

Lecture 6: Bitcoin's Quantum Migration

CPSC 3640/5440: Decentralized Payments, Contracts, and Finance for Humans and AI

Fan Zhang

Yale University

Fall 2026

Where we are going

1. Quantum computers: what they break in Bitcoin, how close they are, and why Bitcoin prepares now
2. Three challenges of migrating
 - exposed public keys
 - bigger post-quantum signatures
 - dormant coins that will never move
3. PACTs: a proposed rescue path for dormant coins

One thing to keep in mind

Much of what you read online blurs what exists with what is proposed. Spoiler: nothing post-quantum is live on Bitcoin as of September 2026.

What is a quantum computer? (ELI5)

Disclaimer: I don't know how quantum computers work. It's apparently also very hard to “explain like I'm 5”. But for today, we just need to know what they can break.

- A normal bit is a coin lying flat: heads or tails.

Take-away for us: Quantum computers breaks Bitcoin's (current) signatures; it does not break hash functions.

Aaronson, Quanta (2021)

What is a quantum computer? (ELI5)

Disclaimer: I don't know how quantum computers work. It's apparently also very hard to “explain like I'm 5”. But for today, we just need to know what they can break.

- A normal bit is a coin lying flat: heads or tails.
- A *qubit* is a coin still spinning. It is not heads or tails yet. Look at it, and it lands on one side at random.

Take-away for us: Quantum computers breaks Bitcoin's (current) signatures; it does not break hash functions.

Aaronson, Quanta (2021)

What is a quantum computer? (ELI5)

Disclaimer: I don't know how quantum computers work. It's apparently also very hard to "explain like I'm 5". But for today, we just need to know what they can break.

- A normal bit is a coin lying flat: heads or tails.
- A *qubit* is a coin still spinning. It is not heads or tails yet. Look at it, and it lands on one side at random.
- Ten spinning coins "contain" all 1,024 outcomes at once.

Take-away for us: Quantum computers breaks Bitcoin's (current) signatures; it does not break hash functions.

Aaronson, Quanta (2021)

What is a quantum computer? (ELI5)

Disclaimer: I don't know how quantum computers work. It's apparently also very hard to "explain like I'm 5". But for today, we just need to know what they can break.

- A normal bit is a coin lying flat: heads or tails.
- A *qubit* is a coin still spinning. It is not heads or tails yet. Look at it, and it lands on one side at random.
- Ten spinning coins "contain" all 1,024 outcomes at once.
- **Inaccurate but perhaps helpful mental model:** the computer tries every answer at the same time for certain problems.

Take-away for us: Quantum computers breaks Bitcoin's (current) signatures; it does not break hash functions.

Aaronson, Quanta (2021)

Shor's algorithm

- For signatures, it's crucial that one cannot recover sk from $pk = sk \cdot G$ where G is a proper group (e.g. points on an elliptic curve).

Shor's algorithm

- For signatures, it's crucial that one cannot recover sk from $pk = sk \cdot G$ where G is a proper group (e.g. points on an elliptic curve).
- We believe it is difficult for **classical computers** to do so (Discrete Log assumption)

Shor's algorithm

- For signatures, it's crucial that one cannot recover sk from $pk = sk \cdot G$ where G is a proper group (e.g. points on an elliptic curve).
- We believe it is difficult for **classical computers** to do so (Discrete Log assumption)
- Foundational for digital signatures, key exchanges, public key encryption, etc.

Shor's algorithm

- For signatures, it's crucial that one cannot recover sk from $pk = sk \cdot G$ where G is a proper group (e.g. points on an elliptic curve).
- We believe it is difficult for **classical computers** to do so (Discrete Log assumption)
- Foundational for digital signatures, key exchanges, public key encryption, etc.
- Shor's algorithm (1994) breaks this assumption: it can solve Discrete Logs in polynomial time using a quantum computer.

Shor's algorithm

- For signatures, it's crucial that one cannot recover sk from $pk = sk \cdot G$ where G is a proper group (e.g. points on an elliptic curve).
- We believe it is difficult for **classical computers** to do so (Discrete Log assumption)
- Foundational for digital signatures, key exchanges, public key encryption, etc.
- Shor's algorithm (1994) breaks this assumption: it can solve Discrete Logs in polynomial time using a quantum computer.
- **A powerful enough quantum computer can break:** signatures based on factoring or discrete log (RSA, ECDSA, Schnorr) and key exchange (Diffie–Hellman). Pretty bad.

Post-quantum cryptography

Difficult problems even for quantum computers:

- Reverting hash functions
- Lattice related problems
- more

Post-quantum cryptography

Difficult problems even for quantum computers:

- Reverting hash functions
- Lattice related problems
- more

Post-quantum cryptography against quantum computers:

- Hash-based signatures: Lamport and Merkle, 1979.
- Lattice-based cryptography: Ajtai and NTRU, 1996.
- 2024: NIST published standards for **post-quantum signatures**: ML-DSA (lattice), SLH-DSA (hash). FN-DSA (lattice) is still a draft.

Post-quantum cryptography

Difficult problems even for quantum computers:

- Reverting hash functions
- Lattice related problems
- more

Post-quantum cryptography against quantum computers:

- Hash-based signatures: Lamport and Merkle, 1979.
- Lattice-based cryptography: Ajtai and NTRU, 1996.
- 2024: NIST published standards for **post-quantum signatures**: ML-DSA (lattice), SLH-DSA (hash). FN-DSA (lattice) is still a draft.

Bitcoin's question is not *whether* a replacement exists, but which one fits a blockchain. More on that shortly.

NIST FIPS 204, 205, draft 206 (2024–2025)

Why migrating now

It's controversial whether Bitcoin should migrate now.

Why migrating now

It's controversial whether Bitcoin should migrate now.

Cryptographically relevant quantum computers (CRQC) don't exist yet

- Largest number factored with Shor on real hardware: **21** (2012). Largest elliptic-curve key broken: **15 bits** (2026). Bitcoin's are 256.
- Ask ten people how close we are and you get ten answers.

Why migrating now

It's controversial whether Bitcoin should migrate now.

Cryptographically relevant quantum computers (CRQC) don't exist yet

- Largest number factored with Shor on real hardware: **21** (2012). Largest elliptic-curve key broken: **15 bits** (2026). Bitcoin's are 256.
- Ask ten people how close we are and you get ten answers.

New signature schemes are not battle tested and they have bugs.

Why migrating now

It's controversial whether Bitcoin should migrate now.

Cryptographically relevant quantum computers (CRQC) don't exist yet

- Largest number factored with Shor on real hardware: **21** (2012). Largest elliptic-curve key broken: **15 bits** (2026). Bitcoin's are 256.
- Ask ten people how close we are and you get ten answers.

New signature schemes are not battle tested and they have bugs.

Why migrating now

- Algorithm improvement (partly by AI): qubits needed to break Bitcoin's elliptic curve went 9 M (2023) $\rightarrow$ 0.5 M $\rightarrow$ 20 k (2026).
- NIST retires elliptic curves for U.S. government systems by 2035.
- Bitcoin upgrades take years to design, activate, and adopt.

Gutmann and Neuhaus (2025); Project Eleven (2026); Babbush et al. (2026); Häner et al. (2026); Chaincode Labs (2025); NIST IR 8547

Challenge 1: the keys are already out

Output type	Public key visible	Exposure window
P2PK (2009)	when UTXO is created	long
P2PKH, P2WPKH	only when spent	short
P2SH, P2WSH	only when spent	short
P2TR (Taproot, 2021)	when UTXO is created	long

Challenge 1: the keys are already out

Output type	Public key visible	Exposure window
P2PK (2009)	when UTXO is created	long
P2PKH, P2WPKH	only when spent	short
P2SH, P2WSH	only when spent	short
P2TR (Taproot, 2021)	when UTXO is created	long

- **Long exposure:** exposed for years; even a slow quantum computer can steal those coins.

Chaincode Labs (2025), Sec. 4; Babbush et al. (2026)

Challenge 1: the keys are already out

Output type	Public key visible	Exposure window
P2PK (2009)	when UTXO is created	long
P2PKH, P2WPKH	only when spent	short
P2SH, P2WSH	only when spent	short
P2TR (Taproot, 2021)	when UTXO is created	long

- **Long exposure:** exposed for years; even a slow quantum computer can steal those coins.
- **Short exposure:** the key shows only while the transaction waits in the mempool (minutes to hours); the attacker must beat the next block.

Chaincode Labs (2025), Sec. 4; Babbush et al. (2026)

Challenge 1: the keys are already out

Output type	Public key visible	Exposure window
P2PK (2009)	when UTXO is created	long
P2PKH, P2WPKH	only when spent	short
P2SH, P2WSH	only when spent	short
P2TR (Taproot, 2021)	when UTXO is created	long

- **Long exposure:** exposed for years; even a slow quantum computer can steal those coins.
- **Short exposure:** the key shows only while the transaction waits in the mempool (minutes to hours); the attacker must beat the next block.
- What is the risk of reusing P2PKH/P2SH addresses?

Chaincode Labs (2025), Sec. 4; Babbush et al. (2026)

How much is exposed

	BTC
Outputs with an exposed key (Chaincode, May 2025)	≈ 6.3 M
P2PK, mostly 2009–2010	≈ 1.7 M
P2TR	≈ 0.15 M
reused addresses	≈ 4.4 M

Who reuses addresses? Mostly exchanges and custodians.

- An exchange pools millions of customers' coins in a handful of wallet addresses, spends from them every day for withdrawals, and sends the change back to the same addresses. The keys have been public for years.
- Custodians holding ETF and corporate treasuries do the same at a smaller number of much larger addresses.

Chaincode Labs (2025), Secs. 4, 6

Challenge 2: new signatures are big

Post-quantum signature schemes cost 10–100× more bytes than what Bitcoin uses today.

Scheme	Assumption	Public key	Signature
Schnorr (today)	elliptic curve	32 B	64 B
ML-DSA-44	lattice	1,312 B	2,420 B
FN-DSA-512 (Falcon)	lattice	897 B	666 B
SLH-DSA-128s (SPHINCS+)	hash only	32 B	7,856 B
SHRINCS (draft BIP)	hash only	48 B	548 B and up

- Block size is limited: bigger signatures mean fewer transactions per block.

Challenge 2: new signatures are big

Post-quantum signature schemes cost 10–100× more bytes than what Bitcoin uses today.

Scheme	Assumption	Public key	Signature
Schnorr (today)	elliptic curve	32 B	64 B
ML-DSA-44	lattice	1,312 B	2,420 B
FN-DSA-512 (Falcon)	lattice	897 B	666 B
SLH-DSA-128s (SPHINCS+)	hash only	32 B	7,856 B
SHRINCS (draft BIP)	hash only	48 B	548 B and up

- Block size is limited: bigger signatures mean fewer transactions per block.
- Beyond size, much software has to be redesigned and rewritten to use the new signatures.

Challenge 2: new signatures are big

Post-quantum signature schemes cost 10–100× more bytes than what Bitcoin uses today.

Scheme	Assumption	Public key	Signature
Schnorr (today)	elliptic curve	32 B	64 B
ML-DSA-44	lattice	1,312 B	2,420 B
FN-DSA-512 (Falcon)	lattice	897 B	666 B
SLH-DSA-128s (SPHINCS+)	hash only	32 B	7,856 B
SHRINCS (draft BIP)	hash only	48 B	548 B and up

- Block size is limited: bigger signatures mean fewer transactions per block.
- Beyond size, much software has to be redesigned and rewritten to use the new signatures.
- Bitcoin developers lean hash-based: no new assumption beyond SHA-256, which Bitcoin already trusts.

NIST FIPS 204/205/206; Kudinov and Nick (2025); SHRINCS BIP draft

Challenge 3: how to move dormant coins

Even with a safe output type available, some coins will never move (keys lost, owners dead, and so on). About 1.7 M BTC sits in P2PK outputs, mostly untouched since 2010.

Some mitigation ideas:

Chaincode Labs (2025), Secs. 4, 6; Hourglass v2 (draft BIP); BIP 361 (draft)

Challenge 3: how to move dormant coins

Even with a safe output type available, some coins will never move (keys lost, owners dead, and so on). About 1.7 M BTC sits in P2PK outputs, mostly untouched since 2010.

Some mitigation ideas:

1. **Throttle.** Allow one P2PK spend per block, so draining takes decades.

Chaincode Labs (2025), Secs. 4, 6; Hourglass v2 (draft BIP); BIP 361 (draft)

Challenge 3: how to move dormant coins

Even with a safe output type available, some coins will never move (keys lost, owners dead, and so on). About 1.7 M BTC sits in P2PK outputs, mostly untouched since 2010.

Some mitigation ideas:

1. **Throttle.** Allow one P2PK spend per block, so draining takes decades.
2. **Sunset.** A soft fork stops accepting old signatures after a chosen block height. Coins that did not move are frozen (proposed as BIP 361, 2026).

Chaincode Labs (2025), Secs. 4, 6; Hourglass v2 (draft BIP); BIP 361 (draft)

Challenge 3: how to move dormant coins

Even with a safe output type available, some coins will never move (keys lost, owners dead, and so on). About 1.7 M BTC sits in P2PK outputs, mostly untouched since 2010.

Some mitigation ideas:

1. **Throttle.** Allow one P2PK spend per block, so draining takes decades.
2. **Sunset.** A soft fork stops accepting old signatures after a chosen block height. Coins that did not move are frozen (proposed as BIP 361, 2026).
 - Problem: moving the coins before the deadline announces that a dormant address has a living owner (e.g., the world will know if Satoshi is still alive or not!)

Chaincode Labs (2025), Secs. 4, 6; Hourglass v2 (draft BIP); BIP 361 (draft)

Challenge 3: how to move dormant coins

Even with a safe output type available, some coins will never move (keys lost, owners dead, and so on). About 1.7 M BTC sits in P2PK outputs, mostly untouched since 2010.

Some mitigation ideas:

1. **Throttle.** Allow one P2PK spend per block, so draining takes decades.
2. **Sunset.** A soft fork stops accepting old signatures after a chosen block height. Coins that did not move are frozen (proposed as BIP 361, 2026).
 - Problem: moving the coins before the deadline announces that a dormant address has a living owner (e.g., the world will know if Satoshi is still alive or not!)
3. **Sunset with a rescue path.** Stop accepting old signatures, but let an owner spend by proving ownership some other way. This is where the interesting cryptography is.

Chaincode Labs (2025), Secs. 4, 6; Hourglass v2 (draft BIP); BIP 361 (draft)

Sunset with a rescue path

The problem

- With a quantum computer, the attacker recovers the secret key from the public key.
- How to tell a signature by the owner from a signature by the attacker?

Sunset with a rescue path

The problem

- With a quantum computer, the attacker recovers the secret key from the public key.
- How to tell a signature by the owner from a signature by the attacker?

The idea behind a recent proposal (PACTs, 2026)

- Suppose we agree that no key-breaking quantum computer exists before some cutoff time T_c , say 2030.

Sunset with a rescue path

The problem

- With a quantum computer, the attacker recovers the secret key from the public key.
- How to tell a signature by the owner from a signature by the attacker?

The idea behind a recent proposal (PACTs, 2026)

- Suppose we agree that no key-breaking quantum computer exists before some cutoff time T_c , say 2030.
- Then a signature produced *before* T_c can only come from the legitimate owner.

Sunset with a rescue path

The problem

- With a quantum computer, the attacker recovers the secret key from the public key.
- How to tell a signature by the owner from a signature by the attacker?

The idea behind a recent proposal (PACTs, 2026)

- Suppose we agree that no key-breaking quantum computer exists before some cutoff time T_c , say 2030.
- Then a signature produced *before* T_c can only come from the legitimate owner.
- So: have legitimate owners get their signatures **timestamped** before T_c . After T_c , only timestamped signatures are accepted. (This requires changing Bitcoin.)

Sunset with a rescue path

The problem

- With a quantum computer, the attacker recovers the secret key from the public key.
- How to tell a signature by the owner from a signature by the attacker?

The idea behind a recent proposal (PACTs, 2026)

- Suppose we agree that no key-breaking quantum computer exists before some cutoff time T_c , say 2030.
- Then a signature produced *before* T_c can only come from the legitimate owner.
- So: have legitimate owners get their signatures **timestamped** before T_c . After T_c , only timestamped signatures are accepted. (This requires changing Bitcoin.)

Sunset with a rescue path

The problem

- With a quantum computer, the attacker recovers the secret key from the public key.
- How to tell a signature by the owner from a signature by the attacker?

The idea behind a recent proposal (PACTs, 2026)

- Suppose we agree that no key-breaking quantum computer exists before some cutoff time T_c , say 2030.
- Then a signature produced *before* T_c can only come from the legitimate owner.
- So: have legitimate owners get their signatures **timestamped** before T_c . After T_c , only timestamped signatures are accepted. (This requires changing Bitcoin.)

Two ingredients needed: a timestamping service, and a way to prove things about a signature without revealing it.

Ingredient 1: a timestamping service

A service TS with two operations:

- $\text{TS.Stamp}(v) \rightarrow t$: record that value v exists now; return the time t .
- $\text{TS.Verify}(v, t) \rightarrow \text{accept or reject}$: was v stamped at time t ?

Ingredient 1: a timestamping service

A service TS with two operations:

- $\text{TS.Stamp}(v) \rightarrow t$: record that value v exists now; return the time t .
- $\text{TS.Verify}(v, t) \rightarrow$ accept or reject: was v stamped at time t ?

What we need from it

- No backdating: a stamp with time t can only be made at time t .

Ingredient 1: a timestamping service

A service TS with two operations:

- $\text{TS.Stamp}(v) \rightarrow t$: record that value v exists now; return the time t .
- $\text{TS.Verify}(v, t) \rightarrow \text{accept or reject}$: was v stamped at time t ?

What we need from it

- No backdating: a stamp with time t can only be made at time t .
- Public verification: anyone can run TS.Verify , years later.

Ingredient 1: a timestamping service

A service TS with two operations:

- $\text{TS.Stamp}(v) \rightarrow t$: record that value v exists now; return the time t .
- $\text{TS.Verify}(v, t) \rightarrow \text{accept or reject}$: was v stamped at time t ?

What we need from it

- No backdating: a stamp with time t can only be made at time t .
- Public verification: anyone can run TS.Verify , years later.

Ingredient 1: a timestamping service

A service TS with two operations:

- $\text{TS.Stamp}(v) \rightarrow t$: record that value v exists now; return the time t .
- $\text{TS.Verify}(v, t) \rightarrow \text{accept or reject}$: was v stamped at time t ?

What we need from it

- No backdating: a stamp with time t can only be made at time t .
- Public verification: anyone can run TS.Verify , years later.

Bitcoin is a timestamping service! Put v (or its hash) in a transaction; the block height is t ; anyone can check the block. Rewriting it would mean rewriting the chain.

In practice a free service, OpenTimestamps, batches thousands of values into one hash per block.

A first attempt: sign a new key with the old one before T_c

Suppose Satoshi holds a vulnerable P2PK coin with key pair (sk, pk) .

A first attempt: sign a new key with the old one before T_c

Suppose Satoshi holds a vulnerable P2PK coin with key pair (sk, pk) .

Before T_c . Generate a fresh key pair (sk', pk') . Sign the new public key with the old secret key, and timestamp that signature:

$$\sigma = \text{Sign}_{sk}(pk'), \quad t = \text{TS.Stamp}(\sigma).$$

A first attempt: sign a new key with the old one before T_c

Suppose Satoshi holds a vulnerable P2PK coin with key pair (sk, pk) .

Before T_c . Generate a fresh key pair (sk', pk') . Sign the new public key with the old secret key, and timestamp that signature:

$$\sigma = \text{Sign}_{sk}(pk'), \quad t = \text{TS.Stamp}(\sigma).$$

At T_c . Bitcoin freezes P2PK coins and adds a rescue path. To spend the coin, show

$$pk', \quad \sigma, \quad t, \quad \text{and a signature on the transaction under } pk'.$$

Nodes check $\text{Verify}_{pk}(\sigma, pk') = 1$, $\text{TS.Verify}(\sigma, t)$ accepts, and $t < T_c$.

A first attempt: sign a new key with the old one before T_c

Suppose Satoshi holds a vulnerable P2PK coin with key pair (sk, pk) .

Before T_c . Generate a fresh key pair (sk', pk') . Sign the new public key with the old secret key, and timestamp that signature:

$$\sigma = \text{Sign}_{sk}(pk'), \quad t = \text{TS.Stamp}(\sigma).$$

At T_c . Bitcoin freezes P2PK coins and adds a rescue path. To spend the coin, show

pk' , σ , t , and a signature on the transaction under pk' .

Nodes check $\text{Verify}_{pk}(\sigma, pk') = 1$, $\text{TS.Verify}(\sigma, t)$ accepts, and $t < T_c$.

Problem 1: privacy. TS sees σ and will learn that Satoshi is alive.

Fix: timestamp $H(\sigma)$; reveal σ only when spending.

A first attempt: sign a new key with the old one before T_c

Suppose Satoshi holds a vulnerable P2PK coin with key pair (sk, pk) .

Before T_c . Generate a fresh key pair (sk', pk') . Sign the new public key with the old secret key, and timestamp that signature:

$$\sigma = \text{Sign}_{sk}(pk'), \quad t = \text{TS.Stamp}(\sigma).$$

At T_c . Bitcoin freezes P2PK coins and adds a rescue path. To spend the coin, show

pk' , σ , t , and a signature on the transaction under pk' .

Nodes check $\text{Verify}_{pk}(\sigma, pk') = 1$, $\text{TS.Verify}(\sigma, t)$ accepts, and $t < T_c$.

Problem 1: privacy. TS sees σ and will learn that Satoshi is alive.

Fix: timestamp $H(\sigma)$; reveal σ only when spending.

Problem 2. This requires Bitcoin to specify a new signature scheme now (pk' must be for this new signature scheme). This choice is the hardest, slowest part of the whole migration.

Ingredient 2: zero-knowledge proofs

Fix a public function f . Two algorithms, for any public value y :

- $\text{Prove}_f(x, y) \rightarrow \pi$: given a secret x with $f(x) = y$, produce a proof π .
- $\text{Verify}_f(y, \pi) \rightarrow \text{accept or reject}$.

Ingredient 2: zero-knowledge proofs

Fix a public function f . Two algorithms, for any public value y :

- $\text{Prove}_f(x, y) \rightarrow \pi$: given a secret x with $f(x) = y$, produce a proof π .
- $\text{Verify}_f(y, \pi) \rightarrow \text{accept or reject}$.

Properties

- If you know such an x , the verifier accepts.

Ingredient 2: zero-knowledge proofs

Fix a public function f . Two algorithms, for any public value y :

- $\text{Prove}_f(x, y) \rightarrow \pi$: given a secret x with $f(x) = y$, produce a proof π .
- $\text{Verify}_f(y, \pi) \rightarrow \text{accept or reject}$.

Properties

- If you know such an x , the verifier accepts.
- If no such x exists, or you do not know one, the verifier rejects.

Ingredient 2: zero-knowledge proofs

Fix a public function f . Two algorithms, for any public value y :

- $\text{Prove}_f(x, y) \rightarrow \pi$: given a secret x with $f(x) = y$, produce a proof π .
- $\text{Verify}_f(y, \pi) \rightarrow \text{accept or reject}$.

Properties

- If you know such an x , the verifier accepts.
- If no such x exists, or you do not know one, the verifier rejects.
- The verifier learns nothing about x beyond the fact that $f(x) = y$.

Ingredient 2: zero-knowledge proofs

Fix a public function f . Two algorithms, for any public value y :

- $\text{Prove}_f(x, y) \rightarrow \pi$: given a secret x with $f(x) = y$, produce a proof π .
- $\text{Verify}_f(y, \pi) \rightarrow \text{accept or reject}$.

Properties

- If you know such an x , the verifier accepts.
- If no such x exists, or you do not know one, the verifier rejects.
- The verifier learns nothing about x beyond the fact that $f(x) = y$.

Ingredient 2: zero-knowledge proofs

Fix a public function f . Two algorithms, for any public value y :

- $\text{Prove}_f(x, y) \rightarrow \pi$: given a secret x with $f(x) = y$, produce a proof π .
- $\text{Verify}_f(y, \pi) \rightarrow \text{accept or reject}$.

Properties

- If you know such an x , the verifier accepts.
- If no such x exists, or you do not know one, the verifier rejects.
- The verifier learns nothing about x beyond the fact that $f(x) = y$.

Example. Convince a friend that you solved a difficult Sudoku puzzle without showing the solution.

Ingredient 2: zero-knowledge proofs

Fix a public function f . Two algorithms, for any public value y :

- $\text{Prove}_f(x, y) \rightarrow \pi$: given a secret x with $f(x) = y$, produce a proof π .
- $\text{Verify}_f(y, \pi) \rightarrow \text{accept or reject}$.

Properties

- If you know such an x , the verifier accepts.
- If no such x exists, or you do not know one, the verifier rejects.
- The verifier learns nothing about x beyond the fact that $f(x) = y$.

Example. Convince a friend that you solved a difficult Sudoku puzzle without showing the solution.

Made much more practical in the last decade. Zcash, Google's privacy-preserving credential systems.

PACTs (2026)

Provable Address-Control Timestamps, proposed by Dan Robinson at Paradigm in May 2026.

Same shape as the first attempt, with one change: **no new key**.

Before T_c . Satoshi signs a random value and stamps the hash of the signature:

$$r \leftarrow \{0, 1\}^{256}, \quad \sigma = \text{Sign}_{sk}(r), \quad c = H(\sigma), \quad t = \text{TS.Stamp}(c).$$

He keeps r and σ secret. The world sees only c .

PACTs (2026)

Provable Address-Control Timestamps, proposed by Dan Robinson at Paradigm in May 2026.

Same shape as the first attempt, with one change: **no new key**.

Before T_c . Satoshi signs a random value and stamps the hash of the signature:

$$r \leftarrow \{0, 1\}^{256}, \quad \sigma = \text{Sign}_{sk}(r), \quad c = H(\sigma), \quad t = \text{TS.Stamp}(c).$$

He keeps r and σ secret. The world sees only c .

After T_c . The network no longer accepts signatures under old keys such as pk . Instead, to spend the coin, Satoshi attaches a zero-knowledge proof:

"I know r , σ , t such that σ is a valid signature on r under pk , the hash $H(\sigma)$ was stamped at time t , and $t < T_c$."

Robinson (2026)

PACTs: why it works

Security

- Timestamps cannot be backdated. The attacker learns sk only after T_c , so any stamp he obtains is dated after T_c .
- Note: r must be kept secret. If not, then the attacker once again knows everything that Satoshi knows, and thus can generate the same proof.

Privacy

- TS sees only a hash of a signature. Nobody can tell from it which pk it belongs to.
- Nobody can detect that Satoshi created a PACT.

Robinson (2026)

PACTs: building blocks

- **TS = OpenTimestamps.** Bitcoin itself is basically a timestamping service.
- **Zero-knowledge proof.** Must be post-quantum, such as STARK.
- **Rescue rule: requires a soft fork.** Stop accepting old signatures after T_c and accept such a proof instead.

Robinson (2026); opentimestamps.org

PACTs: what it assumes

- SHA-256 and STARKs hold up against quantum attackers.

comfortable

Robinson (2026); BIP 361

PACTs: what it assumes

- SHA-256 and STARKs hold up against quantum attackers.
- Chain history before the cutoff is immutable.

comfortable

comfortable

Robinson (2026); BIP 361

PACTs: what it assumes

- SHA-256 and STARKs hold up against quantum attackers.
- Chain history before the cutoff is immutable.
- No secret quantum computer existed before the cutoff.

comfortable

comfortable

uncomfortable

Robinson (2026); BIP 361

PACTs: what it assumes

- SHA-256 and STARKs hold up against quantum attackers.
- Chain history before the cutoff is immutable.
- No secret quantum computer existed before the cutoff.
- Having the key on a date equals rightful ownership.

comfortable

comfortable

uncomfortable

uncomfortable

Robinson (2026); BIP 361

PACTs: what it assumes

- SHA-256 and STARKs hold up against quantum attackers.
- Chain history before the cutoff is immutable.
- No secret quantum computer existed before the cutoff.
- Having the key on a date equals rightful ownership.
- The sunset fork actually includes this rescue rule.

comfortable

comfortable

uncomfortable

uncomfortable

political

Robinson (2026); BIP 361

Take-aways

1. Shor's algorithm breaks the signatures Bitcoin uses. Hash functions survive.
2. Quantum-resistant cryptography is older than Shor's algorithm.
3. No key-breaking quantum computer exists today. Bitcoin prepares anyway, because estimates keep shrinking and upgrades take years.
4. **Challenge 1: exposed keys.** Coins whose public key is already on chain give an attacker years, not minutes.
5. **Challenge 2: new signatures are big.** Post-quantum signatures are 10–100× larger, and wallets must be rewritten to use them.
6. **Challenge 3: how to move dormant coins.** Options: throttle the theft, sunset old signatures, or sunset with a rescue path.
7. **PACTs** are one such proposed rescue path.

Still an ongoing debate and none of this is live on Bitcoin as of September 2026.

References

- D. Robinson. *PACTs: Protecting Your Bitcoin From a Quantum Sunset*. Paradigm, May 2026. <https://www.paradigm.xyz/writing/pacts-protecting-your-bitcoin-from-a-quantum-sunset>
- A. Milton, C. Shikhelman. *Bitcoin and Quantum Computing: Current Status and Future Directions*. Chaincode Labs, May 2025. <https://chaincode.com/bitcoin-post-quantum.pdf>
- Project Eleven. *Q-Day Prize awarded for a 15-bit ECC key broken on quantum hardware*. Apr. 2026. <https://www.projecteleven.com/>
- S. Aaronson. *Why Is Quantum Computing So Hard to Explain?* Quanta Magazine, June 2021.
- NIST. FIPS 204 (ML-DSA), FIPS 205 (SLH-DSA), draft FIPS 206 (FN-DSA), 2024–2025; NIST IR 8547, *Transition to Post-Quantum Cryptography Standards*, Nov. 2024.
- M. Kudinov, J. Nick. *Hash-Based Signature Schemes for Bitcoin*. IACR ePrint 2025/2203.
- SHRINCS working group. *SHRINCS: Shrunk SPHINCS* (draft BIP), Aug. 2026. <https://github.com/SHRINCS/shrincs-bip>
- J. Lopp et al. *BIP 361: Post Quantum Migration and Legacy Signature Sunset*. Draft, 2026. <https://bips.dev/361/>
- H. Beast. *Hourglass v2* (draft BIP). <https://github.com/cryptoquick/bips/blob/hourglass-v2/bip-hourglass-v2.mediawiki>
- OpenTimestamps. <https://opentimestamps.org/>