Chiffrement Transparent pour PostgreSQL

Etat de l’art et Perspectives !

Re-Bonjour

- Damien Clochard
- Co-fondateur de DALIBO
- Vice-Président de l’association PostgreSQLFr
- Développeur Principal de PostgreSQL Anonymizer

Au menu

- Un petit point historique
- Les nouvelles réglementations
- Le paysage actuel
- Perspectives d’avenir

Une longue Histoire

2016: Première implémentation proposée (Cybertec) 1

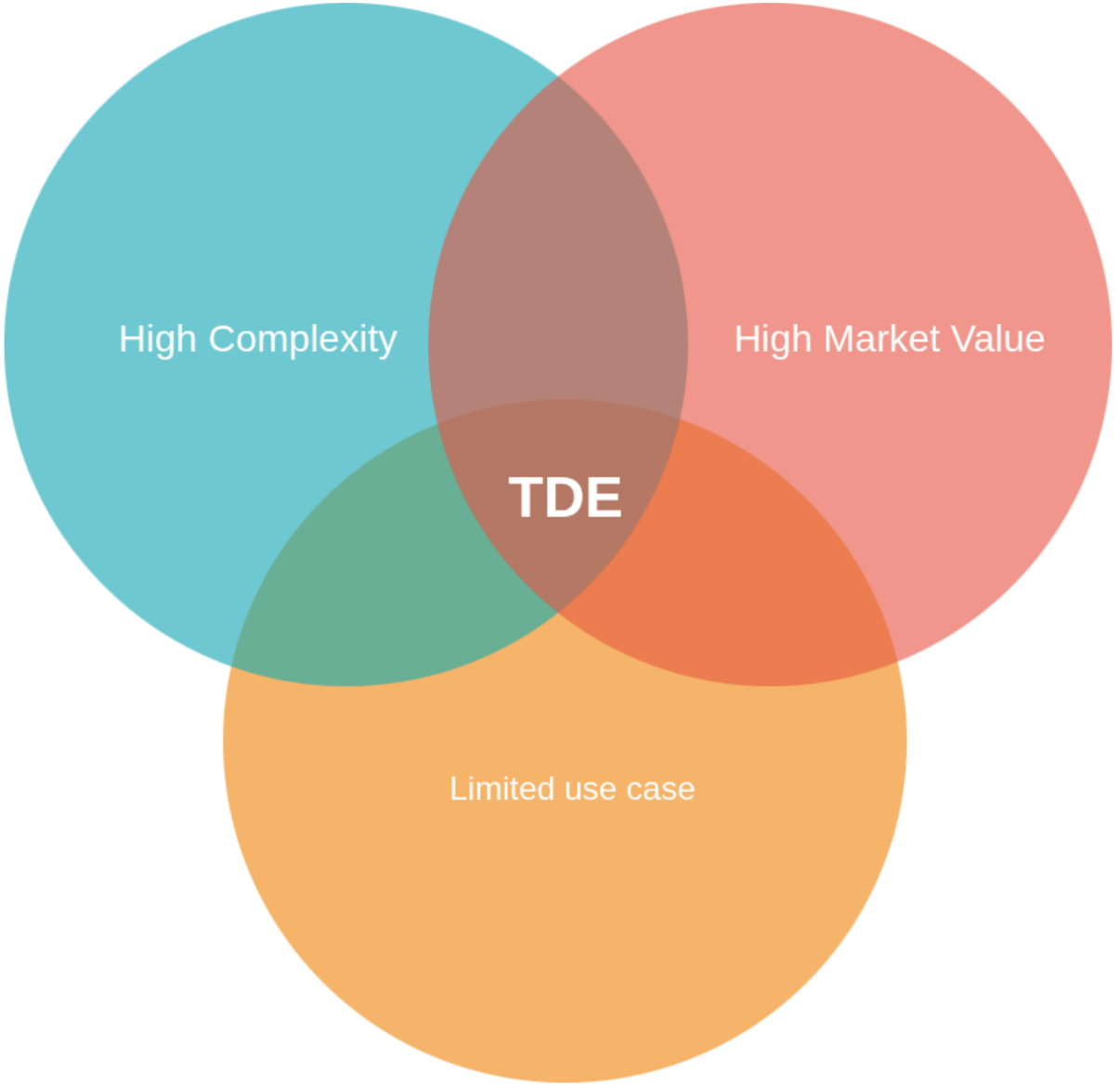
2018-2019: Table-level transparent data encryption (NTT) 2

2019-2021: Plusieurs tentatives refusées (Crunchy Data) 3

2021-2024 : Column-level transparent data encryption (EDB) 4

La fin de l’histoire ?

2025 : « a Postgres core implementation of TDE is unlikely in the near future » 5


High Complexity
High Market Value
TDE
Limited use case

Définitions

- TCE : Transparent Column Encryption
- TDE : Transparent Data Encryption
- CFE : Cluster Files Encryption
- FBE : File-Based Encryption
- FDE : Full Disk Encryption

TDE protège contre :

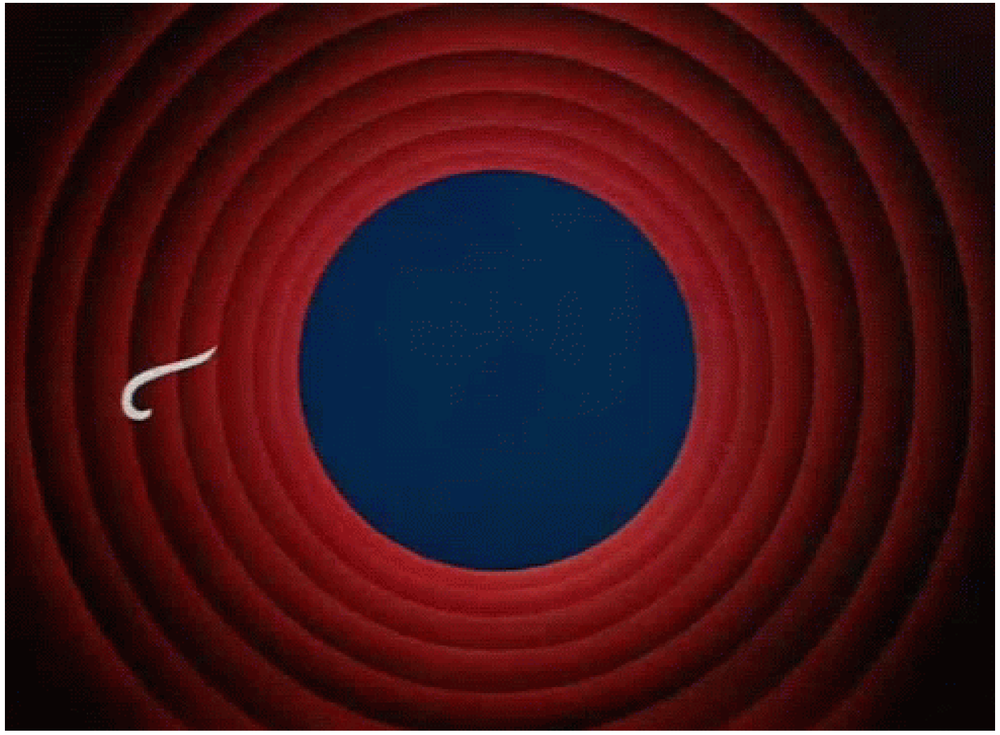
- Vol ou perte d’une unité de stockage (disques durs / SSD)
- Sauvegarde compromise
- Mise au rebut non sécurisée des supports de stockage

TDE NE protège PAS contre

- Attaque sur le système en fonctionnement
- Compromission de l’application
- Injection SQL
- Élévation de privilèges sur le serveur
- Administrateurs malveillants ayant accès au système
- Attaques réseau ou sur les connexions client-serveur

FDE

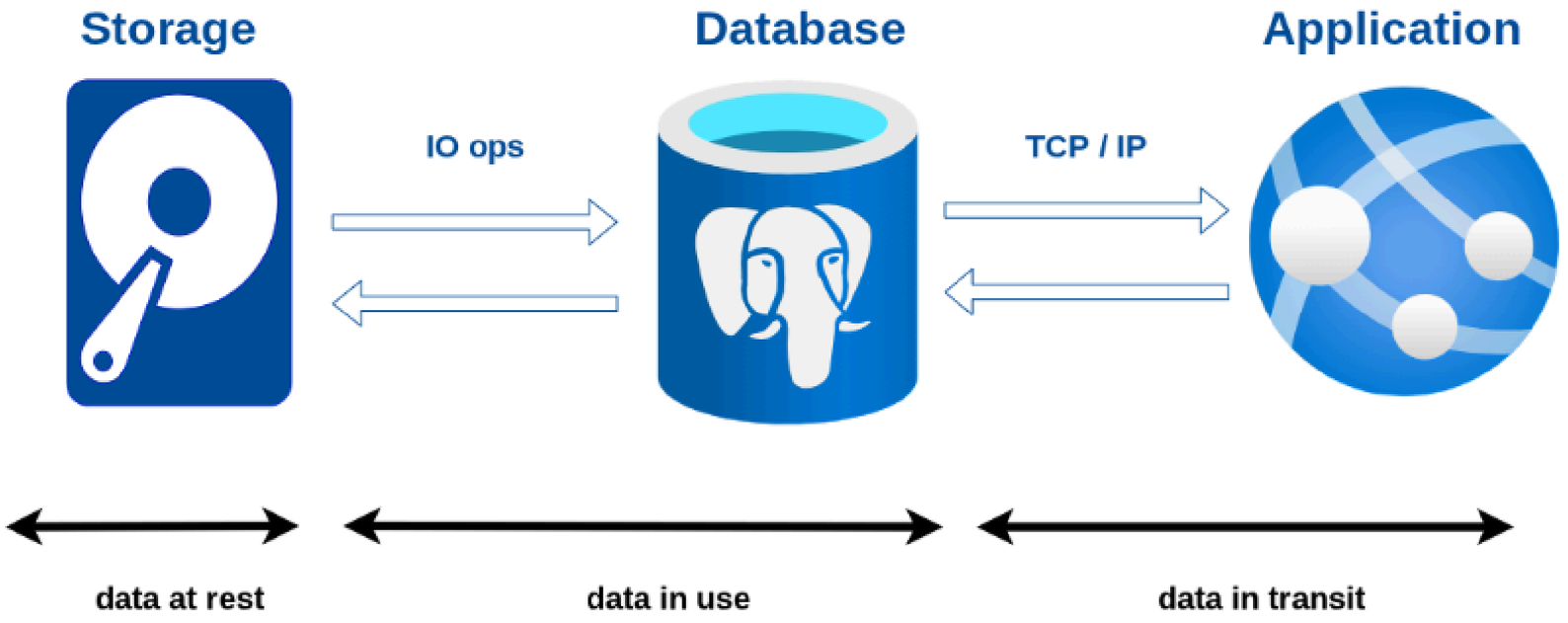
- Simple
- Bien intégré par les constructeurs de baies de stockage
- Compatible avec toutes les versions
- Philosophie Postgres: Déléguer le stockage à la couche inférieure
- Protège contre le vol et la mise au rebut non-securisée



Mais…. les normes évoluent

- DORA => janvier 2025
- PCI DSS 4.0 => mars 2025

data at rest + data in transit + data in use


Storage
Database
Application
IO ops
TCP / IP
data at rest
data in use
data in transit

PCI-DSS dit :

Disk encryption cannot be the only mechanism used to protect PAN on non-removable media.

DORA dit :

The policy must contain rules on:

- Encryption of data at rest and in transit
- Encryption of data in use, **where necessary**

Retour à la case départ

- ~~PostgreSQL~~
- Forks
- Extensions

Forks

- Amazon Aurora
- EDB TDE
- Cybertec
- Crunchy Hardened Postgres
- Postgres Pro TDE

Problèmes

- Coût très élévé
- Fortes Limitations (versions / upgrade / …)
- Solution Propriétaire / Pas d’audits de code possibles
- Vendor Lock In
- Incertitudes sur l’avenir de ces projets

Retour à la case départ

- ~~PostgreSQL~~
- ~~Forks~~
- Extensions

Extensions

- pg_sodium (Supabase)
- pg_tde (Percona)

pg_sodium

- TCE
- Principes similaires à PostgreSQL Anonymizer V1
- Projet à l’arrêt depuis 2023

pg_tde

- `tde_heap_basic`
- `tde_heap`

pg_tde : tde_heap_basic

- Index non chiffrés
- Réplication logique non supportée
- Disponible jusqu’en juin 2025
- Version compatible avec PostgreSQL officiel
- Développé sur GitHub percona/pg_tde

pg_tde : tde_heap

- Basée sur les patchs non-officiels
- Compatible avec Percona Postgres 17 uniquement
- Incompatible avec d'autres extensions (citus/timescale)
- Rapatrié sur GitHub - percona/postgre

pg_tde : Conclusions

Concrètement depuis juin 2025:

- le projet `pg_tde` n’est plus une extension
- de facto c’est un fork de PostgreSQL 17

Retour à la case départ

- ~~PostgreSQL~~
- ~~Forks~~
- ~~Extensions~~

Et Dalibo ?

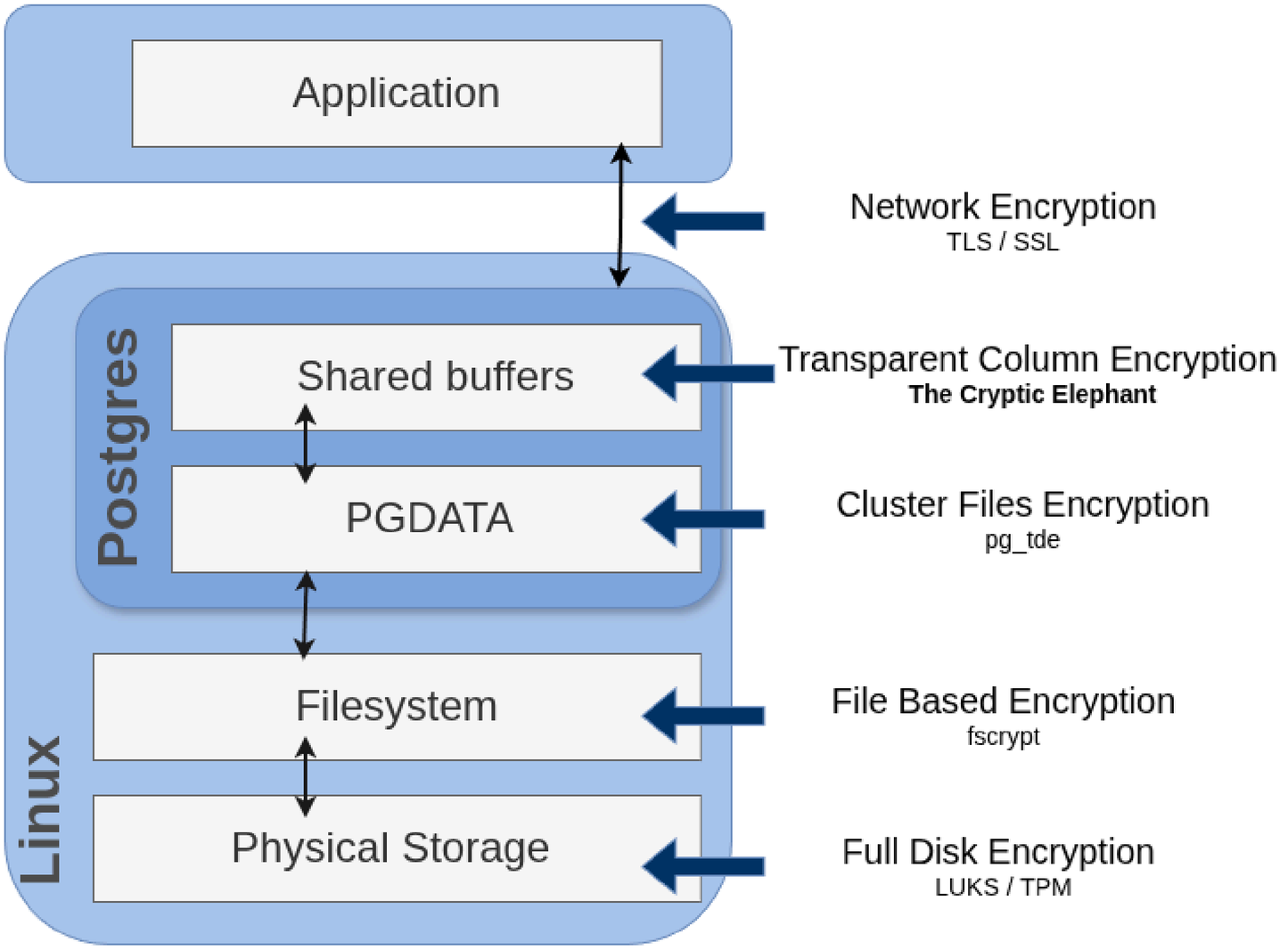
Principes et Roadmap

Investissement significatif sur le sujet en 2026 sur les mêmes bases que nos autres projets:

- Entièrement open-source (pas de modèle open-core)
- Une Extension plutôt qu’un Fork
- S’appuyer sur des librairies déjà auditées
- Intégration dans le Socle Dalibo Postgres 19 (2027)

data-in-use vs. data-at-use

- Tous les forks TDE font du chiffrement au repos (CFE)
- La valeur ajoutée par rapport à FDE est faible
- Cela n’est pas suffisant pour DORA


Application
Postgres
Shared buffers
PGDATA
Filesystem
Physical Storage
Linux
Network Encryption
TLS / SSL
Transparent Column Encryption
The Cryptic Elephant
Cluster Files Encryption
pg_tde
File Based Encryption
fscrypt
Full Disk Encryption
LUKS / TPM

The Cryptic Elephant

- Chiffrement Transparent de Colonnes
- Extension en Rust
- Compatible PostgreSQL 14 à 19
- Licence PostgreSQL

Objectifs

- Chiffrer uniquement quand c’est nécessaire
- Rester transparent pour l’application
- Gérer les clés avec un KMS
- Masquer les données dans le `shared_buffers`

Exemple

```
CREATE EXTENSION tce;

ALTER DATABASE demo SET tce.kms = aws;

CREATE ROLE alice LOGIN;

SECURITY LABEL FOR tce_user_key ON ROLE alice
  IS 'HOSTED BY KMS WITH ID = 98b797ed-e462-45ef-bc6a-4d1dc2434915';
```

Créer une table

```
\connect demo alice

-- Given a classic table
CREATE TABLE spies (
  code TEXT,
  realname TEXT
);

INSERT INTO spies VALUES ('007','James Bond'), ('H21','Mata Hari');
```

Convertir le champs TEXT en TCETEXT

```
ALTER TABLE spies
  ALTER COLUMN realname
  TYPE TCETEXT
  USING realname::TCETEXT;
```

C’est prêt !

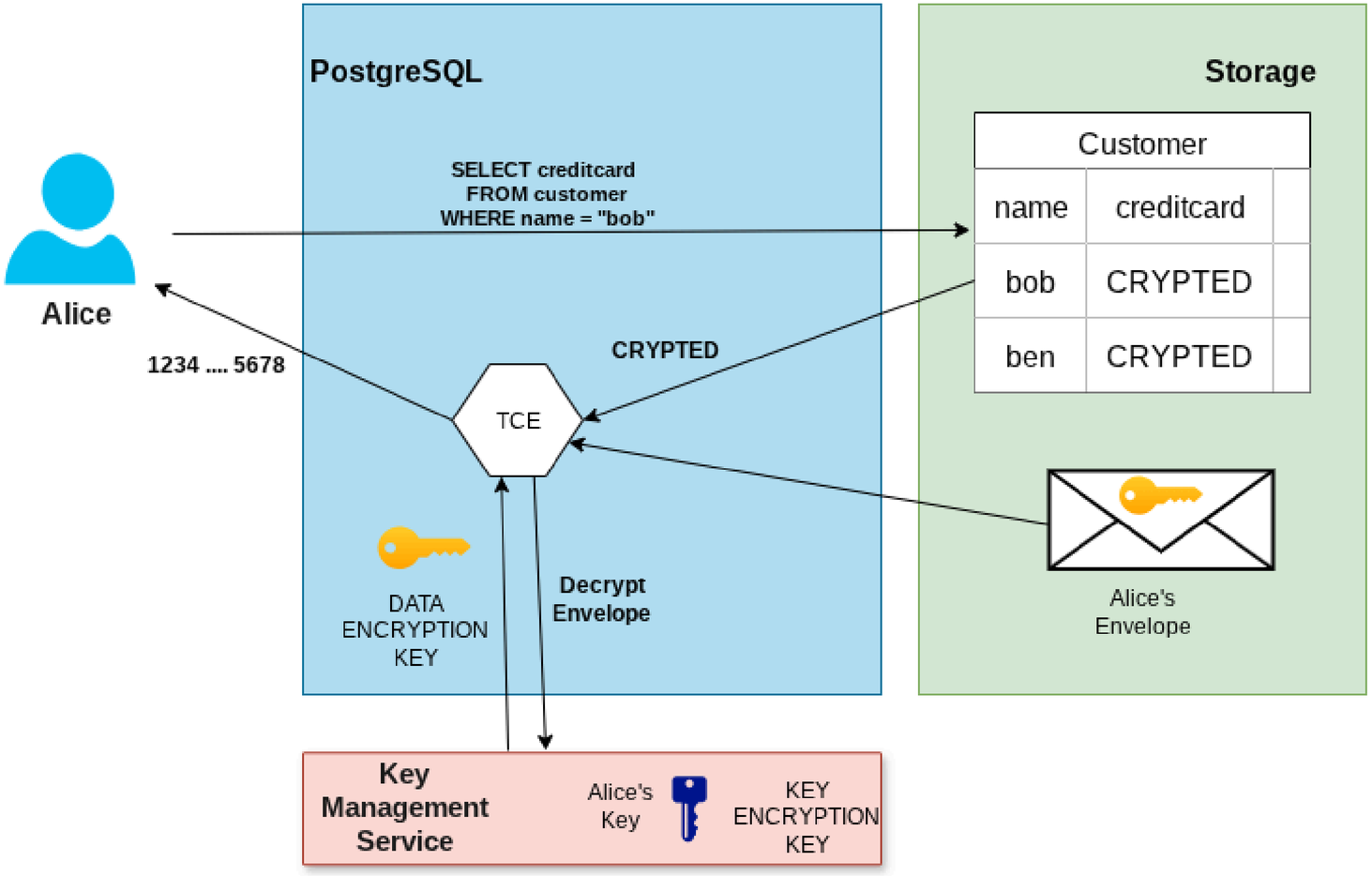
```
SET tce.show_raw TO TRUE;

SELECT realname, raw(realname) FROM spies;
  realname   |                          raw
-------------+-----------------------------------------------
 James Bond  | \x34bebe556560e7fdd4abb441fcb537b7b8c0d67733
 Mata Hari   | \x37bdb54762699534529bb40655fb01232e6f12
(2 rows)
```

Enveloppes Chiffrées

- 1 clé unique de chiffrement des données (DEK)
- Pour chaque utilisateur, la clé DEK est mise dans une envelope
- Chaque utilisateur a une clé pour ouvrir l'envelope (KEK)
- Les enveloppes DEK sont stockés dans la base
- Les KEK sont stockées dans un KMS externe

Fonctionnement Interne



Rotation des clés

- La clé DEK ne change jamais, elle n’est jamais divulguée
- Les clés KEK peuvent être changées à tout moment
- De manière instantanée : on change juste l’enveloppe de la DEK

Sécurité

- En cas de vol du serveur, le KMS est injoignable, les enveloppes ne peuvent pas être ouvertes
- Si un utilisateur est compromis, on désactive sa KEK dans le KMS
- Les données en mémoire sont chiffrées
- Chaque enveloppe est ouverte uniquement pendant la durée d'une session et uniquement **si nécessaire**

Audits

- « Security Through Transparency »
- Chiffrement des données avec la librarie RustCrypto Chacha20Poly1305 Auditée en 2020
- Chiffrement des enveloppes avec la librairie RustCrypto RSA Auditée en 2019

Encore beaucoup de chemin à parcourir

- Input / Output binaire
- Supporter d'autres KMS (Hashicorp Vault / KMIP / …)
- Stocker le DEK dans une enclave mémoire (TPM)
- Certification ANSI CSPN

Feuille de Route

- Janvier 2026 : Publication de la première version Alpha
- Courant 2026 : Premières versions Beta
- 2027 : Version 1.0

https://gitlab.com/dalibo/the_cryptic_elephant

Comment nous aider ?

- Tests & Feedback
- Engagement long terme : Support / Socle / Formation
- Crowd Funding / Co-financement

Conclusion

- Pour les bases non-soumises à DORA ou PCI-DSS, contentez vous du FDE :-)
- Sinon contactez-nous pour une démo !

Questions ?