CYPHERPUNK VALUES · OPEN WEIGHTS · VERIFIABLE INFERENCE

Your prompt is someone else's training data

Privacy for the weak. Transparency for the strong.
Open weights, local inference, and confidential routing
that needs no enclave and no account.

We won the crypto war and lost the terminal

1993 asked: can two strangers talk without a middleman? 2026 asks: can two strangers compute without a middleman?

> **Cypherpunks write code.** Encryption is a property of the protocol, not a favour from the operator.
> **Verify, don't trust.** A claim you cannot check is marketing. A claim you can check is a proof.
> **Exit beats voice.** If you cannot leave, you were never a customer, you were inventory.
> **Strong crypto for everyone**, not a premium tier for the compliant.
> **No permission needed.** No account, no KYC, no email, no flag on a list.
> **Bearers, not balances.** Cash beats an IOU. A key you hold beats a login you rent.
> **Small trust surfaces.** Every trusted component is a future betrayal with a business model.
> **Code you can run offline** is the only code that cannot be revoked from you.

> “Cypherpunks write code. We know that someone has to write software to defend privacy.” — Eric Hughes, A Cypherpunk's Manifesto, 1993

> The 1990s fight was about **messages**. PGP won the argument; the terminals still died, because convenience was never solved.
> The 2020s fight is about **inference**: every prompt is a confession, and the default stack ships it in cleartext to a server that keeps it, reads it, and bills you for the privilege.

Four adversaries, and what actually stops each one

Name the adversary before you name the mitigation. Otherwise you are buying vibes.

ADVERSARY	WHAT THEY SEE TODAY	WHAT BREAKS YOU	WHAT ACTUALLY MITIGATES
Cloud provider	Prompt, response, account, IP, timing, retention	Logs subpoenaed, breached, or sold as "improvement"	Local weights; or E2EE routing where the provider sees one request and no identity
Routing node	Everything, because it is a full MITM	It can log, edit, or swap the model and re-bill you	Confidential mode: node relays ciphertext only, model and cap are ZK-pinned
Network	Ciphertext, volume, endpoints	Traffic analysis reconstructs responses from chunk sizes	TLS to the provider, not to the node; accept the volume leak or pad
Platform	Whatever the app can reach: clipboard, files, screen	The agent is the malware surface	Local model, local app, no account, no telemetry, sandboxing

ASSUME

Provider TLS is real. WebPKI + your own trusted-host pin is the anchor you check yourself.

DON'T ASSUME

That a node's good reviews mean it is honest today. Reputation is a lagging indicator, not a proof.

NEVER ASSUME

That "encrypted" means end-to-end. Encrypted to the operator is a promise; encrypted past the operator is math.

What you can actually download this quarter

Weights you can hold are weights nobody can revoke, rate-limit, or retrain against you.

MODEL	PARAMS	LICENCE	WHY IT MATTERS	CONSUMER REACH
Qwen3.8-27B	~28B	Apache-2.0	The default open workhorse; multimodal, clean licence	24–32 GB at 4-bit — laptop class
Qwen3.8-Flash-Next	~180B MoE	open weights	Frontier-adjacent quality at MoE serving cost	~96 GB at 4-bit — 128 GB workstation
DeepSeek-V4.1-Flash-1017	~763B MoE	MIT	MIT licence on a frontier-class model; sparse, cheap per token	Server / 512 GB+ at low bits
GLM-5.3-Flash	~321B MoE	open weights	Strong agentic and tool-use behaviour	~170 GB at 4-bit — 2×128 GB
GLM-5.3	~753B MoE	MIT	Near-frontier reasoning, permissive	Server class
Kimi-K3 / K3.1 Pro	~2.8T MoE	open weights	Genuinely open and genuinely huge; long context, agentic	Not consumer. Open weight ≠ locally runnable

READ THE LICENCE, NOT THE LEADERBOARD

MIT and Apache-2.0 survive contact with a business. "Open weights" with acceptable-use clauses and MAU caps is a rental agreement with extra steps.

THE GAP IS NOW MONTHS, NOT YEARS

Top open models trail the best closed ones by a release cycle or two. For most work, that gap is not worth giving up exit rights.

What runs fast on a machine you can buy

Decode is memory-bandwidth-bound. Buy RAM before you buy FLOPs.

HARDWARE	REALISTIC MODEL	QUANT	FOOTPRINT	DECODE	VERDICT
16 GB laptop	Qwen3.8-27B	Q4_K_M	~16 GB	4–7 tok/s	Tight; short context only
24–32 GB Apple silicon	**Qwen3.8-27B**	MLX 4-bit	~15–16 GB	**12–18 tok/s**	The sweet spot
32 GB + discrete GPU (24 GB VRAM)	Qwen3.8-27B	Q4 + q8 KV	~16 GB	25–40 tok/s	Fastest per dollar
64 GB unified	Qwen3.8-27B @ 8-bit, long ctx	MLX 8-bit	~30 GB	8–12 tok/s	Quality over speed
128 GB unified	Qwen3.8-Flash-Next	Q4	~96 GB	15–22 tok/s	MoE sparsity pays off
256–512 GB workstation	GLM-5.3-Flash	Q4 / MXFP4	170–190 GB	8–15 tok/s	Homelab frontier
Phone, 8–12 GB	3–8B class distills	Q4_0 / 4-bit	2–5 GB	8–20 tok/s	Always with you, never online

THE RAM RULE

4-bit ≈ **params × 0.55 GB**. Add KV cache: 32k context on a 27B at q8 KV is several GB more. MoE still needs all experts resident.

WHY MOE WINS LOCALLY

You pay RAM for the whole model but compute only the active experts, so a 180B MoE decodes like a small dense model on the same bandwidth.

BANDWIDTH, NOT TOPS

A 27B at 4-bit is ~16 GB read per token. 200 GB/s gives a hard ceiling near 12 tok/s — no kernel will beat arithmetic.

Same weights, four runtimes

Pick by hardware, not by hype. Flags below are illustrative — check the current docs.

```
# ollama — the 30-second path
ollama run qwen3.8:27b
# OpenAI-compatible on :11434
curl localhost:11434/v1/chat/completions \
  -d '{"model":"qwen3.8:27b", ...}'
```

```
# MLX — fastest on Apple silicon
mlx_lm.generate \
  --model mlx-community/Qwen3.8-27B-MLX-4bit \
  --prompt "explain Chaumian ecash" \
  --max-tokens 512
```

```
# llama.cpp — most control
llama-server -hf unsloth/Qwen3.8-27B-GGUF:Q4_K_M \
  -c 32768 -ngl 99 --flash-attn on \
  -ctk q8_0 -ctv q8_0 --jinja
```

```
# vLLM — many users, one box
vllm serve Qwen/Qwen3.8-27B \
  --max-model-len 32768 \
  --gpu-memory-utilization 0.92
```

> **Quantize first.** Q4_K_M or MLX 4-bit costs a little quality and buys 4× speed and 4× smaller footprint. Go 8-bit only when the task is code or math.
> **Quantize the KV cache** (-ctk q8_0): long contexts stop eating your RAM and your bandwidth.
> **Speculative decoding** (draft model or n-gram) buys 1.5–2.5× on predictable text at zero quality cost; it is pure win on a bandwidth-bound box.
> **Pool machines** (exo, llama.cpp RPC) to serve one model across two laptops — slow interconnect, but it turns spare hardware into a bigger one.
> **Prefill is compute, decode is bandwidth.** Batch your long prompts; the expensive part is the part you can amortise.

Routstr: a node market, not an API company

Anyone runs a node. Users pay per request in sats. Nobody onboards anybody.

COMPONENT	ROLE	WHAT IT DOES
routstr-core	node (Python)	Holds the upstream account, reserves and settles payments, signs offers and receipts, serves the OpenAI-compatible API
routstrd	client daemon (Bun)	Runs on your machine, holds your keys, discovers nodes over Nostr, exposes a local OpenAI endpoint on :8008
routstr-sdk	transport (TS)	Offer verification, session driver, payment headers, confidential transport
routstr-chat / routstr-cli	clients	Chat UI and terminal client; any OpenAI-compatible app works against routstrd
routstr-confidential	protocol + provers	**cu-sidecar** (node relay), **cu-2pc** (two-party key schedule), **cu-prover** (ZK proofs)
confidential-web / PWA	browser	Same client compiled to WebAssembly — 2PC and proofs run in a static page, no extension
Nostr relays	discovery	Nodes announce URL and reviews; clients pin pubkeys instead of trusting a directory

LIVE TODAY

routstr.cypherpunk.today serves confidential mode (protocol v2) with a mainstream upstream provider.

STATUS

Confidential mode is **experimental**: on fork branches, not yet in releases. The 2PC has no external review.

WHY IT MATTERS

A market of interchangeable nodes is an exit. A single gateway is a landlord with an API key.

Pay with a token, not with an identity

No email. No password. No KYC. The only credential is a bearer string you can spend or burn.

```
# 1 — fund: mint ecash from sats
#    any Cashu wallet, any mint the node trusts

# 2 — onboard: creates keys locally,
#    asks for nothing else
routstrd onboard

# 3 — fund the daemon
routstrd receive cashuAeyJ0b2tlbiI6...
#    or straight over Lightning:
routstrd receive 21000

# 4 — pin a node (optional, stronger)
#    config.json → node_pubkeys: [npub...]

# 5 — just talk to it
curl 127.0.0.1:8008/v1/chat/completions \
  -H 'content-type: application/json' \
  -d '{"model":"...","messages":[...]}'
```

WHAT A CASHU TOKEN IS

Chaumian ecash: a mint signs blinded notes, so the mint cannot link issuance to redemption. Bearer value, not a balance. Lose the string, lose the money — that is the point.

WHAT THE NODE LEARNS

A token was spent, and how much. Not who you are. Refunds come back as fresh tokens, so even the refund is unlinkable.

WHAT NOSTR IS FOR

Discovery without a directory. Nodes publish signed announcements; clients verify signatures and require a review before trusting an unpinned URL. Pin the pubkey and you trust nothing else.

TWO BEARER FORMS

Prepaid **sk-** key or a raw **Cashu token** — both work as the Authorization header. Receipts are BIP-340 Schnorr signed, so your audit trail is portable.

Bitcoin settles the mint, ecash buys the tokens

Sats anchor the value. Ecash makes it spendable at line rate. Neither one requires your name.

LAYER	WHAT HAPPENS THERE	TRUST REQUIRED	VISIBILITY
Bitcoin L1	Final settlement, miner fees, cold storage	Consensus rules only	Public, permanent
Lightning	Instant funding of the mint and of routstrd	Channels and routing nodes for liveness	Peer-visible amounts, not identity
Cashu mint	Issues and redeems ecash notes	Mint solvency and honesty (the real trust edge)	Mint sees issue/redeem, cannot link them
Routstr node	Reserves sats, serves the model, refunds the remainder	Reputation only; crypto covers content and billing accuracy	Sees model, size, timing — not your prompt
Receipt	Signed usage report: model, tokens, cost, rates, session hash	Verify the signature; read every field from signed bytes	Yours to keep, prove, or publish

WHY ECASH, NOT CARDS

Card rails need identity and chargebacks. Ecash needs a signature and a mint. A protocol with a refund desk is a protocol with a file on you.

THE HONEST TRUST EDGE

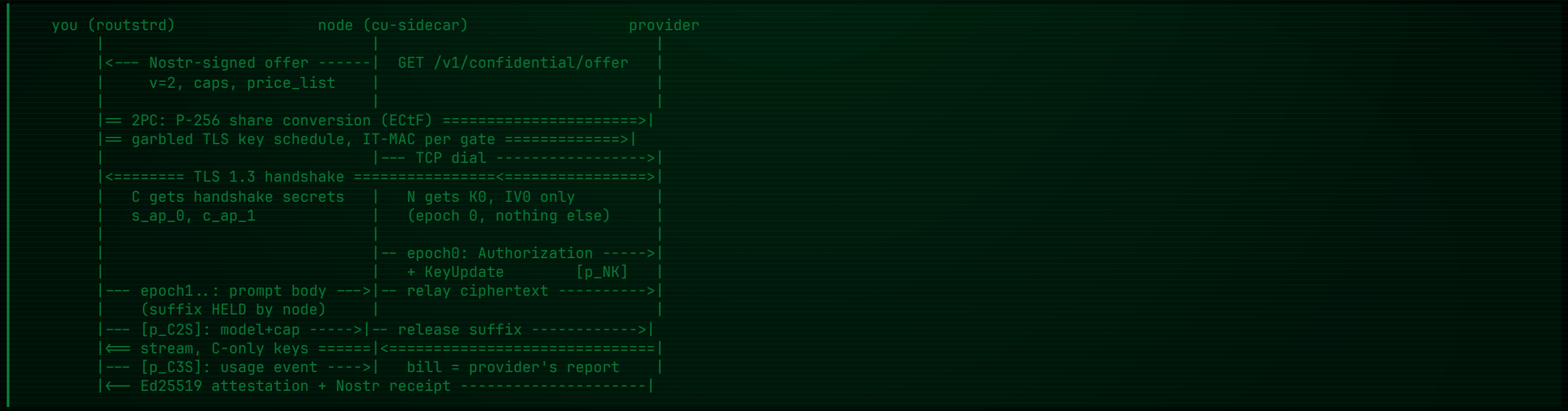
A mint can abscond. Keep balances small, use mints you can name, and treat ecash like a wallet, not like a bank.

BILLING YOU CAN AUDIT

The bill is the provider's own usage report, proven inside the session and charged at rates frozen in a signed offer — not the node's arithmetic.

The node relays ciphertext it cannot read

Not a TEE. Not an attestation badge. A two-party TLS key schedule plus three zero-knowledge proofs.



NOBODY HOLDS THE SECRET

The ECDHE and master secrets are never assembled on either side. The client cannot recover the node's API key even if it records all traffic.

NO PAYLOAD MPC

The only ZK work is tens of SHA-256 compressions and a few AES blocks per session. The prompt then streams at native TLS speed.

V1 IS RETIRED

The single-party variant needed <¾ the bandwidth but protected the key only while no observer of the node's egress colluded with the client.

Three proofs pin the only bytes anyone may author

Each proof is anchored on a hash the two-party computation itself outputs, so it cannot be pointed at other plaintext.

PROOF	PROVER	STATEMENT	ANCHORED ON	WHAT IT STOPS
π_NK	node	The credential record matches the published template, only its key bytes free	hk = SHA256(K0 ‖ IV0)	Node slipping another account or a retention header into epoch 0
π_C2S	client	The body ends in a public JSON suffix naming the agreed model and token cap	h1 = SHA256(c_ap_1)	Model downgrade, cap inflation, silent parameter swaps
π_C3S	client	The provider's own usage/cost event is authentic for this session	h2 = SHA256(s_ap_0)	Invented token counts, creative pricing

THE SUFFIX HOLD

The node keeps the final body record until π_C2S verifies. The provider holds an incomplete request and does nothing. You cannot be billed for a model you did not ask for.

ONE BIT, NOT YOUR BYTES

Proofs compare decrypted ranges against expected bytes *inside* the circuit. The verifier decodes equal / not-equal — never the plaintext, even on failure.

THE RECEIPT

Sidecar signs a session attestation with a per-process Ed25519 key; the node re-signs with its Nostr key. The receipt carries sid, model, usage, cost, reservation, rates and the attestation hash — BIP-340 over canonical JSON.

FAIL CLOSED

No offer, no v2, wrong host, missing signature: the request is refused, never sent in the clear.

Confidentiality costs bandwidth, not latency

Pre-warming moves the two-party computation off the request path. That is the whole trick.

BANDWIDTH

~22 MB node→client, **1.2 MB** back, per session. A whole chat request moves ~28 MB. An image request ~26 MB.

LATENCY

Cold first token **14–16 s** on a good home link, ~1 min on a slow one. Pre-warmed: **2–3 s**, mostly provider time.

MONEY

Reserved at full-context worst case, charged at the provider's reported usage and frozen rates. Unused session or a provider refusal costs nothing beyond advertised fees.

FAILURE	WHO PAYS	WHY
Provider refuses (4xx, 503)	Nothing, or the session fee	No tokens were produced; a 429/503 may still carry a session fee
Pre-warmed session never used	Abandon fee, if the offer has one	The node already paid the bandwidth
Other 5xx, or the stream is cut	The full reservation	The provider got your request but sent no usage report; nobody can prove who cut it

WHAT STILL LEAKS

Provider and model, timing, prompt length, chunk sizes — research has partly reconstructed answers from size patterns. Far less than cleartext, not zero.

WHAT IS STILL UNPROVEN

The 2PC has no published proof. A cheating node could learn at most **~2.6 bits per session** about single-use values, and those bits never accumulate across sessions.

Attestation is a promise with a build pipeline

Enclaves are genuinely useful for a model owner who must hide weights. They are a poor fit for hiding your prompt from a reseller.

PROPERTY	ENCLAVE / TEE	TWO-PARTY KEY SCHEDULE + ZK
What you trust	Chip vendor, microcode, firmware, measured stack, and the operator's build	TLS certificate, a handful of proof statements fixed by the protocol
After every update	Re-review the whole plaintext-touching stack and re-pin a measurement	Nothing. The statements do not change
Who can read your prompt	Nobody inside the boundary — if the boundary is what you think	Nobody at the node; the provider still does, as always
Failure mode	Attestation theatre: a badge nobody actually verifies	Bandwidth cost; a new 2PC with no external review yet
Side channels	Large literature, hard to patch	≤2.6 bits/session leakage, single-use values, no accumulation

WHERE ENCLAVES STILL WIN

Hiding model weights, protecting a key inside a host you do not control, and letting a provider prove *which* model ran. Different problem, different tool.

THE CYPHERPUNK ARGUMENT

A guarantee that requires you to review a build after each patch is a guarantee only auditors enjoy. A guarantee you can check in five lines of code is a guarantee users enjoy.

Five rungs, and what each one costs you

[5] **Air-gapped local** model on a box that never dials out — max secrecy, your own hardware ceiling

[4] **Local model, normal machine** — no network by design, 27B-class quality, laptop speed

[3] **Confidential mode** — open weights or a mainstream upstream, node sees ciphertext, ~22 MB/session

[2] **Plain routstr node** — fast, no account, node sees everything; fine for public questions

[1] **Logging-by-default cloud** — best quality, worst defaults, your prompt is the product

A ROUTER, NOT A RELIGION

Send secrets to rung 4 or 3, boilerplate to rung 2, and the genuinely hard reasoning to whichever rung your threat model allows. Sensitivity is a per-request property, not a lifestyle.

THE PRACTICAL DEFAULT

Local 27B for anything containing code, contracts, medical text, or names. Confidential mode when you need a frontier model. Cleartext only for questions you would ask in public.

Six ways the good version goes bad

FAILURE	HOW IT HAPPENS	COUNTERMEASURE
Metadata leakage	Chunk sizes and timing reconstruct answers; who-talks-to-whom graphs the market	Pad or batch; use onion endpoints; accept that volume is visible
Mint or node absconding	Ecash issuer or node keeps the sats after a failed session	Small balances, mints you can name, signed receipts you keep
Attestation theatre	A TEE badge nobody re-verifies after each firmware update	Prefer proofs of protocol statements over proofs of software identity
Convenience capture	The hosted client is so good nobody runs the local one; the "optional" telemetry becomes mandatory	Ship the CLI and the PWA first; make offline a supported mode, not a demo
Weights pullback	Next release is API-only, or the licence adds acceptable-use vetoes and MAU caps	Keep a working checkpoint; support permissive licences with money and patches
Review debt	The 2PC and its proofs stay unreviewed; everyone treats "experimental" as permanent	Fund external review; keep v1 retired and the threat model published

One question for any provider: can I verify that claim myself, offline, without asking you to be honest?

A minimum viable sovereign stack

LAYER	PICK	WHY
Local runtime	Ollama for convenience, llama.cpp or MLX for speed	OpenAI-compatible on localhost; swaps without touching your apps
Local model	Qwen3.8-27B at 4-bit; Flash-Next on 128 GB	Permissive licence, multimodal, fits real hardware
Frontier access	routstrd + confidential mode, paid in ecash	No account, node sees ciphertext, bill is provable
Wallet	Cashu mint you can name + Lightning for funding	Bearer value, unlinkable redemption, instant top-up
Discovery	Nostr announcements with pinned node pubkeys	No directory to capture, no account to suspend
Audit	BIP-340 receipts, kept locally	Portable proof of what you paid and what ran

```
# the whole onboarding, no email involved
$ ollama run qwen3.8:27b                    # local floor
$ routstrd onboard && routstrd receive 21000    # identity-free funding
$ routstrd start --provider https://routstr.cypherpunk.today
$ curl 127.0.0.1:8008/v1/chat/completions -H 'x-routstr-verify: confidential' ...
```

Three currents, one of them wins

CURRENT — OPEN

Open weights keep landing months behind the frontier. Consumer hardware crosses the "good enough for my real work" line at 128 GB. Local becomes the default for sensitive text; routers become commodity.

COUNTER — ENCLOSURE

Compute concentration, agentic products that need the biggest model, and regulation written by incumbents push everything back behind an account. "Safety" becomes the word for a moat.

DECIDER — CONVENIENCE

Cryptography never lost on math; it lost on user experience. Whichever side makes private inference as easy as a chat box wins the decade.

> **Expect** confidential routing to absorb enclave designs for reseller trust, and enclaves to stay where weight secrecy matters.
> **Expect** proofs to get cheaper and pre-warming to become standard, collapsing the setup cost to background noise.
> **Expect** the hardest problem to stay payment: ecash mint trust is the honest bottleneck, not the crypto.
> **Watch** whether permissive licences survive success, and whether checkpoints stay downloadable after the API launches.
> **Watch** whether the client stays a CLI and a WASM page, or quietly becomes a hosted app with telemetry.

> Privacy is not a feature you enable. It is a stack you assemble, verify, and refuse to give up.