

GL4

ZMATRIX

DATE	DATABASE	ENCRYPTION
April 4, 2026	1GB PostgreSQL 16	ML-KEM-768 + X25519 + AES-256-GCM

Database Migration Security Experiment

Protecting a Live PostgreSQL Migration Against
Quantum Harvest-Now-Decrypt-Later Attacks

CONTENTS

Table Of Contents

- 01 Overview
- 02 The Threat: Harvest Now, Decrypt Later
- 03 Experiment Scenario
- 04 Datacenter Architecture
- 05 Platform Setup & Configuration
- 06 Enterprise Data Generation
- 07 Unprotected Migration (Baseline)
- 08 Where Attackers Harvest Your Data
- 09 ZMatrix PQC Agents
- 10 Protected Migration (PQC)
- 11 Experimental Results
- 12 Conclusions

Overview

This guide walks you through a real, reproducible experiment that demonstrates how a standard database migration exposes every byte of data to interception — and how to protect it with post-quantum cryptography.

The Danger

A standard database migration between two datacenters exposes every byte of data, customer PII, financial records, medical records, employee salaries — to network-level interception.

The Quantum Threat

Data captured today will be decryptable by quantum computers within the next 5–10 years. This "harvest now, decrypt later" strategy is already in active use by nation-state actors.

The Solution

ZMatrix PQC agents wrap the exact same migration in post-quantum encryption (ML-KEM-768 + X25519 + AES-256-GCM) with near-zero performance overhead.

CONTINUED

What You'll Migrate

What You'll Migrate — 1 GB PostgreSQL Database



Customer Records

Names, emails, SSNs, addresses, credit scores



Financial Transactions

Amounts, merchant details, card numbers



Medical Records

Diagnoses, prescriptions, lab results



Employee Records

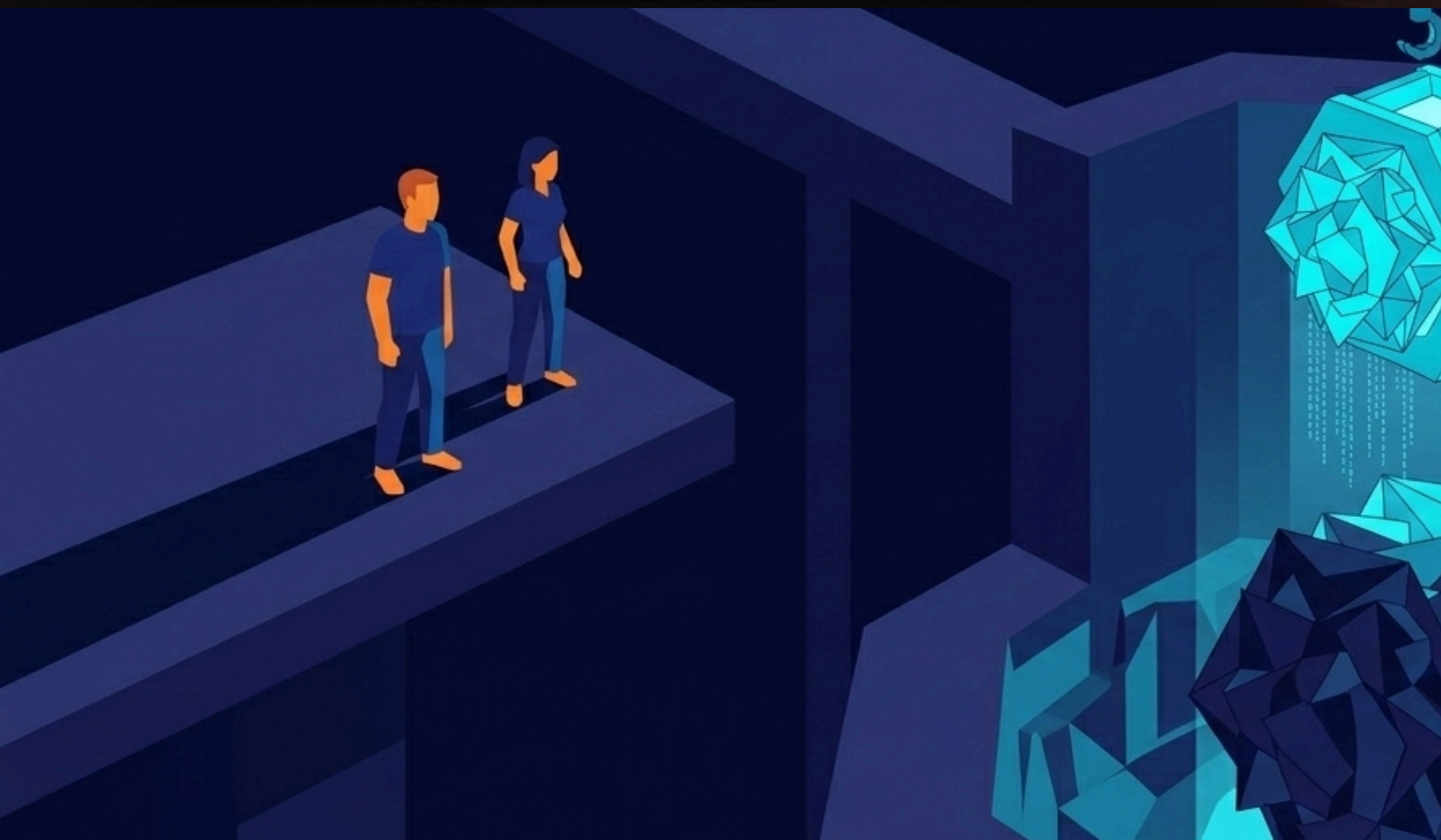
Salaries, performance reviews

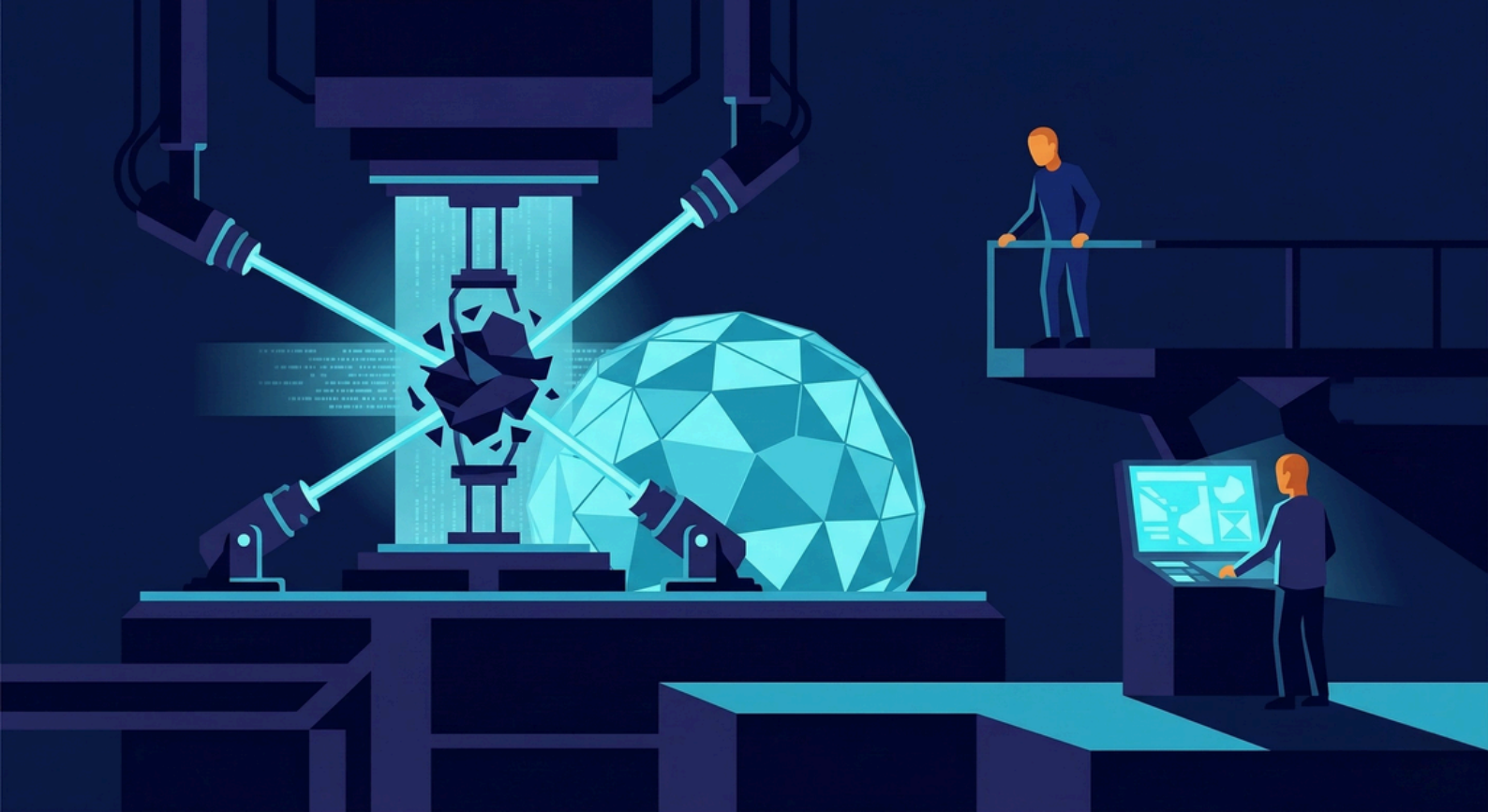


Audit Logs

Full change history with IP addresses

This is exactly the kind of data that nation-state actors and APT groups are harvesting from WAN links *right now*.





— 02

The Threat: Harvest Now, Decrypt Later

Intelligence agencies and advanced persistent threat groups are **recording encrypted network traffic today**, storing it in massive data lakes, and waiting for quantum computers powerful enough to break the encryption.



CONTINUED

Why Database Migrations Are Prime Targets

A database migration is the **single most valuable interception point** for an attacker.

FACTOR	WHY IT'S DANGEROUS
Volume	The <i>entire</i> database crosses the network — not just one query, ALL of it
Structure	pg_dump outputs structured SQL — column names, table definitions, relationships are plaintext
Duration	Migrations run for minutes to hours — plenty of time to capture every packet
Predictability	Migrations are scheduled; attackers know when to listen
Concentration	One interception gives you <i>everything</i> — years of accumulated PII in a single stream

Algorithm Vulnerability Matrix

ALGORITHM	USED BY	BROKEN BY QUANTUM?
RSA-2048/4096	TLS 1.2, most VPNs	YES — Shor's algorithm
ECDH (P-256, P-384)	TLS 1.3 key exchange	YES — Shor's algorithm
X25519	WireGuard, modern TLS	YES — Shor's algorithm
AES-256	Session encryption	Weakened but survivable
ML-KEM-768	ZMatrix PQC	NO — Quantum-resistant (NIST FIPS 203)

Key insight: Even if your pg_dump stream is encrypted with TLS 1.3, the key exchange (ECDH/X25519) will be breakable. ZMatrix adds ML-KEM-768 on top, making it quantum-resistant.

The Regulatory Reality

Governments and standards bodies worldwide are mandating the transition to post-quantum cryptography. Organizations that delay risk non-compliance with emerging frameworks.

NSA

CNSA 2.0 (2022)

All national security systems must migrate to PQC by 2035

NIST

FIPS 203 (2024)

ML-KEM standardized as the federal post-quantum KEM

OMB

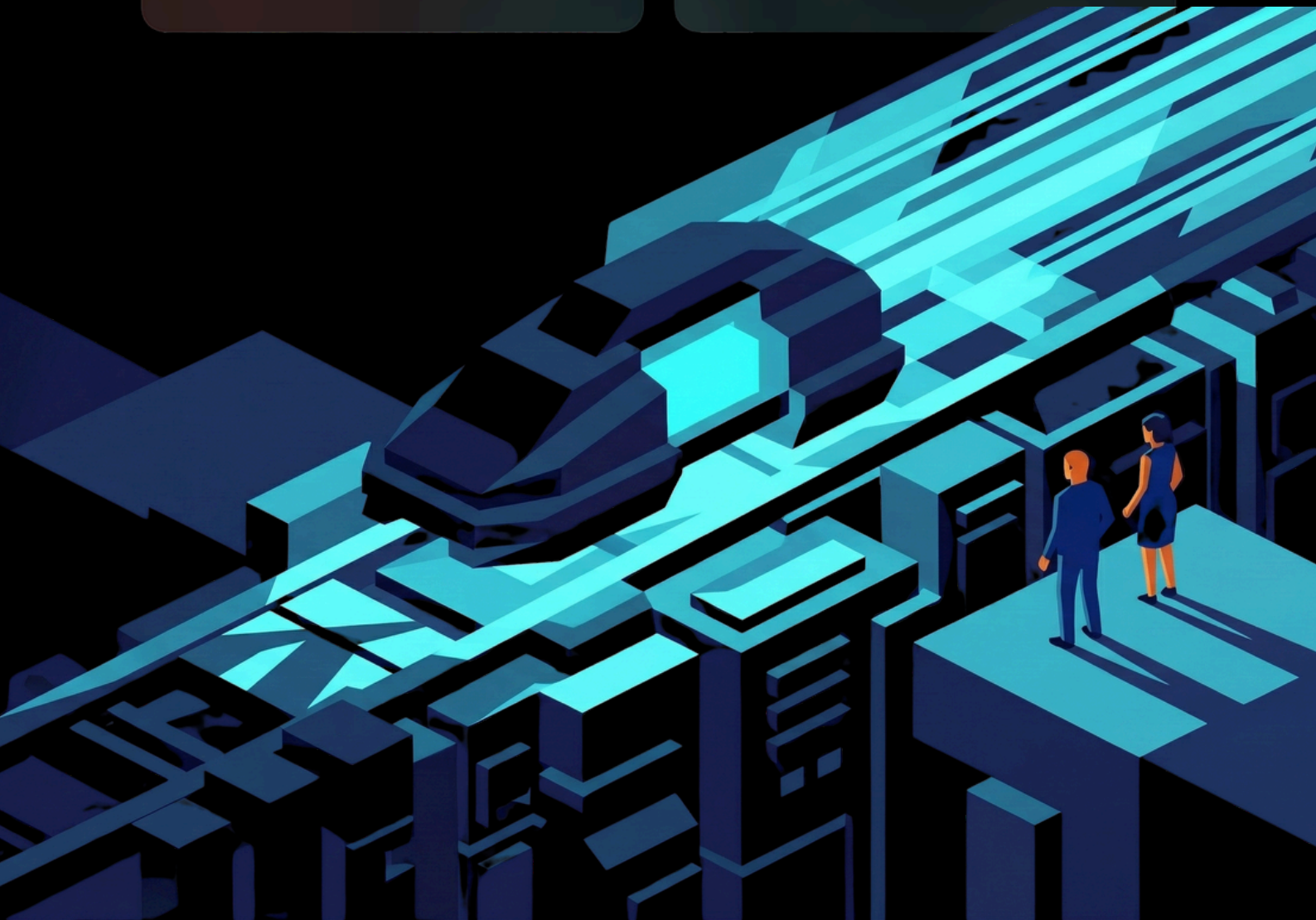
M-23-02

Federal agencies must inventory cryptographic systems and plan migration

EU

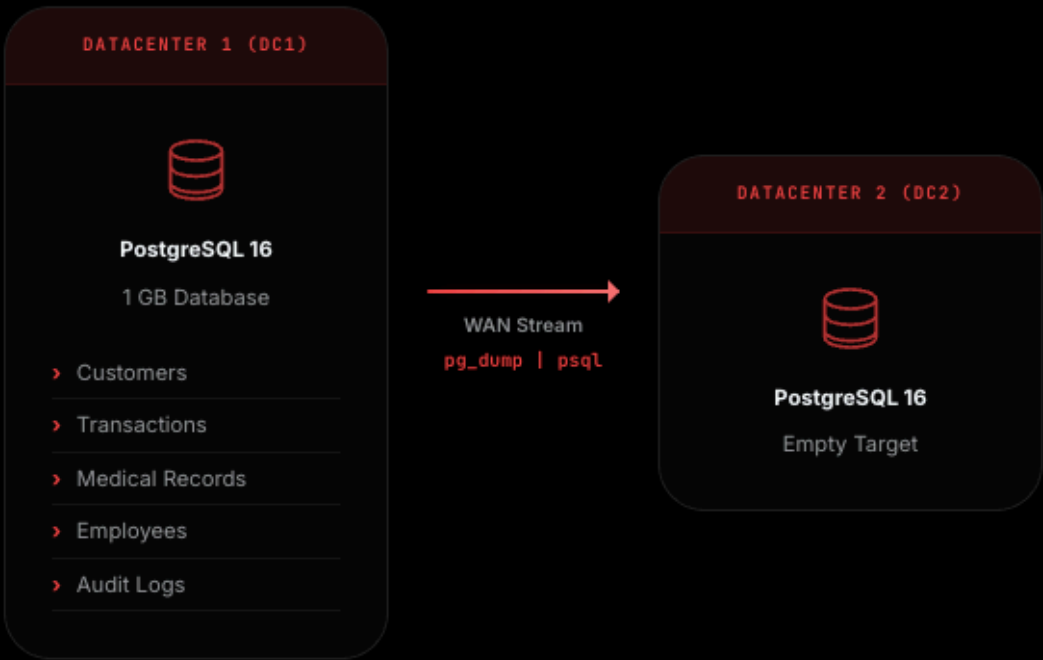
NIS2 Directive

Critical infrastructure must address "future threats" including quantum



Experiment Scenario

A live 1 GB PostgreSQL database migration between two simulated datacenters — first unprotected, then wrapped in post-quantum encryption.



Two Phases

PHASE	METHOD	ENCRYPTION	QUANTUM-SAFE?
Phase 1: Baseline	<code>pg_dump psql</code> direct	None (or TLS — breakable)	NO
Phase 2: PQC Protected	<code>pg_dump psql</code> through ZMatrix	ML-KEM-768 + X25519 + AES-256-GCM	YES

The migration command is **exactly the same** in both phases. The only difference is whether ZMatrix PQC agents are sitting between the datacenters.

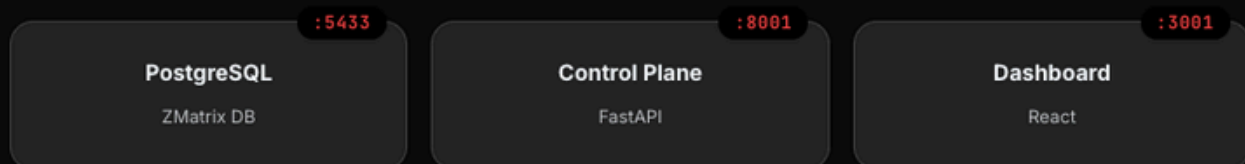


- 04

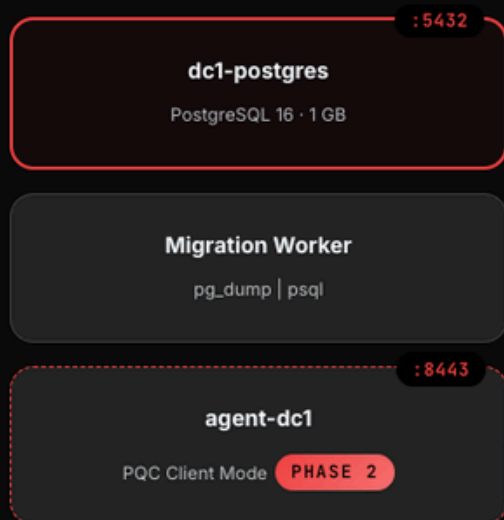
Datacenter Architecture

Docker containers simulate two physical datacenters, connected through a shared network. Each container represents a real production server.

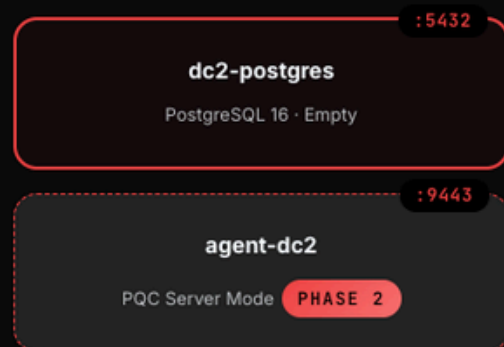
ZMATRIX CLOUD (SAAS CONTROL PLANE)



DATACENTER 1



DATACENTER 2



----- PQC Encrypted Tunnel

CONTINUED

Container Inventory

CONTAINER	ROLE	IMAGE	PORT
zm-postgres	ZMatrix platform database	postgres:16-alpine	5433
zm-control-plane	API server (FastAPI)	Custom	8001
zm-dashboard	Web UI (React)	Custom	3001
zm-dc1-postgres	DC1 — Source database (1 GB)	postgres:16-alpine	5432
zm-dc2-postgres	DC2 — Destination database	postgres:16-alpine	5432
zm-datagen	Data generator (populates DC1)	Custom (Python)	—
zm-migration-baseline	Migration worker (Phase 1)	Custom (Python)	—
zm-migration-pqc	Migration worker (Phase 2)	Custom (Python)	—
zm-agent-dc1	PQC Agent DC1 (client mode)	Custom (Rust)	8443
zm-agent-dc2	PQC Agent DC2 (server mode)	Custom (Rust)	9443

Platform Setup & Configuration

Prerequisites



Docker v24+
& Docker
Compose v2+



Python 3.10+
For orchestrator
script



8 GB RAM
Minimum required



15 GB Disk
Free space
required

Step 1 — Clone & Start the Platform

BASH

Terminal

```
# Clone the repository
git clone https://github.com/zmatrix-security/zm-pqc.git
cd zm-pqc

# Start the core SaaS platform
docker compose build
docker compose up -d
```

SERVICE

URL

PURPOSE

Dashboard

<http://localhost:3001>

Web UI for managing agents

Control Plane API

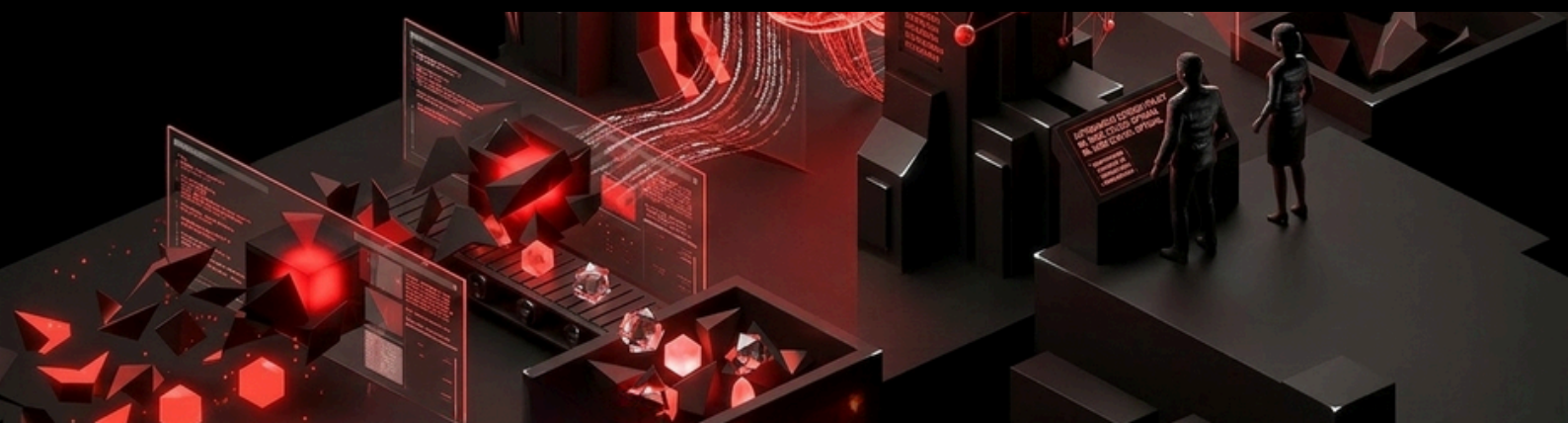
<http://localhost:8001>

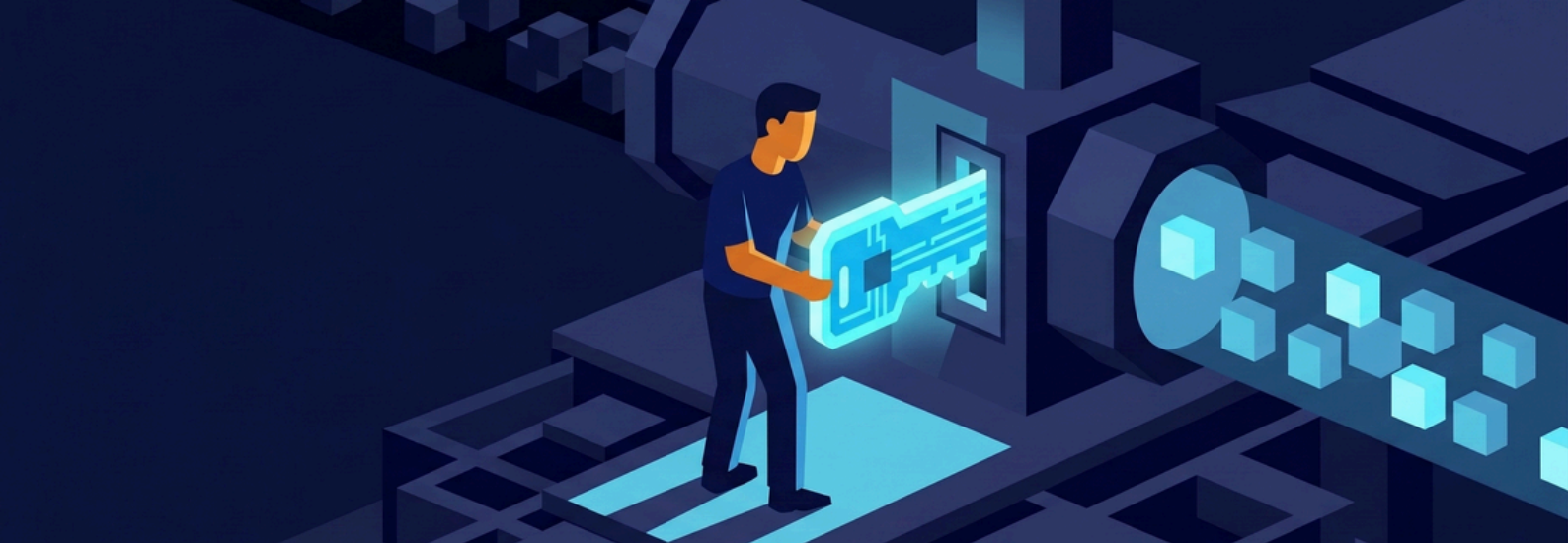
REST API for registration & telemetry

PostgreSQL

<localhost:5433>

ZMatrix platform database





— 05

CONTINUED

Account & API Key Setup

Step 2 — Create Your Account

BASH

API — Create Account

```
curl -X POST http://localhost:8001/api/v1/auth/signup \  
  -H "Content-Type: application/json" \  
  -d '{  
    "email": "benchmark@zmatrix-test.com",  
    "password": "Benchm@rk2026!",  
    "full_name": "Benchmark Operator",  
    "org_name": "Benchmark-DataCenter-Test"  
  }'
```

BASH

Create a Dedicated API Key

```
# Login to get a fresh token  
TOKEN=$(curl -s -X POST http://localhost:8001/api/v1/auth/login \  
  -H "Content-Type: application/json" \  
  -d '{"email": "benchmark@zmatrix-test.com", "password": "Benchm@rk2026!"}' \  
  | python -c "import sys,json; print(json.load(sys.stdin)['access_token'])")  
  
# Create a dedicated key  
curl -X POST http://localhost:8001/api/v1/api-keys \  
  -H "Authorization: Bearer $TOKEN" \  
  -H "Content-Type: application/json" \  
  -d '{"name": "DBMigration"}'
```

Enterprise Data Generation

The data generator creates a realistic enterprise database with **sensitive PII across 5 tables**.

TABLE	CONTENT	SENSITIVITY
customers	Names, emails, SSN hashes, addresses, credit scores, balances	CRITICAL — PII + Financial
transactions	Amounts, merchant details, card numbers, IP addresses	CRITICAL — Financial
medical_records	Diagnoses, prescriptions, lab results, provider notes	CRITICAL — HIPAA
employees	Salaries, bonuses, performance reviews, hire dates	HIGH — HR Confidential
audit_log	Full change history with old/new values, user IDs, IPs	HIGH — Compliance

Generate the Data

BASH

Populate DC1

```
# Start DC1 PostgreSQL and populate with 1 GB of data
docker compose -f docker-compose.yml \
  -f benchmark/db-migration/docker-compose.db-migration.yml \
  --profile datagen up --build

# Once complete, shut down the generator
docker compose -f docker-compose.yml \
  -f benchmark/db-migration/docker-compose.db-migration.yml \
  --profile datagen down
```

Reproducibility: The data generator uses a deterministic seed (`random.seed(42)`), so every run produces identical data.



- 07

The Unprotected Migration (Baseline)

Phase 1: A standard `pg_dump` | `psql` migration directly from DC1 to DC2. This is how **most** organizations migrate databases today.

PostgreSQL

DC1 - 1 GB

PLAINTEXT STREAM

`pg_dump` | `psql`

PostgreSQL

DC2 - Empty



EVERY BYTE crosses the network in the clear
SQL DDL + INSERT statements are human-readable
SSNs, medical records, salaries — ALL exposed

BASH

Run the Baseline Migration

```
export ZM_API_KEY="zm_your_api_key_here"
docker volume rm -f zm-pqc_dc2-pgdata

docker compose -f docker-compose.yml \
  -f benchmark/db-migration/docker-compose.db-migration.yml \
  --profile baseline up --build
```

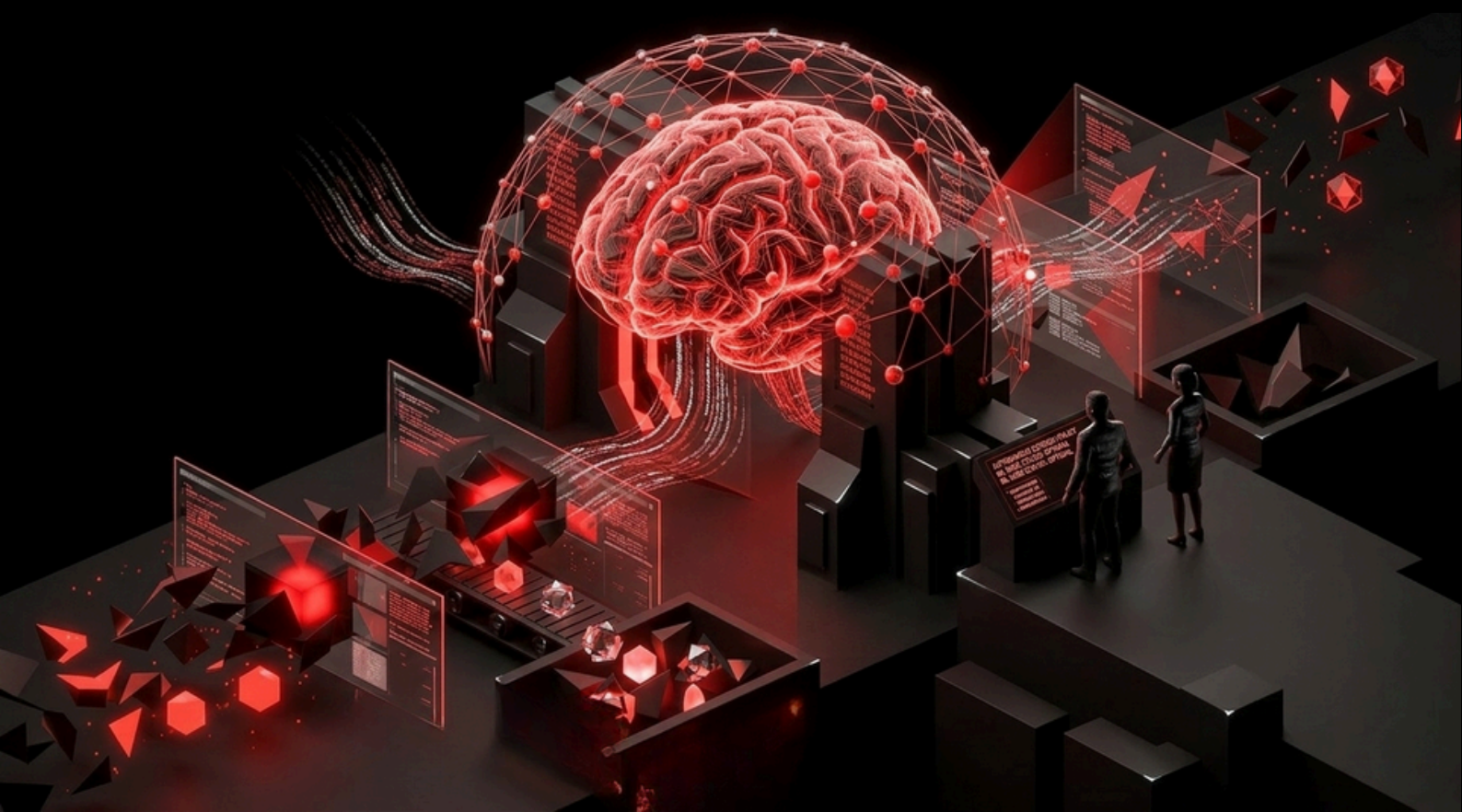
CONTINUED

Baseline Results

The unprotected migration completes in **37.52 seconds** — with every byte of data exposed on the wire.

● ● ● Migration Output

```
[migration] === SOURCE DATABASE (DC1) ===  
[migration] Size: 1.080 GB (1,079,816,675 bytes)  
[migration]   customers:      148,000 rows  
[migration]   transactions: 1,480,000 rows  
[migration]   medical_records: 296,000 rows  
[migration]   employees:      5,000 rows  
[migration]   audit_log:      444,000 rows  
  
[migration] === MIGRATION COMPLETE ===  
[migration] Duration:          37.52s  
[migration] Effective rate:    0.2303 Gbps  
[migration] Data integrity:    VERIFIED
```



Where Attackers Harvest Your Data

Interception Points

INTERCEPTION POINT	METHOD	WHO DOES THIS
ISP/Carrier backbone	Fiber tap, port mirror	Nation-state intelligence agencies
IXP (Internet Exchange)	BGP hijacking, traffic mirroring	State-sponsored APT groups
Cloud provider network	Compromised hypervisor, vSwitch tap	Advanced threat actors
VPN concentrator	Exploit or insider access	Corporate espionage
DNS/routing manipulation	BGP prefix hijack → MitM	Sophisticated adversaries



CONTINUED

What an Attacker Gets

DATA CATEGORY	VOLUME	REAL-WORLD IMPACT
Customer SSNs	148,000 records	Identity theft, tax fraud, credit fraud
Credit scores & balances	148,000 records	Financial profiling, targeted fraud
Medical diagnoses	296,000 records	HIPAA violations, blackmail, insurance fraud
Employee salaries	5,000 records	Corporate espionage, insider trading
Transaction history	1,480,000 records	Financial pattern analysis, fraud
Full audit trail	444,000 records	Understanding security controls to bypass

The Economics Are Devastating

STORAGE COST

\$2.16

To hold 1 GB for 9 years

DATA VALUE

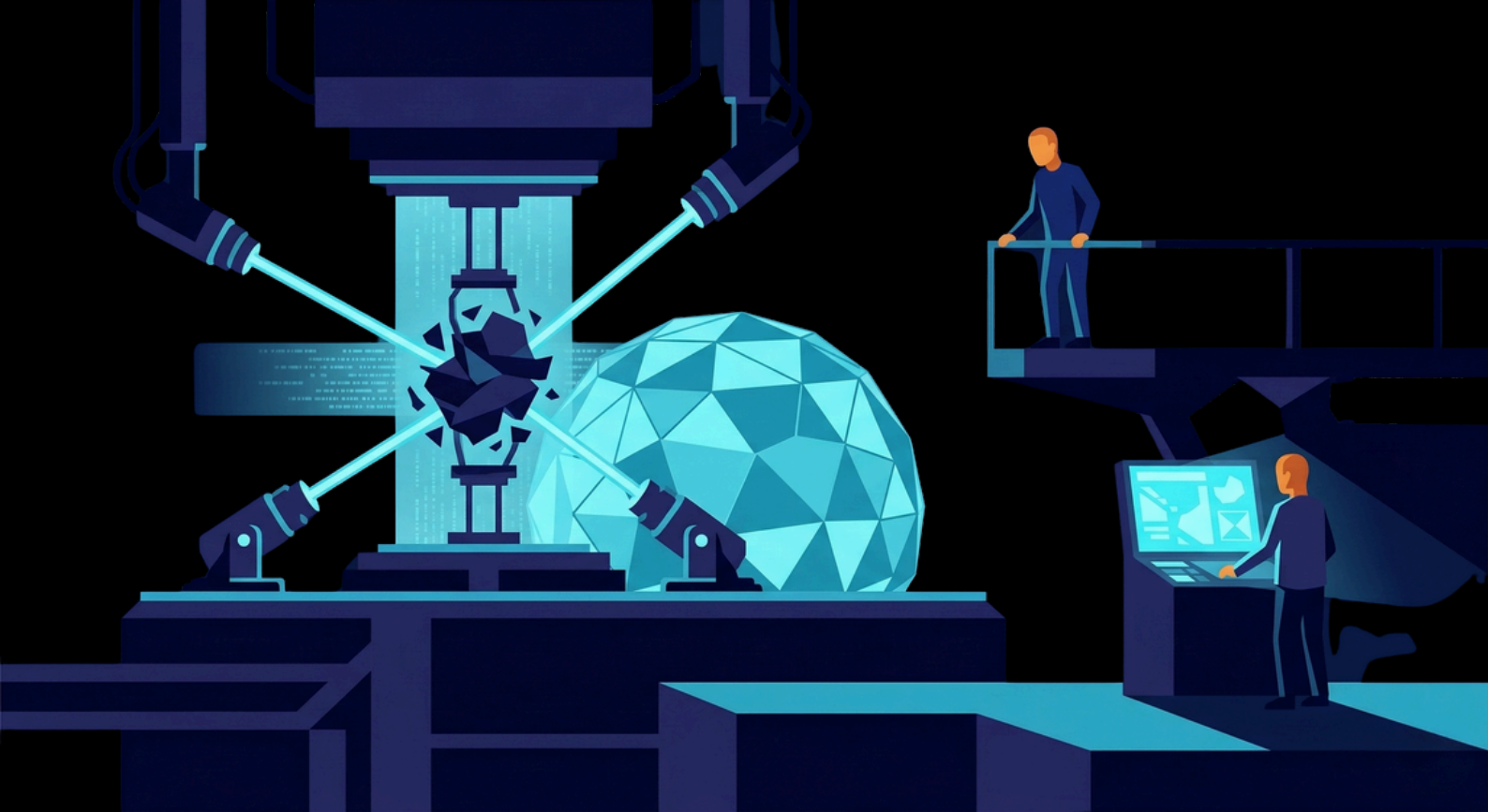
\$1.48M+

VS

148,000 SSNs on dark web

HN DL attacks are considered a **certainty**, not a possibility.





— 09

ZMatrix PQC Agents

ZMatrix PQC works by placing a **pair of agents** — one at each datacenter. They form an encrypted tunnel, and all traffic flows through transparently.

PostgreSQL

DC1 · 1 GB

Agent DC1

Client · :8443

PQC ENCRYPTED TUNNEL

ML-KEM-768 + X25519 + AES-256-GCM

Agent DC2

Server · :9443

PostgreSQL

DC2 · Empty



Attackers see only quantum-resistant ciphertext



pg_dump stream is UNREADABLE on the wire

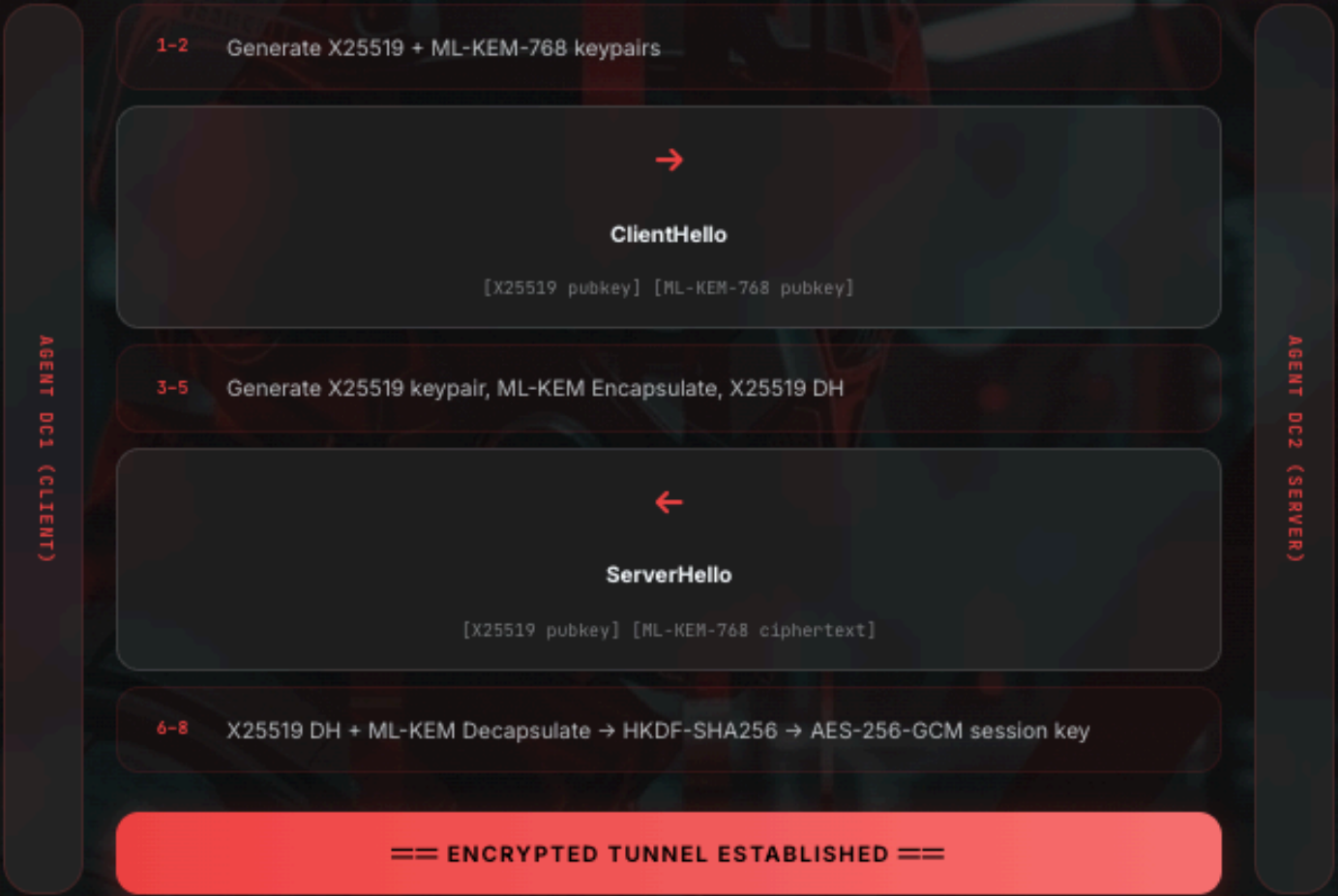


No code changes — migration command is identical

CONTINUED

The PQC Handshake

When the DC1 agent connects to DC2, a hybrid post-quantum key exchange occurs:



Why Hybrid (X25519 + ML-KEM-768)?

REASON	EXPLANATION
Defense in depth	If either algorithm has an undiscovered weakness, the other still protects
X25519	Proven against classical attacks (decades of cryptanalysis)
ML-KEM-768	Proven against quantum attacks (NIST standardized, FIPS 203)
Combined via HKDF	Both shared secrets feed into key derivation — attacker must break <i>both</i>

CONTINUED

Agent Installation

Deploy ZMatrix PQC agents to each datacenter. The agents form the encrypted tunnel automatically on startup.

Agent Installation (Production)

BASH

DC1 Server

```
curl -sSL https://get.zmatrix.io/agent | sh
zm-agent configure \
  --api-key "zm_your_key_here" \
  --mode client \
  --listen-port 8443 \
  --remote-addr dc2.example.com \
  --remote-port 9443
zm-agent start
```

BASH

DC2 Server

```
curl -sSL https://get.zmatrix.io/agent | sh
zm-agent configure \
  --api-key "zm_your_key_here" \
  --mode server \
  --listen-port 9443 \
  --upstream-addr localhost \
  --upstream-port 5432
zm-agent start
```





— 10

The Protected Migration (PQC)

The **exact same migration** — but now `psql` connects through the ZMatrix PQC agent, encrypting everything with quantum-resistant algorithms before it hits the wire.

BASH

Run the PQC Migration

```
docker volume rm -f zm-pqc_dc2-pgdata

docker compose -f docker-compose.yml \
  -f benchmark/db-migration/docker-compose.db-migration.yml \
  --profile pqc up --build
```

The Key Difference

ASPECT	PHASE 1 (BASELINE)	PHASE 2 (PQC)
Migration command	<code>psql dc2-postgres:5432</code>	<code>psql agent-dc1:8443</code>
Data on the wire	Plaintext SQL	AES-256-GCM ciphertext
Key exchange	None	ML-KEM-768 + X25519
Quantum-safe?	NO	YES
Code changes	—	ZERO

What an attacker sees on the wire:

0x7f3a 9c81 d42e f7b0 a523 8d6f 1ce9 4ab8 e2d7 3f19 b086 c54a 7d91 0e3c f682 ...

Quantum-resistant ciphertext — useless now AND later

Experimental Results

Source Database Profile

1.08GB

TOTAL DATABASE
SIZE

2.37M

TOTAL ROWS

5

TABLES

PG16

POSTGRESQL
VERSION

Transactions

1,480,000

Audit Log

444,000

Medical Records

296,000

Customers

148,000

Employees

5,000





- 11

CONTINUED

Head-to-Head Comparison

Phase 1: Baseline

UNPROTECTED

37.52_s

DURATION



INTEGRITY

0.2303_{Gbps}

THROUGHPUT



QUANTUM-SAFE

V S

Phase 2: PQC Protected

QUANTUM-RESISTANT

35.93_s

DURATION



INTEGRITY

0.2404_{Gbps}

THROUGHPUT



QUANTUM-SAFE

4.2%

FASTER WITH PQC

~14ms

HANDSHAKE OVERHEAD

0

CODE CHANGES
REQUIRED

0.04%

AMORTIZED PQC COST

Performance insight: The Rust-based proxy architecture employs optimized TCP buffering (64 KB pre-allocated buffers, `TCP_NODELAY`, 4 MB socket buffers) that actually **reduces syscall overhead** compared to the naive pipe between `pg_dump` and `psql`.

The Business Case

Without ZMatrix PQC

PERFORMANCE

0.2303 Gbps (37.52s)

RISK

ALL data harvestable for quantum decryption

LIABILITY

148,000 SSNs × \$150 = \$22.2M+

COMPLIANCE

Fails CNSA 2.0, NIST PQC, NIS2

With ZMatrix PQC

PERFORMANCE

0.2404 Gbps (35.93s — 4.2% FASTER)

RISK

Quantum-resistant encryption

LIABILITY

Near-zero for HNDL attacks

COMPLIANCE

Satisfies CNSA 2.0, FIPS 203, NIS2

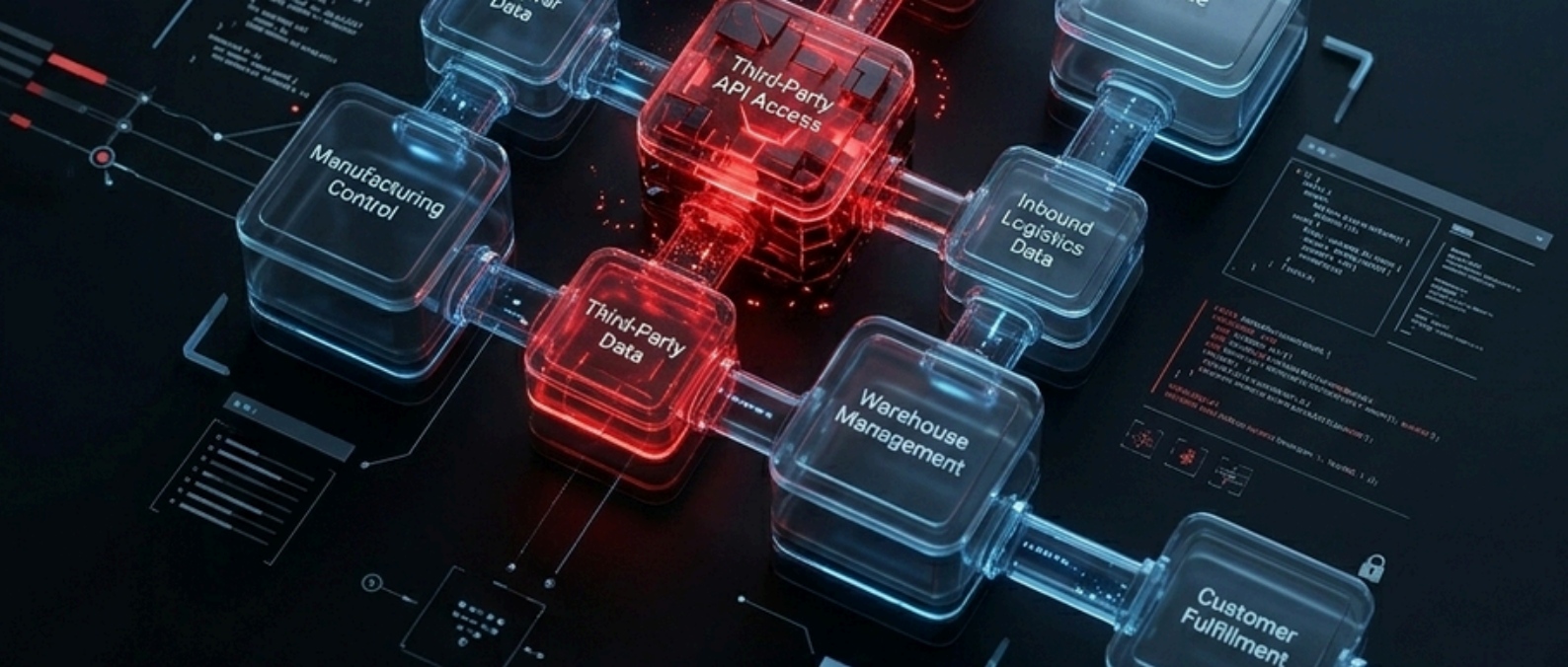
COST OF ZMATRIX

Minutes to install, zero code changes

COST OF NOT USING IT

\$22.2M+ in potential breach damages





— 12

Conclusions

1

Migrations Are High-Value Targets

A single `pg_dump` stream contained the entire database — 148,000 customer SSNs, 1.48M financial transactions, 296,000 medical records, 5,000 employee records, and 444,000 audit entries. All 2.37 million records crossed the network in a single 37-second stream.

2

HNDL Is Economically Rational

Storing 1.08 GB costs \$0.02/month. The 148,000 SSNs inside are worth \$1.48M+ on the dark web. The full dataset triggers \$22.2M+ in potential breach liability. Nation-states are already doing this at scale.

3

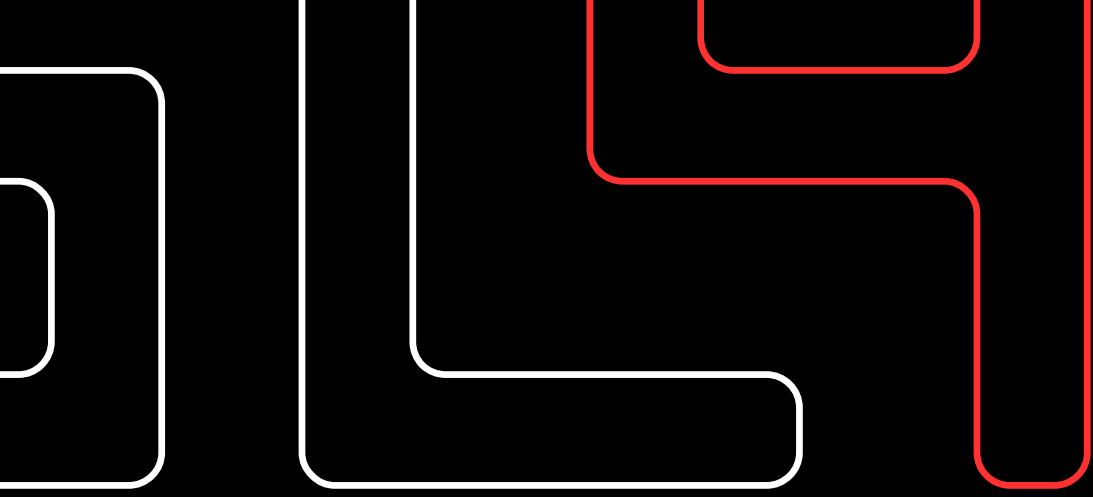
ZMatrix PQC Is Transparent

The migration worker ran the exact same `pg_dump | psql` command in both phases. The only change: destination was `agent-dc1:8443` instead of `dc2-postgres:5432`. Zero code changes, zero application modifications.

4

PQC Actually Improved Performance

The PQC-protected migration completed in 35.93s vs 37.52s baseline — **4.2% faster** — thanks to the Rust agent's optimized TCP buffering. The one-time PQC handshake (~14 ms) is amortized to 0.04% of total migration time.



ZMGL4

ZMatrix Gateway Layer 4

