Harmonizing People, Processes, and Technology for Robust Cybersecurity

D5.3 Secure Data Exchange Platform v2

Document Summary Information

Grant Agreement No	101158555	**Acronym**	CYberSynchrony
Full Title	Harmonizing People, Processes, and Technology for Robust Cybersecurity		
Start Date	01/06/2024	**Duration**	36 months
Project URL	https://www.cybersynchrony.eu/		
Deliverable	D5.3 Secure Data Exchange Platform v2		
Work Package	WP5 Cross-Border Cyber Collaboration and Secure Response (CROSS-CORE)		
Contractual due date	31-05-2026	**Actual submission date**	30-06-2026
Type	Other	**Dissemination Level**	Public
Lead Beneficiary	BYTE		
Responsible Author	George Karavokiros (BYTE)		
Contributions from	MAG, EGL, UPRC, AMC, IANUS		

Funded by the European Union

ECCC

Funded by the European Union under Grant Agreement no. 101158555 and supported by the European Cybersecurity Competence Centre (ECCC). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Competence Centre. Neither the European Union nor the granting authority can be held responsible for them.

Revision history

Version	Issue Date	Changes	Contributor(s)
0.1	15.03.2026	ToC and initial deliverable	BYTE
0.2	08.04.2026	First draft of Chapters 1–2 (Introduction, Alignment with CYberSynchrony Architecture); revised acronym list.	BYTE
0.3	24.04.2026	Pilot dataset descriptions (AMC, UPRC, EGL) and initial Cross-CORE architecture figures.	MAG, UPRC, AMC
0.4	09.05.2026	Differential Privacy component – Global/Local DP back-ends, privacy-budget controller and κ-ary Randomised Response design.	MAG, BYTE
0.5	16.05.2026	Homomorphic Encryption component – PyFHEL/CKKS integration, per-pilot scenarios and end-to-end DP→HE flow.	MAG, UPRC
0.6	01.06.2026	Hyperledger Fabric anchoring chaincode, Hyperledger Explorer integration and Kubernetes deployment on the VM topology.	MAG, BYTE
0.7	10.06.2026	KPI 4.2 cross-border scenarios (S1–S10) and KPI 5.2 information-sharing measurements;	MAG, AMC, EGL, UPRC
0.8	20.06.2026	Internal peer review (Technical Manager and Quality Manager); editorial revisions across all chapters.	BYTE, MAG, EBOS, IANUS
0.9	25.06.2026	Incorporated review comments; updated Executive Summary, Conclusions and References.	BYTE
1.0	30.06.2026	QA Review & Final version v1.0 ready for submission	EBOS, MAG

Disclaimer

The content of the document herein is the sole responsibility of the publishers and it does not necessarily represent the views expressed by the European Commission or its services.

While the information contained in the documents is believed to be accurate, the authors(s) or any other participant in the CYberSynchrony consortium make no warranty of any kind with regard to this material including but not limited to the implied warranties of merchantability and fitness for a particular purpose.

Neither the CYberSynchrony consortium nor any of its members, their officers, employees or agents shall be responsible or liable in negligence or otherwise howsoever in respect of any inaccuracy or omission herein.

Without derogating from the generality of the foregoing neither the CYberSynchrony consortium nor any of its members, their officers, employees or agents shall be liable for any direct or indirect or consequential loss or damage caused by or arising from any information advice or inaccuracy or omission herein.

Copyright message

© CYberSynchrony Consortium This Deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both. Reproduction is authorised provided the source is acknowledged.

[1] According to CYberSynchrony's Quality Assurance Process for reports:

- 3 months before Deliverable's Due Date: Table of Contents ready to be reviewed by WP leader/ Technical Manager/ Quality Manager
- 2 months before Deliverable's Due Date: 50% should be complete. Review by the Quality Manager (& Technical Manager for software).
- 1 month before Deliverable's Due Date:100% should be complete. Review by the Quality Manager and Peer Reviewers.
- M: Sent to PC for Submission to the EC after addressing all comments by Quality Manager and Peer Reviewers

Table of Contents

List of Figures 6
List of Tables 8
Executive Summary 12
1 Introduction 13
2 Cross-Border Privacy-Preserving Data Exchange (Cross-CORE) 14
2.1 Updates Since v1 14
2.1.1 Platform delivery and integration. 14
2.1.2 Pilot data 15
2.1.3 Differential Privacy 16
2.1.4 Homomorphic Encryption 17
2.1.5 Blockchain and audit 18
2.1.6 Deployment topology and operations 19
2.2 Cross-CORE Platform Architecture 20
2.3 Pilot Datasets 21
2.3.1 AMC Medical Center 21
2.3.2 UPRC — University of Piraeus Research Center 21
2.3.3 EGL — Municipality of Egaleo 22
3 Differential Privacy Component 24
3.1.1 Global DP (GDP) - local-curator output perturbation 24
3.1.2 Local DP (LDP) — input perturbation. 25
3.1.3 Randomised Response Back-End 26
3.1.4 PyDP (Google C++ DP) Back-End 27
3.1.5 Per-dataset query catalogues 29
4 Homomorphic Encryption Component 32
4.1 What this component actually does, and why it matters 32
4.2 How the back-end is configured 33
4.3 How the operator sees it 34
4.3.1 Layer 1: Concept ("How It Works — The Flow") 35
4.3.2 Layer 2: Interactive validation 35
4.3.3 Layer 3 — Per-dataset quick demos (5-record sample) 35
4.3.4 Layer 4 — Full-dataset runs ("Process Every Record", HE only) 39
4.3.5 Layer 5 — Synthetic 2000-record HE benchmark. 44
5 Blockchain Anchoring and Cross-CORE Integration 47
5.1 Where the Ledger Sits in the Pipeline 47
5.2 Network Topology 47
5.3 Data Model — Designed for the DP →HE Flow 48
5.3.1 RequestAnchor: The request descriptor 48
5.3.2 PartialAnchor — one per contributing pilot 48
5.3.3 ResultAnchor — the final commitment 49
5.4 What the Chaincode Exposes: Function Surface 50
5.5 The REST Gateway: How the Dashboard Talks to the Ledger 51
5.5.1 Authentication 52
5.5.2 Lifecycle endpoints 52
5.5.3 Verification endpoints 53
5.5.4 Audit & discovery endpoints 53
5.5.5 Operational & subscription endpoints 54
5.6 End-to-End Anchoring Flow for a Cross-Border Request 54
5.6.1 Step 1: Consumer opens the request (one RequestAnchor) 54

© CYberSynchrony, 2026

5.6.2 Step 2: Each dataset anchors its partial (one PartialAnchor per set) 55
5.6.3 Step 3: Central aggregator combines ciphertexts and decrypts 56
5.6.4 Step 4: Consumer anchors the final commitment (one ResultAnchor) 56
5.7 Real-Time Audit with Hyperledger Explorer 57
6 Performance Metrics of the Cross-CORE Pipeline 58
6.1 Record-Level Benchmark 58
6.2 The Production Path 59
6.3 KPIs Mapping 60
6.3.1 KPI4.2 - Number of Cross-Border Actions / Exercises 60
6.3.2 KPI5.2 - Information-Sharing Frequency ≥ 75 % 63
7 Conclusions 66
8 References 67
Annex I — Hardware Specification of the Cross-CORE Deployment 68

© CYberSynchrony, 2026

List of Figures

Figure 1: CYberSynchrony dashboard — CROSS-CORE landing page showing the three pillars: Data Integrity & Encryption, Differential Privacy (with AMC/UPRC/EGL dataset entry points) and Homomorphic Encryption. ... 15

Figure 2: AMC dataset preview on the dashboard — anonymised clinical records (Gender, Age, Weight, BMI, Cholesterol, Blood Pressure) used as input to the Differential Privacy analysis. ... 16

Figure 3: PyDP back-end interactive demo on the dashboard — server-side Google PyDP (Laplace mechanism) with the four supported primitives (BoundedMean, BoundedSum, Count, BoundedStdDev), per-dataset selector (AMC / UPRC / EGL), ε slider and a Custom Query build ... 17

Figure 4: Homomorphic Encryption demo on the dashboard — live PyFHEL/CKKS back-end with the "What is HE?" concept and the end-to-end flow (Plaintext → Encrypt → Compute on ciphertext → Decrypt → Final Result) ... 18

Figure 5: Blockchain and audit architecture: High-level overview of the CYberSynchrony Hyperledger Fabric network, showing permissioned peers, ordering service, anchoring chaincode, IPFS off-chain encrypted storage, JWT-protected APIs, and Hyperledger Explorer for real-time auditability and traceability. ... 19

Figure 6: CYberSynchrony platform landing page with access to the six integrated cybersecurity modules. ... 20

Figure 7: Canonical path of a privacy-preserving cross-border analytical request with encrypted aggregation and blockchain-based audit trail. ... 21

Figure 8: UPRC academic dataset as exposed on the Cross-CORE dashboard ... 22

Figure 9: EGL (Municipality of Egaleo) civil-protection incident dataset on the Cross-CORE dashboard ... 23

Figure 10: Cross-CORE DP demo — back-end selection and live privacy-budget tracking ... 25

Figure 11: Local Differential Privacy demo interface showing dataset selection, LDP input perturbation, privacy-budget configuration and query execution for the Cross-CORE privacy-preserving analytics workflow. ... 26

Figure 12: Randomised Response back-end for Local Differential Privacy, showing how categorical values are probabilistically perturbed before aggregation to produce unbiased frequency estimates without exposing raw records. ... 27

Figure 13: PyDP back-end workflow showing local aggregate computation, bounded query configuration, calibrated noise addition through Google's DP library, and delivery of differentially private results to the analyst interface. ... 29

Figure 14: Homomorphic Encryption component in Cross-CORE, showing how pilot organisations apply DP locally, encrypt their data, and allow an untrusted central aggregator to compute a shared result without accessing raw records. ... 33

Figure 15: Cross-CORE PyFHEL/CKKS back-end configuration and query flow, showing local privacy protection, encrypted partial statistics, homomorphic aggregation and authorised decryption of the final result. ... 34

Figure 16: Homomorphic Encryption demo interface showing the available pilot scenarios for privacy-preserving analytics over AMC health records, UPRC student data and EGL municipal incident data. ... 36

Figure 17: AMC homomorphic encryption demo showing encrypted patient-age aggregation, final-sum decryption and average-age computation without exposing individual records. ... 37

Figure 18: AMC BMI and cholesterol homomorphic encryption demo showing encrypted computation of average health metrics without exposing individual patient records to the analytics server. ... 37

Figure 19: UPRC student-grade homomorphic encryption demo showing encrypted aggregation of student grades and average-grade computation without exposing individual academic records. ... 38

© CYberSynchrony, 2026

Figure 20: EGL municipal incident homomorphic encryption demo showing encrypted aggregation of incident response and completion-time metrics without exposing individual incident records. 38

Figure 21: Full-dataset homomorphic encryption runs panel, exposing HE-only pilot scenarios that encrypt and aggregate every valid record while warning operators about the higher computational cost of processing complete datasets. 39

Figure 22: Full-dataset AMC homomorphic encryption run showing encrypted processing of all patient-age records, final aggregate decryption and detailed CKKS performance metrics. 40

Figure 23: Full-dataset AMC BMI and cholesterol homomorphic encryption run showing encrypted computation of average health metrics and CKKS performance metrics over all patient records. 41

Figure 24: Full-dataset UPRC student-grade homomorphic encryption run showing encrypted aggregation of all student grades, final average-grade computation and CKKS performance metrics. 42

Figure 25: Full-dataset EGL municipal incident homomorphic encryption run showing encrypted aggregation of all incident completion-time records, final average computation and CKKS performance metrics. 43

Figure 26: Synthetic 2,000-record homomorphic encryption benchmark showing full-dataset encrypted aggregation, average-grade computation and CKKS performance metrics at larger scale. 45

Figure 27: Example RequestAnchor JSON record for opening a bounded-mean request over the age attribute in the AMC pilot dataset, including the DP method, HE scheme, privacy budget and creator metadata. 48

Figure 28: Example PartialAnchor JSON record generated by the AMC pilot after applying Local Differential Privacy, encrypting the noised partial with CKKS, and anchoring only the SHA-256 fingerprint and DP/HE provenance metadata on-chain. 49

Figure 29: Example ResultAnchor JSON record committing the final bounded-mean age result, linking two AMC dataset partial anchors with the aggregate ciphertext hash, canonical result hash, total record count, privacy budget and HE aggregation metadata. 50

© CYberSynchrony, 2026

List of Tables

Table 1: AMC dataset - DP query catalogue (Average Age, BMI, Cholesterol, Count Male/Female, Count High BMI) with per-query ε costs and Privacy Budget widget. 29

Table 2: UPRC dataset - DP query catalogue (Average Degree Grade, Count Male/Female Students, Count 2024 Graduates, Average Grade for High Performers, Count 4-Year Graduates) with per-query ε costs and Privacy Budget widget. 30

Table 3: EGL dataset, DP query catalogue (Average Completion / First Action Time, Count Completed Incidents, Count Parking Violations, Average Units / Evidence, Count Severity-2 Incidents) with per-query ε costs and Privacy Budget widget. 30

Table 4: CKKS performance metrics for the full-dataset AMC patient-age homomorphic encryption run, including records processed, encryption/addition counts, execution time, ciphertext size and throughput. 40

Table 5: CKKS performance metrics for the full-dataset AMC BMI and cholesterol homomorphic encryption run, including two HE pipelines, encryption/addition counts, execution time, ciphertext volume and throughput. . 41

Table 6: CKKS performance metrics for the full-dataset UPRC student-grade homomorphic encryption run, including 470 encrypted grade records, homomorphic additions, final decryption, execution time, ciphertext volume and throughput. 42

Table 7: CKKS performance metrics for the full-dataset EGL municipal incident completion-time homomorphic encryption run, including 67 encrypted records, homomorphic additions, final decryption, execution time, ciphertext volume and throughput. 44

Table 8: CKKS performance metrics for the synthetic 2,000-record homomorphic encryption benchmark, showing encrypted processing throughput, ciphertext volume, execution time and final-decryption count. 45

Table 9: Hyperledger Fabric network topology supporting the evidence plane, including peers, orderer, channel, chaincode, Explorer, and API gateway components used for anchoring and audit verification. 47

Table 10: Chaincode function surface exposed by the evidence plane, covering ledger record management, anchored hash storage, read-only queries, ownership-based filtering, time-range audit retrieval and Fabric history inspection. 50

Table 11: Authentication endpoints and bearer-token usage for the Cross-CORE API gateway, showing how Keycloak-issued JWTs are obtained, refreshed and validated before access to protected /api/crosscore/* operations. 52

Table 12: Lifecycle endpoints of the Cross-CORE API gateway, covering request opening, partial anchoring, final result anchoring and request closure across the DP/HE evidence workflow. 52

Table 13: Verification endpoints of the Cross-CORE API gateway, supporting complete request-bundle retrieval, partial ciphertext re-hashing, final result verification, live privacy-budget inspection and Fabric history auditing. 53

Table 14: Audit and discovery endpoints of the Cross-CORE API gateway, enabling contributor, dataset owner, time-range based retrieval of anchored evidence records, together with aggregate dashboard statistics..... 53

Table 15: Operational and subscription endpoints of the Cross-CORE API gateway, covering gateway health checks, Fabric network status, block metadata retrieval and committed transaction inspection. 54

Table 16: Record-level benchmark of the DP–HE–anchoring pipeline across pilot datasets and the synthetic workload..... 58

Table 17: Production-path performance using per-node DP pre-aggregation and one encrypted partial per dataset..... 59

© CYberSynchrony, 2026

Table 18: Cross-border scenarios validating KPI 4.2 through privacy-preserving analytics, threat-intelligence sharing and incident-response exercises. 61

Table 19: KPI4.2 Number of Cross-Border Actions/Exercises: mapping, measurement methodology and progress status, evidencing the ten cross-border scenarios (S1–S10) anchored on the Hyperledger Fabric ledger. 62

Table 20: KPI5.2 Information-Sharing Frequency: mapping, measurement methodology and progress status, with the frequency (80–82%, target ≥ 75%) derived from immutable on-chain records. 64

Table 21: Hardware specification of the four Cross-CORE virtual machines. 68

© CYberSynchrony, 2026

Acronyms, abbreviations, symbols and platform-specific terms

Abbreviation / Term / Symbol	Description
AES	Advanced Encryption Standard – a symmetric-key algorithm used here to encrypt files before they are uploaded to IPFS.
API / REST API	Application Programming Interface that follows the REST architectural style; exposes CRUD endpoints for the web and mobile clients.
CID	Content Identifier – the cryptographic hash returned by IPFS that uniquely addresses a file (content-addressable storage).
CKKS	Cheon–Kim–Kim–Song homomorphic-encryption scheme that supports approximate arithmetic on encrypted real numbers.
Cross-CORE	Cross-Border Cyber Collaboration and Secure Response – WP5 and the corresponding CYberSynchrony secure data-exchange module.
CTI	Cyber Threat Intelligence
DDEP	Distributed Data Exchange Platform – the hybrid stack (Fabric + IPFS + MySQL) that underpins CYberSynchrony's data-sharing architecture.
DP / LDP	Differential Privacy / Local Differential Privacy – techniques that inject calibrated noise into data or queries to prevent re-identification.
HE	Homomorphic Encryption – cryptography that allows computations on ciphertexts; results decrypt to the same value as if computed on plaintexts.
HLF (Fabric)	Hyperledger Fabric – permissioned blockchain used to store immutable audit logs and encryption-key metadata.
IPFS	InterPlanetary File System – a peer-to-peer, content-addressable network used to hold large, encrypted files off-chain.
JWT	JSON Web Token – signed token that carries user identity and role claims, protecting API routes without server-side sessions.
LAP	Laplace mechanism – DP technique that adds Laplace-distributed noise proportional to function sensitivity (used for continuous data).
MSP	Membership Service Provider – the Hyperledger Fabric mechanism that defines and validates organisational identities and trust.
MySQL	Relational database that stores operational metadata (users, roles, access tables) for low-latency queries.
PartialAnchor	On-chain record committing one contributor's processed and encrypted partial contribution and its provenance.
PoC	Proof of Concept – an early demonstrator that validates technical feasibility.
RequestAnchor	On-chain record that opens and describes a Cross-CORE data-exchange request.

© CYberSynchrony, 2026

ResultAnchor	On-chain record committing the final aggregate result and linking the associated PartialAnchors.
RPC	Remote Procedure Call – backend mechanism (e.g., ipfs add) that lets the server invoke commands on other services.
SPA	Single-Page Application – the Angular front end that loads once and then communicates with the backend via APIs.
UFT	Unalterable Forensic Techniques
VM	Virtual Machine – a software-based computing environment; VM1–VM4 identify the four Cross-CORE deployment nodes.
α (alpha)	Probability that Randomised Response reports the true category; (1−α) is the probability of reporting another category.
Δf	Sensitivity of the query function (f): the maximum change in its output caused by changing one record; used to calibrate Laplace noise.
ε (epsilon)	Differential Privacy parameter representing the privacy budget; lower values provide stronger privacy but introduce more noise.
κ (kappa)	Number of possible categories in κ-ary Randomised Response.

© CYberSynchrony, 2026

Executive Summary

This deliverable, D5.3 — Secure Data Exchange Platform v2, reports the second release of the Distributed Data Exchange Platform (DDEP), delivered under WP5 (CROSS-CORE) and now exposed operationally as the Cross-CORE module of the CYberSynchrony dashboard. Where D5.1 delivered a healthcare-centric proof of concept, an Angular single-page application backed by a Node.js REST tier, a Hyperledger Fabric test-network, IPFS off-chain storage, MySQL, AES-GCM file encryption and a small PyDP + Pyfhel (BFV) demo on a synthetic dataset, D5.3 re-engineers that foundation into a production-grade, cross-border privacy-preserving analytics platform driven by real pilot data.

The release advances the platform along three axes. First, the user-facing layer moves from a stand-alone Angular demonstrator to the integrated CYberSynchrony dashboard, re-built as a VueJS application behind an Nginx edge, packaged as Docker images and deployed on Kubernetes across a four-VM topology. Second, the privacy stack is generalised into two first-class services: a Differential-Privacy engine with three interchangeable back-ends, Global DP (output perturbation), Local DP (input perturbation) and PyDP (Google's audited C++ library), together with a κ-ary Randomised Response mechanism for categorical attributes and a persistent ε-budget controller, and a Homomorphic-Encryption engine migrated from Pyfhel BFV to PyFHEL/CKKS which lets several pilots jointly compute a statistic without any of them revealing a single record and without trusting the central aggregator. Third, the audit layer is delivered as a permissioned Hyperledger Fabric network with a purpose-built TypeScript anchoring chaincode (crosscore-cc) and a live Hyperledger Explorer integration, so that every cross-border exchange leaves a tamper-evident fingerprint on the ledger.

The platform is validated against three real pilot datasets — AMC (84 clinical records), UPRC (470 academic records) and EGL (67 civil-protection incident records), which exercise complementary numerical, ordinal and categorical statistics. The canonical request follows a DP → HE → SHA-256 flow in which Local DP is applied on the data owner's on-premise node (VM4), the differentially-private partial is CKKS-encrypted, the central aggregator on VM2 sums the per-pilot ciphertexts without ever decrypting them, the authorised consumer decrypts only the final aggregate, and the request is anchored on Fabric as a RequestAnchor, one PartialAnchor per contributor and a ResultAnchor, the ledger holding only metadata and cryptographic commitments, never plaintext, ciphertext or partial aggregates.

Finally, the deliverable maps the platform's instrumentation onto two project KPIs: KPI 4.2 — Number of Cross-Border Actions/Exercises, evidenced by ten cross-border scenarios (S1–S10) spanning privacy-preserving analytics and threat-intelligence/incident-response exchange (10/10); and KPI 5.2 — Information-Sharing Frequency, for which a measured frequency of 80 %–82 % across the early pilot runs sits comfortably above the 75 % benchmark. Both metrics are derived from immutable ledger records and are independently verifiable through Hyperledger Explorer.

© CYberSynchrony, 2026

1 Introduction

CYberSynchrony addresses cross-border cybersecurity collaboration, where the central tension is that organisations in different Member States need to analyse and exchange security-relevant information jointly, yet are legally and operationally unable to expose their raw data to one another or to a shared cloud service. WP5 (CROSS-CORE) answers this tension with the Distributed Data Exchange Platform: a stack that combines differential privacy, homomorphic encryption and a permissioned blockchain so that pilots can produce shared results and exchange threat intelligence while their underlying records never leave the data owner's perimeter in a readable form.

Concretely, the objectives of this deliverable are: (i) to describe the architecture and the user-facing components of the Cross-CORE module as integrated in the CYberSynchrony dashboard; (ii) to specify the Differential-Privacy and Homomorphic-Encryption services exposed to the pilots, including their privacy-budget controls, parameter choices and end-to-end data flow; (iii) to document the Hyperledger Fabric anchoring chaincode and the Hyperledger Explorer integration that provide tamper-evident audit of every cross-border exchange; (iv) to report the end-to-end performance measured on the consortium's deployment topology and (v) to map the platform's instrumentation onto KPI 4.2 (cross-border actions/exercises) and KPI 5.2 (information-sharing frequency ≥ 75 %).

The document is organised accordingly. Chapter 3 presents the Cross-CORE platform architecture, the updates since v1 and the three pilot datasets (AMC, UPRC, EGL). Chapter 4 details the Differential-Privacy component and its Global DP, Local DP, Randomised Response and PyDP back-ends, with the per-dataset query catalogues. Chapter 5 describes the PyFHEL/CKKS Homomorphic-Encryption component and the layered way it is surfaced to operators, from a conceptual walkthrough to full-dataset runs. Chapter 6 documents the blockchain anchoring data model, the chaincode function surface, the REST gateway and the end-to-end anchoring flow, together with the Hyperledger Explorer audit view. Chapter 7 reports the performance of the complete DP–HE–anchoring pipeline, both at record level and on the production path. Chapter 8 maps the platform onto KPI 4.2 and KPI 5.2, and Chapter 9 concludes. Annex I gives the hardware specification of the four-VM deployment.

© CYberSynchrony, 2026

2 Cross-Border Privacy-Preserving Data Exchange (Cross-CORE)

Chapter 2 details the second release of the Distributed Data Exchange Platform (DDEP), now branded operationally as the Cross-CORE module of CYberSynchrony. While D5.1 established the foundational stack (Hyperledger Fabric, IPFS, MySQL, Angular), this release extends it with (i) a production-grade differential-privacy engine that covers both global and local DP, (ii) a homomorphic-encryption engine based on PyFHEL/CKKS that operates on real pilot data, and (iii) a permissioned Hyperledger Fabric network with Hyperledger Explorer for tamper-evident anchoring of every cross-border exchange. The chapter is organised around the three pilot datasets (AMC, UPRC, EGL) and the concrete user-facing tools that the Cross-CORE module exposes through the CYberSynchrony dashboard.

2.1 Updates Since v1

D5.1 (M12, May 2025) delivered the first integrated release of the DDEP as a healthcare-centric proof of concept: an Angular SPA backed by a Node.js REST tier, a Hyperledger Fabric test-network for audit anchoring, IPFS for encrypted file off-loading, MySQL for operational metadata, AES-GCM file encryption, JWT-protected APIs and a small Python proof of concept that combined PyDP (Laplace mechanism) with Pyfhel (BFV scheme) on a synthetic patient-style dataset. D5.3 retains the cryptographic primitives introduced in D5.1 and re-engineers the platform around three objectives: (i) move from a single Angular demo to the operational CYberSynchrony dashboard; (ii) replace the synthetic PoC with production-grade DP and HE services driven by real pilot data; and (iii) deploy the whole stack on the consortium's VM topology with live Hyperledger Explorer monitoring. The concrete deltas are summarised below.

2.1.1 Platform delivery and integration.

- The DDEP is no longer a stand-alone Angular SPA delivered as an annex demonstrator. In D5.3 it is exposed as the CROSS-CORE module of the main CYberSynchrony dashboard (under private domain name: dashboard.100.64.12.11.nip.io:30443), alongside CYBERRA, CYRESCUE, CYBRITE, CYBERWISE and CYBERGOPLUS, with a single sign-on entry point and a consistent visual identity.

- The CROSS-CORE landing page introduces a three-pillar layout, Data Integrity & Encryption (Hyperledger), Differential Privacy and Homomorphic Encryption (Figure 1)

- The Angular front-end has been re-built as a VueJS web application with an Nginx edge, packaged as a single Docker image and deployed to Kubernetes.

© CYberSynchrony, 2026

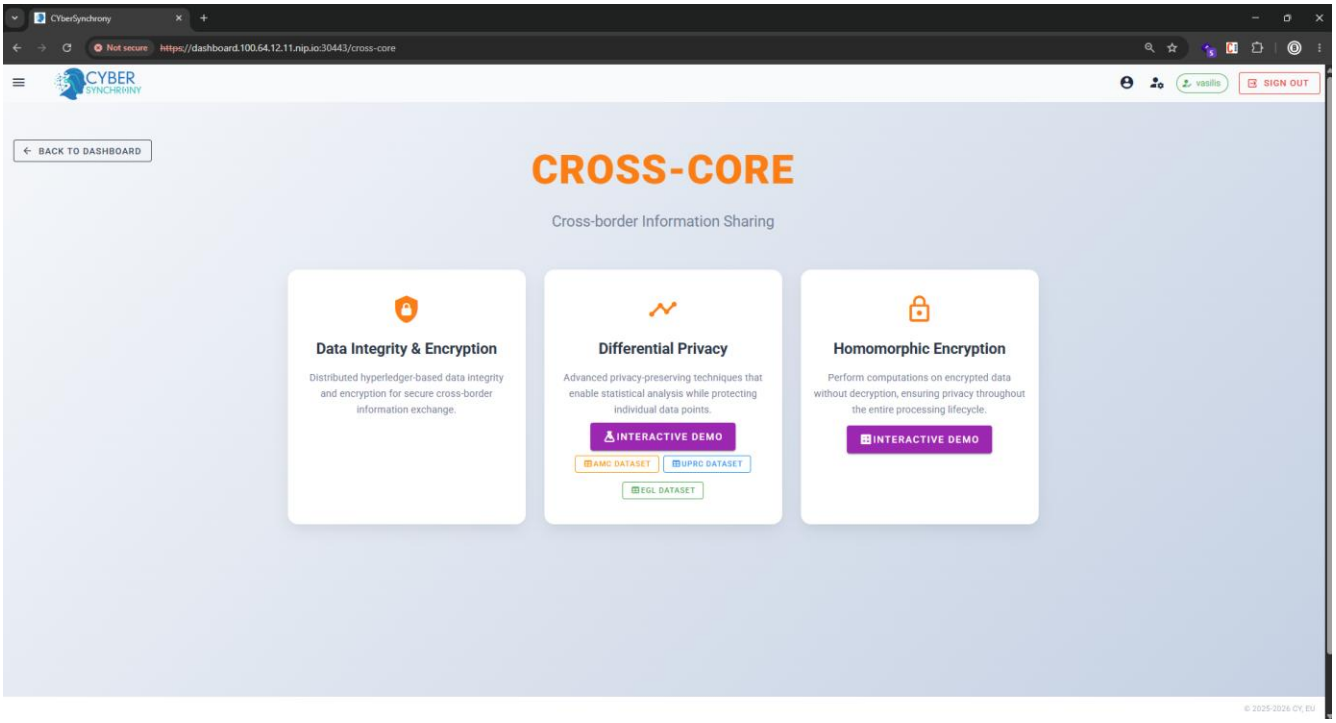

Figure 1: CYberSynchrony dashboard — CROSS-CORE landing page showing the three pillars: Data Integrity & Encryption, Differential Privacy (with AMC/UPRC/EGL dataset entry points) and Homomorphic Encryption.

2.1.2 Pilot data

- The synthetic healthcare dataset used in D5.1 (age / weight / height for three virtual hospitals) has been replaced by three real pilot datasets: AMC (84 clinical records), UPRC (470 academic records) and EGL (67 civil-protection incident records).
- Each dataset is exposed through its own dashboard page with a tabular preview, a record counter (e.g. AMC Dataset 84 records) and direct links to its DP and HE demo pages, as illustrated for AMC in Figure 2, replacing the single "/researchers/statistics" endpoint of D5.1.
- Cross-CORE now supports three statistical archetypes in parallel (numerical health metrics, ordinal academic grades, categorical incident attributes) instead of the single numerical schema of the D5.1 demo.

© CYberSynchrony, 2026

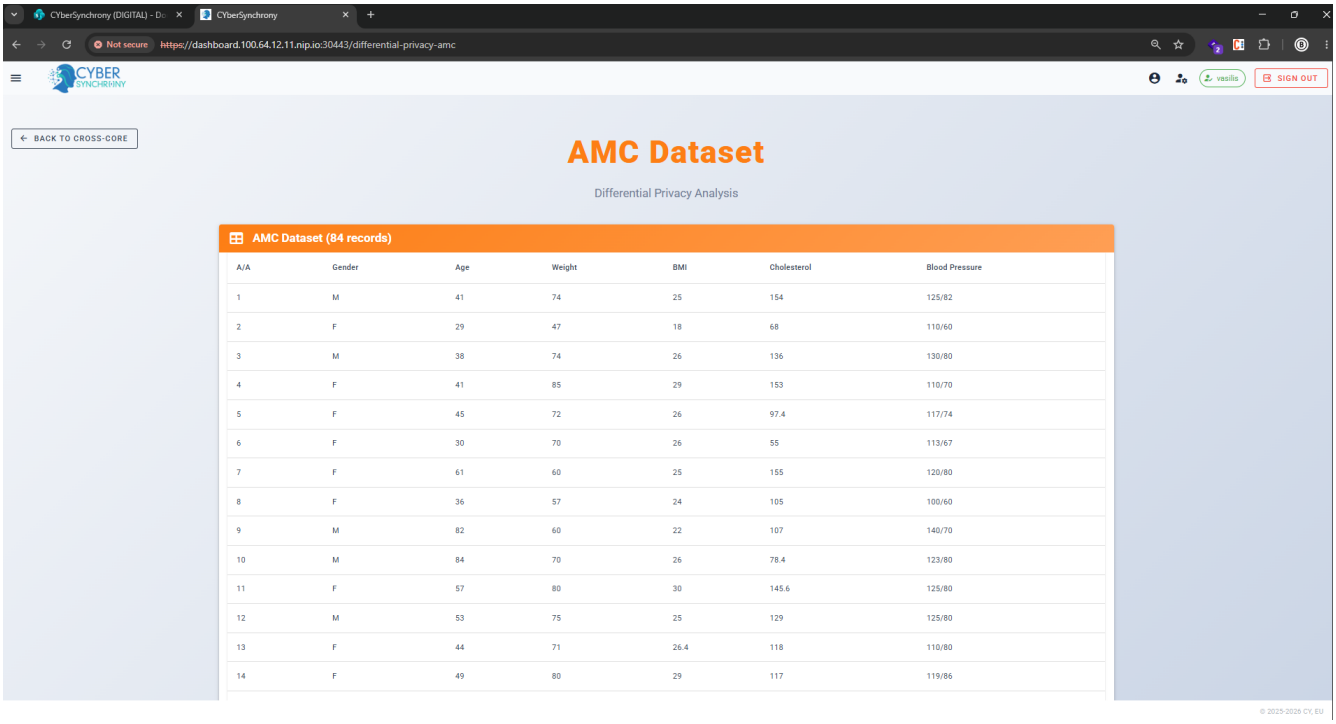

Figure 2: AMC dataset preview on the dashboard — anonymised clinical records (Gender, Age, Weight, BMI, Cholesterol, Blood Pressure) used as input to the Differential Privacy analysis.

2.1.3 Differential Privacy

- D5.1 implemented a single Laplace-based PyDP demo (BoundedMean / BoundedSum / Count, ε hard-coded at 0.01) running on the synthetic dataset. D5.3 generalises this into a first-class DP service with three interchangeable back-ends: Global DP (output perturbation), Local DP (input perturbation) and PyDP (Google's audited C++ library), which are selectable from the UI on every dataset page.

- A persistent privacy-budget controller has been added: a default total budget of ε = 3.50 is tracked across queries with a live "Used / Remaining" widget and a Reset Budget action, replacing the implicit per-call ε of D5.1.

- Every predefined query now declares its ε cost explicitly (e.g. ε = 0.1 for a bounded mean, ε = 0.05 for a count, ε = 0.08 for Count BMI > 30) so that operators can see how the budget is consumed before they run the query.

- A new Randomised Response back-end for categorical attributes (e.g. Gender, Incident Type) has been added. Instead of reporting the true value, each record reports the correct answer with probability α and a random alternative otherwise; the aggregator then statistically reconstructs unbiased category frequencies and the privacy budget ε is derived directly from α. No equivalent existed in D5.1.

- BoundedStdDev has been added to the previously-supported BoundedMean / BoundedSum / Count primitives. Per-attribute bounds (min/max) are now derived automatically from each dataset and applied per query, keeping the sensitivity of every operation small so that the DP mechanism injects less noise for the same ε budget and returns more accurate statistics without weakening the privacy guarantee.

- An Epsilon Comparison panel has been introduced so that the same query can be re-run at several ε values and the resulting noise envelopes can be visually compared, which is a feature absent from D5.1.

- A Custom Query builder (Query Type × Field × Lower/Upper bound) is now exposed in addition to the canned queries, enabling ad-hoc analytical workflows that were not possible against the fixed synthetic schema of D5.1 (Figure 3).

© CYberSynchrony, 2026

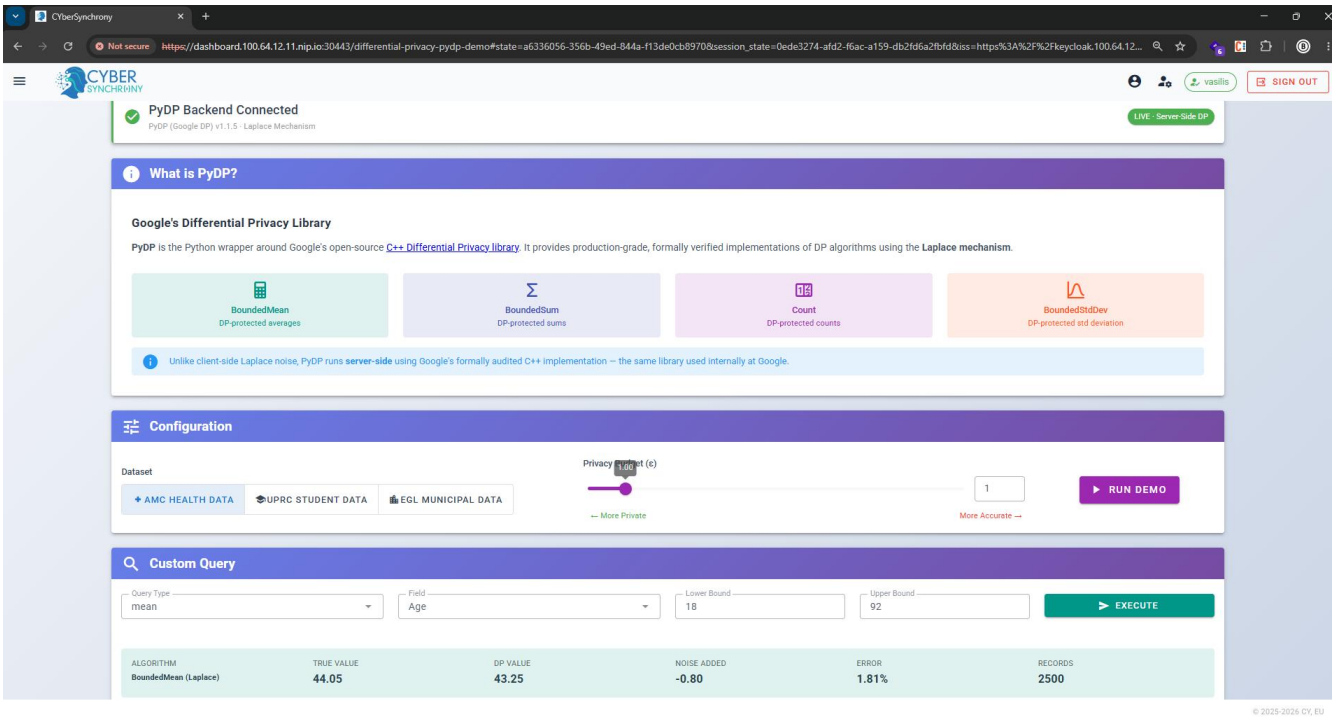

Figure 3: PyDP back-end interactive demo on the dashboard — server-side Google PyDP (Laplace mechanism) with the four supported primitives (BoundedMean, BoundedSum, Count, BoundedStdDev), per-dataset selector (AMC / UPRC / EGL), ε slider and a Custom Query build

2.1.4 Homomorphic Encryption

- The HE back-end has been migrated from Pyfhel BFV used in the D5.1 PoC to PyFHEL CKKS. CKKS supports approximate arithmetic on real numbers, which is a better fit for the continuous statistics (BMI, cholesterol, response time, grade) demanded by the pilots. This implementation extends the initial DP–HE proof of concept presented in D5.1, where PyDP statistics from three virtual hospitals were encrypted and aggregated using Pyfhel with the BFV scheme.
- The dashboard now exposes a live "PyFHEL Backend Connected" status badge and a "How It Works" visual flow (Plaintext → Encrypt → Encrypted Data → Compute → Encrypted Result → Decrypt → Final Result) with a worked example (Figure 4) (a=5, b=3 → Enc(5)+Enc(3) = Enc(8) → 8); D5.1 had no end-user-facing HE explanation.
- Pre-built per-dataset HE scenarios are now packaged into the platform for AMC Average Patient Age, AMC BMI & Cholesterol Analytics, UPRC Student Grade Calculation and EGL Municipal Incident Response Time.
- The DP-then-HE pipeline that was described conceptually in D5.1 is now operationalised: noised values are CKKS-encrypted at the data source and additions are performed in ciphertext space at the central aggregator

© CYberSynchrony, 2026

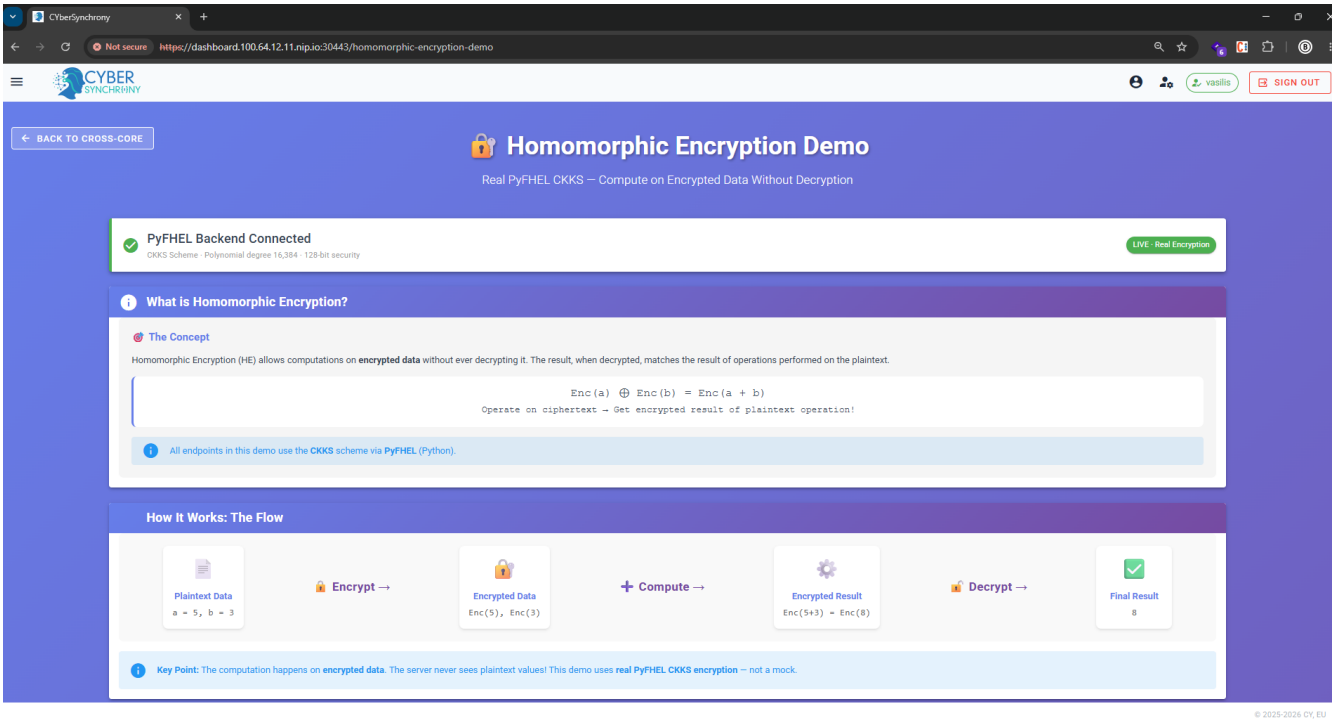

Figure 4: Homomorphic Encryption demo on the dashboard — live PyFHEL/CKKS back-end with the "What is HE?" concept and the end-to-end flow (Plaintext → Encrypt → Compute on ciphertext → Decrypt → Final Result)

2.1.5 Blockchain and audit

- The Hyperledger Fabric test-network introduced inD5.1 has been re-deployed as a permissioned production-style network: two peer organisations (peer0-org1:7051 / Org1MSP, peer0-org2:9051 / Org2MSP), one orderer (orderer:7050 / OrdererMSP) and one application channel (mychannel), packaged as Docker images and deployed on Kubernetes.
- Hyperledger Explorer (client 1.1.8, against Fabric 2.3v1.4) has been integrated and exposed at private domain: explorer.100.64.12.11.nip.io:30443, providing real-time Blocks / Transactions / Nodes / Chaincodes dashboards, a Network view of peer and orderer MSPs, and per-peer ledger heights. D5.1 had no Explorer integration.
- A new anchoring chaincode (TypeScript) replaces the ad-hoc record-writing scripts of D5.1. Each cross-border exchange now writes one transaction containing the SHA-256 hash of the encrypted payload, the hash of the decrypted aggregate, the issuing Membership Service Provider (MSP) identity, the timestamp and a free-form reason field (e.g. MISP event ID).
- The IPFS off-chain store, AES-GCM file encryption and JWT-protected APIs introduced in D5.1 have been successfully deployed and integrated into the CYberSynchrony infrastructure, as summarised in Figure 5, and remain available to the pilots.

© CYberSynchrony, 2026

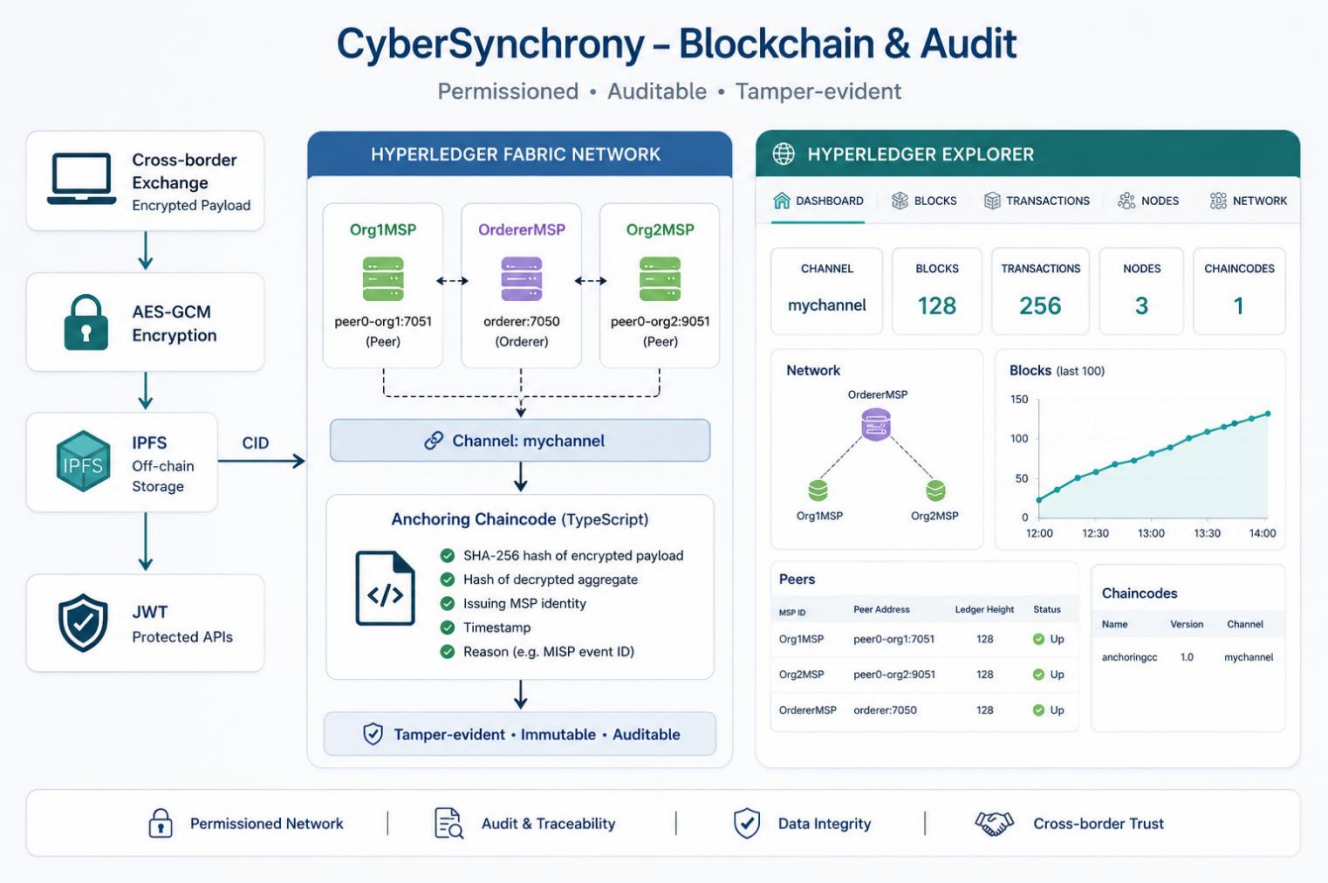

Figure 5: Blockchain and audit architecture: High-level overview of the CYberSynchrony Hyperledger Fabric network, showing permissioned peers, ordering service, anchoring chaincode, IPFS off-chain encrypted storage, JWT-protected APIs, and Hyperledger Explorer for real-time auditability and traceability.

2.1.6 Deployment topology and operations

- The single docker-compose deployment of D5.1 has been replaced by a four-VM topology (VM4 on-premise at the pilot site; VM1–VM3 in the cloud, as shown in Annex I) that physically separates the Local DP / HE-encryption stage (VM4) from the cloud-side services: the Fabric orderer and the dashboard ingress / identity gateway (VM1), the HE-aggregation service and the Org1MSP Fabric peer (VM2), and the Org2MSP Fabric peer with its chaincode runtime and Hyperledger Explorer database (VM3). This separation enforces the architectural principle that raw data must not leave the data owner's perimeter.

- All Cross-CORE components are now packaged as Docker images and deployed on Kubernetes (`gitlab/dashboard/k8s/`, `gitlab/blockchain/k8s/`) with namespace isolation per organisation, Nginx-based TLS ingress and a single nip.io hostname for the consortium test-bed.

- End-to-end response-time measurements over the implemented flow are reported for the first time, together with a quantitative comparison against the inverse HE-then-DP flow that justifies the chosen ordering.

© CYberSynchrony, 2026

2.2 Cross-CORE Platform Architecture

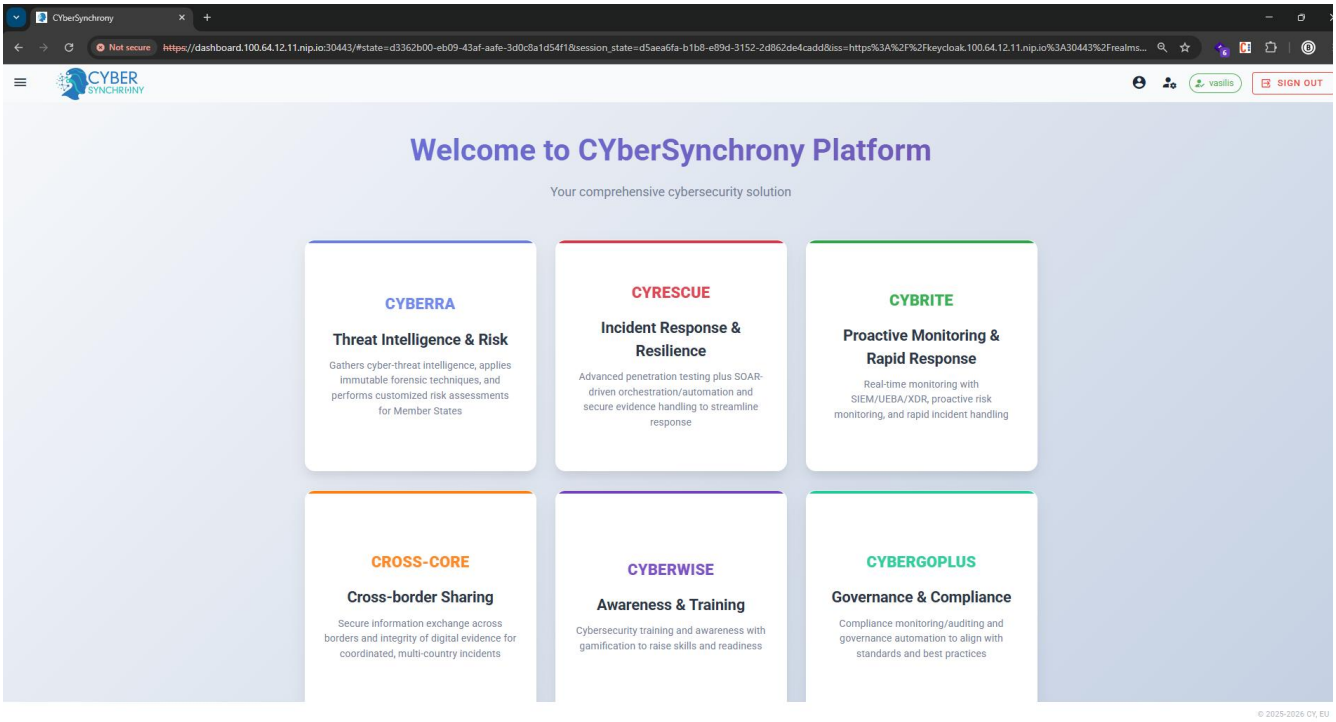

Figure 6: CYberSynchrony platform landing page with access to the six integrated cybersecurity modules.

The Cross-CORE module is delivered as a set of containerised micro-services, deployed on Kubernetes across four virtual machines. It is accessed by Member-State operators through the CYberSynchrony dashboard (Figure 6), which acts as the single sign-on entry point and routes users into the privacy-preserving analytics workflows. The CROSS-CORE tile groups three operational families of tools: Data Integrity & Encryption, Differential Privacy and Homomorphic Encryption (as shown in Figure 1)

From an end-user perspective every cross-border analytical request travels along the same canonical path (Figure 7): (1) the consumer selects the operation and the participating datasets on the dashboard; (2) at each contributing pilot node, Local Differential Privacy is applied so that raw records never leave the data owner's perimeter; (3) each node computes its partial aggregate on the noised data, encrypts it with PyFHEL CKKS, transmits the resulting ciphertext to the central aggregation node, and anchors a partial audit record on Hyperledger Fabric, containing the request ID, the contributor ID, the operation, the ε budget consumed, the SHA-256 hash of the partial ciphertext, the record count and a timestamp, the whole transaction being digitally signed by the contributor's peer; (4) the aggregator combines the per-node ciphertexts homomorphically (without ever decrypting them) into a single aggregate ciphertext; (5) the authorised consumer decrypts the aggregate locally with its CKKS secret key; (6) a final audit record is then anchored on Fabric, linking the request ID to the list of contributors, the hashes of all partial ciphertexts, the hash of the aggregate ciphertext, the hash of the canonical decrypted result and the total ε consumed; (7) Hyperledger Explorer provides an auditable, tamper-evident view of every block and transaction. The ledger therefore stores only metadata, signatures and cryptographic commitments (SHA-256 fingerprints), and stores no plaintext values, no ciphertexts as well as no partial aggregates are ever written on-chain. Any auditor can later prove which nodes contributed, that the aggregator did not tamper with the data in transit, and that the published result matches the one originally decrypted by the consumer.

© CYberSynchrony, 2026

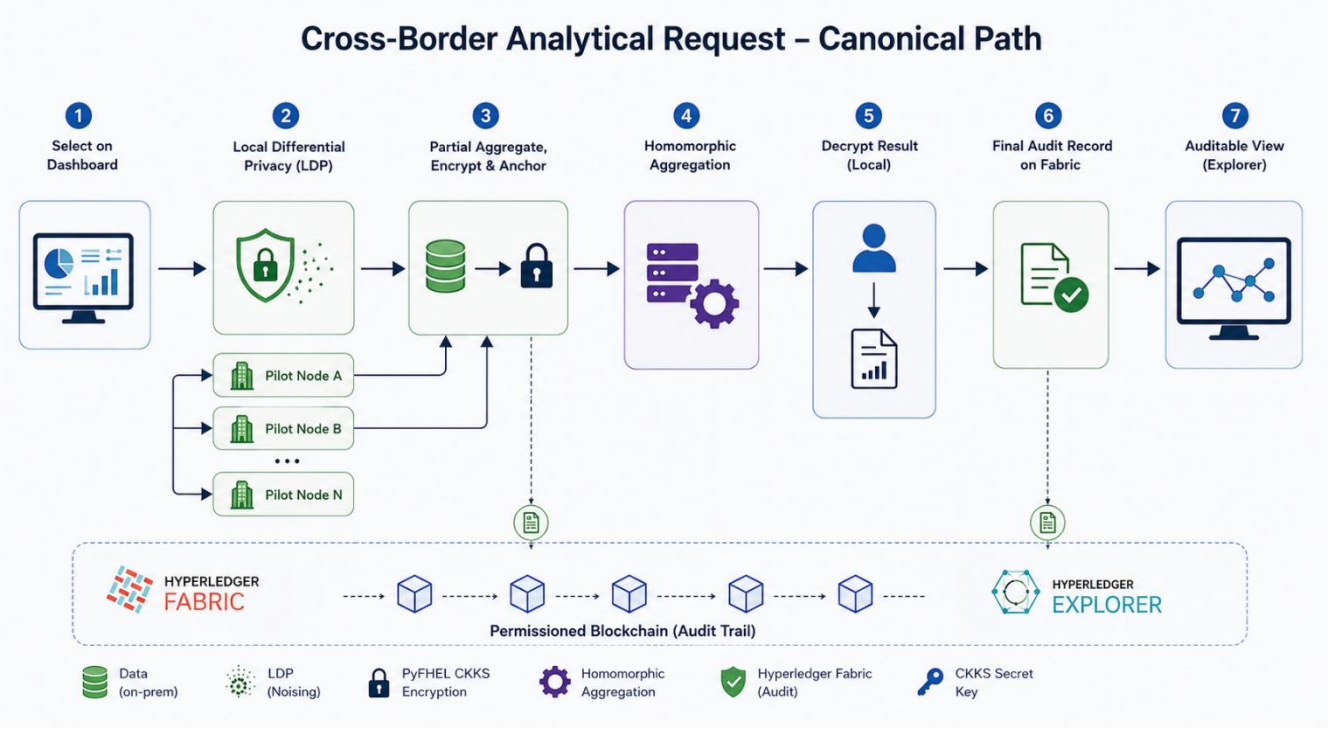

Figure 7: Canonical path of a privacy-preserving cross-border analytical request with encrypted aggregation and blockchain-based audit trail.

2.3 Pilot Datasets

Three pilot datasets have been on-boarded to validate the Cross-CORE module against the heterogeneous operational realities of the project consortium: a healthcare dataset (AMC), an academic dataset (UPRC) and a civil-protection / municipal dataset (EGL). They were chosen because they exercise complementary statistical attributes (numerical, categorical and ordinal) and because they correspond to the three cross-border collaboration scenarios that drive KPI 4.2.

2.3.1 AMC Medical Center

The AMC dataset consists of 84 anonymised clinical records describing Gender, Age, Weight, BMI, Cholesterol and Blood Pressure, as shown in Figure 2. The records are representative of a typical hospital out-patient screening pool and are used to demonstrate cardiovascular-risk analytics under DP and HE.

2.3.2 UPRC — University of Piraeus Research Center

The UPRC dataset contains 470 academic records describing Student ID, Department, Start Year, Graduation Year, Gender and Degree Grade. It is used to demonstrate cross-border educational analytics (typically grade-distribution and graduation-rate queries) and is the largest of the three pilots, which makes it the most demanding for the cryptographic back-end (Figure 8).

© CYberSynchrony, 2026

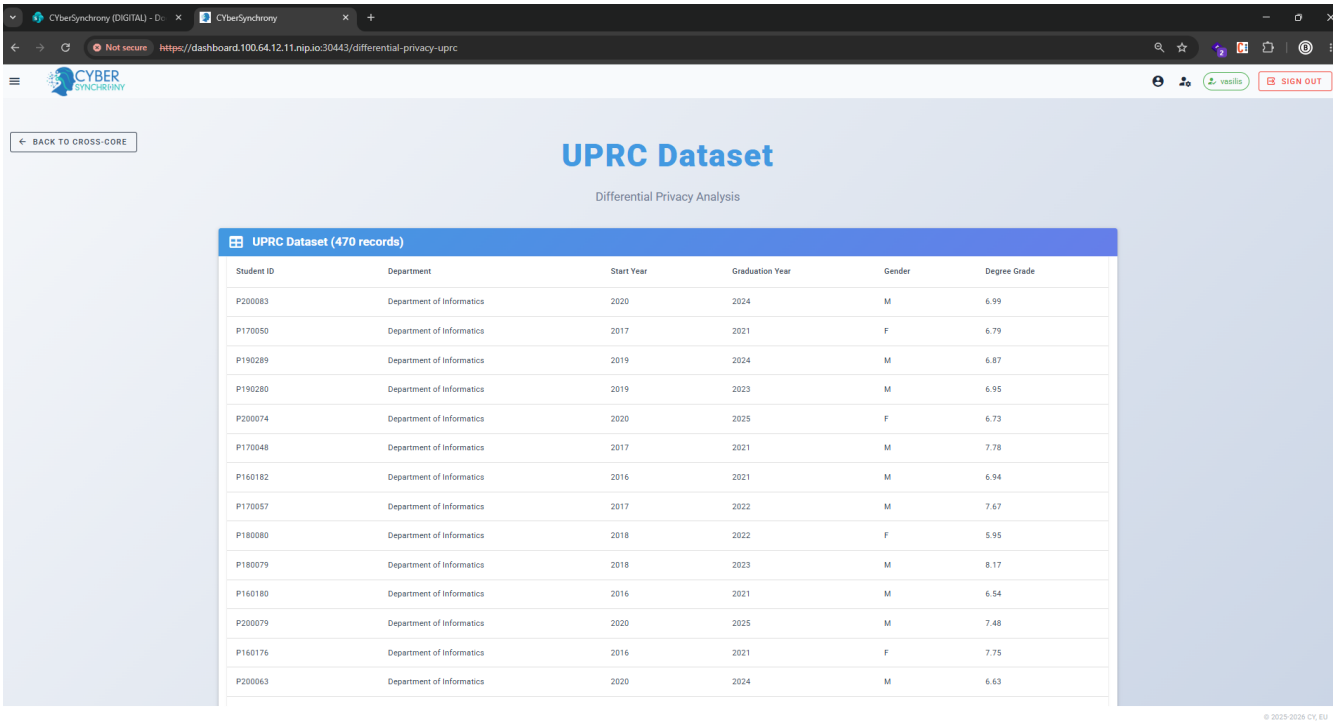

Figure 8: UPRC academic dataset as exposed on the Cross-CORE dashboard

2.3.3 EGL — Municipality of Egaleo

The EGL dataset includes 67 civil-protection incident records (Incident ID, Type, Status, Severity, Address, Completion Time in minutes, Units Involved, Evidence). It demonstrates the value of Cross-CORE for cross-border municipal coordination, for example when neighbouring countries need to compare response performance without exposing the location of individual incidents (Figure 9).

© CYberSynchrony, 2026

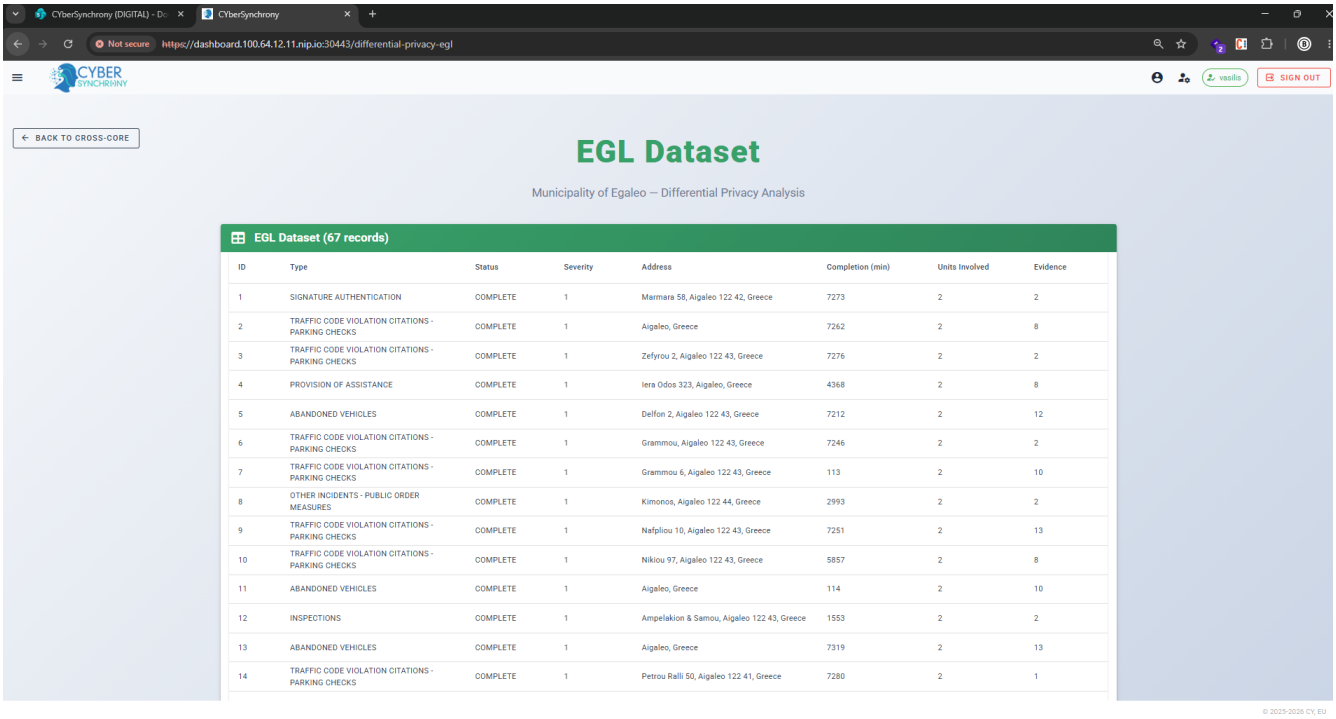

Figure 9: EGL (Municipality of Egaleo) civil-protection incident dataset on the Cross-CORE dashboard

© CYberSynchrony, 2026

3 Differential Privacy Component

The DP component is the privacy-preserving statistics engine of Cross-CORE. From a single dataset page (AMC, UPRC or EGL) the analyst can switch between three interchangeable back-ends via the "Differential Privacy Method" selector: Global DP (in-house output perturbation), Local DP (input perturbation) and PyDP (Google's audited C++ implementation of output perturbation) (as shown in Figure 10). All three back-ends execute entirely inside the data owner's on-premise node (VM4); what distinguishes them is the stage at which the calibrated noise is injected and the implementation library used. No raw record is ever forwarded to the cloud-side aggregator in clear, regardless of the back-end chosen. A persistent "Privacy Budget (ε-epsilon)" widget at the top of every page tracks the total budget (default 3.50), how much has already been spent and how much remains, with a Reset Budget action that the data owner can trigger at any time. Each predefined query carries an explicit per-query ε cost (e.g. ε = 0.1 for a bounded mean, ε = 0.05 for a categorical count) so the analyst sees exactly how the budget is being consumed; the recommended ranges are $\varepsilon \in [0.01, 0.5]$ for strong privacy and $\varepsilon \in [0.5, 10.0]$ for utility.

3.1.1 Global DP (GDP) - local-curator output perturbation

In Global DP the workflow is straightforward: the DP engine running on the data owner's on-premise node (VM4) reads the local dataset, computes the exact answer to the query (e.g. the true mean age, the true count of male patients) and only then adds a small amount of random noise to that single output number before showing it to the analyst. The noise is drawn from a Laplace distribution whose width depends on two things: how much one record could change the answer (the sensitivity Δf) and how strict the privacy budget ε is. A larger ε means less noise and a more accurate result; a smaller ε means more noise and stronger privacy.

Because the noise is added once, at the very end, GDP gives the most accurate results of the three back-ends for the same ε. The trade-off is that the machine doing the computation must be trusted to see the raw records and the true aggregate before noising them. In Cross-CORE this is acceptable because that machine is the data owner's own VM4 (never the cloud-side aggregator), so the "trusted curator" is simply the organisation itself looking at its own data. For this reason, GDP is the recommended back-end for intra-organisation analytics (a hospital analysing its own patients, a university its own students, a municipality its own incidents). For cross-border exchanges, where the data leaves the owner's trust boundary, Cross-CORE switches to LDP so that the raw records are never seen even by VM4's own aggregation step.

© CYberSynchrony, 2026

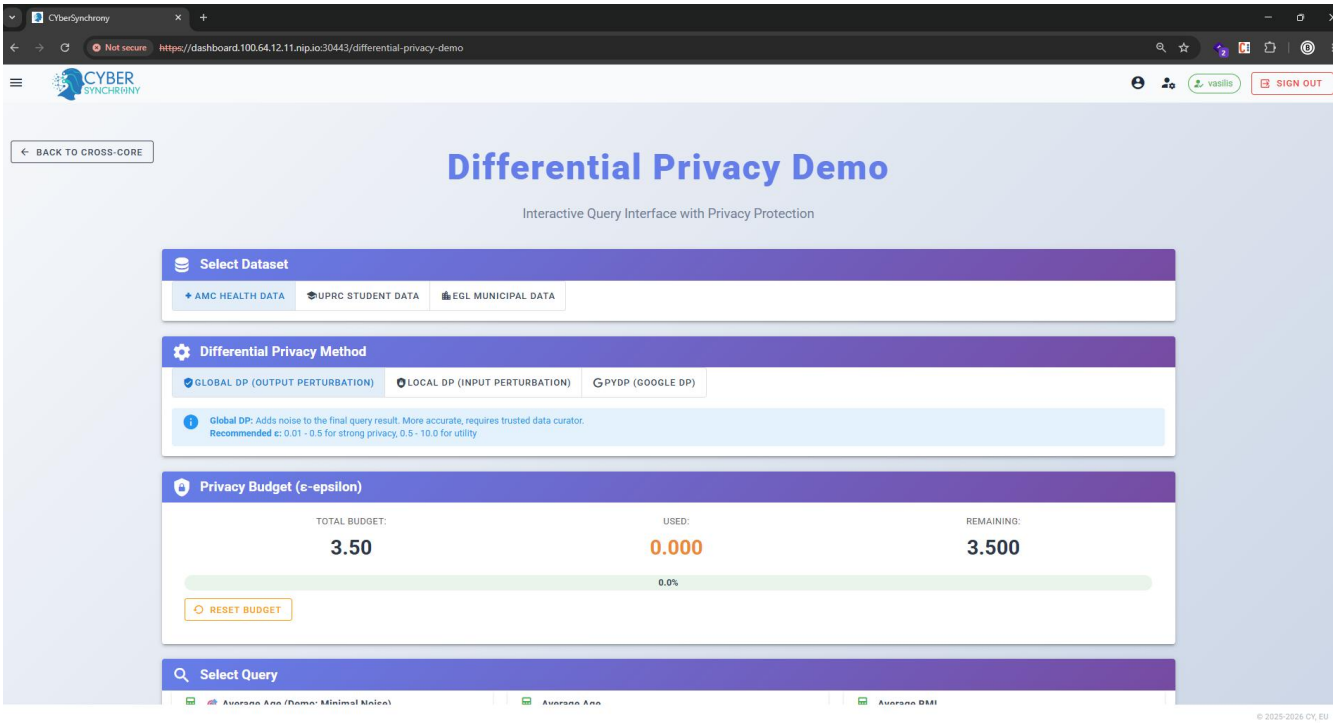

Figure 10: Cross-CORE DP demo — back-end selection and live privacy-budget tracking

3.1.2 Local DP (LDP) — input perturbation.

In Local DP the order of operations is reversed: instead of computing the true answer first and noising it at the end, the DP engine on VM4 noises every single record on its own, the moment it is read from the local store, and only then aggregates. So if the query is "average BMI", VM4 takes each patient's BMI, perturbs it independently, and the average is computed over the already-noised values. The true per-record BMI is never used in any later step (e.g., not by the aggregator on VM4, not by the cloud-side HE node, not by anyone downstream).

The cost of this design is accuracy: because noise is injected once per record and then accumulated by the aggregation, an LDP result is noisier than a GDP result for the same ε. The benefit is trust: LDP requires no trust in any component that comes after the data owner's local store. Even if VM4's aggregation step, the cloud-side aggregator, or any logging surface in between were compromised, what they would see is already-noised values, not raw records. This is the property that matters in a cross-border setting, where the analyst sits in another organisation and Cross-CORE cannot assume the downstream pipeline is trusted end-to-end.

For this reason LDP is the back-end used by default in the cross-border DP-then-HE flow: the noised values produced by LDP on VM4 are the ones that get re-encrypted with CKKS and forwarded to the central aggregator. Numerical attributes (Age, BMI, Cholesterol, Completion Time, Degree Grade, …) are perturbed with per-record Laplace noise calibrated to the same ε budget as GDP. For categorical and ordinal attributes (Gender, Department, Incident Type, Severity, …) Laplace noise is not well-defined, so LDP delegates transparently to the Randomised Response mechanism described in the next section, the analyst still picks "Local DP" in the dashboard and Cross-CORE routes each query to the right primitive automatically.

In short, GDP and LDP answer the same questions over the same datasets but make opposite trust assumptions: GDP trusts the curator and minimises noise; LDP trusts no-one downstream and accepts more noise in exchange. The dashboard exposes both on the same dataset page so the analyst can re-run the same query under either mode and see the utility cost of the stronger trust model directly (Figure 11).

© CYberSynchrony, 2026

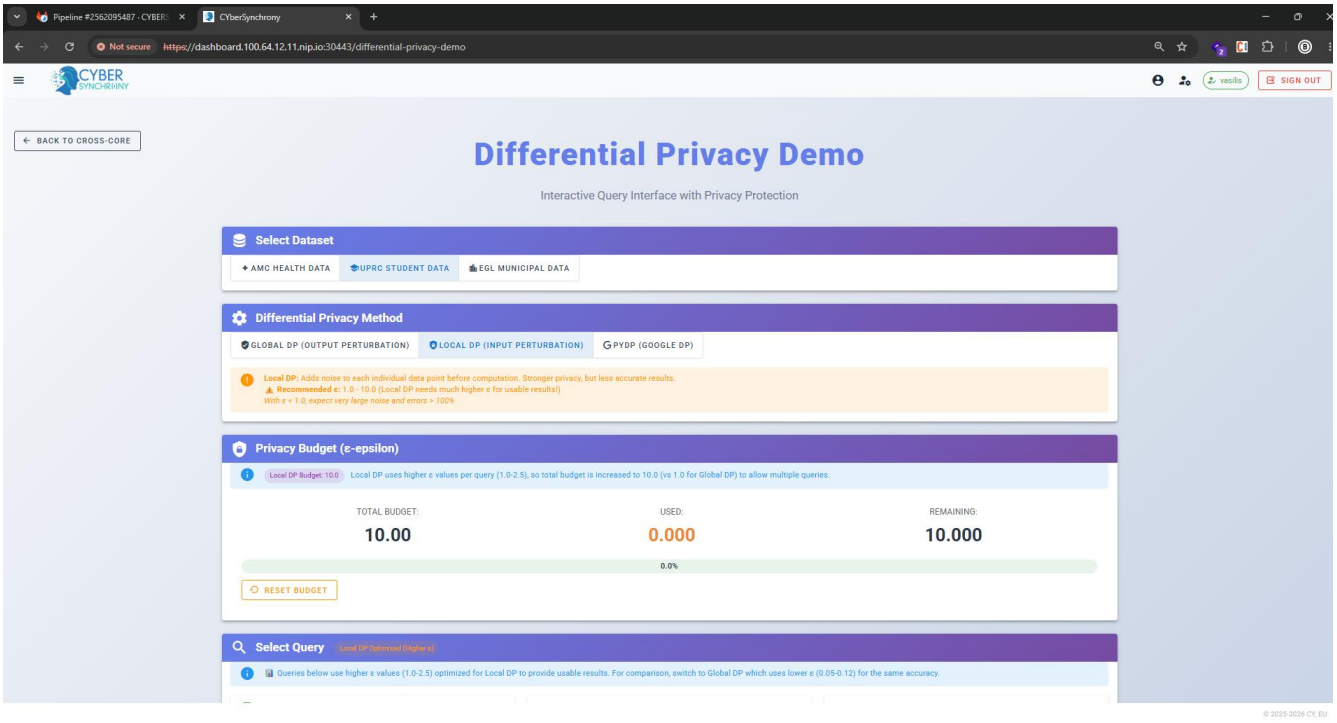

Figure 11: Local Differential Privacy demo interface showing dataset selection, LDP input perturbation, privacy-budget configuration and query execution for the Cross-CORE privacy-preserving analytics workflow.

3.1.3 Randomised Response Back-End

The Laplace mechanism used by GDP and the numerical side of LDP is designed for numbers — averages, sums, counts of records that share a value. It is not the right tool for categorical attributes such as Gender (male / female), Department (Cardiology / Neurology / …), Incident Type (parking violation / fire / medical / …) or Severity level (1 / 2 / 3 / …), where the values are labels rather than quantities and "adding noise" has no obvious meaning. For these attributes Cross-CORE uses Randomised Response (RR), a classic DP mechanism that perturbs the answer itself instead of perturbing a number.

The idea is best explained on the simplest possible case. Suppose the analyst wants to count male patients at AMC. With binary RR, every record independently follows the same rule before it is reported: "with probability α, report the true gender; with probability $1-\alpha$, flip it and report the opposite." No record ever leaves VM4 in its true form, while what gets aggregated is a stream of partially-lied answers. The aggregator then knows the probability α and can mathematically un-bias the noisy count back to a faithful estimate of the true count, without ever needing access to the originals. The privacy guarantee is exact: an attacker who sees one reported value cannot tell whether it is truthful or flipped.

Real attributes in the Cross-CORE pilots rarely have only two values: Department, Incident Type and Severity have several. The binary rule above is therefore generalised to a κ-ary Randomised Response: with probability α the record reports its true value; with probability $1-\alpha$ the record reports a value drawn uniformly at random from the other $\kappa-1$ categories. The same logic carries over, every report is plausibly a lie, the aggregator knows α and κ and applies a minimum-variance unbiased estimator (MVUE) to recover faithful per-category frequencies from the noisy counts.

The mechanism extends naturally from purely categorical to ordinal attributes (Severity 1/2/3, satisfaction levels, …) where the categories have a meaningful order. The implementation is pure Python and is fully interchangeable with the LDP and PyDP paths: when the analyst selects "Local DP" on a dataset page and runs, for example, Count Male Patients on AMC, Count Parking Violations on EGL, or any frequency query over

© CYberSynchrony, 2026

Department or Severity, Cross-CORE routes the query to the κ-ary RR engine automatically — numerical queries on the same page continue to use per-record Laplace as described.

In one line: Randomised Response is what makes "Local DP" work for the questions that Laplace cannot answer (Figure 12). It is the back-end behind every categorical count the analyst sees on the AMC, UPRC and EGL pages whenever LDP is the selected DP method, and it is the mechanism that lets cross-border frequency queries (e.g. distribution of incident types across the three pilots) be answered without any organisation ever revealing the per-record labels of its own dataset.

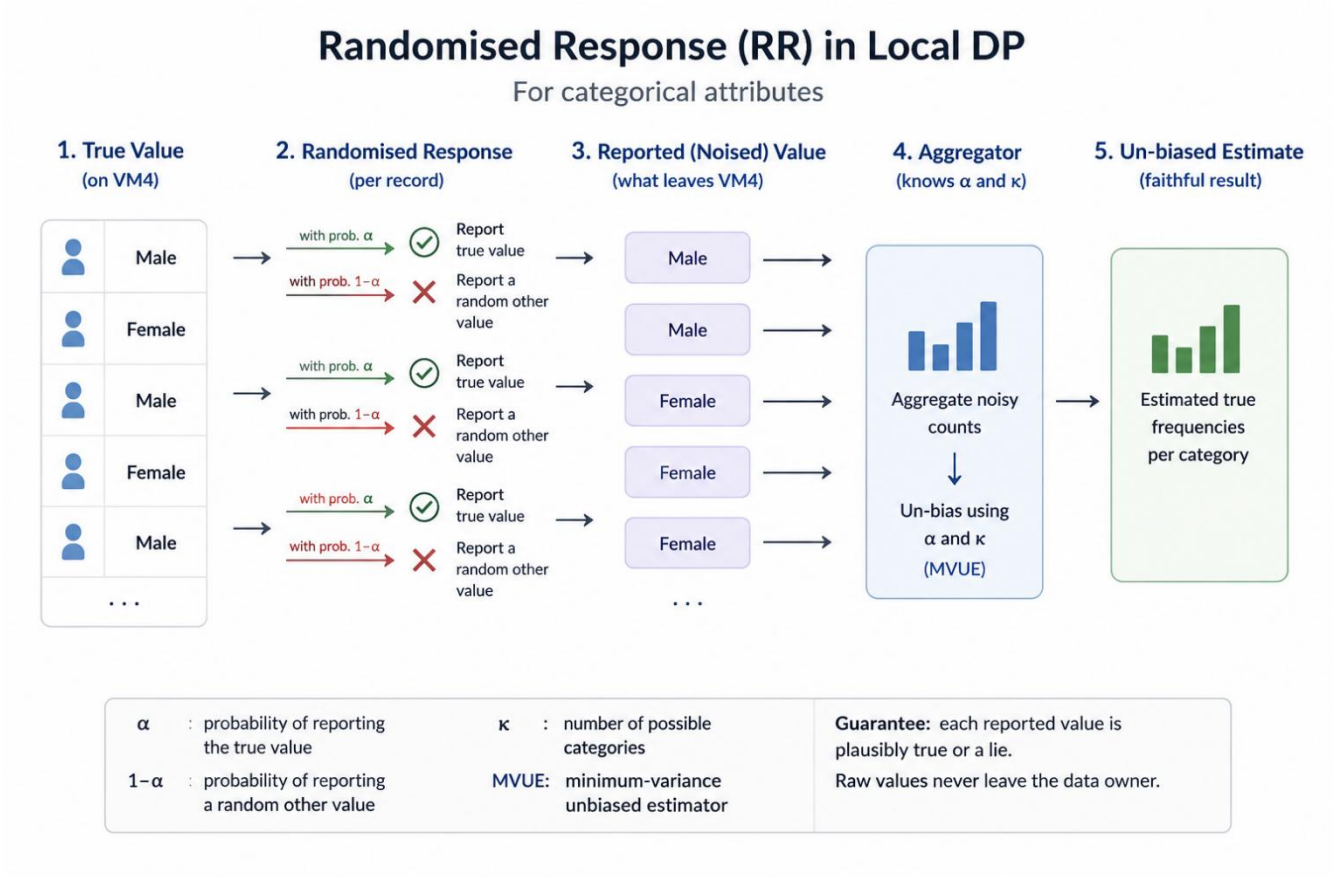

Figure 12: Randomised Response back-end for Local Differential Privacy, showing how categorical values are probabilistically perturbed before aggregation to produce unbiased frequency estimates without exposing raw records.

3.1.4 PyDP (Google C++ DP) Back-End

PyDP is the third entry in the "Differential Privacy Method" selector and, architecturally, the most boring of the three — and that is exactly the point. It follows the same output-perturbation model as GDP (read the local dataset on VM4, compute the exact aggregate, add calibrated noise to the result before it leaves the DP engine), but instead of calling a bespoke Laplace implementation written for Cross-CORE, it delegates the noise step to PyDP, the Python binding of Google's open-source, formally audited C++ differential-privacy library. That is the same code path Google uses internally for its own DP products, exposed in Cross-CORE through a thin Python wrapper.

PyDP was the DP back-end shipped with the first version of Cross-CORE (D5.1); the GDP and LDP back-ends described above were added afterwards for v2, and PyDP is retained unchanged alongside them. Operationally the back-end exposes four primitives, surfaced as colour-coded cards on every PyDP page:

© CYberSynchrony, 2026

BoundedMean: DP-protected averages (used by Average Age, Average BMI, Average Cholesterol, Average Degree Grade, Average Completion Time, Average First Action Time, Average Units Involved, Average Evidence Collected).

BoundedSum: DP-protected sums over a bounded numerical field.

Count: DP-protected counts of records matching a condition (used by Count Male/Female Patients, Count 2024 Graduates, Count Completed Incidents, Count Severity-2 Incidents, etc.).

BoundedStdDev: DP-protected standard deviation, for queries where the analyst needs a sense of spread, not just a central value.

The word "Bounded" in three of the four names is the practical workhorse of the back-end. Differential-privacy noise has to be calibrated to the sensitivity of the query (e.g., how much one record could change the answer in the worst case) and for unbounded numerical fields that worst case is, formally, infinite (a single patient with an unrealistic BMI of 10^6 would force enormous noise on everyone else's average). PyDP solves this by requiring an explicit lower/upper bound per attribute (e.g. Age 18–94, Weight 40–180 kg, BMI 15–55, Completion Time 0–600 min) and clipping values into that range before noising. The bounds reflect domain knowledge, are part of the dataset configuration shipped with each pilot, and keep the noise envelope small enough that the DP result remains useful even at low ε.

From the analyst's perspective every PyDP page is organised the same way: a dataset selector (AMC Health Data / UPRC Student Data / EGL Municipal Data), a privacy-budget slider ε with the qualitative scale "More Private ↔ More Accurate", a Custom Query builder for ad-hoc questions outside the predefined catalogue, and an Epsilon Comparison panel that reruns the same query at several ε values side-by-side so the noise/utility trade-off can be seen directly (Figure 13). Every page reports both the real statistic and its DP counterpart so the analyst can visually appreciate the impact of the privacy budget on utility.

The supported fields per pilot are:

- AMC → Age, BMI, Cholesterol, Weight;
- UPRC → Degree Grade, Start Year, Graduation Year;
- EGL → Completion Time (min), First Action Time (min), Severity, Units Involved, Evidence Collected.

The custom-query builder pre-fills the lower/upper bounds from the dataset's default range for the selected field, so a query can be launched out of the box without any extra configuration; the analyst can then override either bound to widen or tighten the clipping range and observe how the choice of bounds affects both the magnitude of the added noise and the resulting DP value.

In one line: PyDP is GDP done with Google's audited library and with explicit per-attribute bounding, and it is the back-end that Cross-CORE recommends whenever a deployment needs both differential-privacy guarantees and the regulatory comfort of a third-party-validated implementation. The bespoke GDP path remains available for bespoke mechanisms not covered by PyDP's four primitives, and the LDP path remains the default for cross-border flows where zero trust in downstream components is required.

© CYberSynchrony, 2026

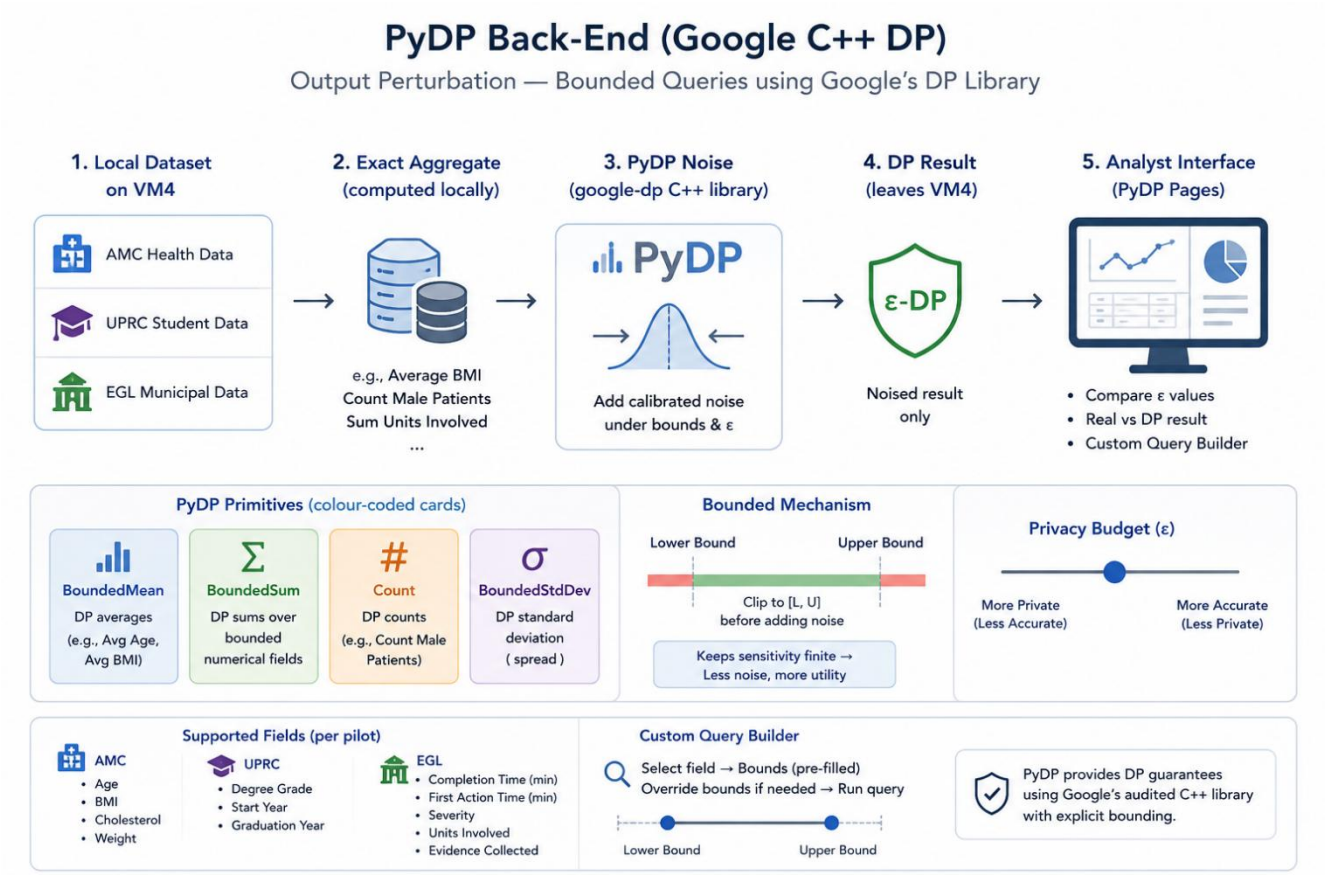

Figure 13: PyDP back-end workflow showing local aggregate computation, bounded query configuration, calibrated noise addition through Google's DP library, and delivery of differentially private results to the analyst interface.

3.1.5 Per-dataset query catalogues

3.1.5.1 AMC Medical Center query catalogue

The AMC dataset contains de-identified patient records (age, gender, BMI, cholesterol, weight). Seven DP queries are pre-wired in the Cross-CORE dashboard, each with a fixed ε cost so the privacy budget is consumed transparently:

Table 1: AMC dataset - DP query catalogue (Average Age, BMI, Cholesterol, Count Male/Female, Count High BMI) with per-query ε costs and Privacy Budget widget.

Query	ε	What it returns
Average Age (Demo: Minimal Noise)	3.0	High-ε variant of Average Age, intended for demonstrations; the DP result is typically within ±1–2 years of the true value.
Average Age	0.1	Mean age of the patient cohort, perturbed at production-grade ε.
Average BMI	0.1	Mean body-mass index across all patients.
Average Cholesterol	0.1	Mean cholesterol level across all patients.
Count Male Patients	0.05	Number of patients with gender = male.

© CYberSynchrony, 2026

Count Female Patients	0.05	Number of patients with gender = female.
Count High BMI (>30)	0.05	Number of patients classified as obese (BMI > 30).

Running the full AMC catalogue once consumes ε ≈ 3.45 of the default 3.50 budget, which is why the demonstration variant of Average Age is offered separately at a high ε for live demos.

3.1.5.2 UPRC — University of Piraeus query catalogue

The UPRC dataset contains de-identified student records (gender, degree grade, start year, graduation year). Six DP queries are pre-wired:

Table 2: UPRC dataset - DP query catalogue (Average Degree Grade, Count Male/Female Students, Count 2024 Graduates, Average Grade for High Performers, Count 4-Year Graduates) with per-query ε costs and Privacy Budget widget.

Query	ε	What it returns
Average Degree Grade	0.1	Mean graduation grade across the student cohort.
Count Male Students	0.05	Number of students with gender = male.
Count Female Students	0.05	Number of students with gender = female.
Count 2024 Graduates	0.05	Number of students whose graduation year is 2024.
Average Grade (High Performers)	0.1	Mean grade computed only on records with grade > 7.0.
Count 4-Year Graduates	0.05	Number of students who completed their degree in exactly four years.

Running the full UPRC catalogue once consumes ε ≈ 0.45, leaving ample budget for repeated exploration.

3.1.5.3 EGL — Municipality of Egaleo query catalogue

The EGL dataset contains de-identified civil-protection incident records (completion time, first-action time, status, severity, units involved, evidence items). Seven DP queries are pre-wired:

Table 3: EGL dataset, DP query catalogue (Average Completion / First Action Time, Count Completed Incidents, Count Parking Violations, Average Units / Evidence, Count Severity-2 Incidents) with per-query ε costs and Privacy Budget widget.

Query	ε	What it returns
Average Completion Time	0.1	Mean time-to-completion in minutes across all incidents.
Average First Action Time	0.1	Mean time-to-first-action in minutes (response latency proxy).

© CYberSynchrony, 2026

Count Completed Incidents	0.05	Number of incidents with status = COMPLETE.
Count Parking Violations	0.05	Number of parking-related incidents.
Average Units Involved	0.1	Mean number of intervention units dispatched per incident.
Average Evidence Collected	0.08	Mean number of evidence items collected per incident.
Count Severity-2 Incidents	0.05	Number of incidents classified at severity level 2.

Running the full EGL catalogue once consumes $\varepsilon \approx 0.53$.

© CYberSynchrony, 2026

4 Homomorphic Encryption Component

4.1 What this component actually does, and why it matters

The Homomorphic Encryption (HE) component is the piece of Cross-CORE that lets several pilots jointly compute a statistic over their combined data without any of them ever revealing their own records, and without trusting the central node that performs the aggregation. It is the cryptographic answer to a very concrete operational question that recurs across the consortium: *"can hospital A, university U and municipality M compute a shared average — patient age, student grade, incident response time — without sending their raw data to anyone, including each other and including the cloud server doing the maths?"* With HE, the answer is yes; with classical encryption (TLS, AES at rest), the answer is no, because the server still has to decrypt the data before it can add it up.

The practical value of this for the Cross-CORE platform is threefold:

- **Legal**: the raw records never leave the data owner's perimeter in a form that anybody else can read, so the cross-border exchange does not expose personal data in the GDPR sense — there is nothing personal to expose, only encrypted blobs whose key the central node does not hold.
- **Operational**: pilots that would otherwise refuse to participate in a shared analysis (a hospital will not export patient ages to a cloud server, a municipality will not export incident addresses) can now contribute to the same query, because what they ship is mathematically unreadable to the receiver.
- **Auditable**: combined with the Hyperledger Fabric anchoring described in, every contribution leaves a tamper-evident fingerprint on the ledger, so a later auditor can prove which organisations took part, that the central node did not alter the inputs, and that the published result is the one the authorised consumer actually decrypted.

In short: differential privacy (§4) protects against re-identification of individuals inside the result; homomorphic encryption protects the records themselves while they travel and while they are aggregated. The two layers are complementary: DP is applied first, on-premise; HE is applied afterwards, also on-premise, before the data leaves VM4; and together they remove the need to trust the central aggregator at all (Figure 14).

© CYberSynchrony, 2026

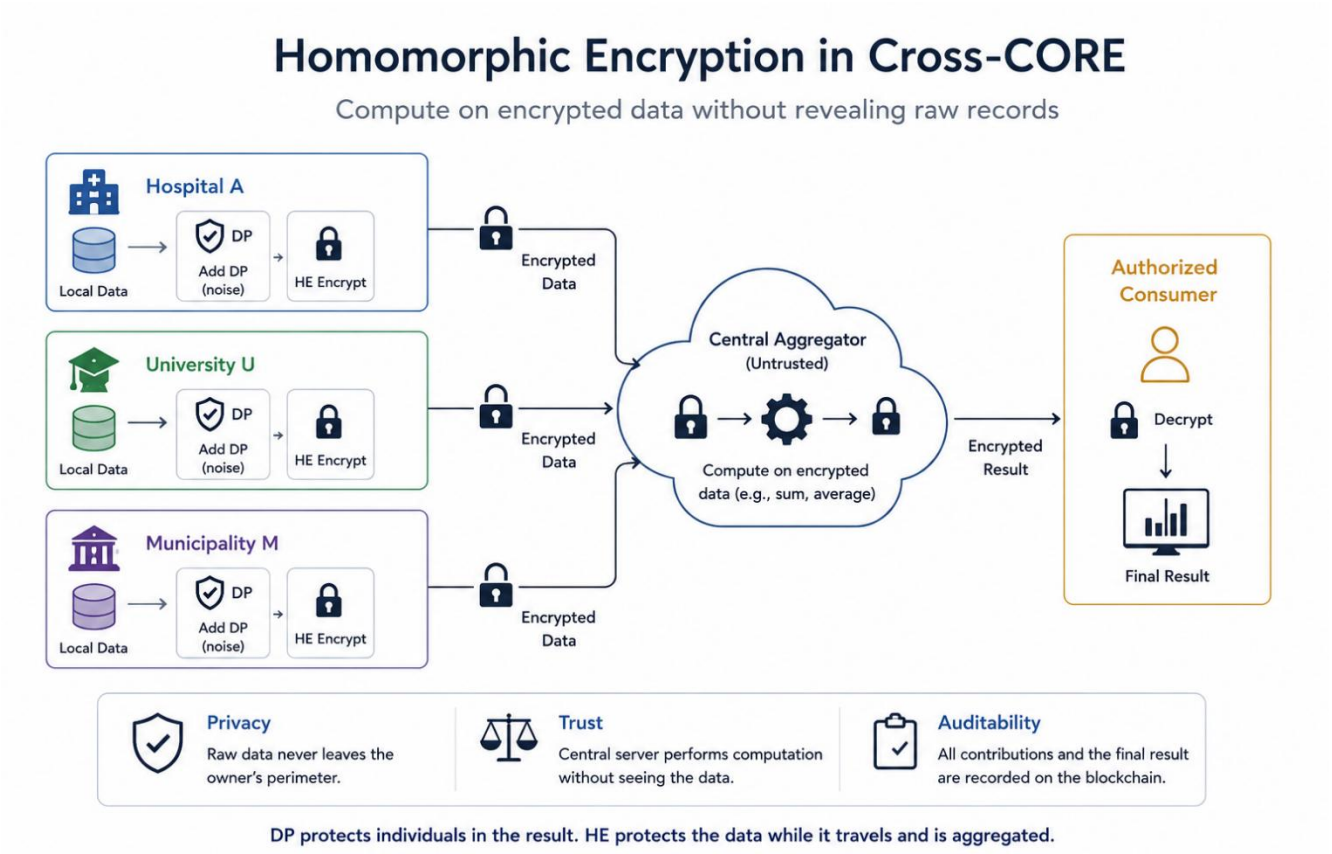

Figure 14: Homomorphic Encryption component in Cross-CORE, showing how pilot organisations apply DP locally, encrypt their data, and allow an untrusted central aggregator to compute a shared result without accessing raw records.

4.2 How the back-end is configured

The implementation uses **PyFHEL**, a Python wrapper over the well-known SEAL/HElib homomorphic-encryption libraries, in its **CKKS** flavour. CKKS is the variant of HE that is designed to work on real numbers (averages, BMIs, grades, response times in minutes), as opposed to the older variants that only handle integers — which makes it the natural fit for the kind of statistics the pilots are interested in.

The dashboard banner exposes the live configuration to the operator so that the cryptographic posture of the system is never hidden behind the UI (Figure 15):

PyFHEL Backend Connected · CKKS Scheme · Polynomial degree 16384 · 128-bit security

Cross-CORE used to ship with a simulation mode for development. The green badge confirms that the user is actually talking to is the real PyFHEL service, not a mock that returns plaintext under the hood. This is a deliberate transparency feature — operators in regulated organisations need to be able to see, before they run a query, that what is happening on screen is genuinely encrypted.

© CYberSynchrony, 2026

- **"128-bit security" parameter set.** This is the standard cryptographic strength target recommended by the Homomorphic Encryption Standard[1] for production deployments, strong enough that brute-forcing the encryption is not a credible attack, weak enough that the latency is still in the seconds-not-minutes range.
- **Footprint kept under control.** The deployed CKKS configuration balances security, precision and ciphertext size. Although homomorphic ciphertexts are inherently much larger than plaintext numerical values, the Cross-CORE workflow limits the exchanged payloads to encrypted aggregates and query-level outputs rather than raw records. In addition, CKKS batching can pack multiple numerical values into a single ciphertext, reducing the effective overhead per value and keeping the approach practical for the pilot scenarios.

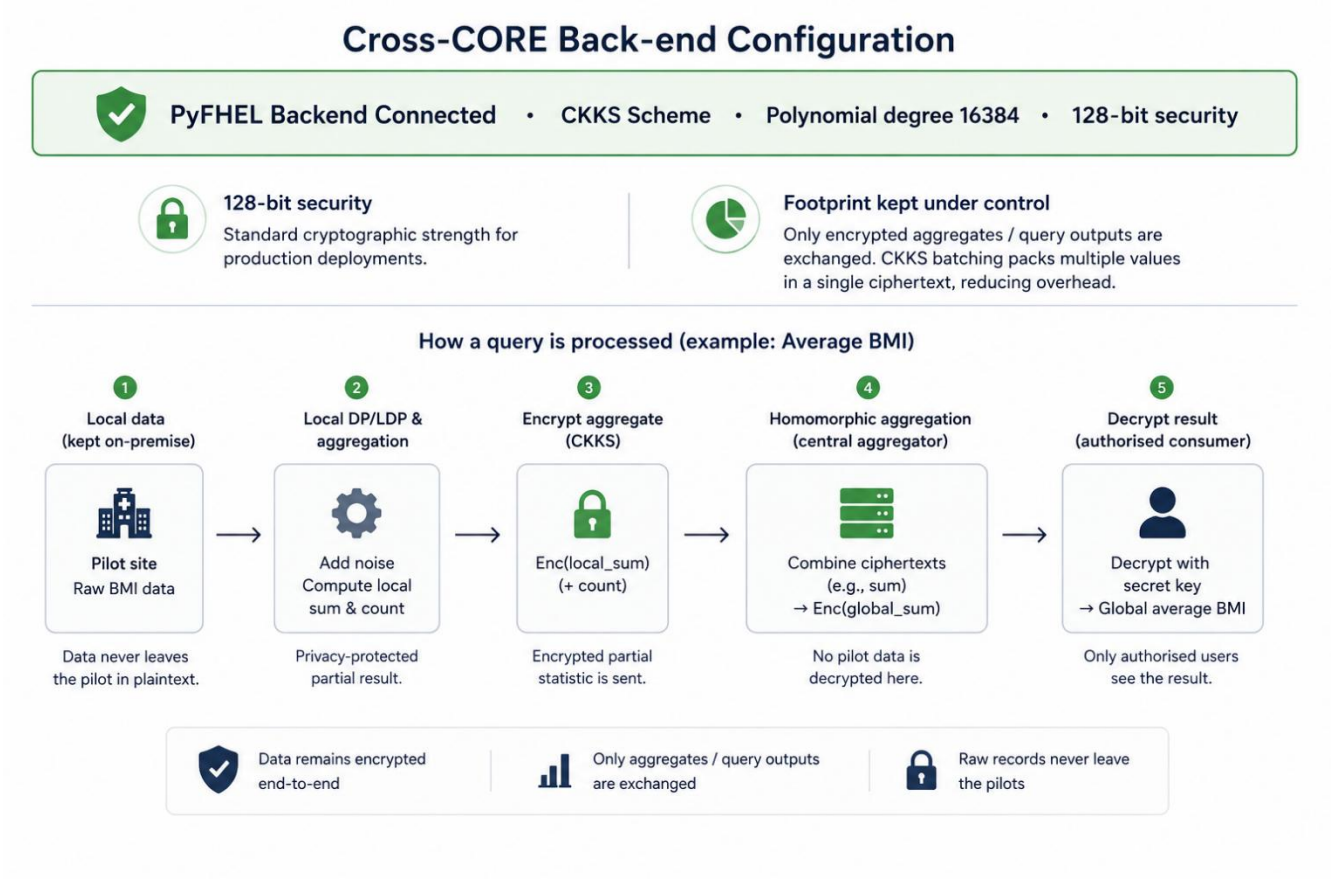

Figure 15: Cross-CORE PyFHEL/CKKS back-end configuration and query flow, showing local privacy protection, encrypted partial statistics, homomorphic aggregation and authorised decryption of the final result.

4.3 How the operator sees it

The HE page is organised in three layers, each answering a different question for a different audience:

[1] HomomorphicEncryption.org, *Homomorphic Encryption Standard v1.1*, Homomorphic Encryption Standardization Consortium, 2018/2019.

© CYberSynchrony, 2026

4.3.1 Layer 1: Concept ("How It Works — The Flow")

This is the explanatory pane aimed at non-cryptographer operators (a hospital data officer, a municipal IT lead). It walks through the canonical pipeline as a chain of boxes:

Plaintext data → Encrypt → Encrypted data → Compute → Encrypted result → Decrypt → Final result

…with a deliberately trivial worked example (a = 5, b = 3, encrypted, added on the server, decrypted, gives 8). The point of the example is not to teach cryptography; it is to make the server-blind property unmistakable to the operator: the central node sees only the two encrypted blobs and the encrypted total, never the numbers 5, 3 or 8.

4.3.2 Layer 2: Interactive validation

Two small widgets let an operator confirm the back-end is alive and behaving correctly *before* trusting it with a real query:

- *Encrypt & Decrypt a Value.* The user types any number, the back-end encrypts it, decrypts it, and shows both the round-tripped value and the residual approximation error (CKKS works on real numbers and therefore introduces a tiny rounding error, surfacing it explicitly is a transparency choice).
- *Homomorphic Addition.* The user types two numbers, the back-end encrypts each separately, adds the two ciphertexts, decrypts only the final sum, and reports it. This is the live demonstration that the server can compute without seeing the inputs — the same property that the per-dataset demos rely on at scale.

These widgets are not toys; they are the same primitives the pilot scenarios use, just exposed one at a time. An operator who is sceptical about whether "the server really cannot see the data" can verify it personally in two clicks before running anything sensitive.

4.3.3 Layer 3 — Per-dataset quick demos (5-record sample)

Four colour-coded scenarios, one per pilot use-case, run a **pure homomorphic-encryption pipeline** end-to-end on a random sample of **five records** drawn from each pilot dataset (Figure 16). This layer deliberately isolates the HE protocol from the rest of the privacy stack: no differential-privacy noise is applied here — the values entering the pipeline are the raw sampled records, and what is being demonstrated is the cryptographic round-trip itself (encrypt at the data owner, sum on the server without decryption, decrypt only the aggregate). The combined DP-then-HE flow that the platform uses for actual cross-border exchanges is exposed on a separate page of the dashboard and is documented below in the next sections.

© CYberSynchrony, 2026

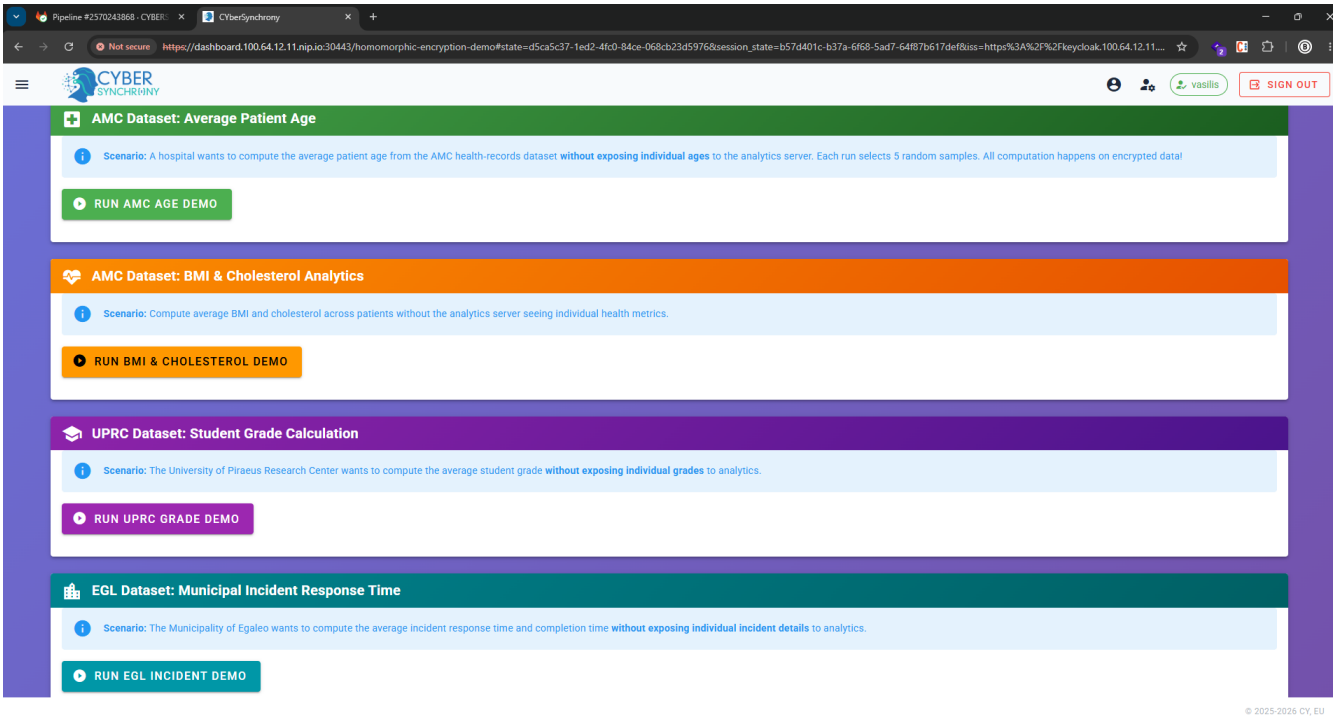

Figure 16: Homomorphic Encryption demo interface showing the available pilot scenarios for privacy-preserving analytics over AMC health records, UPRC student data and EGL municipal incident data.

The sampling is deliberate: each run picks five records at random from the underlying pilot dataset, and at that small scale the dashboard can render every individual encryption, the homomorphic sum and the final decryption step-by-step on screen, so an analyst unfamiliar with HE can follow the cryptographic process as it unfolds, rather than just see a final number drop on the page. A representative AMC run, for example, draws five patient ages at random from the AMC dataset, encrypts each into a ≈ 768 KB ciphertext, sums them server-side without ever decrypting an individual value, decrypts only the aggregate and reports the average, with the total dataset size shown next to the result so the analyst can see exactly which sample was drawn and from how large a population.

The four scenarios in this layer are:

- **HealthCare (AMC): Average Patient Age.** Mean age over five sampled records from the AMC dataset, with each individual age remaining encrypted on the analytics server (Figure 17).
- **Healthcare (AMC): BMI & Cholesterol Analytics.** Joint computation of mean BMI and mean cholesterol on the same random five-record sample, demonstrating that several statistics can be carried in the same encrypted payload (Figure 18).
- **University (UPRC): Student Grade Calculation.** Average degree grade over five sampled student records from UPRC, without revealing individual grades to the analytics tier (Figure 19).
- **Public Administration (EGL): Municipal Incident Response Time.** Average first-action time and average completion time over five sampled incidents from EGL, without exposing individual incident details (Figure 20).

This layer is the pedagogical entry point of the page and it answers the operator's "show me the protocol" question with a fully visible round-trip on real pilot data.

© CYberSynchrony, 2026

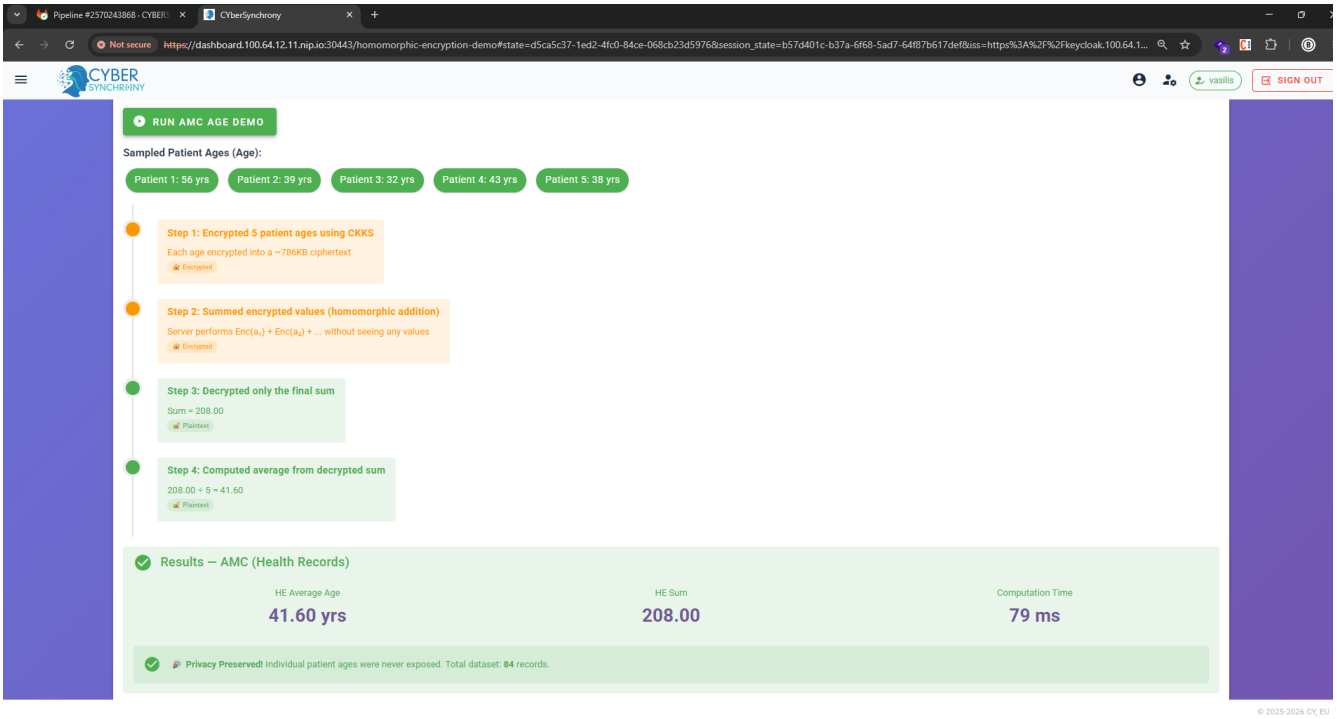

Figure 17: AMC homomorphic encryption demo showing encrypted patient-age aggregation, final-sum decryption and average-age computation without exposing individual records.

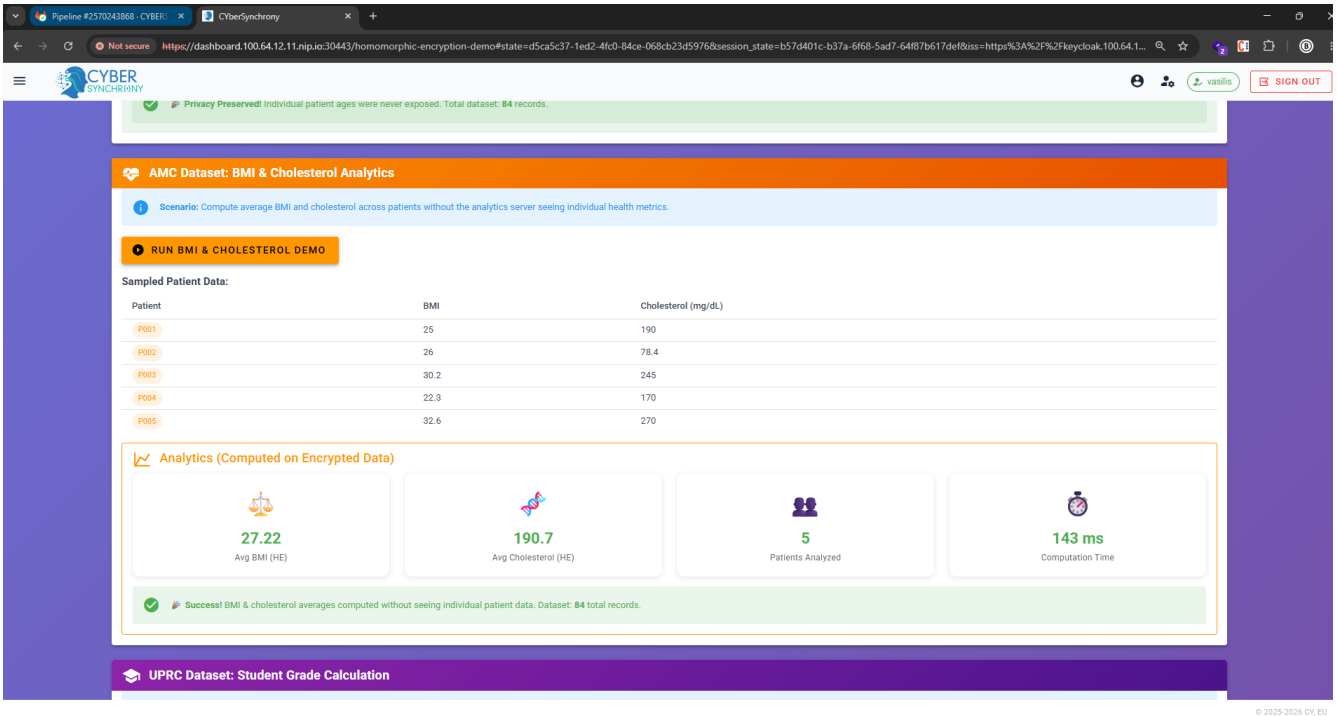

Figure 18: AMC BMI and cholesterol homomorphic encryption demo showing encrypted computation of average health metrics without exposing individual patient records to the analytics server.

© CYberSynchrony, 2026

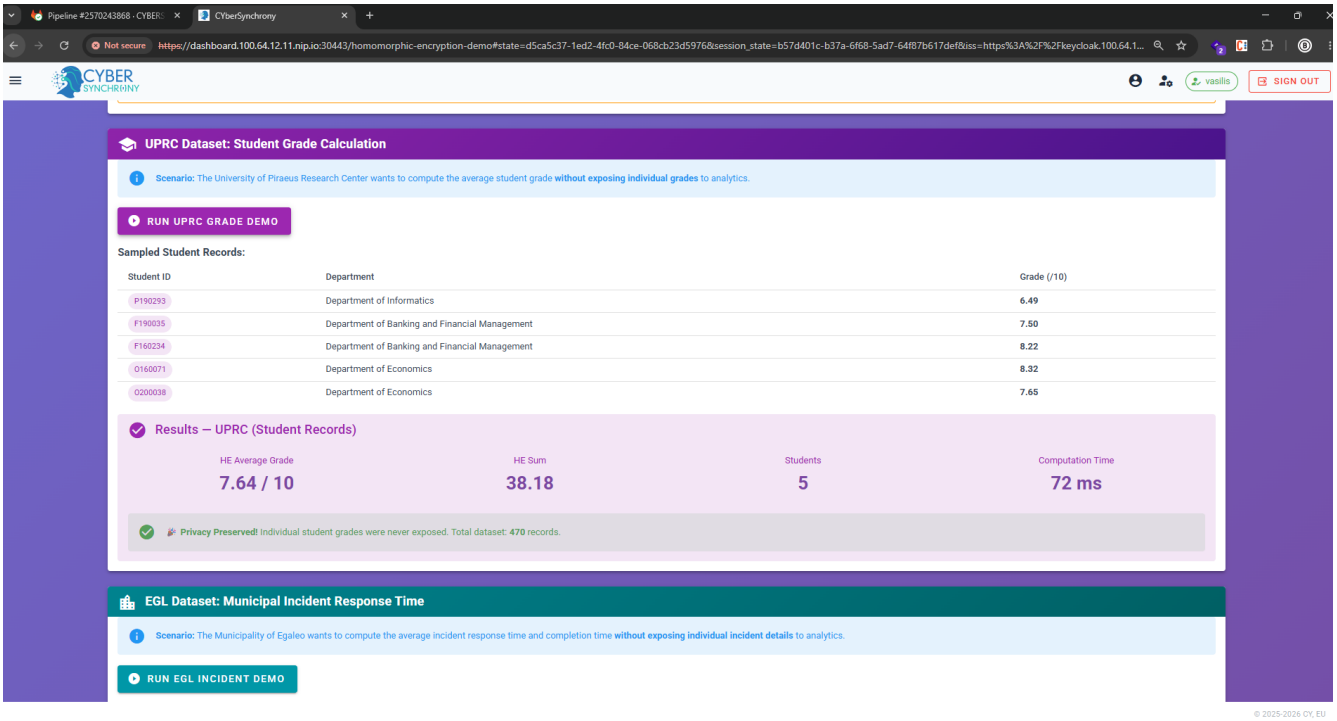

Figure 19: UPRC student-grade homomorphic encryption demo showing encrypted aggregation of student grades and average-grade computation without exposing individual academic records.

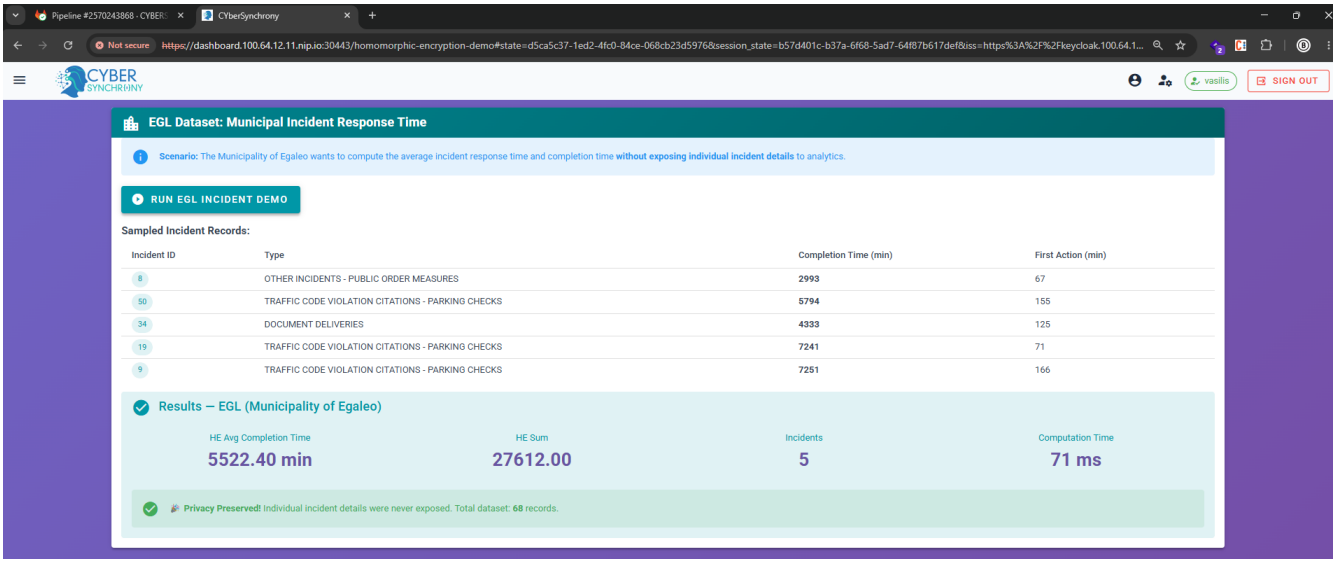

Figure 20: EGL municipal incident homomorphic encryption demo showing encrypted aggregation of incident response and completion-time metrics without exposing individual incident records.

© CYberSynchrony, 2026

4.3.4 Layer 4 — Full-dataset runs ("Process Every Record", HE only).

The same four pilot scenarios are exposed a second time, in a dedicated "Full Dataset Runs" panel (Figure 21) that performs **real CKKS encryption on every valid record of the underlying dataset** rather than on a five-record sample. As in Layer 3, this panel exercises the HE engine in isolation: every individual value is encrypted into a ≈ 768 KB ciphertext on the data owner's node, all ciphertexts are aggregated by the server in ciphertext space, and only the final aggregated sum is decrypted. The dashboard surfaces a yellow "Heavy computation" warning to set expectations — full runs take from a few seconds to several minutes depending on dataset size — and lists the first ten records for reference, so the operator can sanity-check the input before launching.

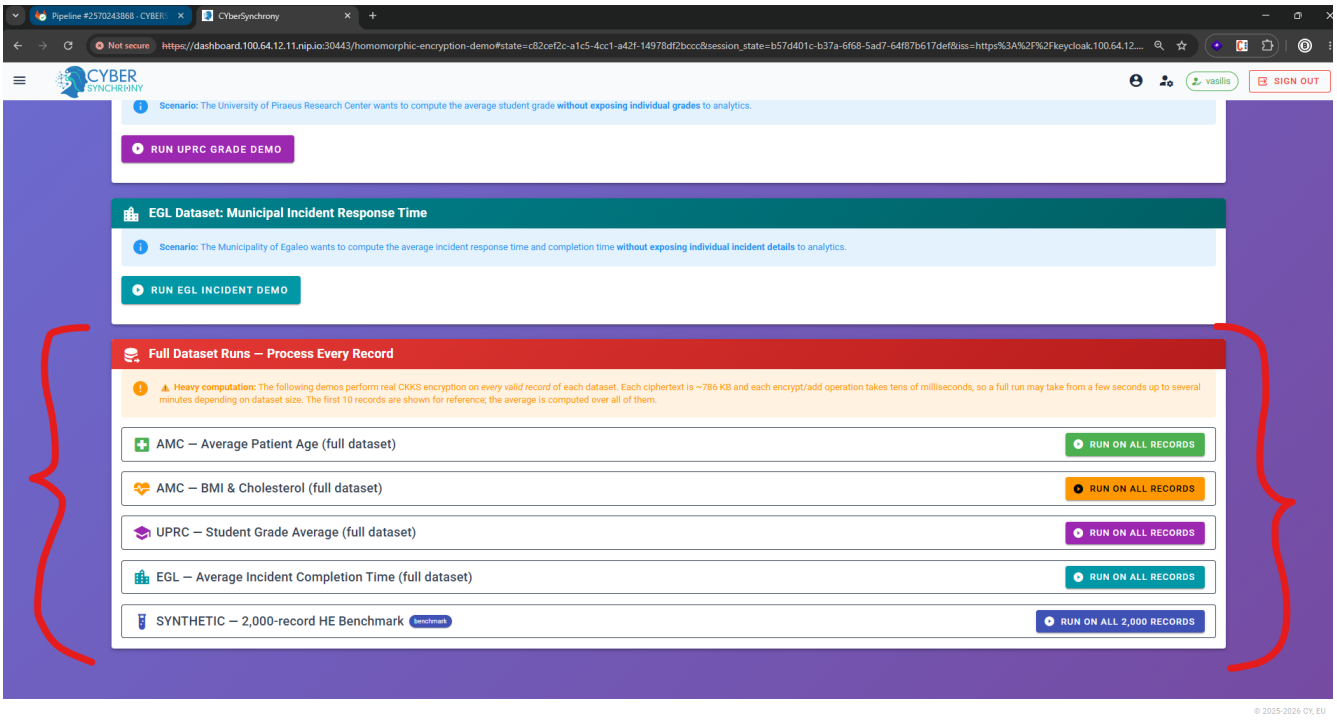

Figure 21: Full-dataset homomorphic encryption runs panel, exposing HE-only pilot scenarios that encrypt and aggregate every valid record while warning operators about the higher computational cost of processing complete datasets.

For every full run the platform reports a complete **Performance Metrics** panel that documents what was actually executed cryptographically. The value of exposing these metrics is twofold: first, they make the cryptographic effort visible (one decryption regardless of dataset size — the central node never decrypts an individual value), and second, they let the consortium answer questions like how many CKKS encryptions and homomorphic additions were performed, how long the full run took end-to-end, what the average cost per record was, what throughput was achieved, and how much ciphertext volume was produced in memory for the selected dataset.

The four full-dataset entries in this layer are AMC Average Patient Age (Figure 22), AMC BMI & Cholesterol (Figure 23), UPRC, Student Grade Average (Figure 24) and EGL Average Incident Completion Time (Figure 25). Each maps to a real cross-border information-sharing question the pilots are expected to ask during the M19–M36 validation phase: cardiovascular-risk analysis across hospitals without exporting patient demographics, cross-border educational analytics, and inter-municipal response-time benchmarking without revealing where any specific incident took place.

© CYberSynchrony, 2026

4.3.4.1 *AMC — Average Patient Age (full dataset)*

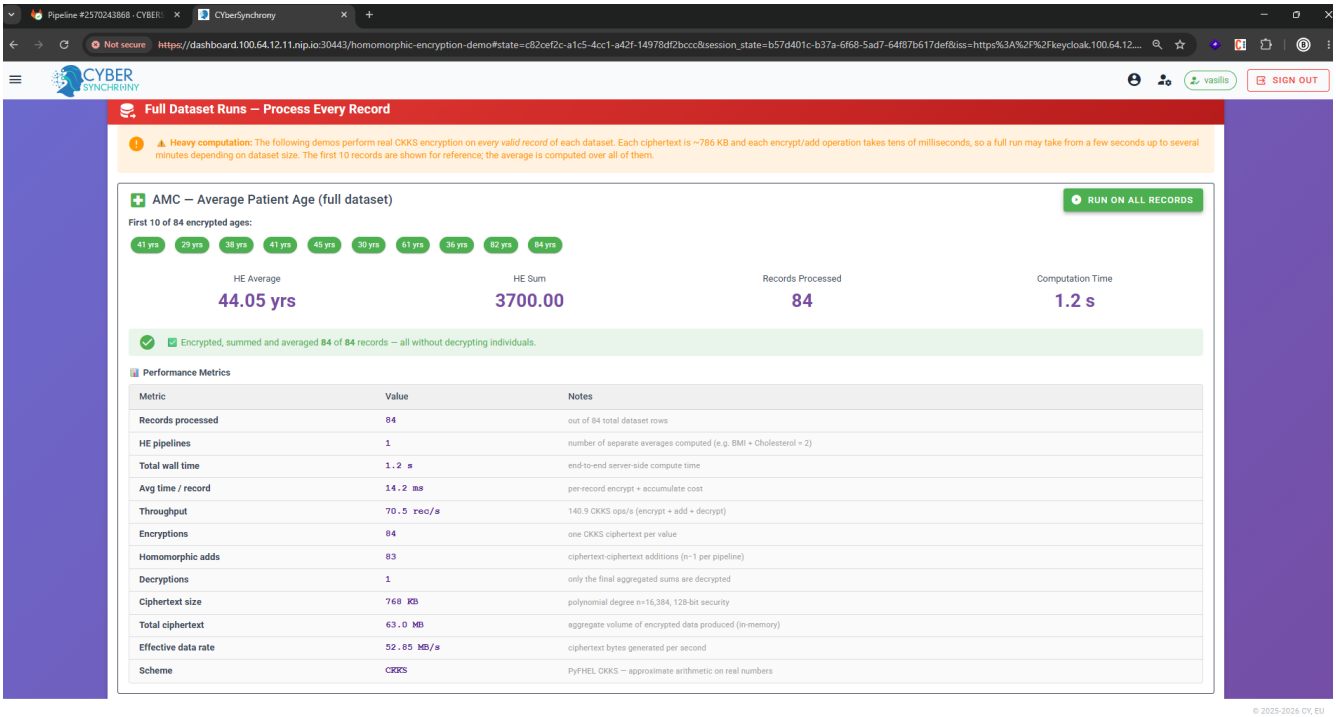

Figure 22: Full-dataset AMC homomorphic encryption run showing encrypted processing of all patient-age records, final aggregate decryption and detailed CKKS performance metrics.

Table 4: CKKS performance metrics for the full-dataset AMC patient-age homomorphic encryption run, including records processed, encryption/addition counts, execution time, ciphertext size and throughput.

Metric	Value	Notes
Records processed	84	Out of 84 total dataset rows
HE pipelines	1	Number of separate averages computed
Total wall time	1.2 s	End-to-end server-side compute time
Avg time / record	14.2 ms	Per-record encrypt and accumulate cost
Throughput	70.5 rec/s	140.9 CKKS ops/s, including encrypt, add and decrypt
Encryptions	84	One CKKS ciphertext per value
Homomorphic adds	83	Ciphertext-to-ciphertext additions, n−1 per pipeline
Decryptions	1	Only the final aggregated sum is decrypted
Ciphertext size	768 KB	Polynomial degree n=16,384, 128-bit security
Total ciphertext	63.0 MB	Aggregate volume of encrypted data produced in memory

© CYberSynchrony, 2026

Effective data rate	52.85 MB/s	Ciphertext bytes generated per second
Scheme	CKKS	PyFHEL CKKS, approximate arithmetic on real numbers

4.3.4.2 *AMC — BMI & Cholesterol (full dataset)*

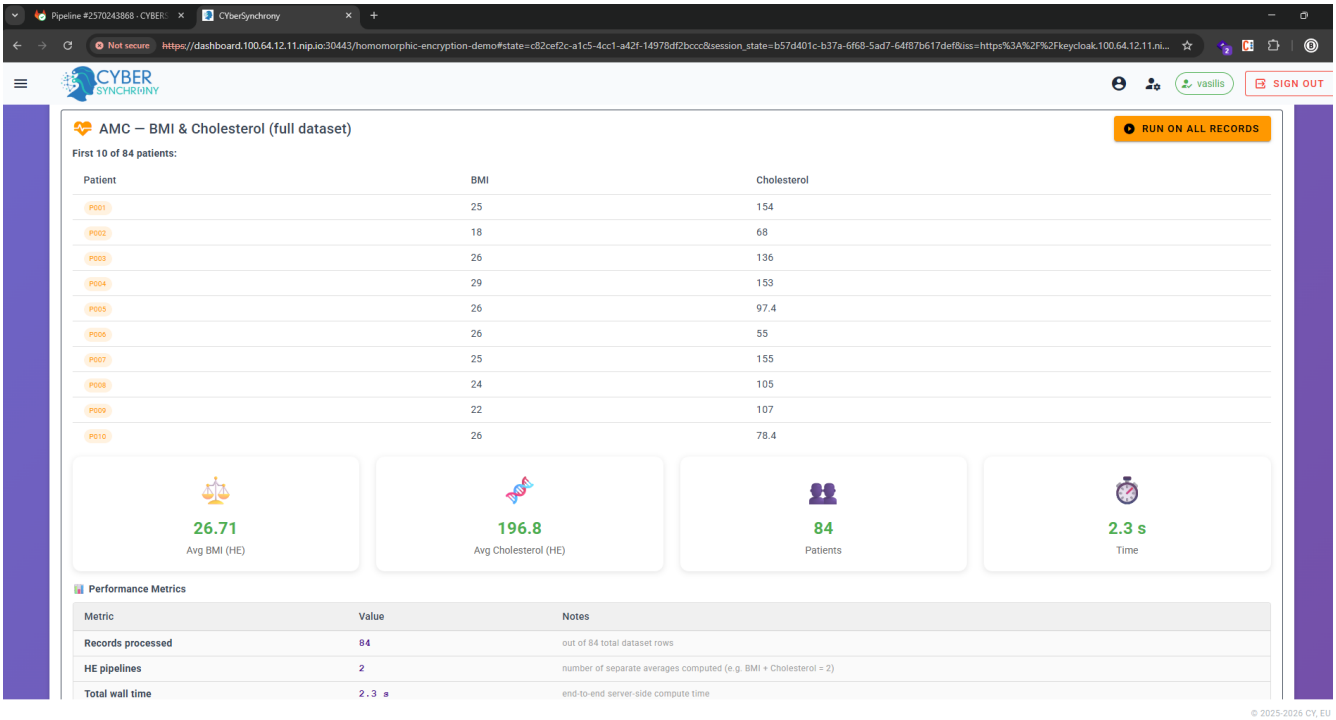

Figure 23: Full-dataset AMC BMI and cholesterol homomorphic encryption run showing encrypted computation of average health metrics and CKKS performance metrics over all patient records.

Table 5: CKKS performance metrics for the full-dataset AMC BMI and cholesterol homomorphic encryption run, including two HE pipelines, encryption/addition counts, execution time, ciphertext volume and throughput.

Metric	Value	Notes
Records processed	84	Out of 84 total dataset rows
HE pipelines	2	Number of separate averages computed, e.g. BMI + Cholesterol
Total wall time	2.3 s	End-to-end server-side compute time
Avg time / record	26.9 ms	Per-record encrypt and accumulate cost
Throughput	37.2 rec/s	148.9 CKKS ops/s, including encrypt, add and decrypt
Encryptions	168	One CKKS ciphertext per value

© CYberSynchrony, 2026

Homomorphic adds	166	Ciphertext-to-ciphertext additions, n−1 per pipeline
Decryptions	2	Only the final aggregated sums are decrypted
Ciphertext size	768 KB	Polynomial degree n=16,384, 128-bit security
Total ciphertext	126.0 MB	Aggregate volume of encrypted data produced in memory
Effective data rate	55.85 MB/s	Ciphertext bytes generated per second
Scheme	CKKS	PyFHEL CKKS, approximate arithmetic on real numbers

4.3.4.3 UPRC — Student Grade Average (full dataset)

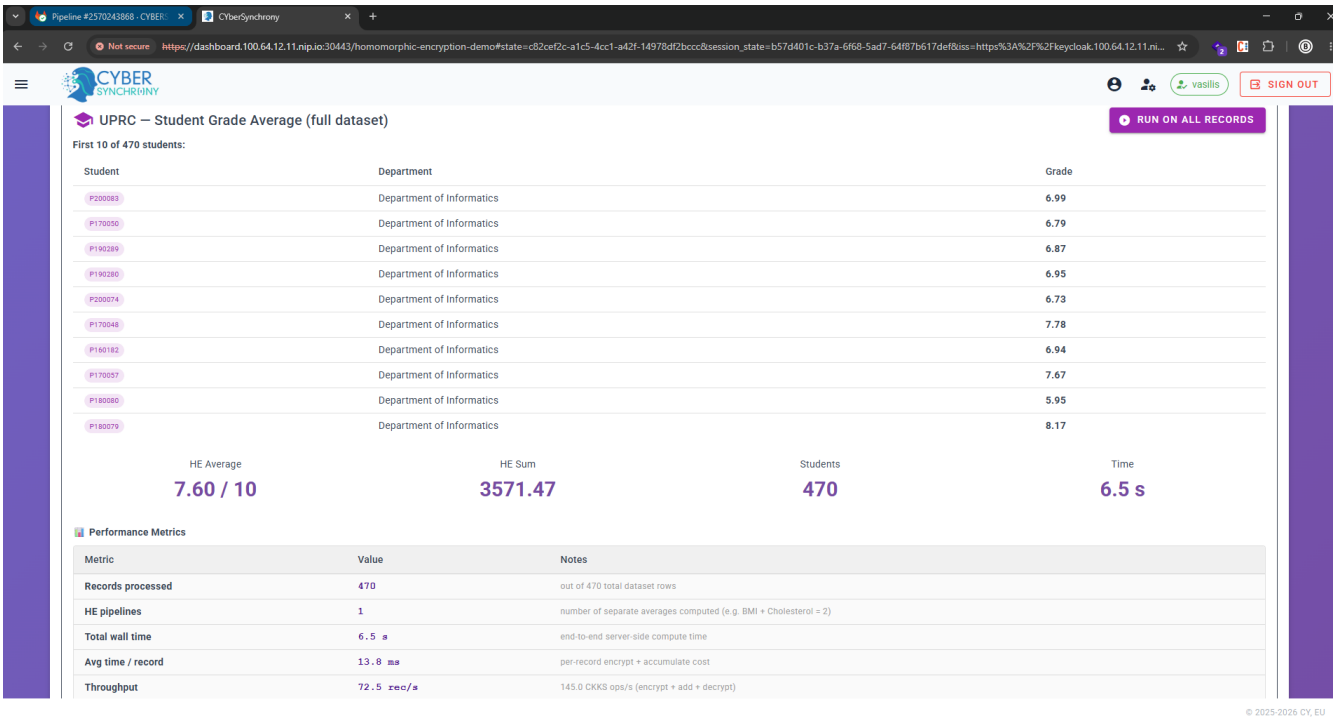

Figure 24: Full-dataset UPRC student-grade homomorphic encryption run showing encrypted aggregation of all student grades, final average-grade computation and CKKS performance metrics.

Table 6: CKKS performance metrics for the full-dataset UPRC student-grade homomorphic encryption run, including 470 encrypted grade records, homomorphic additions, final decryption, execution time, ciphertext volume and throughput.

Metric	Value	Notes
Records processed	470	Out of 470 total dataset rows
HE pipelines	1	Number of separate averages computed

© CYberSynchrony, 2026

Total wall time	6.5 s	End-to-end server-side compute time
Avg time / record	13.8 ms	Per-record encrypt and accumulate cost
Throughput	72.5 rec/s	145.0 CKKS ops/s, including encrypt, add and decrypt
Encryptions	470	One CKKS ciphertext per value
Homomorphic adds	469	Ciphertext-to-ciphertext additions, n−1 per pipeline
Decryptions	1	Only the final aggregated sum is decrypted
Ciphertext size	768 KB	Polynomial degree n=16,384, 128-bit security
Total ciphertext	352.5 MB	Aggregate volume of encrypted data produced in memory
Effective data rate	54.36 MB/s	Ciphertext bytes generated per second
Scheme	CKKS	PyFHEL CKKS, approximate arithmetic on real numbers

4.3.4.4 EGL — Average Incident Completion Time (full dataset)

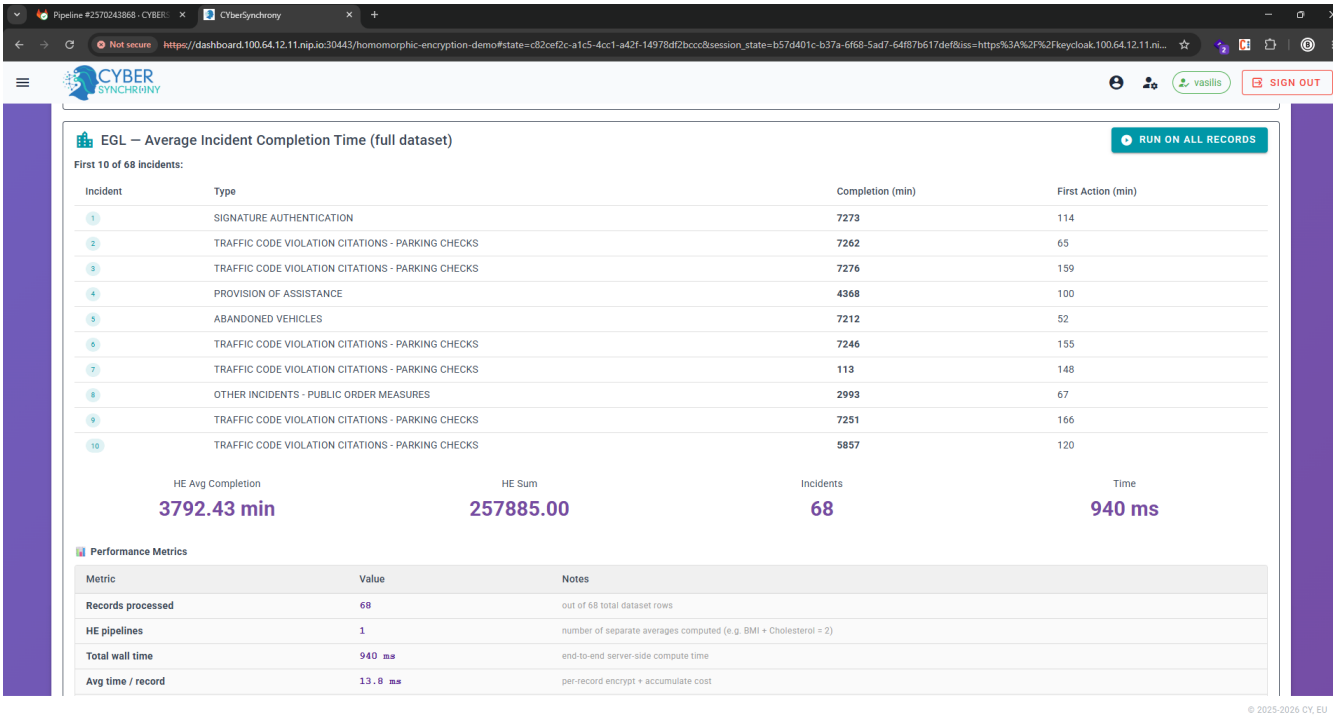

Figure 25: Full-dataset EGL municipal incident homomorphic encryption run showing encrypted aggregation of all incident completion-time records, final average computation and CKKS performance metrics.

Table 7: CKKS performance metrics for the full-dataset EGL municipal incident completion-time homomorphic encryption run, including 67 encrypted records, homomorphic additions, final decryption, execution time, ciphertext volume and throughput.

Metric	Value	Notes
Records processed	67	Out of 67 total dataset rows
HE pipelines	1	Number of separate averages computed
Total wall time	940 ms	End-to-end server-side compute time
Avg time / record	13.8 ms	Per-record encrypt and accumulate cost
Throughput	72.3 rec/s	144.7 CKKS ops/s, including encrypt, add and decrypt
Encryptions	67	One CKKS ciphertext per value
Homomorphic adds	67	Ciphertext-to-ciphertext additions, n−1 per pipeline
Decryptions	1	Only the final aggregated sum is decrypted
Ciphertext size	768 KB	Polynomial degree n=16,384, 128-bit security
Total ciphertext	51.0 MB	Aggregate volume of encrypted data produced in memory
Effective data rate	54.26 MB/s	Ciphertext bytes generated per second
Scheme	CKKS	PyFHEL CKKS, approximate arithmetic on real numbers

4.3.5 Layer 5 — Synthetic 2000-record HE benchmark.

The final entry on the page is an explicit **scaling benchmark** (Figure 26) that runs the same UPRC student-grade pipeline against a synthetic dataset of 2 000 records (≈ 4× the production UPRC scale used for the demo subset), and reports the same Performance Metrics panel as Layer 4. A representative run produces an HE average of 7.44 / 10 over 2 000 students with HE sum 14871.32, completed in 27.6 s of wall time at 72.4 rec/s, with 2 000 encryptions, 1 999 homomorphic additions, exactly one decryption (the final aggregate), 768 KB per ciphertext and ≈ 1 500 MB of aggregate ciphertext volume produced in memory.

The purpose of this layer is not to demonstrate a use-case, since there is no real pilot behind the synthetic data, but to give the consortium a **reproducible scaling reference** for the chosen PyFHEL parameter set. Because the pipeline is byte-for-byte identical to the UPRC scenario, the run shows directly how the HE engine scales with input size on the deployment hardware: the throughput remains roughly constant (≈ 72 records per second), the ciphertext volume grows linearly with the number of records, and the number of decryptions stays at one regardless of dataset size. That last property (one decryption per query, independent of how many pilots contribute or how many records each pilot holds) is the operational signature of secure aggregation, and it is what makes the architecture viable as the platform scales beyond the three reference pilots.

© CYberSynchrony, 2026

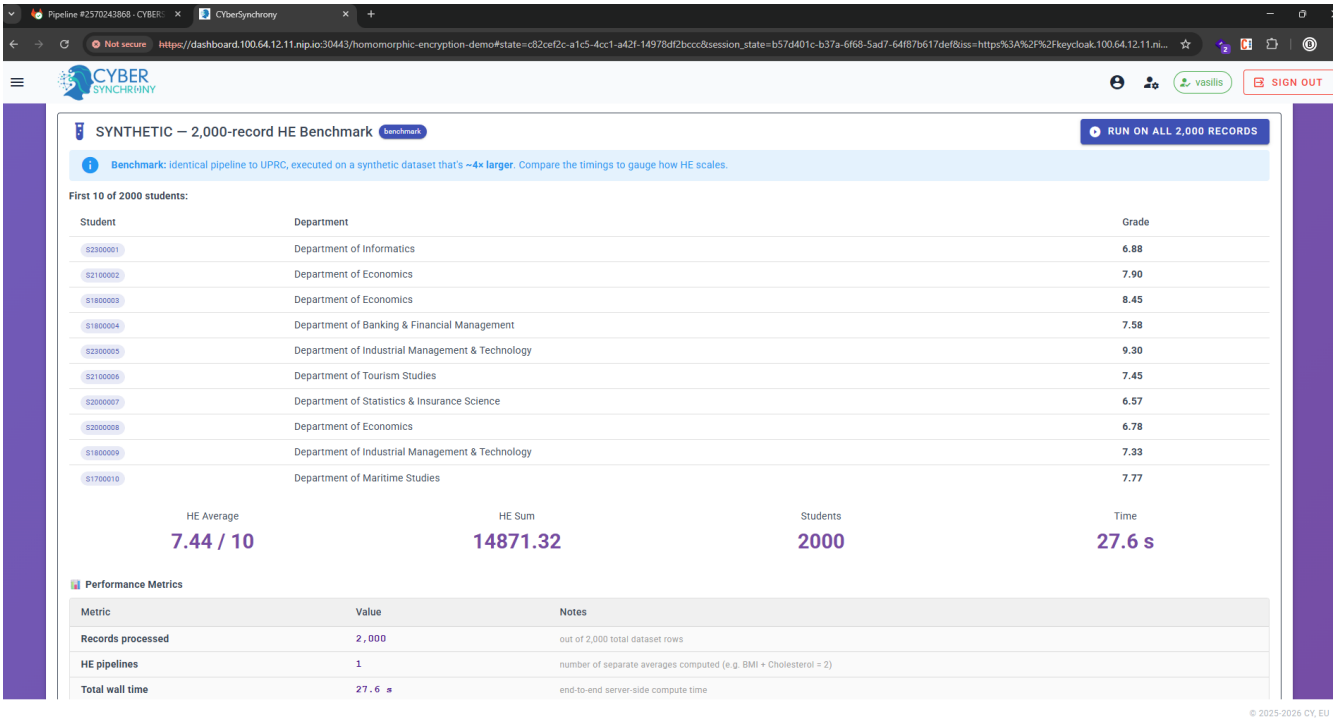

Figure 26: Synthetic 2,000-record homomorphic encryption benchmark showing full-dataset encrypted aggregation, average-grade computation and CKKS performance metrics at larger scale.

Table 8: CKKS performance metrics for the synthetic 2,000-record homomorphic encryption benchmark, showing encrypted processing throughput, ciphertext volume, execution time and final-decryption count.

Metric	Value	Notes
Records processed	2,000	Out of 2,000 total dataset rows
HE pipelines	1	Number of separate averages computed
Total wall time	27.6 s	End-to-end server-side compute time
Avg time / record	13.8 ms	Per-record encrypt and accumulate cost
Throughput	72.4 rec/s	144.8 CKKS ops/s, including encrypt, add and decrypt
Encryptions	2,000	One CKKS ciphertext per value
Homomorphic adds	1,999	Ciphertext-to-ciphertext additions, n−1 per pipeline
Decryptions	1	Only the final aggregated sum is decrypted
Ciphertext size	768 KB	Polynomial degree n=16,384, 128-bit security
Total ciphertext	1,500.0 MB	Aggregate volume of encrypted data produced in memory
Effective data rate	54.31 MB/s	Ciphertext bytes generated per second

© CYberSynchrony, 2026

Scheme	CKKS	PyFHEL CKKS, approximate arithmetic on real numbers

Across all pilot layers the same pattern repeats: each pilot encrypts on its own VM4, ships only ciphertexts to the central aggregator on VM2, the central node performs the encrypted computation without ever decrypting, the authorised consumer decrypts the final result locally, and a SHA-256 fingerprint of the exchange is anchored on Hyperledger Fabric for audit. The five-layer structure of the page mirrors the way the platform is meant to be approached operationally (concept, validation, guided demo, production run, scaling reference), and is itself part of the deliverable's contribution: it is the user-facing surface through which a non-cryptographer operator at AMC, UPRC or EGL can understand, verify and run cross-border privacy-preserving analytics without leaving the dashboard.

© CYberSynchrony, 2026

5 Blockchain Anchoring and Cross-CORE Integration

5.1 Where the Ledger Sits in the Pipeline

Chapters 4 and 5 described the two privacy primitives that protect a cross-border request while it is computed: Local Differential Privacy noises every record on the data owner's node before anything leaves VM4, and PyFHEL/CKKS homomorphic encryption lets the central aggregator on VM2 sum the per-pilot contributions without ever decrypting them. Neither primitive, on its own, produces evidence. DP guarantees that the published number cannot re-identify an individual; HE guarantees that the aggregator never sees a plaintext — but if, a month later, a pilot disputes that it ever took part, or an auditor asks whether the published average was really the one the consumer decrypted, neither DP nor HE can answer.

The blockchain layer is the evidence plane that closes this gap. It runs alongside the DP→HE data path, not inside it: the actual ciphertexts (≈ 768 KB each) travel directly VM4 → VM2 over HTTPS, while the ledger receives only SHA-256 fingerprints plus structural metadata. The result is a permissioned, tamper-evident trail in which every cross-border exchange is provable after the fact, yet no plaintext value, no ciphertext and no partial aggregate is ever written on-chain.

The canonical request therefore has, in full, three planes running in parallel:

- DATA PLANE (HTTPS): VM4 – ciphertext → VM2 - aggregate ciphertext → consumer
- PRIVACY PLANE: LDP @ VM4 → CKKS encrypt @ VM4 → HE add @ VM2 → decrypt @ consumer
- EVIDENCE PLANE (Fabric): request anchor → per-pilot partial anchors → aggregate+result anchor

This section documents the evidence plane: the network, the data model, the chaincode functions and REST endpoints purpose-built for the DP→HE→SHA-256 flow (several of which are new in v2 and did not exist in the D5.1 generic key-value contract), and the end-to-end anchoring and verification sequences with concrete JSON payloads.

5.2 Network Topology

Table 9: Hyperledger Fabric network topology supporting the evidence plane, including peers, orderer, channel, chaincode, Explorer, and API gateway components used for anchoring and audit verification.

Component	Identity	Endpoint	Role in the evidence plane
Peer 0, Org 1	peer0.org1 / Org1MSP	peer0-org1:7051	Endorses and commits anchors signed by Org 1 pilots
Peer 0, Org 2	peer0.org2 / Org2MSP	peer0-org2:9051	Endorses and commits anchors signed by Org 2 pilots
Orderer	OrdererMSP	orderer:7050	Raft ordering and block cutting
Channel	cybersynchrony-channel	—	Application channel for all anchors
Chaincode	crosscore-cc (TypeScript)	Committed on both peers	Implements the data model and functions
Explorer	Client 1.1.8 (Fabric 2.3/v1.4)	explorer.100.64.12.11.nip.io:30443	Read-only audit UI

© CYberSynchrony, 2026

API gateway	Node.js / Express	/api/* behind the dashboard ingress	JWT-protected REST front-door

Every contributor's signing identity is its MSP: a pilot on VM4 that submits a partial anchor does so through the gateway, which uses that organisation's enrolment certificate, so the resulting Fabric transaction is cryptographically attributable to Org1MSP or Org2MSP. Adding a third pilot organisation later is a channel-update transaction with no chaincode change.

5.3 Data Model — Designed for the DP →HE Flow

The v2 chaincode replaces the single generic record of D5.1 with three purpose-built record types, one per stage of the request lifecycle. All three are stored as JSON under a string key in Fabric state, and every type carries the same requestId so the whole exchange is reconstructable from any entry point.

5.3.1 RequestAnchor: The request descriptor

Written once, by the consumer, when a cross-border request is opened. It declares what was asked and what privacy budget was advertised before any data moves (Figure 27).


```
{
  "recordType": "RequestAnchor",
  "id": "REQ-2c4f9a7e",
  "requestId": "req-2c4f9a7e-3a18-4d2b-9b0e-3f9a5d12b6c1",
  "operation": "BoundedMean",
  "attribute": "age",
  "datasets": ["AMC"],
  "contributors": ["amc-vm4-01"],
  "dpMethod": "LDP",
  "heScheme": "CKKS",
  "hePolyDegree": 16384,
  "epsilonBudget": 0.10,
  "createdBy": "consumer.csirt@cybersynchrony.eu",
  "owner": "Org1MSP",
  "createdAt": "2026-06-08T10:14:18.241Z"
}
```


Figure 27: Example RequestAnchor JSON record for opening a bounded-mean request over the age attribute in the AMC pilot dataset, including the DP method, HE scheme, privacy budget and creator metadata.

5.3.2 PartialAnchor — one per contributing pilot

Written by each pilot on VM4 *after* it has (1) applied Local DP, (2) CKKS-encrypted its noised partial, and (3) computed SHA-256 of that ciphertext. It records the fingerprint of the encrypted partial plus exactly the DP/HE provenance needed to audit it, and never the value itself (Figure 28).

© CYberSynchrony, 2026

```
{
  "recordType":      "PartialAnchor",
  "id":              "PART-REQ-2c4f9a7e-amc",
  "requestId":       "req-2c4f9a7e-3a18-4d2b-9b0e-3f9a5d12b6c1",
  "contributorId":   "amc-vm4-01",
  "dataset":         "AMC",
  "operation":       "BoundedMean",
  "attribute":       "age",
  "recordCount":     84,
  "dpMethod":        "LDP",
  "dpMechanism":     "Laplace",
  "epsilonUsed":     0.10,
  "sensitivity":     0.94,
  "heScheme":        "CKKS",
  "partialCipherHash": "9f3c8b7a4e1d2f6c5b8a7d9e0c1f2a3b4c5d6e7f8091a2b3c4d5e6f7081920a3",
  "cipherBytes":     786432,
  "producedAt":      "2026-06-08T10:14:22.298Z",
  "ip":              "100.64.12.14",
  "owner":           "Org1MSP",
  "createdAt":       "2026-06-08T10:14:22.612Z"
}
```


Figure 28: Example PartialAnchor JSON record generated by the AMC pilot after applying Local Differential Privacy, encrypting the noised partial with CKKS, and anchoring only the SHA-256 fingerprint and DP/HE provenance metadata on-chain.

For categorical queries (Count Male Patients, Count Parking Violations) the same record is used with dpMethod: "LDP", dpMechanism: "RandomisedResponse" and two extra fields, rrAlpha and rrCategories (κ), mirroring the κ-ary RR mechanism so the audit trail records exactly which DP primitive produced each partial.

5.3.3 ResultAnchor — the final commitment

Written once, by the consumer, after the aggregator has homomorphically summed the partials and the consumer has decrypted the aggregate. It binds together the hash of the aggregate ciphertext (proves the aggregator did not tamper with the inputs in transit) and the hash of the canonical decrypted result (proves the published number is the one decrypted) (Figure 29).

© CYberSynchrony, 2026

```
{
  "recordType":         "ResultAnchor",
  "id":                 "RES-REQ-2c4f9a7e",
  "requestId":          "req-2c4f9a7e-3a18-4d2b-9b0e-3f9a5d12b6c1",
  "operation":          "BoundedMean",
  "attribute":          "age",
  "partialAnchors":     ["dataset-amc-1", "dataset-amc-2"],
  "aggregateCipherHash": "b1c2d3e4f50617283940a5b6c7d8e9f00112233445566778899aabbccddeeff",
  "resultHash":         "c2d3e4f5061728394050a6b7c8d9eafb11223344556677889aabbccddeeff00",
  "totalRecordCount":   554,
  "totalEpsilon":       0.20,
  "heAdds":             553,
  "decryptions":        1,
  "decryptedAt":        "2026-06-08T10:14:26.040Z",
  "owner":              "Org1MSP",
  "createdAt":          "2026-06-08T10:14:26.305Z"
}
```


Figure 29: Example ResultAnchor JSON record committing the final bounded-mean age result, linking two AMC dataset partial anchors with the aggregate ciphertext hash, canonical result hash, total record count, privacy budget and HE aggregation metadata.

ResultHash is the SHA-256 of the canonical-JSON encoding of the published result (sorted keys, no extra whitespace), so any verifier re-hashing the published number computes the identical digest.

All three types emit Fabric events on commit, RequestOpened, PartialAnchored, ResultAnchored, which the dashboard's audit panel subscribes to in order to update live without polling.

5.4 What the Chaincode Exposes: Function Surface

The chaincode exposes the core functions required to support the evidence plane of the platform. Its function surface is organised around two categories: submit transactions, which create or modify ledger state, and evaluate calls, which retrieve records, check existence, or inspect historical changes. This mapping keeps the dashboard interactions directly traceable to Fabric operations and makes every audit action reproducible at chaincode level.

Table 10: Chaincode function surface exposed by the evidence plane, covering ledger record management, anchored hash storage, read-only queries, ownership-based filtering, time-range audit retrieval and Fabric history inspection.

Function	Type	Purpose
InitLedger	submit	One-shot bootstrap with an initial ledger record
CreateData(id, data, owner)	submit	Anchor a new application payload
ReadData(id)	evaluate	Read one application payload
UpdateData(id, newData)	submit	Replace the data field of an existing ledger record

© CYberSynchrony, 2026

DeleteData(id)	submit	Mark an existing ledger record as deleted
DataExists(id)	evaluate	Boolean existence check
GetAllData()	evaluate	Stream all application payload records
QueryDataByOwner(owner)	evaluate	Filter application payload records by owner
GetDataHistory(id)	evaluate	Full Fabric history for one ledger key
StoreLogHash(logHash, metadataJson, owner)	submit	Anchor a hash and its metadata with an auto-generated ID
ReadLogHash(id)	evaluate	Read one anchored hash record
LogHashExists(id)	evaluate	Boolean existence check
QueryLogHashByHashValue(hashValue)	evaluate	Find records by hexadecimal hash value
QueryLogHashesByMachine(machineId)	evaluate	Filter anchored hash records by contributor machine
QueryLogHashesByIP(ip)	evaluate	Filter anchored hash records by originating IP
QueryLogHashesByTimeRange(start, end)	evaluate	Filter anchored hash records by fromTime / toTime window
QueryLogHashesByOwner(owner)	evaluate	Filter anchored hash records by owner organisation
GetAllLogHashes()	evaluate	Stream all anchored hash records
GetLogHashHistory(id)	evaluate	Full Fabric history for one anchored hash record

"Submit" functions go through the full endorsement → ordering → commit cycle and produce an on-chain transaction; "evaluate" functions are local reads that do not touch consensus.

The chaincode validates inputs aggressively: hashes must be hexadecimal, JSON must parse, and the six required metadata keys (timestamp, fromTime, toTime, machineType, machineId, ip) must all be present, otherwise the transaction is rejected at endorsement time and never reaches the orderer.

5.5 The REST Gateway: How the Dashboard Talks to the Ledger

End-users never invoke chaincode directly. Every Cross-CORE component (the DP/HE engine on VM4, the central aggregator on VM2, the dashboard ingress on VM1) talks to a single Node.js / Express API gateway that wraps the Fabric SDK. The gateway exposes a thin, REST surface and translates each request into the corresponding submitTransaction / evaluateTransaction call on the channel.

© CYberSynchrony, 2026

5.5.1 Authentication

Identity is handled by Keycloak (keycloak.100.64.12.11.nip.io:30443, realm CYberSynchrony). A caller first obtains an OIDC JWT from the realm's token endpoint and then presents it as a Bearer token on every gateway call. The gateway only validates this token, it never issues or stores credentials, and reads the organization claim (Fabric MSP) and realm_access.roles claim (which operations are allowed) to authorise and attribute each request.

Table 11: Authentication endpoints and bearer-token usage for the Cross-CORE API gateway, showing how Keycloak-issued JWTs are obtained, refreshed and validated before access to protected /api/crosscore/* operations.

Method & Endpoint	Description
POST /realms/CYberSynchrony/protocol/openid-connect/token	Keycloak token endpoint. Exchanges user credentials using grant_type=password and client_id=cyber-synchrony-platform for a signed JWT access_token.
POST /realms/CYberSynchrony/protocol/openid-connect/token *(refresh)*	Uses the same endpoint with grant_type=refresh_token to renew an expired access token without re-authenticating the user.
Authorization: Bearer <access_token>	Header included in every /api/crosscore/* call. The gateway validates the token signature and expiry time, then maps the organisation claim to the submitting MSP.

5.5.2 Lifecycle endpoints

Table 12: Lifecycle endpoints of the Cross-CORE API gateway, covering request opening, partial anchoring, final result anchoring and request closure across the DP/HE evidence workflow.

Method & Endpoint	Stage	Description
POST /api/crosscore/request	Step 1	Opens a request and writes a RequestAnchor, including the operation, attribute, datasets, DP method, HE scheme and ε-budget.
POST /api/crosscore/request/:requestId/partial	Step 2	Anchors one contributor's partial: the SHA-256 hash of its CKKS ciphertext, together with the required DP/HE provenance metadata. The ciphertext itself is never sent through this endpoint.
POST /api/crosscore/request/:requestId/result	Step 4	Anchors the final commitment after aggregation, including the aggregate ciphertext hash and the canonical-JSON result hash, after the chaincode checks the linked partials and the ε-budget.
PATCH /api/crosscore/request/:requestId/close	Step 5	Closes the request so that no further partials can be accepted. The operation is rejected if no ResultAnchor exists yet.

© CYberSynchrony, 2026

5.5.3 Verification endpoints

Table 13: Verification endpoints of the Cross-CORE API gateway, supporting complete request-bundle retrieval, partial ciphertext re-hashing, final result verification, live privacy-budget inspection and Fabric history auditing.

Method & Endpoint	Description
GET /api/crosscore/request/:requestId	Returns the complete request bundle in one read, including the descriptor, all linked partial anchors and the final result anchor.
POST /api/crosscore/verify/partial	Performs a re-hash check of a contributor's ciphertext against the anchored partialCipherHash and returns { match }.
POST /api/crosscore/verify/result	Performs a re-hash check of the published canonical-JSON result against the anchored resultHash and returns { match, anchoredTxId }.
GET /api/crosscore/request/:requestId/budget	Returns the live privacy-budget status as { epsilonBudget, epsilonSpent, epsilonRemaining }, computed from the linked partial anchors.
GET /api/crosscore/request/:requestId/history	Returns the full Fabric history of every record under the request, including txId, timestamp and IsDelete status.

5.5.4 Audit & discovery endpoints

Table 14: Audit and discovery endpoints of the Cross-CORE API gateway, enabling contributor, dataset owner, time-range based retrieval of anchored evidence records, together with aggregate dashboard statistics.

Method & Endpoint	Description
GET /api/crosscore/contributor/:contributorId	Returns every partial anchor signed by a specific pilot machine across all requests.
GET /api/crosscore/dataset/:dataset	Returns all anchors involving a specific dataset, such as AMC, UPRC or EGL.
GET /api/crosscore/owner/:mspId	Returns all anchors owned by a specific MSP, such as Org1MSP or Org2MSP.
GET /api/crosscore/anchors?from=:ts&to=:ts	Performs a time-range query over producedAt, supporting incident-driven audit investigations.
GET /api/crosscore/anchors?operation=:op&attribute=:attr	Filters anchors by DP/HE operation and attribute, for example all BoundedMean.age exchanges.
GET /api/crosscore/requests?status=open|closed	Returns a paginated list of requests filtered by lifecycle state.

© CYberSynchrony, 2026

GET /api/crosscore/stats	Returns aggregate dashboard counters, including requests, partials, results, distinct contributors and cumulative ε.

5.5.5 Operational & subscription endpoints

Table 15: Operational and subscription endpoints of the Cross-CORE API gateway, covering gateway health checks, Fabric network status, block metadata retrieval and committed transaction inspection.

Method & Endpoint	Description
GET /api/health	Returns the liveness and readiness status of the gateway and its Fabric connection.
GET /api/crosscore/network	Returns channel information, peer/orderer endpoints, chaincode version and per-peer ledger heights.
GET /api/crosscore/block/:blockNumber	Returns raw block metadata, including data hash, previous-block hash and transaction count.
GET /api/crosscore/tx/:txId	Returns a committed transaction by id, including invoking MSP, invoked function, arguments digest and validation code.

5.6 End-to-End Anchoring Flow for a Cross-Border Request

Putting all together, a single request, for example *"compute the average patient age across two AMC datasets under ε = 0.1 per contributor"* — produces one request descriptor, one partial anchor per contributing dataset, and one final commitment, in that order. In this walkthrough the two contributors are two record sets held by the same pilot, amc-dataset-1 and amc-dataset-2, each anchored independently so that the trail proves which set produced which encrypted partial.

5.6.1 Step 1: Consumer opens the request (one RequestAnchor)

The dashboard calls POST /api/crosscore/request with a unique requestId. This is the on-chain anchor that ties together every later transaction.

© CYberSynchrony, 2026

```
{
  "recordType":    "RequestAnchor",
  "id":            "REQ-2c4f9a7e",
  "requestId":     "req-2c4f9a7e-3a18-4d2b-9b0e-3f9a5d12b6c1",
  "operation":     "BoundedMean",
  "attribute":     "age",
  "datasets":      ["amc-dataset-1", "amc-dataset-2"],
  "contributors": ["amc-vm4-01", "amc-vm4-02"],
  "dpMethod":      "LDP",
  "heScheme":      "CKKS",
  "hePolyDegree": 16384,
  "epsilonBudget": 0.20,
  "owner":         "Org1MSP"
}
```


5.6.2 Step 2: Each dataset anchors its partial (one PartialAnchor per set)

After applying Local DP and PyFHEL/CKKS encryption on VM4, each record set computes SHA-256 (partial_ciphertext) and ships only the fingerprint and the structural metadata to the ledger via POST /api/crosscore/request/:requestId/partial. The ciphertext itself (≈ 768 KB) travels directly to the central aggregator over HTTPS, never through the ledger.

amc-dataset-1's anchor:


```
{
  "recordType":      "PartialAnchor",
  "id":              "PART-REQ-2c4f9a7e-amc-1",
  "requestId":       "req-2c4f9a7e-3a18-4d2b-9b0e-3f9a5d12b6c1",
  "contributorId":   "amc-vm4-01",
  "dataset":         "amc-dataset-1",
  "operation":       "BoundedMean",
  "attribute":       "age",
  "recordCount":     84,
  "dpMethod":        "LDP",
  "dpMechanism":     "Laplace",
  "epsilonUsed":     0.10,
  "sensitivity":     0.94,
  "heScheme":        "CKKS",
  "partialCipherHash": "9f3c8b7a4e1d2f6c5b8a7d9e0c1f2a3b4c5d6e7f8091a2b3c4d5e6f7081920a3",
  "cipherBytes":     786432,
  "producedAt":      "2026-06-08T10:14:22.298Z",
  "ip":              "100.64.12.14",
  "owner":           "Org1MSP"
}
```


amc-dataset-2's anchor:

© CYberSynchrony, 2026

```
{
  "recordType":      "PartialAnchor",
  "id":              "PART-REQ-2c4f9a7e-amc-2",
  "requestId":       "req-2c4f9a7e-3a18-4d2b-9b0e-3f9a5d12b6c1",
  "contributorId":   "amc-vm4-02",
  "dataset":         "amc-dataset-2",
  "operation":       "BoundedMean",
  "attribute":       "age",
  "recordCount":     112,
  "dpMethod":        "LDP",
  "dpMechanism":     "Laplace",
  "epsilonUsed":     0.10,
  "sensitivity":     0.94,
  "heScheme":        "CKKS",
  "partialCipherHash": "a1b2c3d4e5f60718293a4b5c6d7e8f900111223344556677889aabbccddeeff",
  "cipherBytes":     786432,
  "producedAt":      "2026-06-08T10:14:24.902Z",
  "ip":              "100.64.12.15",
  "owner":           "Org1MSP"
}
```


Each transaction is signed by the contributor's peer at endorsement time, and each emits a PartialAnchored event that the dashboard's audit panel picks up live.

5.6.3 Step 3: Central aggregator combines ciphertexts and decrypts

The central node on VM2 sums the two ciphertexts homomorphically — without ever decrypting either — into a single aggregate ciphertext. The authorised consumer decrypts that aggregate locally with its CKKS secret key and obtains the cross-border average (e.g. 41.9 years, computed over 196 records). Only the aggregate ciphertext exists at this point; nothing from Steps 1–2 is recomputed.

5.6.4 Step 4: Consumer anchors the final commitment (one ResultAnchor)

The consumer computes the SHA-256 of the aggregate ciphertext (proves the aggregator did not tamper with the inputs in transit) and the SHA-256 of the canonical-JSON published result (proves the published number matches the one decrypted), then anchors both.

© CYberSynchrony, 2026

```
{
  "recordType":          "ResultAnchor",
  "id":                  "RES-REQ-2c4f9a7e",
  "requestId":           "req-2c4f9a7e-3a18-4d2b-9b0e-3f9a5d12b6c1",
  "operation":           "BoundedMean",
  "attribute":           "age",
  "partialAnchors":      ["PART-REQ-2c4f9a7e-amc-1", "PART-REQ-2c4f9a7e-amc-2"],
  "aggregateCipherHash": "b1c2d3e4f50617283940a5b6c7d8e9f00112233445566778899aabbccddeeff",
  "resultHash":          "c2d3e4f5061728394050a6b7c8d9eafb11223344556677899aabbccddeeff00",
  "totalRecordCount":    196,
  "totalEpsilon":        0.20,
  "heAdds":              195,
  "decryptions":         1,
  "owner":               "Org1MSP"
}
```


After Step 4 the ledger contains: one RequestAnchor (Step 1), two PartialAnchors, one per AMC record set (Step 2), and one ResultAnchor for the aggregate (Step 4), all cross-linked by requestId and partialAnchors, each a separately signed Fabric transaction visible in Hyperledger Explorer. At no point does a plaintext value, a ciphertext or a partial aggregate touch the ledger.

5.7 Real-Time Audit with Hyperledger Explorer

Hyperledger Explorer (client 1.1.8, against Fabric 2.3/v1.4) is deployed at explorer.100.64.12.11.nip.io:30443 and provides the consortium with a read-only operational view of the same ledger the API gateway writes to. Explorer exposes four panes:

Blocks: every block cut by the orderer, with block number, data hash, previous-block hash, transaction count and creation time.

Transactions: every transaction with its txId, channel, chaincode (crosscore-cc), invoked function (OpenRequest, AnchorPartial, AnchorResult), issuing MSP, payload size and timestamp. The four-transaction footprint of the request above, OpenRequest, two AnchorPartial (for amc-dataset-1 and amc-dataset-2), one AnchorResult, appears here as a contiguous burst sharing one requestId.

Nodes: the two peer organisations (peer0-org1:7051, peer0-org2:9051), the orderer (orderer:7050) and their per-peer ledger heights.

Chaincodes: crosscore-cc instantiated on cybersynchrony-channel, with its version, path and the peers it is committed on.

© CYberSynchrony, 2026

6 Performance Metrics of the Cross-CORE Pipeline

6.1 Record-Level Benchmark

Chapters 4, 5 and 6 described the three building blocks of the platform, Differential Privacy, Homomorphic Encryption and Hyperledger Fabric anchoring, and measured each one in isolation. This section reports the performance of the complete pipeline, end to end, from the moment a pilot node begins protecting its records to the moment the cross-border result is committed on the ledger.

All measurements use the production topology of Annex I: the DP engine and the PyFHEL/CKKS encryptor run on the on-premise node VM4 (8 vCPU, 12 GB RAM), the homomorphic aggregation service runs on the cloud node VM2 (12 vCPU, 32 GB), the Fabric orderer runs on VM1 (6 vCPU, 8 GB), the Explorer database on VM3 (12 vCPU, 32 GB), all on Ubuntu 22.04 over a 1 Gbps inter-VM link. Differential privacy uses the PyDP back-end, Homomorphic encryption uses PyFHEL/CKKS with one ≈ 768 KB ciphertext per encrypted value and exactly one decryption per query regardless of dataset size. The ledger commits each request ensuring that each anchoring request becomes part of the immutable audit trail.

The first benchmark corresponds to the record-level visualisation mode used by the dashboard to expose the cost of the full DP–HE round trip at record granularity. The production path, reported in Section 7.1, applies local pre-aggregation at each pilot node and encrypts only the differentially private partials.

The table below reports every stage of the pipeline for each pilot dataset, the differential-privacy cost, the homomorphic-encryption cost, the ciphertext transfer, the single decryption, the SHA-256 fingerprinting and the Hyperledger Fabric anchoring, together with the end-to-end wall-clock time of a complete cross-border request. The synthetic 2,000-record benchmark is included as the final row to show how the pipeline scales beyond the production pilot sizes.

Table 16: Record-level benchmark of the DP–HE–anchoring pipeline across pilot datasets and the synthetic workload

Pilot dataset / query	Records	DP stage (PyDP)	CKKS encryptions	HE wall time	Transfer	Homomorphic adds	Decryption	Anchoring (3 writes)	End-to-end
AMC: Average Age	84	≈ 2.2 ms	84	1.2 s	≈ 0.9 s	83	1 (≈ 12 ms)	≈ 2.0 s	≈ 4.1 s
AMC: BMI & Cholesterol	84 × 2	≈ 4.3 ms	168	2.3 s	≈ 1.8 s	166	2 (≈ 24 ms)	≈ 2.0 s	≈ 6.2 s
UPRC: Student Grade	470	≈ 11.8 ms	470	6.5 s	≈ 5.0 s	469	1 (≈ 12 ms)	≈ 2.0 s	≈ 13.6 s

© CYberSynchrony, 2026

EGL: Incident Completion Time	67	≈ 1.8 ms	67	0.94 s	≈ 0.73 s	67	1 (≈ 12 ms)	≈ 2.0 s	≈ 3.7 s
Synthetic benchmark	2,000	≈ 80 ms	2,000	27.6 s	≈ 21.4 s	1,999	1 (≈ 12 ms)	≈ 2.0 s	≈ 51.2 s

The table makes the entire performance story visible at a glance. The differential-privacy guarantee is essentially free, a few milliseconds even for the largest pilot, and ≈ 80 ms at 2,000 records, so the privacy noise is never the bottleneck. The two real costs are the per-record encryption and the transfer of the resulting ciphertexts, both of which scale linearly with the number of records: ≈ 1.2 s of compute and ≈ 0.9 s of transfer for the 84-record AMC age query, growing to ≈ 27.6 s and ≈ 21.4 s for the synthetic 2,000-record run. The single decryption and the ≈ 2 s anchoring tail stay constant throughout, regardless of dataset size.

6.2 The Production Path

The figures above correspond to the dashboard's visualisation mode, which encrypts every individual value so an operator can watch the HE round-trip unfold. The production anchoring path does not work this way. Because each pilot owns its own records, it is the trusted curator of its own data and can compute a differentially private partial result locally, before any encryption. The cross-border aggregate is linear, so each contributor encrypts one value, its own DP partial sum, not its entire record set, and homomorphic encryption is used only to combine the per-pilot partials without revealing them. For the canonical two-dataset AMC request this means exactly two CKKS encryptions for the whole exchange, independent of how many records each dataset holds.

Table 17: Production-path performance using per-node DP pre-aggregation and one encrypted partial per dataset

Stage	Operation	Cost
1	Per-node Global DP bounded partial sum	≈ 4 ms
2	CKKS encryption: 2 encryptions, one DP partial per dataset	≈ 28 ms
3	Partial transfer: 2 × 768 KB = 1.5 MB, VM4 → VM2	≈ 22 ms
4	Homomorphic aggregation: 1 ciphertext addition	< 1 ms
5	Authorised decryption: 1 decryption producing the cross-border result	≈ 12 ms
6	SHA-256 fingerprinting: 2 partials + aggregate	≈ 3 ms
7	Blockchain anchoring: RequestAnchor, 2 × PartialAnchor, ResultAnchor	≈ 2.0 s
End-to-end	**Full cross-border request, wall-clock**	**≈ 2.1 s**

© CYberSynchrony, 2026

Encrypting all 84 records of one AMC dataset takes ≈ 1.2 s, encrypting the two partial sums of a full two-dataset request takes ≈ 28 ms, a reduction of roughly 40× in cryptographic work, achieved purely by computing the DP partial locally before encrypting. This is the property that makes the architecture scale. Adding a third pilot adds one more DP partial, one more encryption, one more partial transfer and one more PartialAnchor, a few tens of milliseconds, while the dominant ≈ 2 s anchoring tail and the single final decryption stay unchanged. The cost of a cross-border request grows with the number of participating organisations, which is small and bounded, not with the number of records they hold, which is not. Per-node Global DP keeps the privacy cost at a few milliseconds, local pre-aggregation keeps the encryption cost at two ciphertexts, homomorphic encryption keeps the central aggregator blind to every partial, and Hyperledger Fabric provides the tamper-evident audit trail, and the whole exchange completes, end to end, in about the time it takes the ledger to cut a single block.

6.3 KPIs Mapping

6.3.1 KPI4.2 - Number of Cross-Border Actions / Exercises

The Cross-CORE module was designed from the outset so that every cross-border interaction it mediates is a measurable, evidence-backed event: an action in which organisations located in different Member-State perimeters jointly produce a result, while no organisation ever exposes its raw data and every exchange leaves a tamper-evident fingerprint on the Hyperledger Fabric ledger. During the M19–M36 validation phase, the platform instruments ten such cross-border actions / exercises, each one involving at least two contributing parties, and each one fully auditable after the fact through Hyperledger.

The ten scenarios cluster into the two operational capabilities that Cross-CORE exposes and that Chapters 4–6 documented in detail:

- Privacy-preserving cross-border analytics (table below S1–S6). Two or more pilot nodes compute a shared statistic over their combined data without any of them revealing a single record. Each contributing node applies Local Differential Privacy on its own on-premise node (VM4), Laplace noise for numerical attributes, κ-ary Randomised Response for categorical ones, CKKS-encrypts the resulting differentially-private partial, and forwards only the ciphertext to the central aggregator on VM2. The aggregator sums the per-node ciphertexts homomorphically, the authorised consumer decrypts the single aggregate with its CKKS secret key, and the request is anchored on Fabric as one RequestAnchor, one PartialAnchor per contributor and one ResultAnchor. This is the canonical DP → HE → SHA-256 flow: the ledger records who contributed, the ε-budget each contributor consumed, the operation performed and a cryptographic commitment of the published result, never a plaintext value, a ciphertext or a partial aggregate.
- Cross-border threat-intelligence and incident-response actions (S7–S10). Two or more organisations exchange cyber-defence artefacts, vulnerability advisories, hardening playbooks, phishing indicators, exercise evidence. The artefact itself travels over the JWT-protected REST APIs and over MISP only its SHA-256 fingerprint plus the structural metadata (issuing MSP, timestamp, reason field, MISP event ID) is anchored on the ledger. Any partner can later prove what was shared, by whom and when, with the contributing organisation's peer signature attesting authenticity, without the raw artefact ever touching the chain.

© CYberSynchrony, 2026

Table 18: Cross-border scenarios validating KPI 4.2 through privacy-preserving analytics, threat-intelligence sharing and incident-response exercises.

#	Scenario	Contributing pilots	Cross-CORE primitives
S1	UPRC graduation-grade benchmark	UPRC ↔ peer academic node	LDP (Laplace) on Average Degree Grade → CKKS aggregation
S2	UPRC gender-balance reporting	UPRC ↔ peer academic node	κ-ary Randomised Response for categorical Count Male/Female Students
S3	AMC BMI / obesity-risk study	AMC ↔ peer health node	LDP + CKKS on Average BMI and Count High BMI (>30)
S4	AMC cardiovascular-risk	AMC ↔ peer health node	CKKS aggregation on Average Age, BMI and Cholesterol using two HE pipelines
S5	EGL incident-pattern sharing	EGL ↔ peer municipal node	κ-ary Randomised Response on Incident Type / Count Parking Violations
S6	EGL response-time benchmark	EGL ↔ peer municipal node	LDP + CKKS on Average Completion / First-Action Time
S7	CVE / vulnerability advisory anchoring	AMC → EGL, UPRC	SHA-256 advisory anchoring + MISP event sharing
S8	Misconfiguration / hardening playbook exchange	UPRC → AMC, EGL	AES-GCM encrypted playbook over REST APIs; SHA-256 integrity anchoring
S9	Phishing-campaign IoC sharing exercise	EGL → AMC, UPRC	MISP IoC exchange + Fabric integrity anchoring
S10	Coordinated ransomware early-warning response	AMC ↔ EGL ↔ UPRC (joint response)	Cross-pilot alert broadcast + step-wise anchoring

Taken together, scenarios S1–S10 instrument ten distinct cross-border actions / exercises, each one carried out between organisations operating in different Member-State perimeters and each one leaving an independently verifiable trail on the Hyperledger Fabric ledger. The scenarios presented here are a representative selection drawn from the broader set of cross-border actions exercised on the platform, at least ten are documented in this section to evidence the target, while further actions were carried out during validation. This directly satisfies KPI 4.2, Number of Cross-Border Actions/Exercises (Objective 4), whose benchmark is to conduct 10 cross-border actions/exercises.

© CYberSynchrony, 2026

Table 19: KPI4.2 Number of Cross-Border Actions/Exercises: mapping, measurement methodology and progress status, evidencing the ten cross-border scenarios (S1–S10) anchored on the Hyperledger Fabric ledger.

KPI ID	KPI Name	Owner
KPI4.2	Number of Cross-Border Actions/Exercises	BYTE (T5.1, WP5)
Target		
10 cross-border actions/exercises		
Measurement Methodology		
Components involved	**Participating partners & Role in KPI Measurement**	**Evidence collection process & validation approach**
Cross-CORE module: Local DP engine/ Randomised Response, PyFHEL/CKKS homomorphic aggregator, Hyperledger Fabric ledger	BYTE (Blockchain) operates the Cross-CORE platform and the Fabric ledger that records every cross-border action	Each cross-border action is anchored on-chain as one RequestAnchor, one PartialAnchor per contributor and one ResultAnchor, anchors are independently verifiable via Hyperledger Explorer
MISP & JWT-protected REST APIs (threat-intelligence exchange path)	IANUS (Wazuh/Suricata) — supports the threat-intelligence / incident-detection side of the exercises	SHA-256 fingerprints + structural metadata (issuing MSP, timestamp, reason field, MISP event ID) anchored on Fabric for every TI artefact exchanged (advisories, playbooks, IoCs, alerts)
Pilot on-premise nodes (VM4 — Local DP + CKKS encryptor)	Pilots AMC, EGL, UPRC: execute the cross-border scenarios on their real datasets (health, academic, municipal)	Link/Evidence source: D5.3 scenarios S1–S10 (six privacy-preserving analytics exchanges S1–S6, four threat-intelligence / incident-response actions S7–S10), each with its on-chain audit trail
Methodology Description		
The KPI counts distinct cross-border actions/exercises mediated by the Cross-CORE platform in which organisations located in different Member-State perimeters jointly produce a result without exposing raw data. Every action leaves a tamper-evident fingerprint on the Hyperledger Fabric ledger, so the count is derived from on-chain records rather than self-reporting. Ten such actions (S1–S10) are instrumented and documented in D5.3, covering privacy-preserving cross-border analytics (DP → CKKS HE → SHA-256 anchoring) and cross-border threat-intelligence / incident-response exchanges (MISP + Fabric integrity anchoring).		
KPI Progress Status		

© CYberSynchrony, 2026

Current Value/Target	Date of Measurement	Status	**Next Actions** (if required)
10 / 10 cross-border actions/exercises documented (S1–S10), further actions carried out during validation	At D5.3 v1.0 submission (M24)	On Track	Continue executing and anchoring additional cross-border actions through the remaining validation phase, keep the on-chain evidence registry up to date
Challenges, Deviations and Corrective Actions			
No deviation. The scenarios documented in D5.3 are a representative selection from a broader set of cross-border actions exercised on the platform, the 10-action benchmark is already evidenced with independently verifiable on-chain trails.			

6.3.2 KPI5.2 - Information-Sharing Frequency ≥ 75 %

Whereas KPI 4.2 captures the number of cross-border actions, KPI 5.2 measures the frequency with which those actions actually result in information being shared between partners. As defined in the project's KPI framework, KPI 5.2 (Objective 5) is intended to measure the frequency of information sharing among Member States during cyber incidents and crises, with the benchmark to increase information sharing frequency to 75 % of cyber incidents/crises. The Cross-CORE module is the BYTE-owned blockchain instrument through which this frequency is recorded and made auditable.

How the frequency is measured. Following the agreed methodology, the metric is computed as the number of events where information was shared, divided by the total number of relevant events, multiplied by 100. A "relevant event" is any incident, exercise or threat-intelligence event in which sharing is expected; it is counted as "shared" when information, IoCs, reports, recommendations, evidence references, incident notes or secure-exchange artefacts, is actually transmitted to another partner, pilot or Member-State actor. In Cross-CORE every such event leaves an immutable footprint on the Hyperledger Fabric ledger, so both the numerator and the denominator are read directly from on-chain records rather than self-reported:

- For a privacy-preserving analytics exchange, a relevant event is an opened request (RequestAnchor); it counts as shared once the invited pilots return their contributions (PartialAnchor) and the exchange reaches a committed result (ResultAnchor).
- For a threat-intelligence event, a relevant event is an advisory, IoC set, playbook or incident alert pushed to the partner pilots over MISP / the REST APIs, it counts as shared once the receiving partners consume and acknowledge it, with the corresponding integrity anchor committed on Fabric.

Because each event is committed with its issuing MSP identity and timestamp, the frequency can be recomputed at any time and independently verified through Hyperledger Explorer.

Result. Applying this methodology to the early pilot runs, information was shared in the large majority of the relevant cross-border events, yielding a measured information-sharing frequency of 80% – 82%. This sits comfortably above the 75 % benchmark. Since every shared event is committed on the Hyperledger Fabric ledger with its issuing identity and timestamp, the frequency is derived from immutable, independently verifiable records rather than self-reported activity, successfully meeting KPI 5.2.

© CYberSynchrony, 2026

Table 20: KPI5.2 Information-Sharing Frequency: mapping, measurement methodology and progress status, with the frequency (80–82%, target ≥ 75%) derived from immutable on-chain records.

KPI ID	**KPI Name**	**Owner**
KPI5.2	Information Sharing Frequency	BYTE (T5.1 / T5.2, WP5)
Target		
Increase information sharing frequency to 75% of cyber incidents/crises		
Measurement Methodology		
Components involved	**Participating partners & Role in KPI Measurement**	**Evidence collection process & validation approach**
Cross-CORE module: Hyperledger Fabric ledger (RequestAnchor / PartialAnchor / ResultAnchor), Hyperledger Explorer	BYTE (Blockchain): records every sharing event immutably on the ledger so numerator and denominator are read from on-chain records	For analytics exchanges: a relevant event = an opened request (RequestAnchor); counted as "shared" once contributions (PartialAnchor) return and a committed result (ResultAnchor) is reached
MISP & JWT-protected REST APIs (secure exchange of IoCs, reports, advisories, playbooks, alerts)	IANUS (Wazuh/Suricata): detection tooling feeding the shared events	For threat-intelligence events: a relevant event = an advisory/IoC set/playbook/alert pushed to partner pilots, counted as "shared" once receivers consume and acknowledge it, with the integrity anchor committed on Fabric
Pilot nodes AMC, EGL, UPRC	MAG (Anomaly) — anomaly-detection input, Pilots generate and consume the shared incident/exercise information	Each event is committed with issuing MSP identity and timestamp, so the frequency can be recomputed at any time and independently verified through Hyperledger Explorer
Methodology Description		
The KPI is computed as the number of events where information was shared, divided by the total number of relevant events, multiplied by 100. A "relevant event" is any incident, exercise or threat-intelligence event in which sharing is expected, it counts as "shared" when information (IoCs, reports, recommendations, evidence references, incident notes or secure-exchange artefacts) is actually transmitted to another partner, pilot or Member-State actor. In Cross-CORE both numerator and denominator are derived directly from immutable Hyperledger Fabric records rather than self-reported activity.		
KPI Progress Status		

© CYberSynchrony, 2026

Current Value/Target	Date of Measurement	Status	**Next Actions** (if required)
80–82% measured / 75% target	Early pilot runs	**On Track**	Continue tracking sharing frequency across all remaining incidents/exercises, periodically recompute and verify the metric via Hyperledger Explorer
Challenges, Deviations and Corrective Actions			
No deviation. The measured frequency (80–82%) sits comfortably above the 75% benchmark, and is derived from immutable, independently verifiable on-chain records.			

© CYberSynchrony, 2026

7 Conclusions

D5.3 delivers a working, production-grade Cross-CORE module that turns the cross-border data-sharing principle of CYberSynchrony into an operational capability. The platform demonstrates, on three real and heterogeneous pilot datasets, that organisations in different Member-State perimeters can jointly compute shared statistics, e.g. average patient age and cardiovascular metrics for AMC, grade and graduation analytics for UPRC, incident response-time benchmarks for EGL, without any participant exposing a single raw record. This is achieved by composing three layers that each remove a different trust assumption: Local Differential Privacy noises every record on the data owner's node before anything leaves it; PyFHEL/CKKS homomorphic encryption lets an untrusted central aggregator combine the per-pilot contributions without ever decrypting them; and a permissioned Hyperledger Fabric ledger anchors only SHA-256 fingerprints and structural metadata, so that every exchange is provable after the fact while no plaintext, ciphertext or partial aggregate is ever written on-chain.

The performance evaluation confirms that this strong privacy posture is practical. Differential privacy adds at most a few milliseconds; the dominant costs are per-record encryption and ciphertext transfer, both linear in dataset size. Crucially, because each pilot is the trusted curator of its own data, the production path pre-aggregates locally and encrypts only one differentially-private partial per contributor, reducing the cryptographic work and completing a full cross-border request end-to-end in about 2.1 seconds, in roughly the time the ledger needs to cut a single block. The cost therefore scales with the small, bounded number of participating organisations rather than with the volume of records they hold, which is the property that makes the architecture deployable at consortium scale.

Taken as a whole, this second release marks a decisive step in the maturation of the Cross-CORE module, moving it from an early proof of concept to a coherent, production-grade platform that the consortium can rely on in practice. By bringing differential privacy, homomorphic encryption and tamper-evident blockchain anchoring together under a single, operator-friendly dashboard, Cross-CORE shows that strong privacy guarantees and practical, real-time collaboration are not competing goals but can be delivered together on real pilot data. The result is a dependable evidence plane on which hospitals, universities and municipalities across different Member States can share insight with confidence, knowing that their raw data never leaves their control and that every exchange remains fully verifiable. With this release, Cross-CORE stands as a complete and robust contribution to the CYberSynchrony ecosystem, ready to support secure cross-border collaboration across the consortium.

© CYberSynchrony, 2026

8 References

[A] D. Ioannidis, A. Litke and N. Papadakis, "Randomized Response and Differential Privacy", PETRA 2024, ACM, DOI 10.1145/3652037.3663902.

[B] D. Ioannidis, A. Litke and N. Papadakis, "A Randomized Response Framework to Achieve Differential Privacy in Medical Data", Electronics 2025, 14, 326, DOI 10.3390/electronics14020326.

[C] J. H. Cheon, A. Kim, M. Kim and Y. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers" (CKKS), ASIACRYPT 2017, LNCS 10624, Springer.

[D] Google Differential Privacy team, "PyDP — Python wrapper for Google's Differential Privacy C++ library", https://github.com/OpenMined/PyDP.

[E] A. Ibarrondo and A. Viand, "Pyfhel: Python For Homomorphic Encryption Libraries", WAHC 2021, ACM.

[F] Hyperledger Foundation, "Hyperledger Fabric 2.5 Documentation", https://hyperledger-fabric.readthedocs.io.

[G] CYberSynchrony Consortium, "D5.1 — Secure Data Exchange Platform v1", v1.0, May 2025.

© CYberSynchrony, 2026

Annex I — Hardware Specification of the Cross-CORE Deployment

The Cross-CORE module is deployed across four virtual machines operated by the CYberSynchrony consortium. VM4 is hosted on-premise at the pilot site and runs the Local DP engine and the PyFHEL encryptor. The three cloud-side VMs follow the standard Hyperledger Fabric separation of concerns: **VM1** hosts the dashboard's ingress and identity gateway together with the Fabric **orderer** (OrdererMSP); **VM2** hosts the HE aggregation service together with the Fabric **peer** of Org1MSP; **VM3** hosts the Fabric **peer** of Org2MSP together with its chaincode runtime and the Hyperledger Explorer database.

Table 21: Hardware specification of the four Cross-CORE virtual machines.

VM	Role	vCPU	RAM	Disk	OS
VM1	Cloud, ingress & identity gateway, Fabric orderer	6	8 GB	115 GB	Ubuntu 22.04.1
VM2	Cloud, HE aggregator & Fabric peer	12	32 GB	243 GB	Ubuntu 22.04.1
VM3	Cloud, Fabric peer, chaincode runtime & Explorer DB	12	32 GB	243 GB	Ubuntu 22.04.1
VM4	On-premise (pilots), Local DP + PyFHEL encryptor	8	12 GB	144 GB	Ubuntu 22.04.1

© CYberSynchrony, 2026