

Carmentis — Documentation fonctionnelle

Oct 30, 2026

Synthèse

Carmentis est une blockchain de preuve : elle ancre des données structurées, signées, horodatées et chiffrées par canal, puis permet de prouver l'authenticité de tout ou partie de ces données sans révéler le reste. Le réseau tourne sur CometBFT avec une application ABCI en TypeScript ; les entreprises s'y branchent via un serveur Opérateur, les utilisateurs via un wallet (Desk).

Ce document décrit les fonctionnalités telles qu'elles existent dans le code public au 30 octobre 2026 (**SDK core 1.11.4**, **Opérateur 1.16.1**, **nœud 1.6.28**, **carmentis-indexer 1.0.12**, **carmentis-incentive 0.0.5**), dans la documentation officielle docs.carmentis.io et dans la spécification protocolaire (v2, juin 2025).

Ce que Carmentis fait, en cinq verbes:

Verbe	Fonction Carmentis	État
S'authentifier	Signature d'un challenge par la clé du wallet (Desk), via un relais chiffré	En production
S'identifier	Ancrage et résolution de SD-JWT VC / VP, certificats X.509 (PEM ou chaîne x5c)	En production
Signer	Signature de JSON canonique (ou JWS) par un wallet, ancrée sous <code>__signature__</code> et vérifiée à la résolution	En production
Payer	Comptes en token CMTS : transferts, vesting, séquestre, staking. Tokenized Account en EUR en cours de développement	En production
Prouver	Preuves Merkle d'authenticité sélective, preuves de résolution, Proof Checker en ligne	En production

Trois différenciateurs ressortent du projet : un registre découpé en **blockchains virtuelles** (une par objet métier), une **confidentialité native par canal** avec divulgation sélective, et une **cryptographie agile** incluant le post-quantique (par exemple ML-DSA-65 ou ML-KEM-768).

Concepts fondamentaux

Une seule blockchain physique porte une multitude de **blockchains virtuelles** (VB), comme une machine physique porte des machines virtuelles. Chaque objet métier (un compte, une organisation, un dossier client) a sa propre VB, ce qui rend l'historique de chaque objet lisible et vérifiable indépendamment.

Blockchain virtuelle

- **Identifiant** : le hash du premier microbloc. Le « hash précédent » du premier bloc est remplacé par un identifiant de genèse (incluant une seed aléatoire, un horodatage et une date d'expiration).
- **Expiration** : chaque VB peut être conservée sans limite ou pour un nombre de jours donné ; la durée de rétention entre dans le calcul des frais.
- **Six types de VB** sont définis dans le SDK :

Type	Rôle	Visibilité
Protocol	Paramètres globaux du protocole et leurs versions successives	Publique
Account	Compte en CMTS : clé publique, solde, transferts, staking	Publique
Validator node	Déclaration d'un nœud valideur, sa clé CometBFT, son endpoint RPC, son approbation	Publique
Organization	Entreprise inscrite : nom, ville, pays, site web, compte rattaché	Publique
Application	Application déclarée par une organisation : nom, logo, description	Publique
Application ledger	Une instance d'application (un dossier, une transaction) : acteurs, canaux, données	Publique et/ou chiffrée

Microbloc

Une VB est une chaîne de **microbloccs**. Tous partagent la même structure : un en-tête puis des sections.

- **En-tête** : version de protocole, hauteur dans la VB, hash du microbloc précédent, horodatage, gas consommé, prix du gas, payeur des frais.
- **Contrôle d'horodatage** : un microbloc est rejeté s'il a plus de 300 s de retard ou plus de 120 s d'avance sur l'horloge du nœud.
- **Scellement** : le microbloc se termine par une section de signature qui couvre tout le contenu.

Sections

La section est l'unité de contenu typée. Le SDK en définit une trentaine, regroupées par type de VB ; les plus structurantes :

Famille	Sections
Génériques	SIGNATURE, AUXILIARY_SIGNATURE, ALLOWED_ADDITIONAL_WRITER, CUSTOM

Famille	Sections
Protocol	PROTOCOL_CREATION, PROTOCOL_UPDATE, PROTOCOL_VARIABLES
Account	PUBLIC_KEY, TOKEN_ISSUANCE, CREATION, TRANSFER, VESTING_TRANSFER, ESCROW_TRANSFER, ESCROW_SETTLEMENT, STAKE, UNSTAKE
Validator node	VN_CREATION, COMETBFT_PUBLIC_KEY_DECLARATION, RPC_ENDPOINT, APPROVAL, SLASHING_CANCELLATION
Organization / Application	ORG_CREATION, ORG_DESCRIPTION, APP_CREATION, APP_DESCRIPTION
Application ledger	ALLOWED_SIG_SCHEMES, ALLOWED_PKE_SCHEMES, CREATION, ACTOR_CREATION, CHANNEL_CREATION, SHARED_SECRET, CHANNEL_INVITATION, ACTOR_SUBSCRIPTION, PUBLIC_CHANNEL_DATA, PRIVATE_CHANNEL_DATA, AUTHOR, ENDORSEMENT_REQUEST

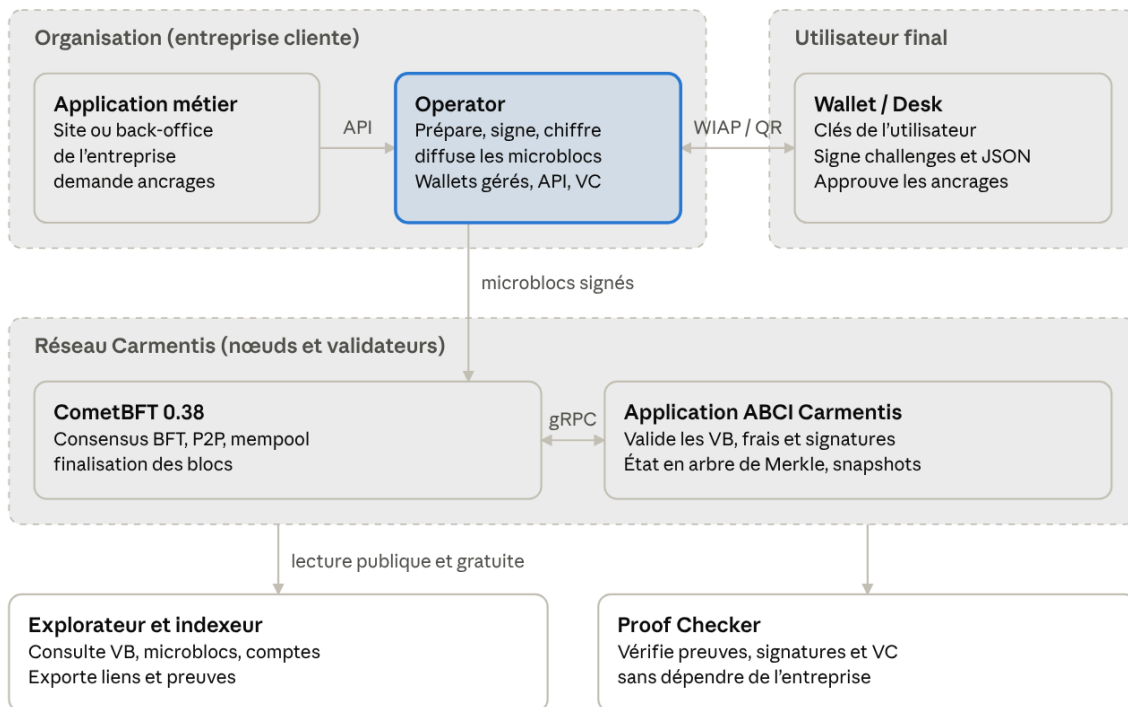
La section **CUSTOM** permet d'ancrer du JSON libre sur les VB publiques (organisation, application, compte) : c'est elle qui porte aujourd'hui les certificats X.509 et les objets « taggés » (voir Preuves).

Acteurs et canaux

Dans une application ledger, les participants sont des **acteurs** nommés (ex. « vendeur », « acheteur », « notaire »), de type Wallet ou Opérateur. Les données sont rangées dans des **canaux**, publics ou privés. Un canal privé n'est lisible que par les acteurs qui y ont été invités (détail en section Confidentialité).

Architecture de l'écosystème

L'Operator relie l'entreprise, le wallet et le réseau ; tout le monde peut vérifier



L'entreprise opérant un service ne parle jamais directement à la chaîne : son application passe par l'Opérateur, qui obtient l'accord de l'utilisateur via son wallet puis diffuse le microbloc aux nœuds. La vérification, elle, ne demande l'accord de personne.

- **Nœuds et validateurs** : chaque nœud associe CometBFT (consensus) et l'application ABCI Carmentis (règles métier), reliés en gRPC. Les validateurs sont déclarés dans leur propre VB, approuvés, et peuvent être pénalisés (slashing).
- **Gouvernance protocolaire** : les paramètres globaux (versions des formules de frais et des règles de mise à jour d'état) vivent dans la VB Protocol, contrôlée par une organisation. Changer une règle passe par une section PROTOCOL_UPDATE versionnée, avec journal des modifications.
- **Opérateur** : serveur déployé par chaque entreprise souhaitant opérer un service. Il signe en tant qu'auteur de microbloc, paie les frais, chiffre les canaux et distribue les clés.
- **Explorateur, indexeur et Proof Checker** : outils de lecture ; un tiers peut développer les siens, ce qui évite toute dépendance à Carmentis pour vérifier.

Identité et comptes

L'identité sur Carmentis est **centrée sur la clé publique** : un utilisateur est identifié par la clé qui signe, et tout le reste (organisation, certificat, VC) est rattaché à cette clé. La hiérarchie se construit comme suit :

1. **Compte** (Account VB) : porte une clé publique et un solde en CMTS. C'est le payeur des frais.
2. **Organisation** (Organization VB) : créée à partir d'un compte ; décrit l'entreprise (nom, ville, code pays, site web).
3. **Application** (Application VB) : déclarée par une organisation ; décrit un service (nom, logo, page d'accueil, description).
4. **Application ledger** : chaque usage concret d'une application ouvre une VB dédiée, dans laquelle les acteurs s'abonnent avec leurs propres clés, sous contrôle des utilisateurs déjà présents sur le VB d'application ledger.

Cette chaîne permet à un vérificateur de remonter d'une donnée ancrée à l'application, puis à l'organisation qui l'opère, puis au compte qui l'a payée.

Desk

Carmentis Desk est une application de bureau (Linux, Windows, macOS publiée sur [GitHub](#)) dont la fonction première est de faciliter la gestion de wallets. Un wallet dans l'écosystème Carmentis est un couple (clé privée de signature, seed). Deux variantes existent :

- **Wallet à seed** : les clés sont dérivées de façon déterministe d'une phrase de 12 mots (BIP39). La seed est stockée chiffrée par une clé dérivée du mot de passe (PBKDF2 ou scrypt).
- **Wallet à fournisseur** : la clé privée est générée et gardée par un tiers (Desk, Opérateur pour un stockage local ou stocker sur un serveur tiers comme un KMS).

Pour chaque application ledger, le wallet dérive une **paire de clés distincte** (HKDF sur la seed et l'identifiant de genèse de la VB). Un observateur externe ne peut donc pas relier un même utilisateur sur deux VB Application Ledger distinctes: c'est la propriété d'anonymat visée par la spécification.

Elle vise d'abord les développeurs et les opérateurs de nœuds, et sert aussi de wallet pour l'utilisateur final.

Fonction	Ce que fait l'utilisateur	État
Wallets/Desk	Crée plusieurs wallets ; publie organisations, applications et nœuds	En production
Connexion Opérateur	Se connecte à l'interface d'administration d'un Opérateur en signant un challenge (aucun mot de passe). Desk est <i>obligatoire</i> pour configurer un Opérateur	En production
Nœuds	Réclame un nœud, y stake des tokens, suit son statut	En production
Jetons	Transfert, solde et historique	En production
Certificats	Stocke des SD-JWT et des X509. Les présente sur demande	En production

Fonction	Ce que fait l'utilisateur	État
Approbations	Approuve authentification, signature de message, présentation de certificat	En production
Preuves	Génère et vérifie un fichier de preuve	En production

Intégrer Desk dans un site. Le site envoie des requêtes JSON-RPC 2.0 à Desk via un **relais** fourni par Carmentis (ou auto-hébergé). Les messages sont chiffrés et authentifiés (AEAD) de bout en bout, le relais n'est donc que semi-de-confiance. Bibliothèques recommandées : `carmentis-desk-connect-js` et `carmentis-desk-connect-vuejs`.

Export de wallet. Un wallet peut être exporté dans un fichier de clés chiffré protégé par mot de passe et réimporter. L'Opérateur accepte l'import de ce fichier dans son interface d'administration.

Cryptographie

Carmentis est **agile cryptographiquement** : chaque clé et chaque signature porte l'identifiant de son schéma, et chaque application ledger déclare les schémas qu'elle accepte. On peut donc ajouter un algorithme sans casser l'existant.

Usage	Options disponibles	Remarque
Signature	secp256k1, ML-DSA-65 (post-quantique), Ed25519, JWK	Le JWK est recommandé
Chiffrement à clé publique	ML-KEM-768 + AES-256-GCM (post-quantique)	Sert à transmettre les clés de canal entre acteurs
Chiffrement symétrique	AES-256-GCM	Chiffrement authentifié des données de canal
Hachage	SHA-256 (empreintes, arbres de Merkle), SHA-512 (génération des sels)	
Dérivation de clés	HKDF ; PBKDF2, scrypt et Argon2id pour les mots de passe.	

Encodage des clés. Carmentis propose un formatage des clés utilisé dans l'ensemble de l'écosystème (configuration, API) :

- Le support des JWK permet une intégration **native** de [l'identité décentralisée](#): `did:jwk:ey...`
- L'ancien formatage suit cette nomenclature: `SIG:SECP256K1:PK{...} / SIG:SECP256K1:SK{...}`,

Authentification par signature

Carmentis propose nativement l'authentification d'un utilisateur via son Desk. Cette méthode évite de recourir au mot de passe.

1. Le site génère un challenge aléatoire de 256 bits.
2. Le site renvoie un identifiant de QR, affiché par le site.
3. Le wallet scanne le QR et récupère le challenge et des méta-données de connexion en passant par le relais.
4. Le wallet signe le challenge avec sa clé et renvoie sa clé publique, le challenge et la signature.
5. Le site accepte l'utilisateur si la signature est valide et si la clé fait partie des clés autorisées. La vérification est réalisée par l'opérateur.

Avec cette approche, *l'identité de l'utilisateur est la clé publique.*

Confidentialité et divulgation sélective

Carmentis refuse le « tout ou rien » : dans un même dossier, chaque partie ne lit que les canaux qui la concernent, et chacun peut ensuite prouver un sous-ensemble des données sans révéler le reste.

Canaux publics et privés

- **Canal public**: données en clair, lisibles et vérifiables par tous.
- **Canal privé**: données chiffrées en AES-256-GCM, accompagnées de la racine d'un arbre de Merkle calculé sur les données en clair avant ancrage.
- Un même microbloc peut contenir des données de plusieurs canaux, et un canal s'étend sur plusieurs microblocs.

Gestion des clés (key schedule)

1. À la création d'un canal privé, une **clé de canal** de 256 bits est tirée au hasard ; elle ne change pas pendant la vie du canal.
2. Chaque bloc de données est chiffré avec une clé dérivée (HKDF) de la clé de canal et de sa position (hauteur, section).
3. Pour donner accès à un acteur, un hôte déjà membre publie une **invitation** contenant la clé de canal chiffrée pour l'invité (ML-KEM-768).
4. Un acteur s'abonne en publiant sa clé publique de signature et sa clé publique de chiffrement.

Conséquence : ni les validateurs ni Carmentis ne peuvent lire un canal privé ; la réplique publique de la chaîne ne compromet pas la confidentialité.

Divulgateion sélective

- Chaque champ devient une feuille d'un **arbre de Merkle salé** : un sel unique par champ, dérivé d'un « pepper » aléatoire (SHA-512), empêche de deviner une valeur non révélée.
- Pour prouver un champ, on fournit sa valeur, son sel et le chemin vers la racine ancrée : la preuve fait une taille logarithmique et ne révèle rien des autres champs.

Ancrage

Prérequis : un wallet et une organisation créés dans Desk, un Opérateur déployé dont le compte est approvisionné en CMTS (sinon les microblobs sont rejetés), et une **clé d'API** `cmts...` créée dans l'interface d'administration.

1. **Créer la demande d'ancrage** : le site contacte l'opérateur avec en en-tête sa clé d'API. Il doit renseigner l'identifiant d'application utilisé, l'identifiant de la VB (application ledger) pour prolonger une VB application ledger existante, ainsi qu'une durée de stockage sur la chaîne (valable seulement pour le premier microblob; à expiration toute la VB est supprimée).
2. **Suivre le statut** : Le site interroge périodiquement l'opérateur pour récupérer le statut d'ancrage de sa demande. Ci-dessous est indiqué le statut possible d'un ancrage:

Statut	Signification
created	Demande enregistrée, rien de publié
submitted	Microbloc envoyé au nœud, pas encore confirmé par l'indexeur
anchored	Présence sur la chaîne confirmée par l'indexeur (succès)
failed	Non vue par l'indexeur dans le délai (final)

La confirmation est vérifiée chaque minute avec un délai d'échec de 600 s (configurable ; si l'indexeur est injoignable, la demande reste en l'état.

Clés spéciales (tags)

Pour travailler avec les données, Carmentis met à disposition des développeurs des clés réservées au format `__nom__`. Les données ancrées contiennent ces clés réservées. Le résolveur les interprète et remplace chaque objet tagué par son contenu vérifié.

Clé	Contenu attendu (format du code)	Résultat après résolution
<code>__link__</code>	Une chaîne <code>cmts://resolve/...</code>	Les données du microblob ou de la section ciblée, avec leur preuve
<code>__digest__</code>	<code>{ digest, type: "json", encoding, digestAlg }</code>	La donnée hors chaîne (si fournie).
<code>__digest_data__</code>	<code>{ <empreinte>: { __salt__, ... } }</code>	Rien en sortie : il est utilisé pour remplacer le hash.
<code>__signature__</code>	Objet de signature JSON	Le payload signé, après vérification
<code>__vc__ / __vp__</code>	<code>{ type: "SDJWT_IETF", sdjwt_ietf: "..." }</code>	Les revendications du certificat (<code>iss</code> , <code>sub</code> , <code>iat</code> , <code>exp</code> , <code>nbf</code> , <code>claims</code>)

Clé	Contenu attendu (format du code)	Résultat après résolution
<code>__certificate__</code>	<code>{ type: "PEM", pem } ou { type: "X5C", x5c: [...] }</code>	x509 ou x509Chain : sujet, émetteur, extensions
<code>__representation__</code>	<code>{ templateEngine: "Liquid", format: "markdown", template }</code>	Rien en sortie : sert au rendu lisible d'une preuve au format PDF.

Liens de résolution

Un lien de résolution permet de référencer un microbloc ou une section sur la chaîne. Chaque lien débute par **cmts://** et suit cette nomenclature :

- `cmts://resolve/<type>/<vbId>/mb/h/<hauteur>` : microbloc par VB et hauteur (à partir de 1) ; `<type>` vaut vb (type quelconque), al, acc, app, org, node ou prot.
- `cmts://resolve/<type>/<vbId>/mb/<hash>` : microbloc par hash dans une VB donnée ;
- `cmts://resolve/mb/<hash>` : microbloc par hash seul ;
- suffixe `/sect/<index>` ou `/sect/n/<nom>` : une section précise.

Une référence ne fonctionne qu'entre microblochs gérés par le même Opérateur. Un microbloc qui en référence un autre n'est vérifiable que tant que la VB référencée existe : donner aux VB référencées une durée de conservation au moins égale à celle des VB qui les citent.

Données hors chaîne (RGPD, gros volumes)

Documenté par le guide [Off-chain data](#) et implémenté dans le SDK.

1. L'application envoie l'objet JSON à l'Opérateur et reçoit les **données salées** (l'objet + un champ aléatoire) ainsi que le **hash** associé aux données salées hachées.
2. Elle ancre le hash à la place de la donnée, et conserve les données salées de son côté.
3. À la résolution, elle fournit les données salées ; le résolveur recalcule l'empreinte (JSON canonique) et refuse toute donnée qui ne correspond pas.
4. Pour figer une copie dans un rapport, on peut ré-ancrer sur la chaîne un lien vers le microbloc d'empreinte et les données en clair. La résolution s'occupera alors de remplacer l'empreinte par les données.

Effacer les données salées côté application satisfait le droit à l'effacement : seule une empreinte salée reste sur la chaîne. Le droit de rectification d'une donnée déjà ancrée n'est pas couvert. Un microbloc d'empreinte peut être conservé *sans limite*.

Certificat x509, certificat et présentation vérifiable

Carmentis supporte à la fois les certificats x509, mais également les certificats vérifiables (ainsi que leurs présentations), en les ancrant sur la chaîne. Cela permet d'établir un lien explicite entre une identité, une

chaîne de confiance ou un attribut à une clé publique, et de rendre l'identification des clés transparentes pour les utilisateurs.

Signature de JSON

Une signature est un objet JSON autonome, qui peut être demandée à un utilisateur via son Desk ou encore à un Opérateur, vérifiable via un Opérateur et ancrable tel quel. Carmentis propose deux variantes de signatures d'un JSON : une signature canonique et une signature JWS. La signature embarque la clé publique, ce qui permet d'établir un lien entre la signature et d'autres éléments liés à la clé publique (comme un certificat vérifiable). Chaque signature embarque également un contexte, qui limite sa validité à un usage spécifique.

Preuves, Résolution et vérification

Une preuve Carmentis est un fichier JSON autonome : n'importe qui peut la vérifier contre la chaîne publique, *sans passer par l'entreprise qui l'a produite*. La consultation de la chaîne est gratuite.

Il existe *quatre* types de preuves :

Preuve	Ce qu'elle établit	Produite par
Preuve de compte	État et historique d'un compte	Explorateur
Preuve d'ancrage de microbloc	Le microbloc existe sur la chaîne, à telle hauteur et telle date	Explorateur
Preuve d'authenticité d'enregistrement	Telle valeur (ou tel sous-ensemble de champs) figure bien dans tel microbloc, via le chemin de Merkle	Opérateur (nécessite la clé de déchiffrement pour les données chiffrées).
Preuve de résolution	Un JSON assemblé à partir de plusieurs microblochs est authentique, champ par champ	Opérateur (nécessite la clé de déchiffrement pour les données chiffrées).

Chaque preuve suit une enveloppe commune : type, en-tête (version, identifiant de chaîne, auteur, date), contenu, signature optionnelle de la preuve.

Résolution

La résolution est une opérateur supportée par l'opérateur, et nécessite donc une clé d'API. La résolution commence par un JSON de départ, contenant généralement des clés spéciales (typiquement "`__link__`": "`cmts://...`"). La demande de résolution peut également spécifier des données hors chaîne indexées par empreinte. La réponse est une preuve de résolution contient en-tête la version de preuve, un identifiant de chaîne, un auteur de la preuve ainsi qu'une date de résolution. La preuve embarque le JSON de départ ainsi que le JSON résolu, et les preuves d'intégrité (une par microbloc résolu). La structure de la preuve est pensée pour pouvoir être signée (fonctionnalité optionnelle) ce qui permet d'engager la responsabilité de l'opérateur et donc du service qui s'appuie sur Carmentis pour générer la preuve. La résolution est récursive et procède en plusieurs passes ; le SDK produit en interne

un rapport de « nœuds de résolution » typés (microbloc, section, signature, digest, certificat, VC, VP) avec leur validité.

La clé `__representation__` porte un gabarit Liquid au format Markdown pour produire un rendu lisible (et un PDF) à partir des données résolues.

Proof Checker

Le Proof Checker est une application web de vérification indépendante, publiée en open source et pensée pour être auto-hébergée. Elle accepte une preuve en JSON ou un PDF (qui contient la preuve JSON en pièce jointe), et traite les preuves de compte, d'ancrage, d'application ledger et de résolution.

Elle permet de :

- **Rejouer la résolution** à partir de la chaîne et comparer le résultat à la résolution annoncée par le fichier : toute valeur ajoutée ou modifiée à la main est signalée.
- **Attribuer un niveau de confiance à chaque valeur** : non fiable, authentifiée (signée), ancrée, ancrée et signée, ou en échec. Chaque niveau s'appuie sur des « éléments de preuve » typés : ancrage, signature, empreinte hors chaîne, certificat vérifiable, certificat X.509.
- **Identifier les parties** : pour chaque microbloc, l'application et l'organisation qui l'ont ancré (nom, site, pays), lues sur la chaîne ; pour chaque signature ou certificat, la clé ou l'émetteur, avec les rôles et une mise en avant des parties principales.
- **Revérifier les SD-JWT localement** : signature de l'émetteur et lien de possession du détenteur pour une présentation.

Token CMTS et économie

Le CMTS est le token utilitaire natif : il paie l'ancrage, rémunère les validateurs et porte la gouvernance. Offre initiale codée dans le SDK : **1 milliard de CMTS**, chaque token étant divisible en 100 000 unités atomiques.

Calcul des frais (en production)

La formule de frais (version 1) dépend uniquement des informations signées du microbloc :

1. **Gas de stockage** = 1 gas par octet du microbloc, multiplié selon la **durée de rétention** demandée (grille de prix par tranche de jours, définie dans les paramètres du protocole).
2. **Surcoût de signature** selon le schéma utilisé (secp256k1, ML-DSA-65, etc.).
3. **Frais** = gas × prix du gas ; prix minimum 1 unité atomique, prix par défaut appliqué si l'émetteur n'en fixe pas.

Les frais sont versés sur un compte de frais de bloc, puis redistribués. Le payeur des frais est inscrit dans l'en-tête et contrôlé par le nœud (v1.6.26). La formule est versionnée dans la VB Protocol, ce qui permet de la faire évoluer par mise à jour de protocole.

Opérations sur les comptes (en production)

Opération	Usage
Émission initiale	Création de l'offre au lancement de la chaîne
Création de compte	Un compte vendeur finance l'ouverture d'un nouveau compte
Transfert	Paiement avec référence publique et référence privée
Transfert avec vesting	Durée de « cliff » et durée de libération en jours
Séquestre (escrow)	Fonds bloqués avec un agent et une durée ; règlement confirmé ou remboursé ; expiration automatique
Staking / unstaking	Immobilisation de tokens sur un objet (ex. un nœud validateur)
Slashing	Pénalité sur les tokens d'un validateur, annulable par section dédiée

Trois types de comptes existent : standard, frais de bloc, et compte de tokens brûlés.

Modèle économique

- Frais payés en CMTS par les opérateurs à chaque ancrage, redistribués aux validateurs ; mode « Pay for me / gas station » pour qu'un partenaire paie à la place de son client.
- Validateurs : 1 000 000 CMTS minimum 10 000 000 maximum pour opérer un nœud ; récompense de 10% du montant mis en garantie par an payé quotidiennement dans la limite du plafond de 200m de tokens.
- Gouvernance en trois phases, du conseil d'administration vers un vote ouvert ; vesting linéaire de 60 mois pour les fondateurs
- Prix du token au 30 octobre 2026 : EUR 1,5cts

Devenir validateur et programme d'incentive

Les récompenses des validateurs ne sont pas créées par le protocole : elles sont **versées par un programme hors chaîne**, publié dans [carmentis-incentive](#) (v0.0.5, licence LGPL-3.0, TypeScript).

Parcours du validateur (pages [Nodes](#) de la documentation) :

1. Déployer un nœud réplicateur (CLI `@cmts-dev/carmentis-cli`, CometBFT 0.38, Docker), synchronisé par state sync sur le testnet.
2. Détenir au moins **1 000 000 CMTS** et une organisation publiée, puis **réclamer** le nœud depuis l'Opérateur : il est rattaché à l'organisation et apparaît dans l'explorateur.
3. **Staker** des tokens sur le nœud depuis l'Opérateur. Un seul destakage à la fois par nœud ; il prend effet **30 jours** après la demande.
4. Obtenir l'**approbation de la gouvernance** (chaîne permissionnée) : signer un message avec la clé CometBFT du nœud (`cmts cometbft sign ./cometbft/config/priv_validator_key.json <message>`) et le renvoyer.

Fonctionnement du programme de paiement :

1. Toutes les 120 secondes (paramètre `period_in_seconds`), le service lit la liste des nœuds validateurs auprès de l'indexeur et retire les nœuds exclus par configuration.
2. Pour chaque nœud, il demande à l'indexeur la récompense due depuis la date de début du programme. L'indexeur renvoie le cumul acquis, payé et restant dû, et un détail par période : puissance de vote, heures de disponibilité, heures d'indisponibilité, récompense.
3. Il paie à chaque tour **un seul nœud**, celui dont le reste dû est le plus élevé, par un transfert CMTS depuis le compte payeur vers le compte de l'organisation du nœud, avec la référence publique « Carmentis incentive program: reward payment for node <NODE_ID> ».

Outils et références pour le développeur

Carmentis suit une philosophie open-source, le code est donc public et accessible sur github.com/Carmentis. La documentation développeur est sur docs.carmentis.io.

Dépôt	Rôle
carmentis-sdk-core	Cœur du protocole : VB, microblochs, crypto, preuves, résolveur, tags, frais
carmentis-operator	Passerelle API des entreprises, wallets gérés, administration
carmentis-node	Application ABCI derrière CometBFT 0.38
carmentis-desk	Wallet de bureau, connexion aux Opérateurs, gestion des nœuds
carmentis-cli	Initialisation et signature côté nœud (<code>cmts node init-config</code> , <code>cmts cometbft sign</code>)
<ul style="list-style-type: none">• carmentis-desk-connect-js• carmentis-desk-connect-vuejs	Client JSON-RPC pour dialoguer avec Desk depuis un site

SDK (Typescript)

Modules principaux : blockchains virtuelles, états internes versionnés, calcul des frais, cryptographie (signature, chiffrement, KDF dont Argon2id, hachage), enregistrements et arbres de Merkle, preuves, résolveur et gestionnaires de tags, credentials (SD-JWT, X.509), export de wallet chiffré, fournisseurs d'accès au nœud.

Opérateur

Toutes les routes publiques sont sous `/api/v1`, authentifiées par une clé d'API. L'Opérateur embarque sa propre documentation Swagger sur `/swagger`. L'administration se fait sur `/admin`. Lors de la première configuration, l'Opérateur demandera à l'utilisateur de se connecter sur `/admin/setup` et sera le premier administrateur.

L'interface d'administration couvre la gestion des wallets (création, import BIP39 ou fichier de clés, endpoints RPC et indexeur, soldes), la gestion des organisations et applications (créer, importer, publier), des clés d'API, des invitations, des réclamations et staking de nœuds. Le choix de la base est libre: SQLite, PostgreSQL ou MySQL avec chiffrement au repos. L'Opérateur a besoin d'un nœud **et d'un indexeur** joignables. Il est recommandé d'avoir un Opérateur dédié par entreprise en production.

Nœud

- CometBFT gère réseau, mempool et finalisation ; l'application ABCI applique les règles Carmentis. État dans un arbre de Merkle persistant, avec snapshots.
- Ports : 26656 (P2P), 26657 (RPC CometBFT), 26658 (ABCI gRPC, à ne jamais exposer), 26659 (requêtes REST Carmentis).
- Configuration minimale conseillée : 2 cœurs, 4 Go de RAM ; un validateur a besoin d'une IP publique et d'un nom de domaine (Caddy suggéré pour TLS).

Connecteur Desk VueJS/JS

Un site peut facilement intégrer notre connecteur pour interagir avec un utilisateur via son Desk. Nous proposons deux connecteurs : un premier développé en Javascript qui est le plus générique, et un second plus spécialisé pour une intégration naturelle en VueJS. Le connecteur est pensé pour recevoir une requête écrite en JSON-RPC s'occupe de la transmettre au Desk via un relais.

Cas d'usage et positionnement réglementaire

Carmentis se positionne comme une couche de preuve neutre quant aux formats, complémentaire des infrastructures financières existantes, et non comme un rail de règlement.

Démonstration : File Sign

File Sign est l'application de démonstration : on envoie un fichier à un ou plusieurs destinataires, chacun l'accepte avec son wallet, et l'échange est ancré et prouvable sur la chaîne. Elle montre de bout en bout l'Opérateur, le wallet (bureau ou mobile) et le Proof Checker.

- Application : file-sign.server4.devnet.carmentis.io (devnet ; la documentation cite encore l'adresse testnet file-sign.testnet.carmentis.io)
- Vidéo de démonstration : youtube.com/watch?v=AH1souVIIHs

Familles de cas d'usage (pitch d'avril 2026)

Famille	Ce que Carmentis apporte	Exemple cité
Authentifier, identifier, se conformer	Identité par clé, VC vérifiables, piste d'audit probante	Réduction des coûts KYC/AML ; Finctek (prestataire crypto) pour lier authentification, identité et ordre de paiement
Tracer et prouver	Intégrité de documents qui évoluent dans un flux multipartite	Rapprochement de paiements
Échanger des données de confiance	Partage contrôlé en environnement réglementé (finance, santé, défense, justice)	Recouvrement de créances (Blitz)
Propriété intellectuelle et passeports produit	Gestion des ayants droit, Digital Product Passport	DPP obligatoire en 2027
Identité d'entreprise	Certificats vLEI émis par les partenaires de la GLEIF, signature sur carte	Partenariat GLEIF

Correspondance fonctionnalités / cadres réglementaires

Cadre	Exigence visée	Fonctionnalité Carmentis
eIDAS 2 (registre électronique, EUDI Wallet)	Registre horodaté, intègre, ordonné ; certificats vérifiables	Chaîne de microblochs signés et horodatés ; SD-JWT VC/VP ; statut QTSP visé en feuille de route
RGPD	Minimisation, droit à l'effacement	Champs hachés hors chaîne, divulgation sélective, expiration des VB
MiCA	Encadrement du token CMTS	Token utilitaire ; obligations de livre blanc probables (Titre II)
DORA / NIS2	Résilience et sécurité des prestataires	Consensus BFT, réplication publique sans exposition des données, cryptographie post-quantique
Facturation électronique (2026), MLETR	Preuve d'origine et de non-altération de documents	Ancrage de documents signés, preuves d'authenticité exportables

La conformité eIDAS 2 en tant que registre électronique qualifié n'est pas acquise : elle figure en feuille de route (« QTSP registre électronique distribué », fin 2026), avec des preuves formelles du protocole prévues en amont.

Glossaire

Terme	Définition
ABCI	Interface entre CometBFT (consensus) et l'application qui applique les règles Carmentis
Acteur	Participant nommé d'un application ledger (wallet ou operator)
Application ledger	VB qui porte une instance d'application : un dossier, une transaction
Canal	Espace de données d'un application ledger, public ou chiffré
CometBFT	Moteur de consensus tolérant aux fautes byzantines, version 0.38
Desk	Application wallet de l'utilisateur final
Endosseur	Contre-signataire d'un microbloc
Microbloc	Bloc d'une VB : en-tête + sections + signature
ML-DSA / ML-KEM	Algorithmes post-quantiques de signature et d'encapsulation de clé normalisés par le NIST
Opérateur	Serveur de l'entreprise qui prépare, signe, chiffre et diffuse les microblocs
Pepper / sel	Valeurs aléatoires qui rendent impossible de deviner un champ non révélé
Proof Checker	Outil de vérification indépendante des preuves
SD-JWT VC	Format de certificat vérifiable à divulgation sélective (IETF), retenu par l'écosystème EUDI
Section	Unité de contenu typée d'un microbloc
Tag	Propriété réservée __nom__ (par exemple __link__, __digest__, __signature__, __vp__) interprétée par le résolveur
VB	Blockchain virtuelle : chaîne de microblocs dédiée à un objet
WIAP	Protocole d'échange wallet ↔ Opérateur
Indexeur	Service qui indexe la chaîne ; confirme les ancrages pour l'Opérateur et calcule les récompenses des validateurs
Relais Desk	Serveur qui achemine les requêtes JSON-RPC chiffrées entre un site et le Desk de l'utilisateur
Réplicateur	Nœud qui suit la chaîne sans participer au consensus ; première étape avant de devenir validateur

Sources

- Code (branches principales, lues le 29 octobre 2026) : [carmentis-sdk-core](#) 1.11.4, [carmentis-operator](#) 1.16.1, [carmentis-node](#) 1.6.28, [carmentis-incentive](#) 0.0.5, [carmentis-node-docker](#)
- Documentation : [Concepts](#), [Getting started \(Build\)](#), [Anchor data on-chain](#), [Reference microblocks](#), [Off-chain data](#), [Signed data](#), [Verifiable credentials](#), [Resolve and check proofs](#), [Interact with Desk](#), [Operator API](#), [Operator](#), [Replicator node](#), [Validator node](#), [Get Carmentis Desk](#), [Manage proofs](#), [Demo applications](#)
- Spécification : *Carmentis Protocol Specification*, version 2, 16 juin 2025
- Notes de conception du projet : Operator, Wallet, Desk, Tags, Resolution, Resolution link, Hashed Data, Proof checker, JWK public key, Decentralized identifiers, Système de résolution des microblocks
- *Carmentis Tokenomics Overview*, brouillon du 2 décembre 2024 ; pitch deck Carmentis, avril 2026
- [Roadmap Carmentis](#) (Google Docs)