	Expected Cardinality
Accounts & Auth	accounts	User account anchors	Tens of thousands
Accounts & Auth	webauthn_credentials	WebAuthn public key credentials	1–3 per account
Accounts & Auth	recovery_verifiers	BIP39 recovery verification hashes	1 per account
Accounts & Auth	auth_challenges	Ephemeral WebAuthn challenges	Transient, pruned by TTL
Lookup Tables	countries	ISO 3166-1 country reference	~250 (seeded)
Lookup Tables	vault_types	Data scope categories	10 (seeded)
Lookup Tables	crypto_algorithms	Encryption algorithm registry	5 (seeded)
Encrypted Records	encrypted_records	Per-record encrypted user data	Hundreds per account
Encrypted Records	record_history	Version history for conflict resolution	Low; on-conflict only
Advisor Accounts	advisor_accounts	Advisor profiles with KYC status	Subset of accounts
Advisor Grants	advisor_grants	Period key grants to advisors	4–20 per advisor relationship
Advisor Workspace	advisor_workspace_messages	Encrypted messages (stub)	Not implemented v1
Reference Data	jurisdictions	Tax jurisdictions	~300
Reference Data	rulesets	Versioned rule sets per jurisdiction	1–3 per jurisdiction
Reference Data	rules	Individual residency rules	5–15 per ruleset
Reference Data	rule_parameters	Rule configuration parameters	1–5 per rule
Reference Data	treaties	Bilateral tax treaties	~3,000
Reference Data	treaty_tiebreaker_steps	Treaty tiebreaker test sequence	3–5 per treaty
Ref Versioning	reference_data_versions	Version tracking for ref data sync	5 rows
Sync Metadata	sync_cursors	Per-device sync cursor state	10–50 per account
Rate Limiting	rate_limit_events	Abuse prevention event log	High volume, pruned

3.1 Accounts & Authentication

3.1.1 accounts

Anchor table for all user accounts. Contains no PII. The account_fingerprint is a derived, non-reversible identifier used for deduplication.

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	UUID
account_fingerprint	TEXT	NOT NULL, UNIQUE	Derived identifier for dedup
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	
last_seen_at	TIMESTAMP		Updated on each authenticated request

3.1.2 webauthn_credentials

Stores WebAuthn credential public keys. Each account may have multiple credentials (multi-device). The credential_id is the WebAuthn-standard identifier returned by the authenticator.

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	UUID
account_id	TEXT	NOT NULL, FK → accounts(id)	
credential_id	TEXT	NOT NULL, UNIQUE	WebAuthn credential identifier
public_key	TEXT	NOT NULL	COSE public key (base64)
sign_count	INTEGER	NOT NULL, DEFAULT 0	Authenticator sign counter
transports	TEXT		JSON array: usb, ble, nfc, internal
device_name	TEXT		User-assigned device label
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	
last_used_at	TIMESTAMP		Updated on each authentication

FK: account_id → accounts(id)

3.1.3 recovery_verifiers

Stores the hashed recovery verifier derived from the BIP39 mnemonic. One per account. The verifier_hash is computed as HKDF(master_seed, "recovery-verify") then hashed with argon2id.

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	UUID
account_id	TEXT	NOT NULL, FK → accounts(id), UNIQUE	One verifier per account
verifier_hash	TEXT	NOT NULL	Argon2id hash of recovery verifier
algorithm	TEXT	NOT NULL, DEFAULT 'argon2id'	Hash algorithm identifier

Column	Type	Constraints	Notes
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	

UNIQUE: (account_id)

FK: account_id → accounts(id)

3.1.4 auth_challenges

Ephemeral challenges for WebAuthn registration, authentication, and recovery flows. Pruned by TTL based on expires_at.

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	UUID
challenge	TEXT	NOT NULL, UNIQUE	Random challenge bytes (base64)
challenge_type	TEXT	NOT NULL, CHECK	registration | authentication | recovery
account_id	TEXT		NULL for registration challenges
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	
expires_at	TIMESTAMP	NOT NULL	TTL expiry for housekeeping

CHECK: challenge_type IN ('registration', 'authentication', 'recovery')

3.2 Lookup Tables

3.2.1 countries

Canonical ISO 3166-1 country reference. Added in v3.1. FK target for jurisdictions.country_code, treaties.country_a_code, and treaties.country_b_code. Also synced to the client as a reference data cache.

Column	Type	Constraints	Notes
code	TEXT	PRIMARY KEY	ISO 3166-1 alpha-2 (e.g. 'US', 'PT')
name	TEXT	NOT NULL	Country common name
alpha3	TEXT	NOT NULL, UNIQUE	ISO 3166-1 alpha-3 (e.g. 'USA')
numeric_code	TEXT		ISO 3166-1 numeric (e.g. '840')
region	TEXT		e.g. 'Europe', 'Asia', 'Americas'
sub_region	TEXT		e.g. 'Southern Europe', 'Southeast Asia'
is_eu_member	INTEGER	NOT NULL, DEFAULT 0	Boolean: EU member state
is_oecd_member	INTEGER	NOT NULL, DEFAULT 0	Boolean: OECD member
has_dn_visa	INTEGER	NOT NULL, DEFAULT 0	Boolean: digital nomad visa programme
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	

Column	Type	Constraints	Notes
updated_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	

3.2.2 vault_types

Canonical list of data scope categories. All encrypted_records and sync_cursors reference this table via vault_type_id. The is_temporal flag determines whether period key derivation is applied.

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	Scope identifier (e.g. 'presence')
display_name	TEXT	NOT NULL	Human-readable label
is_temporal	INTEGER	NOT NULL, DEFAULT 1	1 = period-scoped, 0 = not
description	TEXT		Scope purpose description
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	

Seed Data:

ID	Display Name	Temporal	Description
presence	Presence Data	Yes	Travel intervals and family presence
status_modifiers	Status Modifiers	Yes	Visas, tax registrations, domicile, homes, employment
assertions	Assertions	Yes	User assertions for subjective rules
evaluations	Evaluations	Yes	Residency evaluations and rule results
treaty_evals	Treaty Evaluations	Yes	Treaty tiebreaker evaluations
day_counts	Day Counts	Yes	Derived day count cache
risk_alerts	Risk & Projections	Yes	Risk alerts and simulations
audit_log	Audit Log	Yes	Client-side audit trail
identity	Identity	No	Nationalities and family members
settings	Settings	No	User preferences and UI state

3.2.3 crypto_algorithms

Canonical list of encryption and key-wrapping algorithms. Added in v3.1. FK target for encrypted_records.encryption_algo_id, advisor_grants.wrapping_algo_id, and advisor_workspace_messages.encryption_algo_id.

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	e.g. 'aes-256-gcm'
algorithm_type	TEXT	NOT NULL, CHECK	symmetric | asymmetric | wrapping | kdf

Column	Type	Constraints	Notes
display_name	TEXT	NOT NULL	Human-readable name
key_bits	INTEGER		Key length in bits
nonce_bits	INTEGER		Nonce/IV length in bits
webcrypto_native	INTEGER	NOT NULL, DEFAULT 0	1 = available via WebCrypto API
library	TEXT		Required library if not native
notes	TEXT		Implementation notes
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	

CHECK: algorithm_type IN ('symmetric', 'asymmetric', 'wrapping', 'kdf')

3.3 Encrypted Records

3.3.1 encrypted_records

Core table for per-record encrypted user data. Each row is one logical record (e.g. a single travel interval, a single visa record) encrypted with the period key for its (vault_type_id, period_label) scope. The server sees only structural metadata; the encrypted_payload is opaque.

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	UUID
account_id	TEXT	NOT NULL, FK → accounts(id)	
vault_type_id	TEXT	NOT NULL, FK → vault_types(id)	Data scope category
period_label	TEXT		e.g. '2025-Q1'; NULL for non-temporal
record_date	TEXT		Plaintext ISO 8601 date for filtering
encrypted_payload	TEXT	NOT NULL	Base64-encoded ciphertext
nonce	TEXT	NOT NULL	Base64-encoded, unique per record
encryption_algo_id	TEXT	NOT NULL, DEFAULT 'aes-256-gcm', FK → crypto_algorithms(id)	
data_version	INTEGER	NOT NULL, DEFAULT 1	Schema version of plaintext record
size_bytes	INTEGER		Payload size for quota tracking
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	
updated_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	

FK: account_id → accounts(id), vault_type_id → vault_types(id), encryption_algo_id → crypto_algorithms(id)

Design Decision: Per-Record vs Blob Encryption
v1–v2 used monolithic encrypted blobs per vault scope. v3+ uses per-record encryption where each logical record is a separate encrypted row. This enables:
• Server-side filtering and pagination without decryption
• Granular sync (only changed records transferred)
• Grant enforcement at individual record level (via record_date filtering)
• Conflict resolution via record_history without re-encrypting entire vaults

Plaintext Metadata Rationale
record_date is a coarse temporal position (date for presence, month for evaluations). It does not leak record content — only temporal position. This enables server-side filtering for advisor grant enforcement (start_offset) and client sync optimisation. The trade-off is minimal: an observer learns when a record was created, not what it contains.

3.3.2 record_history

Stores previous versions of encrypted records for conflict resolution and rollback. Created when a record is updated. The encrypted_payload and nonce are the previous version's values.

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	UUID
record_id	TEXT	NOT NULL, FK → encrypted_records(id)	Parent record
data_version	INTEGER	NOT NULL	Version number of this snapshot
encrypted_payload	TEXT	NOT NULL	Previous ciphertext
nonce	TEXT	NOT NULL	Previous nonce
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	

FK: record_id → encrypted_records(id)

3.4 Advisor Accounts & KYC

3.4.1 advisor_accounts

Advisor profile linked 1:1 to an accounts row. Contains the advisor's delegation public key (for key wrapping) and KYC verification state. display_name_encrypted is encrypted with the advisor's own key and opaque to the server.

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	UUID
account_id	TEXT	NOT NULL, FK → accounts(id), UNIQUE	1:1 with accounts
delegation_public_key	TEXT	NOT NULL	Public key for key wrapping
kyc_status	TEXT	NOT NULL, DEFAULT 'pending', CHECK	KYC state machine
kyc_verified_at	TIMESTAMP		Set when status = verified
kyc_reference	TEXT		External KYC provider reference
display_name_encrypted	TEXT		Encrypted advisor display name

Column	Type	Constraints	Notes
jurisdiction_tags	TEXT		JSON array of jurisdiction specialisations
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	
updated_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	

CHECK: kyc_status IN ('pending', 'submitted', 'verified', 'rejected', 'suspended')

UNIQUE: (account_id)

KYC State Machine:

pending → submitted → verified | rejected | suspended. Transitions are server-enforced. A rejected advisor may resubmit (rejected → submitted). A suspended advisor's existing grants remain but no new grants are accepted.

3.5 Advisor Grants

3.5.1 advisor_grants

Stores wrapped period keys granted to advisors. Each grant gives an advisor access to one (vault_type_id, period_label) scope. The wrapped_period_key is encrypted with the advisor's delegation public key. The server enforces grant boundaries by filtering encrypted_records before serving ciphertext.

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	UUID
account_id	TEXT	NOT NULL, FK → accounts(id)	Granting user
recipient_type	TEXT	NOT NULL, CHECK	advisor_id | opaque_token
advisor_account_id	TEXT	FK → advisor_accounts(id)	For KYC'd advisors
opaque_token	TEXT		For non-onboarded recipients
vault_type_id	TEXT	NOT NULL, FK → vault_types(id)	Data scope
period_label	TEXT	NOT NULL	Scoped period (e.g. '2025-Q1')
wrapped_period_key	TEXT	NOT NULL	Period key encrypted with advisor pubkey
wrapping_algo_id	TEXT	NOT NULL, DEFAULT 'x25519-xsalsa20-poly1305', FK → crypto_algorithms(id)	
start_offset	TEXT		ISO 8601 date; server-enforced filter
granted_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	
expires_at	TIMESTAMP		Optional grant expiry
tombstoned_at	TIMESTAMP		Set on revocation

CHECK: recipient_type IN ('advisor_id', 'opaque_token')

CHECK: (recipient_type = 'advisor_id' AND advisor_account_id IS NOT NULL AND opaque_token IS NULL) OR (recipient_type = 'opaque_token' AND opaque_token IS NOT NULL AND advisor_account_id IS NULL)

Partial Unique Indexes:

idx_grants_active_advisor: UNIQUE(account_id, advisor_account_id, vault_type_id, period_label) WHERE advisor_account_id IS NOT NULL AND tombstoned_at IS NULL

idx_grants_active_token: UNIQUE(account_id, opaque_token, vault_type_id, period_label) WHERE opaque_token IS NOT NULL AND tombstoned_at IS NULL

Grant Enforcement Query
The server enforces grants by filtering encrypted_records with the following WHERE clause:
WHERE vault_type_id = grant.vault_type_id
AND period_label = grant.period_label
AND (grant.start_offset IS NULL OR record_date >= grant.start_offset)
AND grant.tombstoned_at IS NULL
AND (grant.expires_at IS NULL OR grant.expires_at > NOW())

Tombstone Revocation
Revocation sets tombstoned_at. This immediately stops the server from serving new ciphertext to the advisor. However, any data the advisor has already received and decrypted cannot be un-disclosed — you cannot unring the bell. Tombstoning cuts off future access only.

start_offset Is Access Control, Not Cryptographic
start_offset is a server-side filter on record_date. It is not a cryptographic boundary — the advisor holds the full period key and could theoretically decrypt any record in that period if they received the ciphertext. The server withholds records before start_offset, so the advisor never receives that ciphertext.
Residual risk: a compromised server could serve records before start_offset. Mitigation: the advisor is a KYC'd party, the overshare is at most ~2 extra months within the same quarter, and the content is still encrypted tax data. Severity: low.

3.6 Advisor Workspace

3.6.1 advisor_workspace_messages

Stub table for encrypted communication between users and advisors within a grant context. Not implemented in v1. Schema is defined for forward compatibility.

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	UUID
grant_id	TEXT	NOT NULL, FK → advisor_grants(id)	Parent grant
sender_type	TEXT	NOT NULL, CHECK	user | advisor
encrypted_payload	TEXT	NOT NULL	Encrypted message content
nonce	TEXT	NOT NULL	Unique nonce
encryption_algo_id	TEXT	NOT NULL, DEFAULT 'aes-256-gcm', FK → crypto_algorithms(id)	
payload_type	TEXT	NOT NULL, DEFAULT 'message', CHECK	message | document | annotation
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	

CHECK: sender_type IN ('user', 'advisor')

CHECK: payload_type IN ('message', 'document', 'annotation')

3.7 Public Reference Data

All reference data tables contain plaintext published law. This data is not user-specific and is not encrypted.

3.7.1 jurisdictions

Tax jurisdictions. Each jurisdiction belongs to a country (FK to countries table, added in v3.1). Sub-regions support sub-national jurisdictions (e.g. US states, Canadian provinces).

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	e.g. 'PT', 'US-CA'
country_code	TEXT	NOT NULL, FK → countries(code)	ISO 3166-1 alpha-2
country_name	TEXT	NOT NULL	Denormalised for convenience
sub_region	TEXT		State/province for sub-national
tax_year_type	TEXT	NOT NULL, DEFAULT 'calendar', CHECK	calendar | fiscal | custom
tax_year_start	TEXT		e.g. '04-06' for UK fiscal year
notes	TEXT		

CHECK: tax_year_type IN ('calendar', 'fiscal', 'custom')

FK: country_code → countries(code)

3.7.2 rulesets

Versioned sets of residency rules per jurisdiction. Each ruleset has an effective date range and a status lifecycle: draft → active → superseded.

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	UUID
jurisdiction_id	TEXT	NOT NULL, FK → jurisdictions(id)	
version	INTEGER	NOT NULL	Monotonic version number
effective_from	DATE	NOT NULL	Start of applicability
effective_to	DATE		NULL if currently active
status	TEXT	NOT NULL, DEFAULT 'draft', CHECK	draft | active | superseded
author	TEXT		Rule author/contributor
change_notes	TEXT		Changelog for this version
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	

CHECK: status IN ('draft', 'active', 'superseded')

UNIQUE: (jurisdiction_id, version)

3.7.3 rules

Individual residency rules within a ruleset. Each rule has a determination_type that classifies how it is evaluated:

- mechanical — Fully deterministic from data (e.g. day counts). No user input needed.
- structured_subjective — Requires user assertions but has a structured evaluation (e.g. centre of vital interests with enumerated factors).
- irreducibly_subjective — Requires professional judgment; cannot be mechanically resolved (e.g. mutual agreement procedure).

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	UUID
ruleset_id	TEXT	NOT NULL, FK → rulesets(id)	
rule_code	TEXT	NOT NULL	Short code (e.g. 'SRT-183')
rule_name	TEXT	NOT NULL	Human-readable rule name
description	TEXT		Detailed rule description
determination_type	TEXT	NOT NULL, CHECK	mechanical | structured_subjective | irreducibly_subjective
authority_type	TEXT	NOT NULL, CHECK	statute | regulation | admin_guidance | case_law
authority_reference	TEXT		Legal citation
evaluation_order	INTEGER	NOT NULL	Order within ruleset
rule_logic_ref	TEXT	NOT NULL	Client-side logic function reference
output_template	TEXT		Template for evaluation output
notes	TEXT		
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	

CHECK: determination_type IN ('mechanical', 'structured_subjective', 'irreducibly_subjective')

CHECK: authority_type IN ('statute', 'regulation', 'admin_guidance', 'case_law')

UNIQUE: (ruleset_id, rule_code)

3.7.4 rule_parameters

Configurable parameters for rules (e.g. day count thresholds, lookback periods). Each parameter is a typed key-value pair.

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	UUID
rule_id	TEXT	NOT NULL, FK → rules(id)	
param_key	TEXT	NOT NULL	Parameter name (e.g. 'threshold_days')
param_value	TEXT	NOT NULL	Value as string
param_type	TEXT	NOT NULL, CHECK	integer | boolean | text | date | decimal

Column	Type	Constraints	Notes
description	TEXT		Parameter purpose

CHECK: param_type IN ('integer', 'boolean', 'text', 'date', 'decimal')

UNIQUE: (rule_id, param_key)

3.7.5 treaties

Bilateral tax treaties between country pairs. Both country codes FK to the countries table (added in v3.1). The saving_clause flags indicate whether each country's saving clause overrides treaty benefits for its own citizens.

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	UUID
country_a_code	TEXT	NOT NULL, FK → countries(code)	First treaty party
country_b_code	TEXT	NOT NULL, FK → countries(code)	Second treaty party
treaty_name	TEXT	NOT NULL	Official treaty title
effective_from	DATE	NOT NULL	Treaty effective date
effective_to	DATE		NULL if currently in force
country_a_saving_clause	INTEGER	NOT NULL, DEFAULT 0	Boolean
country_b_saving_clause	INTEGER	NOT NULL, DEFAULT 0	Boolean
treaty_reference	TEXT		Official publication reference
notes	TEXT		
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	

UNIQUE: (country_a_code, country_b_code, effective_from)

FK: country_a_code → countries(code), country_b_code → countries(code)

3.7.6 treaty_tiebreaker_steps

Ordered sequence of tiebreaker tests for treaty dispute resolution. Based on OECD Model Tax Convention Article 4. Each step has a test_name from the canonical set and a determination_type indicating how it is evaluated.

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	UUID
treaty_id	TEXT	NOT NULL, FK → treaties(id)	
step_order	INTEGER	NOT NULL	Evaluation sequence (1-based)
test_name	TEXT	NOT NULL, CHECK	See CHECK values below
determination_type	TEXT	NOT NULL, CHECK	mechanical | structured_subjective | irreducibly_subjective
rule_logic_ref	TEXT		Client-side logic function reference
description	TEXT		Test description

CHECK: test_name IN ('permanent_home', 'vital_interests', 'habitual_abode', 'nationality', 'mutual_agreement')

CHECK: determination_type IN ('mechanical', 'structured_subjective', 'irreducibly_subjective')

UNIQUE: (treaty_id, step_order)

3.8 Reference Data Versioning

3.8.1 reference_data_versions

Tracks the current version of each reference data type. Clients check this table during sync to determine whether their cached reference data is stale. The 'countries' data type was added in v3.1.

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	UUID
data_type	TEXT	NOT NULL, UNIQUE, CHECK	countries | jurisdictions | rulesets | rules | treaties
current_version	INTEGER	NOT NULL, DEFAULT 1	Monotonic version counter
updated_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	

CHECK: data_type IN ('countries', 'jurisdictions', 'rulesets', 'rules', 'treaties')

3.9 Sync Metadata

3.9.1 sync_cursors

Tracks per-device sync state for each (vault_type, period_label) combination. The client sends its last_version and the server returns records updated since that version. Non-temporal vaults use NULL period_label.

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	UUID
account_id	TEXT	NOT NULL, FK → accounts(id)	
device_id	TEXT	NOT NULL	Client-generated device UUID
vault_type_id	TEXT	NOT NULL, FK → vault_types(id)	Data scope
period_label	TEXT		NULL for non-temporal vaults
last_synced_at	TIMESTAMP	NOT NULL	Last successful sync time
last_version	INTEGER	NOT NULL, DEFAULT 0	Cursor position

UNIQUE: (account_id, device_id, vault_type_id, period_label)

3.10 Rate Limiting

3.10.1 rate_limit_events

Event log for rate limiting and abuse prevention. account_id is NULL for unauthenticated requests (e.g. registration attempts). ip_hash is a one-way hash of the client IP for privacy.

Column	Type	Constraints	Notes
id	TEXT	PRIMARY KEY	UUID

Column	Type	Constraints	Notes
account_id	TEXT		NULL for unauthenticated requests
ip_hash	TEXT		One-way hash of client IP
event_type	TEXT	NOT NULL, CHECK	auth_attempt | vault_write | recovery_attempt | api_call
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	

CHECK: event_type IN ('auth_attempt', 'vault_write', 'recovery_attempt', 'api_call')

4. Indexes

All indexes defined in the schema, grouped by functional area.

4.1 Authentication

Index Name	Table	Columns	Partial WHERE	Purpose
idx_webauthn_account	webauthn_credentials	account_id	—	Look up credentials by account
idx_webauthn_credential_id	webauthn_credentials	credential_id	—	WebAuthn authentication lookup
idx_challenges_expires	auth_challenges	expires_at	—	TTL pruning of expired challenges

4.2 Lookup Tables

Index Name	Table	Columns	Partial WHERE	Purpose
idx_countries_region	countries	region	—	Filter countries by region
idx_countries_alpha3	countries	alpha3	—	Lookup by alpha-3 code
idx_vault_types_temporal	vault_types	is_temporal	—	Filter temporal vs non-temporal scopes
idx_crypto_algo_type	crypto_algorithms	algorithm_type	—	Filter by algorithm category

4.3 Encrypted Records

Index Name	Table	Columns	Partial WHERE	Purpose
idx_records_account	encrypted_records	account_id	—	All records for account
idx_records_account_type	encrypted_records	account_id, vault_type_id	—	Records by scope
idx_records_account_type_period	encrypted_records	account_id, vault_type_id, period_label	—	Records by scope + period

Index Name	Table	Columns	Partial WHERE	Purpose
idx_records_account_type_date	encrypted_records	account_id, vault_type_id, record_date	—	Records by scope + date (grant enforcement)
idx_records_updated	encrypted_records	account_id, updated_at	—	Sync: records changed since cursor
idx_records_algo	encrypted_records	encryption_algo_id	—	Algorithm usage tracking
idx_record_history_record	record_history	record_id, data_version	—	Version history lookup

4.4 Advisor

Index Name	Table	Columns	Partial WHERE	Purpose
idx_advisor_account	advisor_accounts	account_id	—	Lookup advisor by account
idx_advisor_kyc_status	advisor_accounts	kyc_status	—	Filter by KYC state
idx_grants_account	advisor_grants	account_id	—	All grants for account
idx_grants_advisor	advisor_grants	advisor_account_id	advisor_account_id IS NOT NULL	Grants for specific advisor
idx_grants_token	advisor_grants	opaque_token	opaque_token IS NOT NULL	Grants by opaque token
idx_grants_period	advisor_grants	account_id, vault_type_id, period_label	—	Grants by scope + period
idx_grants_wrapping_algo	advisor_grants	wrapping_algo_id	—	Algorithm usage tracking
idx_grants_active_advisor	advisor_grants	account_id, advisor_account_id, vault_type_id, period_label	advisor_account_id IS NOT NULL AND tombstoned_at IS NULL	UNIQUE: one active grant per advisor per scope
idx_grants_active_token	advisor_grants	account_id, opaque_token, vault_type_id, period_label	opaque_token IS NOT NULL AND tombstoned_at IS NULL	UNIQUE: one active grant per token per scope
idx_workspace_grant	advisor_workspace_messages	grant_id, created_at	—	Messages within a grant
idx_workspace_algo	advisor_workspace_messages	encryption_algo_id	—	Algorithm usage tracking

4.5 Sync

Index Name	Table	Columns	Partial WHERE	Purpose
idx_sync_cursors_account	sync_cursors	account_id, device_id	—	Cursors for account + device

4.6 Reference Data

Index Name	Table	Columns	Partial WHERE	Purpose
idx_jurisdictions_country	jurisdictions	country_code	—	Jurisdictions by country
idx_rulesets_jurisdiction	rulesets	jurisdiction_id	—	Rulesets for jurisdiction
idx_rules_ruleset	rules	ruleset_id, evaluation_order	—	Rules in evaluation order
idx_treaties_country_a	treaties	country_a_code	—	Treaties involving country A
idx_treaties_country_b	treaties	country_b_code	—	Treaties involving country B
idx_treaty_steps	treaty_tiebreaker_steps	treaty_id, step_order	—	Steps in tiebreaker order

4.7 Rate Limiting

Index Name	Table	Columns	Partial WHERE	Purpose
idx_rate_limit_account	rate_limit_events	account_id, event_type, created_at	—	Rate check by account + event type
idx_rate_limit_ip	rate_limit_events	ip_hash, event_type, created_at	—	Rate check by IP + event type

5. Relationships & Entity Model

5.1 Foreign Key Summary

Every foreign key relationship defined in the schema:

Source Table.Column	Target Table.Column	Cardinality
webauthn_credentials.account_id	accounts.id	Many-to-one
recovery_verifiers.account_id	accounts.id	One-to-one
encrypted_records.account_id	accounts.id	Many-to-one
encrypted_records.vault_type_id	vault_types.id	Many-to-one
encrypted_records.encryption_algo_id	crypto_algorithms.id	Many-to-one
record_history.record_id	encrypted_records.id	Many-to-one
advisor_accounts.account_id	accounts.id	One-to-one
advisor_grants.account_id	accounts.id	Many-to-one
advisor_grants.advisor_account_id	advisor_accounts.id	Many-to-one
advisor_grants.vault_type_id	vault_types.id	Many-to-one
advisor_grants.wrapping_algo_id	crypto_algorithms.id	Many-to-one
advisor_workspace_messages.grant_id	advisor_grants.id	Many-to-one

Source Table.Column	Target Table.Column	Cardinality
advisor_workspace_messages.encryption_algo_id	crypto_algorithms.id	Many-to-one
jurisdictions.country_code	countries.code	Many-to-one
rulesets.jurisdiction_id	jurisdictions.id	Many-to-one
rules.ruleset_id	rulesets.id	Many-to-one
rule_parameters.rule_id	rules.id	Many-to-one
treaties.country_a_code	countries.code	Many-to-one
treaties.country_b_code	countries.code	Many-to-one
treaty_tiebreaker_steps.treaty_id	treaties.id	Many-to-one
sync_cursors.account_id	accounts.id	Many-to-one
sync_cursors.vault_type_id	vault_types.id	Many-to-one

5.2 Three Zones

The schema is organised into three logical zones:

Auth Zone: accounts, webauthn_credentials, recovery_verifiers, auth_challenges. This zone handles identity establishment and authentication. The accounts table is the root anchor for the entire schema.

Encrypted Zone: encrypted_records, record_history, advisor_accounts, advisor_grants, advisor_workspace_messages. This zone handles encrypted user data and advisor delegation. Connected to the Auth zone via accounts.id (encrypted_records.account_id, advisor_accounts.account_id, advisor_grants.account_id). Connected to Lookup zone via vault_types.id and crypto_algorithms.id.

Reference Zone: countries, jurisdictions, rulesets, rules, rule_parameters, treaties, treaty_tiebreaker_steps, reference_data_versions. This zone is entirely plaintext published law. No connection to user data. Synced to clients as read-only reference data.

Cross-zone bridges:

- accounts → encrypted_records: The account_id FK bridges Auth to Encrypted zone.
- accounts → advisor_accounts: The account_id FK bridges Auth to Advisor subsystem within the Encrypted zone.
- vault_types and crypto_algorithms: Lookup tables are shared across Encrypted zone tables (encrypted_records, advisor_grants, advisor_workspace_messages, sync_cursors).
- sync_cursors bridges Auth and Encrypted zones for per-device sync state.
- rate_limit_events loosely connects to Auth zone via optional account_id.
- Reference zone has no FK connections to Auth or Encrypted zones. The client joins reference data with decrypted user data locally.

6. Advisor Delegation Flow

This section walks through the full lifecycle of advisor delegation, from grant creation through data access to revocation.

6.1 Grant Creation (Server-Mediated)

For KYC-verified advisors with an advisor_accounts record:

- User selects advisor by advisor_account_id and chooses scope (vault_type_id) and period (period_label).
- Client derives the period key: HKDF(scope_master_key, period_label).
- Client retrieves advisor's delegation_public_key from advisor_accounts.
- Client wraps the period key with the advisor's public key using the wrapping algorithm (default: X25519 + XSalsa20-Poly1305).
- Client sends wrapped_period_key, vault_type_id, period_label, and optional start_offset to the server.
- Server creates advisor_grants row with recipient_type = 'advisor_id'. The partial unique index prevents duplicate active grants for the same scope.

6.2 Grant Creation (Peer-to-Peer via Opaque Token)

For non-onboarded recipients who do not yet have an advisor_accounts record:

- User generates an opaque_token (random string) and shares it out-of-band with the recipient.
- Client wraps the period key with a key derived from the opaque token (or a pre-exchanged public key).
- Server creates advisor_grants row with recipient_type = 'opaque_token'. The opaque_token is stored for grant lookup.
- Recipient redeems the token to retrieve the wrapped period key. If the recipient later completes KYC onboarding, the grant can be migrated to an advisor_id grant.

6.3 Data Access by Advisor

When an advisor requests data for a granted scope:

- Advisor authenticates via WebAuthn (they have their own accounts row).
- Server looks up active grants: advisor_grants WHERE advisor_account_id = ? AND tombstoned_at IS NULL AND (expires_at IS NULL OR expires_at > NOW()).
- For each matching grant, server filters encrypted_records: WHERE account_id = grant.account_id AND vault_type_id = grant.vault_type_id AND period_label = grant.period_label AND (grant.start_offset IS NULL OR record_date >= grant.start_offset).
- Server returns the filtered ciphertext rows plus the wrapped_period_key from the grant.
- Advisor unwraps the period key with their private key, then decrypts each record's encrypted_payload using the period key and per-record nonce.

6.4 Rolling Window Management

For ongoing advisory relationships, the client manages a rolling window of grants:

- At the start of each new quarter, the client automatically derives the next period key.
- Client creates a new grant for the new period, wrapping the new period key with the advisor's public key.
- Optionally, the client tombstones the oldest grant in the rolling window to maintain a fixed window size (e.g. 4 quarters = ~1 year of access).
- The _advisor_grants_local table on the client tracks rolling_window and window_periods for automation.

6.5 Revocation

To revoke an advisor's access:

- Client sends a revocation request specifying the grant ID.
- Server sets tombstoned_at = NOW() on the grant.
- Immediately, the server stops serving ciphertext for that grant. The partial unique index allows a new grant to be created for the same scope if needed.
- Previously decrypted data cannot be reclaimed. Revocation is forward-only.

7. Sync Protocol

The sync protocol enables multi-device access to encrypted records using cursor-based synchronisation via the sync_cursors table.

7.1 Sync Flow

- Client connects and authenticates via WebAuthn. Each device has a unique device_id (generated on first run, stored in client _device table).
- Client sends its last_version per (vault_type_id, period_label) combination from its local _sync_state table.
- Server looks up the corresponding sync_cursors row for (account_id, device_id, vault_type_id, period_label).
- Server returns all encrypted_records updated since the cursor's last_version (WHERE updated_at > cursor timestamp or data_version > cursor version).
- Client receives ciphertext rows, decrypts each with the appropriate period key (derived from scope_master_key + period_label), and merges into the local SQLite database.
- Client updates local _sync_state. Server updates sync_cursors.last_version and last_synced_at.

7.2 Non-Temporal Vaults

Non-temporal vaults (identity, settings) use NULL period_label in both sync_cursors and encrypted_records. The scope master key is used directly for encryption without period key derivation. Sync follows the same cursor pattern.

7.3 Reference Data Sync

Reference data (countries, jurisdictions, rulesets, rules, treaties) is synced separately:

- Client checks reference_data_versions to compare its cached_version (in _reference_cache) against server's current_version.
- If stale, client fetches the full updated dataset for that data_type.
- Client replaces its local reference data cache and updates _reference_cache.cached_version.
- Reference data is plaintext and not encrypted. It is the same for all users.

8. Security Considerations

8.1 Threat Model

The server is assumed to be potentially compromisable. The zero-knowledge encryption architecture ensures that even a fully compromised server exposes only ciphertext and structural metadata. Key threat actors:

- Compromised server: Attacker gains database access. Can see all ciphertext, structural metadata, reference data. Cannot see plaintext user data, key material, or user identity beyond account_fingerprint.
- Malicious advisor: KYC'd advisor with granted access. Can decrypt records within grant scope. Cannot access records outside their grant boundaries (vault_type + period_label + start_offset). Revocation via tombstoning cuts off future access.
- Intercepted traffic: TLS protects data in transit. Even if intercepted, payloads are encrypted. Nonces prevent replay attacks against the encryption layer.

8.2 Plaintext Surface

What the server CAN see:

- account_fingerprint (non-reversible identifier)
- WebAuthn credential public keys and sign counts
- Recovery verifier hash (argon2id)
- vault_type_id, period_label, record_date on encrypted_records
- data_version, size_bytes, created_at, updated_at timestamps
- Advisor KYC status, delegation public keys, jurisdiction_tags
- Grant metadata: vault_type_id, period_label, start_offset, timestamps
- All reference data (public law, not user data)
- Rate limit events: ip_hash, event_type, timestamps

What the server CANNOT see:

- Encrypted payload content (travel data, visa records, evaluations, etc.)
- User identity (no email, no username, no PII)
- Record meaning (the server knows a record exists in 'presence' scope for '2025-Q1' but not which country or dates)
- Key material (master seed, scope keys, period keys never leave the client)
- Advisor display names (display_name_encrypted is opaque to server)
- Workspace message content (encrypted with grant-scoped keys)

8.3 Nonce Management

Each encrypted record has a unique nonce stored alongside the ciphertext:

- AES-256-GCM: 96-bit (12 byte) nonce. Strict non-reuse requirement. Client generates a cryptographically random nonce for each record. With 96-bit nonces, the birthday bound collision probability reaches 50% at ~2^48 encryptions under a single key. With quarterly key rotation and per-scope keys, each key encrypts at most a few thousand records, well within safe bounds.
- XChaCha20-Poly1305: 192-bit (24 byte) nonce. Safe to generate randomly without collision tracking. The birthday bound is ~2^96, making random nonce collision negligible even at massive scale.

8.4 Recovery Flow

Account recovery uses the BIP39 24-word mnemonic:

- User enters mnemonic. Client derives master_seed from BIP39.
- Client derives recovery verifier: HKDF(master_seed, "recovery-verify").
- Client hashes the verifier with argon2id and sends the hash to the server.
- Server compares against recovery_verifiers.verifier_hash for the account.
- On match, client re-derives all scope keys from the master seed: HKDF("scope:") for each vault_type.
- Client registers a new WebAuthn credential for the recovery device.
- All encrypted data is accessible again because keys are deterministically derived from the same master seed.

9. Operational Notes

9.1 Housekeeping

- auth_challenges: TTL pruning based on expires_at. Recommended: run a scheduled job to DELETE WHERE expires_at < NOW() at least hourly.
- rate_limit_events: Retention policy of 30 days. DELETE WHERE created_at < NOW() - INTERVAL 30 DAYS. Events older than the retention window are not useful for rate limiting and consume storage.
- record_history: Retention policy should be defined per deployment. Options: keep last N versions, keep for M days, or keep indefinitely. Recommended: keep last 5 versions per record, prune older entries periodically.

9.2 Scaling

Cloudflare D1 is SQLite-based with the following scaling characteristics:

- Single-tenant query patterns: All queries are scoped by account_id, meaning cross-account contention is minimal.
- D1 read replication: Read queries are served from edge replicas, providing low-latency reads globally.
- Low write contention: Each user writes to their own records. Concurrent writes to different accounts do not conflict.
- SQLite limitations: No concurrent writers (WAL mode mitigates this for D1). Large reference data updates should be batched.

9.3 Migration Path

Schema migrations follow a forward-only ALTER TABLE approach:

- New columns are added with DEFAULT values to avoid breaking existing queries.
- Column renames (e.g. encryption_algo to encryption_algo_id in v3.1) require creating the new column, copying data, and dropping the old column in SQLite.
- The per-record encryption model means no bulk re-encryption is needed on schema changes. Each record's ciphertext is independent and can be individually migrated if the plaintext schema version changes (tracked via data_version).
- reference_data_versions.current_version is incremented when reference data is updated, triggering client re-sync of affected data types.