POST-QUANTUM BITCOIN

Blockstream

OP_CHECKSHRINCS

A hash-based signature opcode for **post-quantum Bitcoin**.

SPEAKER

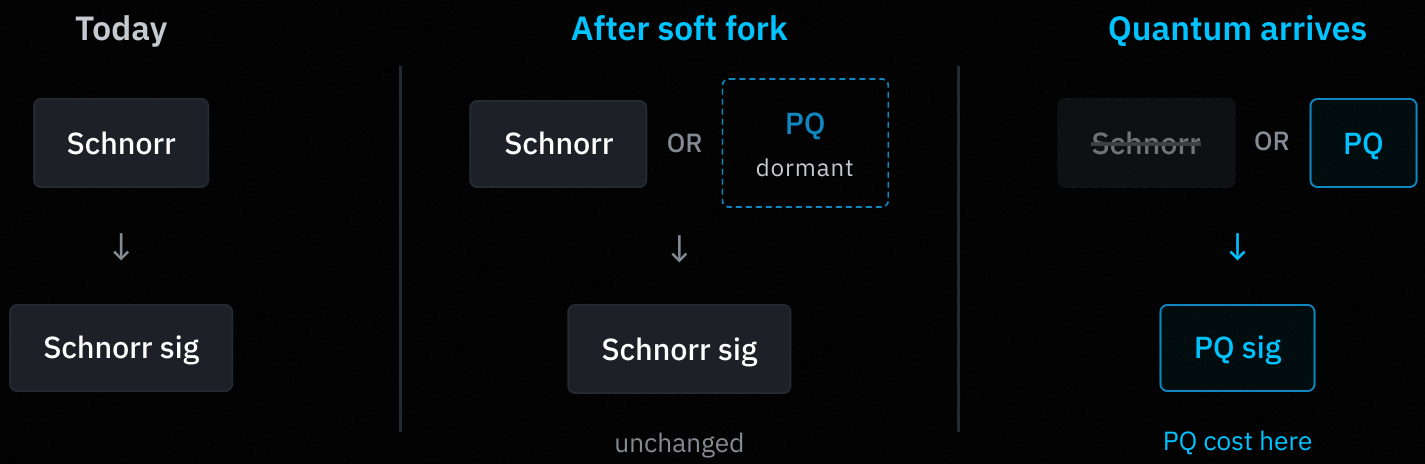
Jonas Nick

DATE

April 16, 2026

Blockstream RESEARCH

Quantum-ready without paying for it



Failure modes

01

Throughput

Signatures so large some users can't transact at all.

02

Verification

Cost exceeds what full nodes can handle.

03

Signing

Too expensive for constrained devices.

04

Assumptions

The cryptography breaks.

05

Adoption

Too complex, doesn't integrate with existing infrastructure.

06

Implementation

Broken — or becomes unmaintainable.

A proposal that avoids these pitfalls has the highest chance of rough consensus.

Standard candidates

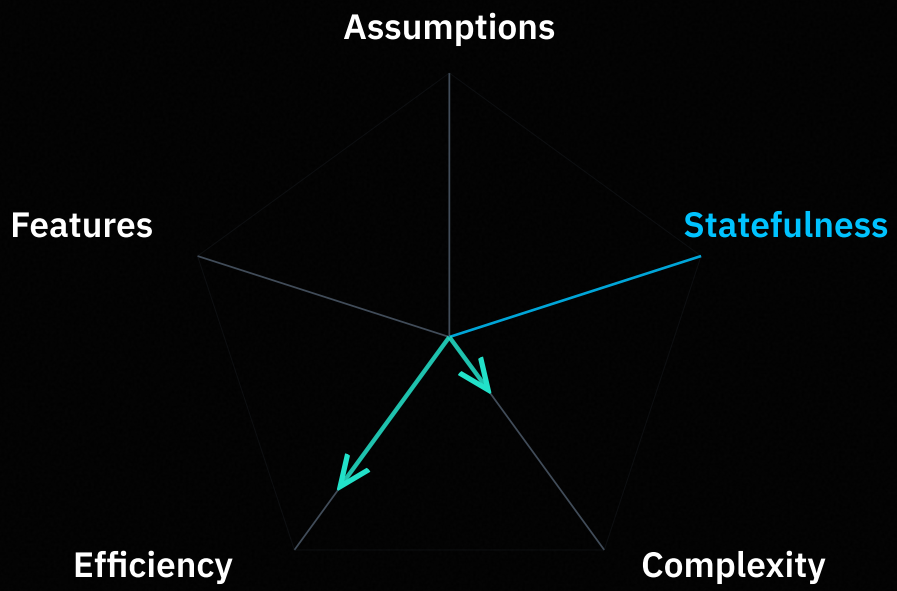
Scheme	Assumption	Throughput (tx/s)	Features BIP 32, threshold sigs, silent payments
Schnorr	DLP	6.55	yes
ML-DSA	LATTICE	0.50	no
SLH-DSA	HASH	0.36	no

tx/s assumes avg tx shape: 2.27 inputs, 2.64 outputs — transactionfee.info

Candidates needing more research

Scheme	Assumption	Throughput (tx/s)	Features
Falcon-WS	LATTICE	~1.05	no
Lattice w/ features	LATTICE	<1.05	depends
SQIsign	ISOGENY	~3.60	maybe
Block-wide SNARK	VARIOUS	~6 ?	depends

The tradeoff space



Signature budget

How many times you can securely sign with one key.

SLH-DSA: 2^{64} — essentially unlimited.

Smaller budget

Smaller signature.

Exceed the budget

Security breaks down.

Choosing a smaller budget

Budget: 2^{64}

SLH-DSA

Size	~7.9 KB
tx/s	0.36

→

+33% tx/s

Budget: 2^{40}

~1 trillion signatures

Size	~5.7 KB
tx/s	0.48

What if the budget is tiny?

One public key

Compact path

Small sig, tiny budget

`Stateful`

Fallback path

Larger sig, high budget

`Stateless`

Signer must track how often they signed

The state problem

Sign → `state: 1/3`

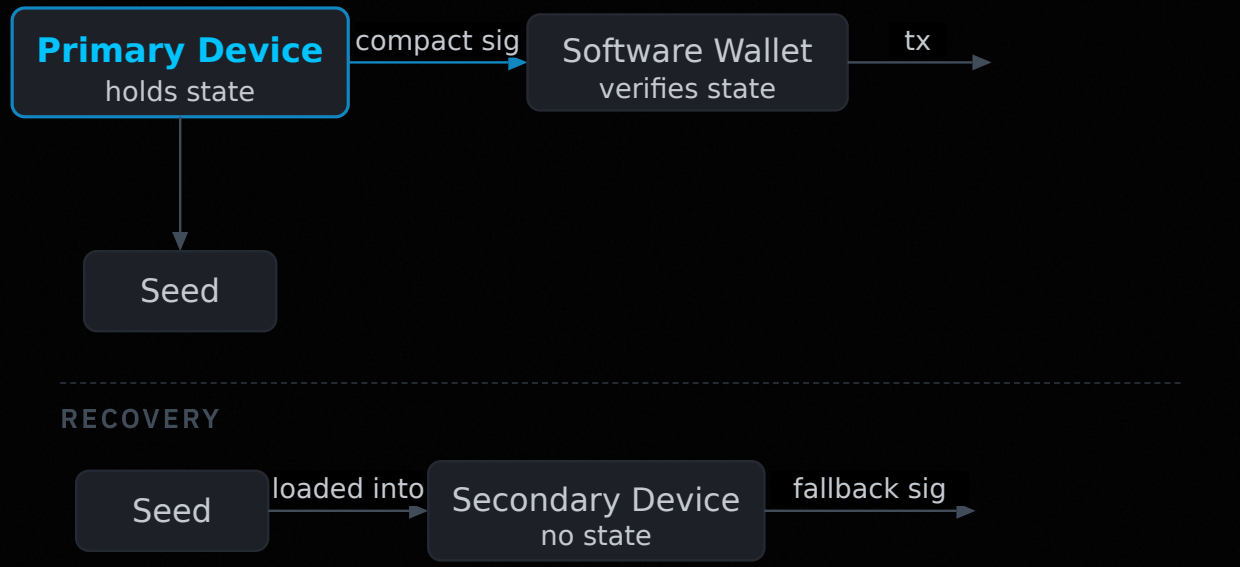
Backup disk

Sign → `state: 2/3`

Restore backup → `state: 1/3`

Sign → state 1/3 reused → **security compromised**

Managing statefulness



State is managed entirely by the device. The user has no opportunity to corrupt it.

SHRINCS 🛋️

A post-quantum signature scheme with

(1) two signing paths under a single public key: **compact** (stateful) + **fallback** (stateless),

(2) an extremely efficient compact path.

Scheme	Throughput (tx/s)
SLH-DSA NIST standard	0.36
SLH-DSA, reduced budget 2^{40}	0.48
SHRINCS compact path, 580 B	3.04

SHRINCS trades systemic risk (low throughput, assumptions, complexity) for localized risk (single device failure).

SHRIMPS

One public key

Compact path
580 B, tiny budget
SHRINCS

Second compact path
~3 KB, budget ≥ 1000
SHRIMPS

Fallback path
Larger sig, high budget
Stateless

Number of initialized devices ≤ 1000

Restored-device signature	Throughput (tx/s)
SLH-DSA NIST, budget 2^{64}	0.36
SLH-DSA, reduced budget 2^{40}	0.48
SHRIMPS second compact path, ~3000 B	0.87

Further SPHINCS+ optimizations

Optimization	Size	tx/s
SLH-DSA NIST, budget 2^{64}	7872 B	0.36
SLH-DSA, reduced budget 2^{40}	5792 B	0.48
+ WOTS+C, PORS+FP	5060 B	0.54
+ 5× sign / keygen cost	4496 B	0.60
+ verify ≤ 1.5× Schnorr / byte	3896 B	0.69

A toy proposal

Two design principles: keep verification as fast as SLH-DSA, and use one scheme per use case rather than one-size-fits-all.

Use case	Stateful?	Sign time	Signing budget	Size
Desktop / mobile L1	no	slow size-optimized	2^{40}	4496 B
Signing device	yes	fast	2^{32} + SHRINCS / SHRIMPS	580 B – 4336 B
Lightning node	yes	fast	2^{32} + SHRINCS	580 B – 4336 B

4 signature variants **0.60 to 3.04 tx/s** vs 0.36 baseline SLH-DSA

Downsides and remaining questions

01

Not size-optimized

Prioritizes verification time: 6× lower cost per byte than Schnorr — room for a 6× block size increase. Need better benchmarks.

02

Fungibility & privacy

Combining multiple signature schemes reduces fungibility and privacy. Is that acceptable?

03

L2 coverage

Which use cases aren't covered? L2 devs — how does this interact with your protocols?

04

SNARK aggregation

How to design schemes for future SNARK aggregation? Should that be a consideration now?

05

Signing time

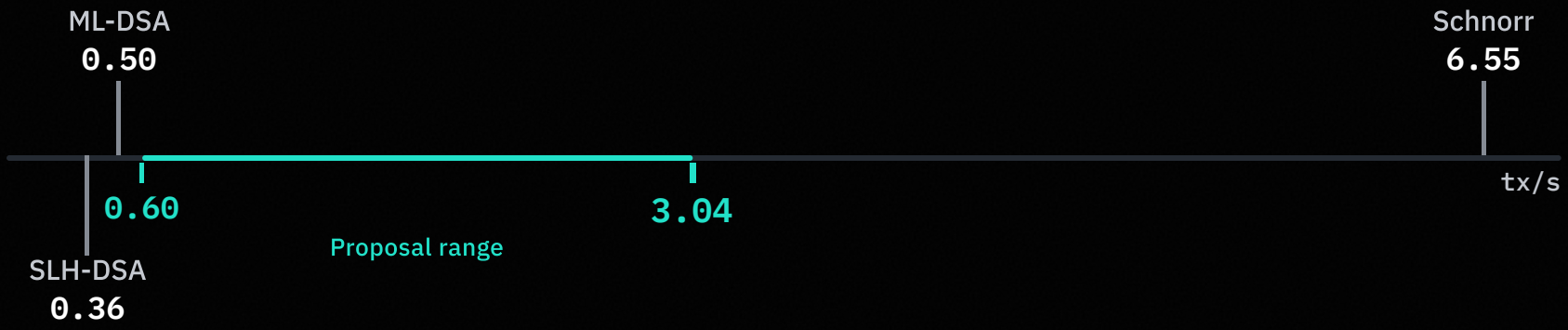
Tolerable on which platforms? Longer signing would shrink signatures further — please benchmark.

06

Implementation

C++ or a formal spec? LLMs make formal verification of consensus code more feasible than ever.

Conclusion & next steps



OPEN QUESTIONS

- Other stateful optimizations beyond SHRINCS / SHRIMPS?
- How can careful engineering guarantee the safety of stateful setups?
- Can L2 protocols leverage stateful signatures?

LINKS

- **Paper** — Hash-based Signatures for Bitcoin
  `eprint.iacr.org/2025/2203`
- **Code** — `github.com/BlockstreamResearch`
  `SPHINCS-Parameters`
  `shrincs-cpp`
  `shrincs-simplicity-verifier`