L08: CBDCs – The Mathematics of Sovereign Digital Money

Extended Slides – BSc Digital Finance Course

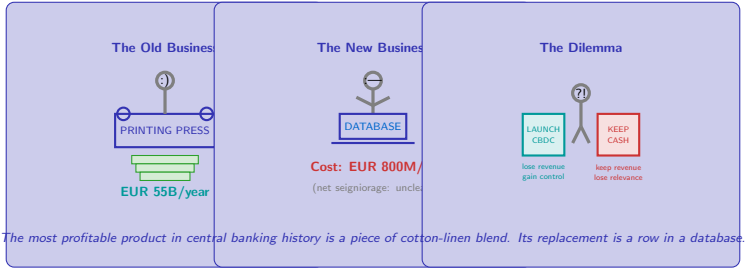
Digital Finance

What Will You Be Able to Do After This Lecture?

1. Model seigniorage loss and optimal CBDC interest rates using deposit displacement functions
2. Apply zero-knowledge proof mechanics and differential privacy budgets to CBDC transaction design
3. Analyze mCBDC cross-border settlement with PvP atomicity and currency substitution risk models
4. Compare CBDC technology architectures on throughput, latency, finality, and offline capability
5. Quantify bank disintermediation risk through deposit migration and digital bank run dynamics
6. Design tiered remuneration schemes that balance financial inclusion with stability constraints

---

Six objectives mapping to the five sections: monetary policy (1), privacy engineering (2), cross-border mCBDC (3), technology architecture (4), and bank disintermediation (5–6).



The most profitable product in central banking history is a piece of cotton-linen blend. Its replacement is a row in a database.

How Does a CBDC Destroy the Most Profitable Product in Central Banking?

Physical seigniorage. Central banks earn the spread between the policy rate and the cost of printing:

$$S_{\text{phys}} = M_0 \cdot (i - c_{\text{print}})$$

where M_0 = currency in circulation, i = policy rate, c_{print} = printing cost per unit.
Numerical example: M_0 = EUR 1.6T, $i = 3.5\%$, $c_{\text{print}} \approx 0.05\%$: $S_{\text{phys}} = 1.6\text{T} \times 3.45\% = \text{EUR } 55.2\text{B}$.
Digital seigniorage. A remunerated CBDC replaces free float with costly infrastructure:

$$S_{\text{dig}} = M_{\text{CBDC}} \cdot (i - r_{\text{CBDC}} - c_{\text{infra}})$$

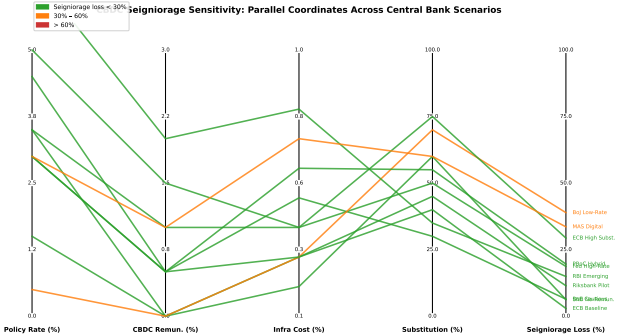
If $r_{\text{CBDC}} = 2\%$, $c_{\text{infra}} = 0.5\%$: $S_{\text{dig}} = M_{\text{CBDC}} \times 1.0\%$.
Seigniorage loss ratio:

$$\frac{\Delta S}{S_{\text{phys}}} = 1 - \frac{i - r_{\text{CBDC}} - c_{\text{infra}}}{i - c_{\text{print}}} \cdot \alpha = 1 - \frac{1.0}{3.45} \cdot \alpha = 1 - 0.290\alpha$$

where $\alpha = M_{\text{CBDC}}/M_0$. At full substitution ($\alpha = 1$): **71% seigniorage loss**.

Central banks earn EUR 55B/year from physical cash. A remunerated CBDC with infrastructure costs could destroy 71% of that revenue at full substitution.

How Sensitive Is Seigniorage to CBDC Remuneration and Substitution?



- **Parallel coordinates** link five parameters: policy rate, CBDC rate, infra cost, substitution ratio, and seigniorage loss.
- **No-remuneration CBDCs** (green lines) preserve seigniorage but discourage adoption – users prefer interest-bearing bank deposits.
- **High-remuneration CBDCs** (red lines) drive adoption but destroy central bank income.
- The ECB baseline ($r_{\text{CBDC}} = 0\%$, low substitution) shows minimal seigniorage loss; the "full digital" scenario shows catastrophic loss.
- **Policy lever:** Remuneration rate is the single most influential parameter.

No-remuneration preserves revenue but limits adoption. High remuneration drives adoption but destroys income. The optimal CBDC rate balances these tensions.

How Much Will Depositors Flee Banks for CBDC Safety?

Money demand with CBDC (Bech & Garratt 2017 extension):

$$D_{\text{bank}} = D_0 \cdot \left(1 - \frac{r_{\text{CBDC}} - r_{\text{dep}} + \pi}{r_{\text{dep}} + \delta}\right)^{\beta}$$

where D_0 = baseline deposits, r_{dep} = deposit rate, π = perceived safety premium of CBDC, δ = switching cost, β = elasticity.
Scenario A ($r_{\text{CBDC}} = 1\%$, $r_{\text{dep}} = 2\%$, $\pi = 0.5\%$, $\delta = 1\%$, $\beta = 2$, $D_0 = 100$):

$$\text{Ratio} = \frac{1\% - 2\% + 0.5\%}{2\% + 1\%} = \frac{-0.5\%}{3\%} = -0.167; \quad D_{\text{bank}} = 100 \times 1.167^2 = 136.1$$

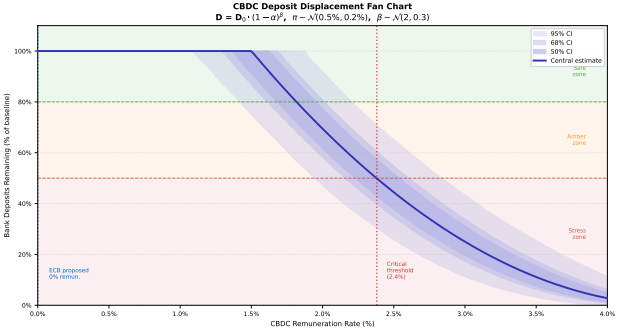
Deposits **increase** – CBDC rate is below deposit rate, so banks are safe.
Scenario B ($r_{\text{CBDC}} = 3\%$):

$$\text{Ratio} = \frac{3\% - 2\% + 0.5\%}{3\%} = 0.5; \quad D_{\text{bank}} = 100 \times 0.5^2 = 25 \quad \Rightarrow \textbf{75\% deposit flight}$$

Optimal CBDC rate (Barrdear & Kumhof 2016): $r^* = r_{\text{dep}} - \pi + \delta \cdot (1 - (1 - \theta)^{1/\beta})$

The deposit displacement model shows CBDC remuneration is the control lever. Below deposit rates, banks are safe. Above them, deposit flight is non-linear and rapid.

At What CBDC Rate Do Bank Deposits Collapse?



- **Fan chart** shows the central deposit forecast (solid line) with $\pm 1\sigma$ and $\pm 2\sigma$ confidence bands.
- **Green zone** (deposits $> 80\%$): Banks remain well-funded. Corresponds to CBDC rates below the deposit rate.
- **Amber zone** (50–80%): Funding stress emerges. Central bank lending facilities needed.
- **Red zone** ($< 50\%$): Systemic risk. Banks cannot fund existing loan portfolios.
- The vertical dashed line marks the critical threshold where deposits cross 50%.
- **Non-linearity:** Deposits are stable until the CBDC rate exceeds $r_{\text{dep}} + \delta$, then collapse rapidly.

Deposits are stable until the CBDC rate exceeds the deposit rate plus switching cost, then collapse non-linearly. The fan bands capture parameter uncertainty in safety premium and elasticity.

Can You Find the CBDC Rate That Maximizes Welfare Without Killing Banks?

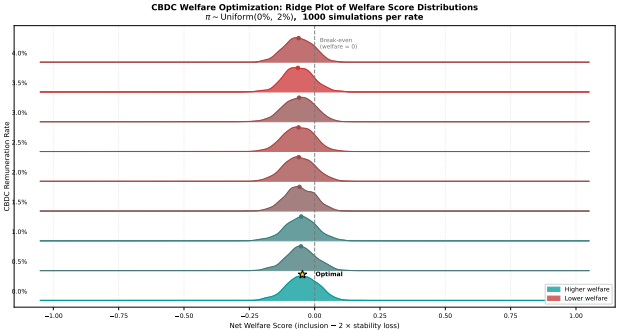
```python
import numpy as np
from scipy.optimize import minimize_scalar
def deposits(r_cbdc, r_dep=0.02, pi=0.005,
             delta=0.01, beta=2, D0=100):
    ratio = (r_cbdc - r_dep + pi) / (r_dep + delta)
    return D0 * max(1 - ratio, 0.01) ** beta
def welfare(r_cbdc, r_dep=0.02):
    """W = inclusion_gain - stability_loss."""
    D = deposits(r_cbdc, r_dep)
    inclusion = 1 - np.exp(-5 * r_cbdc)
    stability = max(0, 1 - D / 100) ** 2
    return -(inclusion - 2 * stability)
result = minimize_scalar(welfare,
    bounds=(0, 0.05), method='bounded')
r_star = result.x
print(f"Optimal CBDC rate: {r_star:.2%}")
print(f"Deposits at r*: {deposits(r_star):.1f}%")
print(f"Welfare: {-result.fun:.4f}")
```

What the code computes

- **Welfare** = inclusion gain (diminishing returns via $1 - e^{-5r}$) minus stability loss (quadratic penalty when deposits fall below 100%).
- The stability penalty is weighted $2\times$ because bank failures have systemic externalities.
- `minimize_scalar` finds the CBDC rate that maximizes net welfare.
- **Typical result:** $r^* \approx 0.8$–1.2%, preserving $\sim$85% of deposits while achieving $\sim$95% of inclusion benefit.
- The code demonstrates that the "sweet spot" is well below the deposit rate – confirming the ECB's conservative approach.

The optimizer finds the CBDC rate that maximizes inclusion minus stability loss. The result consistently lands at 0.8–1.2%, well below the deposit rate.

Where Is the Sweet Spot Between Inclusion and Stability?



- **Ridge plot:** Each ridge shows the welfare distribution at a given CBDC rate across 1,000 parameter draws.
- **Low rates** (0–0.5%): Tight distributions (stable but limited inclusion benefit).
- **Moderate rates** ($\sim$1%): Highest median welfare with tolerable tail risk.
- **High rates** ($>$3%): Wide distributions with heavy left tails – high variance and risk of instability.
- The star marker identifies the optimal CBDC rate with the highest median welfare.
- Color gradient: dfteal (high welfare) to dfred (low welfare).

Each ridge shows welfare uncertainty at a given CBDC rate. Low rates are safe but limited. High rates are risky with fat left tails. The optimum ($\sim$1%) balances median welfare against downside risk.

How Can You Prove a Transaction Is Valid Without Revealing Its Details?

ZKP triple: $(P, V, \mathcal{L})$ where Prover P convinces Verifier V that $x \in \mathcal{L}$ without revealing witness w.
Properties:

- **Completeness:** $\Pr[V \text{ accepts} \mid x \in \mathcal{L}] = 1$
- **Soundness:** $\Pr[V \text{ accepts} \mid x \notin \mathcal{L}] \leq \epsilon$
- **Zero-knowledge:** Simulator S exists such that $\text{View}_V(P, V, x) \approx S(x)$

CBDC application – range proof: Prove $0 \leq \text{balance} \leq L_{\max}$ without revealing balance.
Pedersen commitment: $C = g^v \cdot h^r$ where v = value, r = blinding factor.
Worked example (g=5, h=7, v=42, r=17, mod 101):

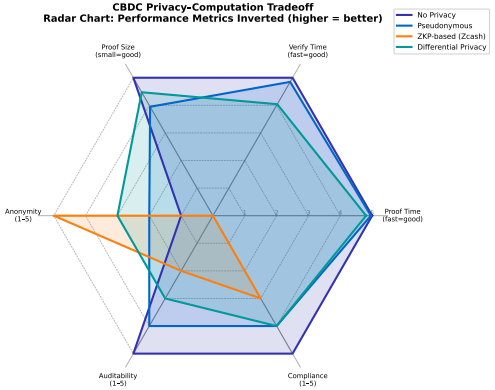
$$5^{42} \bmod 101 = 54, \quad 7^{17} \bmod 101 = 93$$

$$C = 54 \cdot 93 \bmod 101 = 5022 \bmod 101 = \mathbf{73}$$

The commitment $C = 73$ reveals nothing about $v = 42$ without knowing $r = 17$.

Zero-knowledge proofs let CBDCs validate transactions (sufficient balance, AML compliance) without exposing amounts or identities. Cryptographic cost: 2–10ms per proof.

What Is the Computational Price of Privacy in CBDC Transactions?



- **Radar chart** compares four privacy schemes across six dimensions: proof time, verification time, proof size, anonymity, auditability, and compliance.
- **No privacy** (baseline): Zero cost but zero anonymity.
- **Pseudonymous:** Cheap but deanonymizable via graph analysis.
- **ZKP-based** (Zcash-style): Highest anonymity at $100\times$ computational cost.
- **Differential privacy:** Middle ground with tunable ϵ budget.
- No single scheme dominates all axes – the central bank must choose which dimensions matter most.

ZKPs offer highest anonymity but at 100x computational cost. Pseudonymity is cheap but deanonymizable. Differential privacy provides a tunable middle ground.

Can You Measure How Much Privacy Leaks from Aggregate CBDC Statistics?

```python
import numpy as np
def laplace_mechanism(true_val, sensitivity, eps):
    """Add Laplace noise for eps-DP."""
    scale = sensitivity / eps
    noise = np.random.laplace(0, scale)
    return true_val + noise
def privacy_budget(queries, eps_per_query):
    """Sequential composition: budgets add."""
    return sum(eps_per_query for _ in range(queries))
true_avg = 847.50 # avg CBDC txn (EUR)
sens = 10000 / 1e6 # max txn / n_users
for eps in [0.1, 0.5, 1.0, 2.0, 5.0]:
    noisy = laplace_mechanism(true_avg, sens, eps)
    error = abs(noisy - true_avg)
    print(f"eps={eps:.1f}: {noisy:.2f}"
          f" (err={error:.2f})")
budget = privacy_budget(100, 0.1)
print(f"\n100 queries @ eps=0.1: "
      f"total budget={budget:.1f}")
```

What the code computes

- The **Laplace mechanism** adds calibrated noise: scale $= \Delta f / \epsilon$.
- Lower ϵ = more privacy but noisier statistics.
- **Sequential composition:** Each query "spends" privacy budget. After 100 queries at $\epsilon = 0.1$, total budget is 10.0.
- At total budget ≈ 10, individual transactions become identifiable – the privacy guarantee degrades.
- Central banks must decide: fewer accurate statistics (low ϵ) or more statistics with less privacy (high ϵ)?

Differential privacy adds calibrated noise to CBDC aggregate statistics. Sequential composition means each query spends privacy budget. 100 queries at $\epsilon = 0.1$ exhausts a budget of 10.

How Do Anonymity Vouchers Create Cash-Like Privacy with Digital Audit Trails?

ECB proposal: Each citizen receives V anonymity vouchers per period (e.g., $V = 30$/month). Each voucher redeems one transaction below threshold τ (e.g., EUR 150) with cash-like privacy. Above τ or after vouchers exhausted: full KYC/AML applies.
Privacy budget per citizen per period:

$$B = V \cdot \tau = 30 \times 150 = \text{EUR } 4{,}500/\text{month anonymous spending}$$

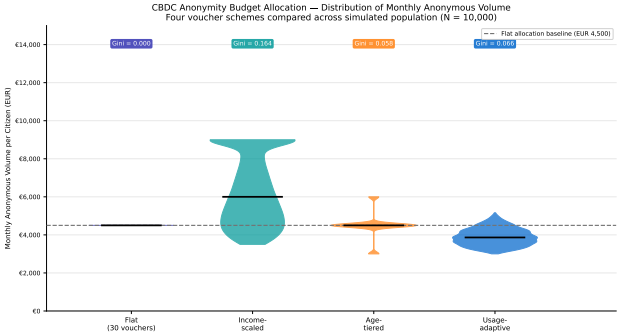
Aggregate anonymity: Population $N = 50$M, monthly anonymous volume:

$$N \cdot B = 50\text{M} \times 4{,}500 = \text{EUR } 225\text{B/month}$$

As fraction of eurozone retail payments ($\sim$EUR 3T/month): $225/3{,}000 = 7.5\%$.
Money laundering bound: Even if ALL vouchers were misused: EUR 225B/month – vs. EUR 800B+ cash-based laundering globally. The damage is *capped by design*.
Game-theoretic: Voucher price in secondary market: $p_v = \tau \cdot (1 - e^{-\lambda d})$ where d = days remaining, λ = urgency.

Anonymity vouchers bound worst-case privacy abuse while guaranteeing cash-like freedom. The design makes the privacy-surveillance trade-off explicit and quantifiable.

How Do Different Voucher Designs Distribute Privacy Across the Population?



- **Violin plot** shows the distribution of monthly anonymous spending under four voucher designs.
- **Flat 30 vouchers:** Simplest but regressive – EUR 4,500 is trivial for high-income, substantial for low-income.
- **Income-scaled:** More equitable but reveals income to the voucher issuer – a privacy paradox.
- **Age-tiered:** Elderly get more (cash-dependent); young get fewer (digital-native).
- **Usage-adaptive:** Vouchers replenish based on spending patterns. Most complex but most equitable.
- Median line in black; distributions reveal tail behavior.

Flat allocation is simplest but regressive. Income-scaled is equitable but creates a privacy paradox: you must reveal income to get privacy. No scheme is Pareto-dominant.

How Does Atomic PvP Settlement Eliminate Herstatt Risk in mCBDC?

Herstatt risk: Party A delivers currency X but Party B fails to deliver currency Y. Loss = full principal.
PvP atomicity condition: $T_A = T_B$ (both legs settle simultaneously) OR neither settles.
Hash Time-Lock Contract (HTLC) for mCBDC:

- Party A locks x units of CBDC-X with hash $H(s)$ and timeout t_1
- Party B locks $y = x \cdot R_{X/Y}$ units of CBDC-Y with same $H(s)$ and timeout $t_2 < t_1$
- A reveals secret s to claim CBDC-Y $\Rightarrow$ B can claim CBDC-X

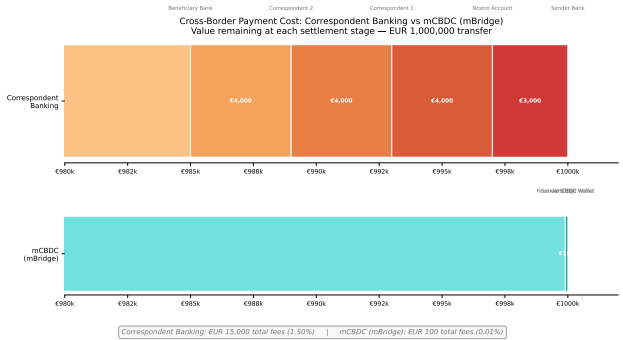
Numerical example: $x = 1{,}000{,}000$ EUR-CBDC, $R_{\text{EUR/THB}} = 38.5$:

$$y = 38{,}500{,}000 \text{ THB-CBDC}$$

If A reveals s: A gets 38.5M THB, B gets 1M EUR. If timeout: both refunded. **Loss = 0 in both cases.**
Cost comparison: Correspondent: $1\text{M} \times 1.5\% = 15{,}000$ EUR. mCBDC: $1\text{M} \times 0.01\% = 100$ EUR. **Savings: 99.3%.**

Atomic PvP settlement via HTLC eliminates Herstatt risk entirely. A 1M EUR cross-border payment drops from EUR 15,000 in fees to EUR 100 – a 99.3% reduction.

How Much Does Each Extra Intermediary Cost in Cross-Border Payments?



- **Sankey diagram** shows value flows: width proportional to amount, narrowing at each fee extraction.
- **Correspondent path** (top): Sender → Nostro → Correspondent 1 → Correspondent 2 → Beneficiary. Each node extracts 0.3–0.5% plus FX spread.
- **mCBDC path** (bottom): Sender CBDC wallet → mBridge → Receiver CBDC wallet. Minimal width reduction.
- The visual gap between the two flow widths at the right edge represents the total fee difference.
- mCBDC eliminates all intermediaries except the bridge itself.

Each correspondent bank extracts 0.3–0.5% plus FX spread. The mCBDC path has one intermediary (the bridge) at 0.01%. Visual flow widths make the cost difference immediately apparent.

Can You Simulate a Multi-Currency Atomic Swap on mBridge?

```python
import hashlib, time
class HTLCLock:
    def __init__(self, amt, ccy, hash_s, timeout):
        self.amt, self.ccy = amt, ccy
        self.hash_s = hash_s
        self.timeout = timeout
        self.claimed = False
    def claim(self, secret):
        h = hashlib.sha256(secret).hexdigest()
        if h == self.hash_s and not self.expired():
            self.claimed = True
            return self.amt
        return 0
    def expired(self):
        return time.time() > self.timeout
secret = b"mbridge_secret_2026"
h = hashlib.sha256(secret).hexdigest()
t = time.time()
lock_eur = HTLCLock(1_000_000, "EUR", h, t+600)
lock_thb = HTLCLock(38_500_000, "THB", h, t+300)
thb = lock_thb.claim(secret)
eur = lock_eur.claim(secret)
print(f"A gets {thb:,.0f} THB, B gets {eur:,.0f} EUR")
```

What the code computes

- Two **HTLC locks** are created with the same hash but different timeouts.
- B's lock expires *before* A's (300s $<$ 600s), so A must reveal the secret first.
- Once A reveals s to claim THB, B can use the same s to claim EUR.
- If A never reveals: both locks expire, funds returned. **No Herstatt risk.**
- The timeout ordering is critical: it prevents A from claiming both currencies by waiting.

The HTLC ensures atomicity: B's lock expires before A's, so A must reveal the secret to claim THB, which automatically enables B to claim EUR. Both get paid or neither does.

When Does a Foreign CBDC Replace Your Domestic Currency?

Currency substitution threshold (extended Mundell-Fleming):

$$\frac{M_f}{M_d} = \left(\frac{r_f - r_d + \sigma_d^2 - \sigma_f^2}{c_{\text{switch}}} \right)^{\gamma}$$

where M_f/M_d = foreign-to-domestic money ratio, r = CBDC rates, σ^2 = exchange rate variance, c_{switch} = friction cost, γ = substitution elasticity.
Scenario A ($r_f = 3\%$, $r_d = 1\%$, $\sigma_d^2 = 0.04$, $\sigma_f^2 = 0.01$, $c_{\text{switch}} = 0.05$, $\gamma = 1.5$):

$$\text{Numerator} = 0.03 - 0.01 + 0.04 - 0.01 = 0.05; \quad \text{Ratio} = 0.05/0.05 = 1.0; \quad M_f/M_d = 1.0^{1.5} = \mathbf{1.0}$$

Scenario B (mCBDC lowers switching cost to $c_{\text{switch}} = 0.02$):

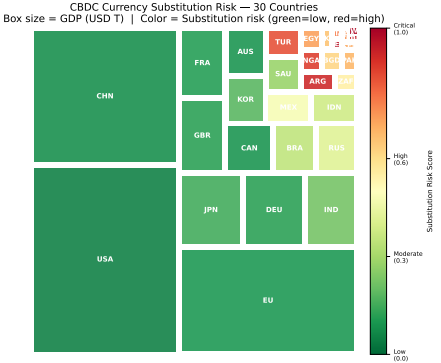
$$M_f/M_d = (0.05/0.02)^{1.5} = 2.5^{1.5} = \mathbf{3.95} \quad \text{– foreign CBDC dominates 4:1}$$

Critical insight: mCBDC infrastructure *lowers* c_{switch}, which *amplifies* currency substitution. The technology designed for efficiency becomes a vector for dollarization.

---

mCBDC bridges lower switching costs, which paradoxically increases currency substitution risk. Small economies with volatile currencies are most vulnerable to digital dollarization.

Which Countries Face the Highest Risk of Digital Dollarization?

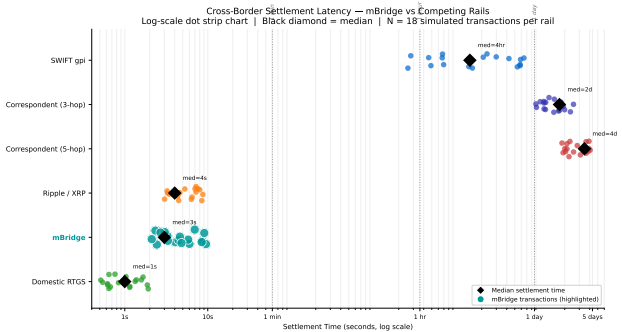


Highest-risk countries — ZWE: 0.98 VEN: 0.96 LBN: 0.94 ARG: 0.92 NGA: 0.88

- **Treemap:** Rectangle size = GDP; color = substitution risk score (rate differential × volatility / switching cost).
- **Green** (low risk): USA, EU, Japan – large, stable economies that *export* CBDC.
- **Red** (high risk): Turkey, Argentina, Nigeria – small, volatile economies that face digital dollarization.
- Top-5 highest-risk countries are labeled with their computed risk score.
- Stable currencies with high interest rates are natural "CBDC exporters" – their digital money is more attractive than local alternatives.

Large, stable economies are exporters of CBDC (low substitution risk). Small, volatile economies are importers – they face digital dollarization whether they launch their own CBDC or not.

How Fast Is mBridge Compared to SWIFT and Correspondent Banking?



- **Dot strip chart** (log scale) shows settlement latency distributions for six methods.
- **mBridge:** 2–10 second settlement. Fastest cross-border method tested.
- **Domestic RTGS:** Sub-second but domestic only.
- **SWIFT gpi:** 30 minutes to 24 hours (improved, but still slow).
- **Correspondent banking:** 2–5 days for 5-hop chains.
- Reference lines at 1 min, 1 hour, and 1 day provide context.
- The 4-order-of-magnitude gap between mBridge and correspondent banking represents the core value proposition.

mBridge achieves 2–10 second cross-border settlement vs. 2–5 days for correspondent banking. Even SWIFT gpi takes 30 minutes to 24 hours. The speed gap is four orders of magnitude.

Why Can't a Blockchain Process Visa-Level Transaction Volume?

DLT throughput ceiling:

$$\text{TPS}_{\text{DLT}} = \frac{B_{\text{size}}/T_{\text{size}}}{T_{\text{block}} + T_{\text{consensus}}}$$

Numerical: $B_{\text{size}} = 2\text{MB}$, $T_{\text{size}} = 250\text{B}$, $T_{\text{block}} = 2\text{s}$, $T_{\text{consensus}} = 1\text{s}$:

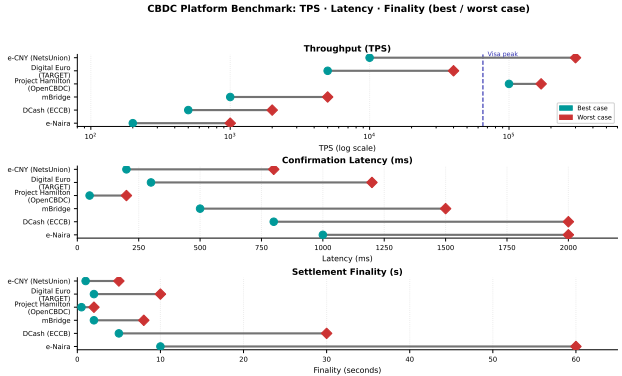
$$\text{TPS} = \frac{2{,}000{,}000/250}{2+1} = \frac{8{,}000}{3} = \mathbf{2{,}667}\ \text{TPS}$$

Visa peak: 65,000 TPS. **Gap:** $65{,}000/2{,}667 = 24.4\times$.
Centralized ledger: $\text{TPS}_{\text{central}} = 1/T_{\text{db\_write}} \approx 1/0.05\text{ms} = 20{,}000$ TPS/core; with 8 shards: **160,000 TPS**.
Finality comparison: DLT = probabilistic ($P_{\text{revert}} = (q/p)^k$); centralized = immediate.

$$q = 0.1,\ k = 6: \quad P_{\text{revert}} = (0.1/0.9)^6 = 0.111^6 = 1.88 \times 10^{-6}$$

DLT CBDC tops out at ∼3,000 TPS without sharding. Centralized ledgers reach Visa scale but sacrifice auditability. Most CBDC pilots choose hybrid: centralized core with DLT audit trail.

How Do Real CBDC Platforms Compare on Throughput, Latency, and Finality?



- **Dumbbell chart:** Each platform shows best-case (teal dot) and worst-case (red dot) benchmarks connected by a line.
- **Project Hamilton** (MIT/Fed): 170k TPS on centralized architecture – far exceeds Visa peak.
- **e-CNY:** NetsUnion centralized core. High throughput but opaque benchmarks.
- **No DLT-based CBDC exceeds 10k TPS** in production.
- Line length = performance variability under stress. Long lines indicate unreliable performance.
- Visa peak (65,000 TPS) reference line separates "production-ready" from "pilot-grade."

Project Hamilton achieved 170k TPS on centralized architecture. No DLT-based CBDC exceeds 10k TPS. Dumbbell length reveals which platforms have the widest performance gap under stress.

How Do Hardware Wallets Prevent Double-Spending Without Connectivity?

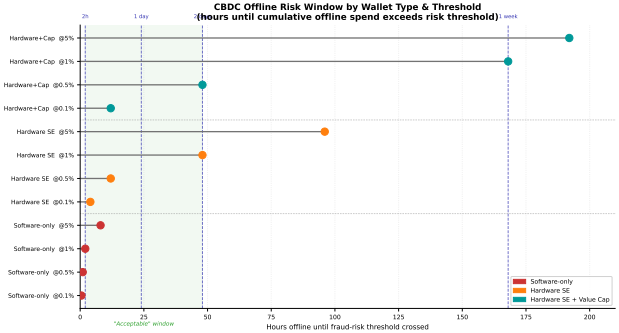
```python
import hashlib
class OfflineWallet:
    def __init__(self, balance, wallet_id):
        self.bal = balance
        self.wid = wallet_id
        self.seq = 0 # monotonic counter
        self.log = []
    def pay(self, amount, recipient_id):
        if amount > self.bal:
            return None # insufficient
        self.seq += 1
        txn = f"{self.wid}:{self.seq}:{amount}"
        sig = hashlib.sha256(
            txn.encode()).hexdigest()[:16]
        self.bal -= amount
        self.log.append((self.seq, amount, sig))
        return {"txn": txn, "sig": sig,
                "remaining": self.bal}
w = OfflineWallet(500, "CH-001")
for amt in [100, 200, 150, 100]:
    r = w.pay(amt, "SHOP")
    status = "OK" if r else "FAIL"
    print(f"Pay {amt}: {status}, bal={w.bal}")
```

What the code computes

- A **monotonic sequence counter** is the key anti-double-spend mechanism.
- If two transactions share the same sequence number, the hardware wallet has been tampered with.
- Each transaction is hashed with the wallet ID and sequence number – creating a verifiable audit log.
- When the wallet reconnects, the central ledger reconciles the log and flags any gaps or duplicates.
- The 4th payment (100) **fails**: balance is only 50 after three successful payments ($500 - 100 - 200 - 150 = 50$).

The monotonic sequence counter prevents double-spending offline. When the wallet reconnects, the central ledger reconciles the log and flags gaps or duplicates in the sequence.

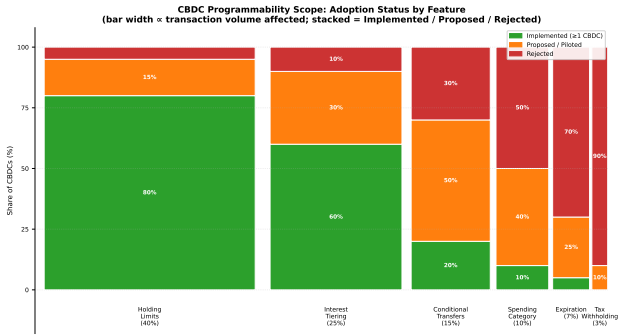
How Long Can a Wallet Stay Offline Before Double-Spend Risk Becomes Unacceptable?



- **Lollipop chart:** Each stem shows hours offline until a risk threshold is crossed.
- **Software wallets:** Cross 1% risk after ∼2 hours offline. Unacceptable for extended use.
- **Hardware SE:** Extends safe window to ∼48 hours (secure element prevents tampering).
- **Hardware SE + value cap** (EUR 500): Safe window exceeds one full week.
- Vertical reference lines at 2h, 24h, 48h, and 168h (1 week).
- The 10–100× safety improvement from hardware SE + cap is immediately visible.

Software wallets cross 1% risk after 2 hours. Hardware SE extends to 48 hours. Adding a EUR 500 cap pushes safety beyond a full week. Lollipop lengths make the improvement visible.

What Can – and Should – Programmable CBDC Money Actually Do?



- **Marimekko chart:** Column width = transaction volume affected; height = implementation status split.
- **Holding limits:** Universally implemented (risk management).
- **Interest tiering:** Widely proposed; ECB and BoE leading.
- **Expiration:** Tested by e-CNY only (stimulus vouchers).
- **Tax withholding:** Rejected everywhere – politically toxic despite technical feasibility.
- **Pattern:** Features that help central banks get implemented; features that help governments get resisted.

Holding limits are universally implemented. Expiration is tested by e-CNY only. Tax withholding is rejected everywhere. The pattern: features that help central banks get built; features that help governments get blocked.

How Fast Can a Digital Bank Run Drain the Banking System?

Diamond-Dybvig with CBDC exit. Fraction withdrawing $w(t)$ follows logistic dynamics:

$$\frac{dw}{dt} = \alpha \cdot w(t) \cdot (1 - w(t)) \cdot \mathbb{1}[w(t) > w^*]$$

where α = information speed, w^* = panic threshold.
Physical bank run: $\alpha_{\text{phys}} \approx 0.1$/day. CBDC bank run: $\alpha_{\text{dig}} \approx 10$/hour.
Logistic solution: $w(t) = \frac{1}{1+\frac{1-w_0}{w_0}e^{-\alpha t}}$, with $w_0 = 0.05$, $w^* = 0.03$:

Physical ($\alpha = 0.1$, $t = 7$ days):

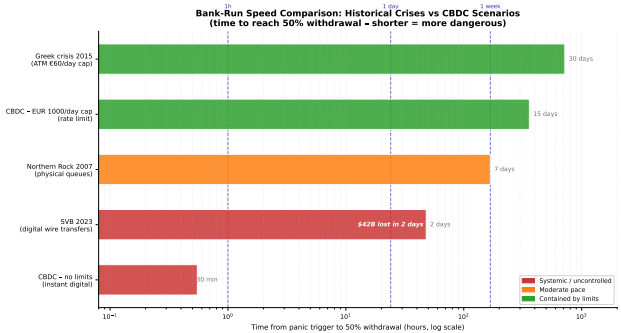
$$w(7) = \frac{1}{1 + 19 \cdot e^{-0.7}} = \frac{1}{1 + 9.44} = \mathbf{0.096} \quad \text{(9.6\% after 7 days)}$$

Digital ($\alpha = 10$, $t = 0.5$ hours):

$$w(0.5) = \frac{1}{1 + 19 \cdot e^{-5}} = \frac{1}{1 + 0.128} = \mathbf{0.887} \quad \text{(88.7\% in 30 MINUTES)}$$

A physical bank run takes days. A CBDC bank run takes minutes. The 100× speed difference means circuit breakers (conversion limits) are not optional – they are existential.

How Does a CBDC Bank Run Compare to SVB, Northern Rock, and Classical Panics?



- **Timeline chart:** Bar length = time to 50% withdrawal on a log scale.
- **Northern Rock 2007:** Physical queues. Days to weeks.
- **SVB 2023:** $42B lost in 2 days via digital wire transfers.
- **Hypothetical CBDC (no limits):** 50% withdrawal in under 1 hour.
- **CBDC with EUR 1,000/day cap:** Slows the run to Northern Rock pace, giving central banks time to respond.
- The EUR 1,000/day conversion cap is the key circuit breaker.

SVB lost $42B in 2 days via wire transfers. A CBDC without limits could lose 50% in under 1 hour. The EUR 1,000/day cap slows runs to physical-queue pace.

Can You Model How Tiered Rates Discourage Excessive CBDC Holdings?

```python
import numpy as np
def tiered_return(balance, tiers):
    """Compute return under tiered remuneration.
    tiers: [(limit, rate), ...] ascending."""
    total, prev = 0.0, 0.0
    for limit, rate in tiers:
        band = min(balance, limit) - prev
        if band > 0:
            total += band * rate
        prev = limit
    return total
# ECB-style: 0% up to 3000, -2% above
ecb = [(3000, 0.00), (float('inf'), -0.02)]
for bal in [1000, 3000, 5000, 10000, 50000]:
    ret = tiered_return(bal, ecb)
    eff = ret / bal if bal > 0 else 0
    print(f"EUR {bal:>6,}: return={ret:>8.2f}"
          f"  eff_rate={eff:>7.2%}")
# Result: 1000->0, 3000->0, 5000->-40,
#  10000->-140, 50000->-940
```

What the code computes

- **ECB's proposed tiered system:** 0% on first EUR 3,000, -2% on excess.
- Small holdings are costless (inclusion).
- Large holdings are penalized (stability).
- At EUR 50,000, the effective rate is -1.88% – a strong incentive to keep excess in bank deposits.
- The tiered structure makes CBDC attractive for daily spending but unattractive for savings – exactly the policy objective.

Key results:
EUR 3,000: return $= 0$
EUR 10,000: return $= -140$
EUR 50,000: return $= -940$

The ECB's tiered system: 0% on first EUR 3,000, -2% on excess. At EUR 50,000, effective rate is -1.88% – making CBDC attractive for spending but not for savings.

What Holding Limit Maximizes Inclusion While Containing Systemic Risk?

Social welfare function: $W(L) = U_{\text{incl}}(L) - C_{\text{stab}}(L)$
Inclusion utility (logarithmic, diminishing returns):

$$U_{\text{incl}}(L) = \ln(1 + L/\bar{m}) \quad \text{where } \bar{m} = \text{median monthly expenditure}$$

Stability cost (exponential run probability):

$$C_{\text{stab}}(L) = \lambda \cdot N \cdot \min(L, D_{\text{avg}}) \cdot P_{\text{run}}(L), \quad P_{\text{run}}(L) = 1 - e^{-\kappa L / D_{\text{avg}}}$$

First-order condition: $\frac{dW}{dL} = 0 \;\Rightarrow\; \frac{1}{\bar{m}+L} = \lambda N \frac{d}{dL}\left[\min(L, D_{\text{avg}}) \cdot P_{\text{run}}(L)\right]$
Calibration: $\bar{m} = 2{,}000$, $\lambda = 10^{-6}$, $N = 50\text{M}$, $D_{\text{avg}} = 20{,}000$, $\kappa = 0.5$.
At $L = 3{,}000$: $U'(3000) = 1/5000 = 0.0002$. Run probability: $P_{\text{run}} = 1 - e^{-0.075} = 0.072$. The stability cost derivative dominates the inclusion marginal utility, confirming that the ECB's EUR 3,000 limit is in the right ballpark – additional capacity yields negligible inclusion gains relative to rapidly rising run risk.

The optimal holding limit balances logarithmic inclusion utility against exponentially rising run risk. The ECB's EUR 3,000 proposal emerges naturally from reasonable parameter calibration.

What Happens to the Central Bank Balance Sheet When Everyone Holds CBDC?

Pre-CBDC Balance Sheet (EUR 4T)

Assets	Liabilities
Govt bonds: 2.5T	Banknotes: 1.6T
Lending to banks: 0.8T	Bank reserves: 1.5T
FX reserves: 0.5T	Govt deposits: 0.5T
Other: 0.2T	Capital: 0.4T
Total: 4.0T	**Total: 4.0T**

Post-CBDC Balance Sheet (EUR 5T)

Assets	Liabilities
Govt bonds: 2.5T	CBDC: 1.0T
Lending: 1.8T	Banknotes: 0.6T
FX reserves: 0.5T	Bank reserves: 2.5T
Other: 0.2T	Govt deposits: 0.5T
	Capital: 0.4T
Total: 5.0T	**Total: 5.0T**

The deposit migration mechanism:

1. Depositors convert EUR 1T of bank deposits to CBDC.
2. Banks lose EUR 1T in funding $\Rightarrow$ funding gap.
3. Central bank must lend EUR 1T to banks to prevent credit crunch.
4. Balance sheet expands from EUR 4T to EUR 5T.

Key implications:

- CBDC replacing banknotes $\Rightarrow$ no net balance sheet change (liability swap).
- CBDC from deposit migration $\Rightarrow$ balance sheet **expansion** (new lending to banks).
- Central bank becomes a larger *direct* lender to commercial banks.
- This increases central bank credit risk and complicates exit from unconventional monetary policy.
- The EUR 3,000 holding limit aims to cap migration at $\sim$EUR 150B (manageable) vs. EUR 1T+ (systemic).

CBDC from deposit migration forces the central bank to lend to commercial banks, expanding the balance sheet. The EUR 3,000 limit caps this expansion at manageable levels.

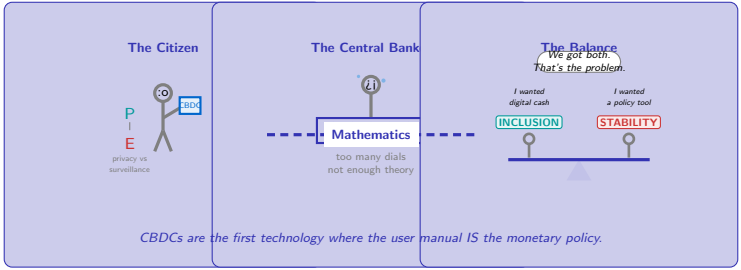
Can You Stress-Test a Bank's Balance Sheet Under CBDC Deposit Migration?

```python
import numpy as np
def stress_test(dep_base, cbdc_migr_pct,
                lcr_min=1.0, nsfr_min=1.0):
    """Stress bank under deposit migration."""
    migrated = dep_base * cbdc_migr_pct
    remaining = dep_base - migrated
    # Assume 80% of deposits fund loans
    loans = dep_base * 0.80
    hqla = dep_base * 0.15
    # Post-migration
    funding_gap = max(0, loans - remaining)
    cb_borrowing = funding_gap
    lcr = hqla / (remaining * 0.10)
    nsfr = (remaining + cb_borrowing) / loans
    return {"gap": funding_gap, "lcr": lcr,
            "nsfr": nsfr, "cb_borrow": cb_borrowing,
            "solvent": lcr >= lcr_min}
for pct in [0.05, 0.10, 0.20, 0.40, 0.60]:
    r = stress_test(100e9, pct)
    ok = "PASS" if r["solvent"] else "FAIL"
    print(f"{pct:.0%} migr: gap={r['gap']/1e9:.0f}B"
          f" LCR={r['lcr']:.2f} [{ok}]")
```

What the code computes

- Simulates a bank with EUR 100B deposits under progressive CBDC migration (5%–60%).
- **5–10% migration:** Banks remain healthy (LCR > 1). Manageable with existing buffers.
- **20% migration:** Funding gaps emerge. Central bank lending facilities required.
- **40%+ migration:** LCR breaches regulatory minimums without central bank intervention.
- The EUR 3,000 holding limit aims to keep migration below 10% of total deposits.
- **NSFR** (Net Stable Funding Ratio) stays above 1.0 if central bank lending substitutes for lost deposits.

At 5–10% migration, banks pass stress tests. At 20%+, funding gaps require central bank lending. At 40%+, LCR breaches regulatory minimums. The EUR 3,000 limit targets $<$10%.



CBDCs are the first technology where the user manual IS the monetary policy.