TECHNICAL WHITEPAPER — V0.6

FalconDraw

A Provably Fair On-Chain Draw on a Bitcoin PoW Foundation

Issuer	Ticker	Date	Protocol	Mainnet launch
Falcon Draw LLC (Wyoming)	FDRW	June 2026	v0.6	July 1, 2026

ABSTRACT

FalconDraw is a proof-of-work blockchain forked from Bitcoin's btcd implementation in Go. It preserves Bitcoin's security model — SHA256d mining, 10-minute blocks, UTXO accounting, and 210,000-block halving — and adds exactly one new primitive: a provably fair draw that runs every 36 blocks (~6 hours), funded entirely by participant stakes.

There is no house edge. One winner receives 100% of the accumulated pool. Win probability is proportional to stake size and exactly 10% per draw. Every draw outcome is verifiable by any node from on-chain data alone. Falcon Draw LLC is a software firm only — it does not operate the draw, does not hold participant funds, and has no special network authority.

0%
HOUSE EDGE

10%
WIN FREQUENCY

4/day
DRAWS

~430M
FDRW SUPPLY

SHA256d
POW ALGORITHM

100%
ON-CHAIN VERIFIABLE

1. Introduction

Online lotteries are operated by entities that hold and disburse funds. Players must trust the operator to compute outcomes honestly, pay winners promptly, and not abscond with pooled funds. Even regulated lotteries take 40–50% as a "house edge."

FalconDraw removes the operator entirely. The draw runs on a PoW chain. Outcomes are determined by the chain's own block hashes — fully verifiable by any node from public on-chain data. A node operator cannot bias the outcome without controlling more than 50% of mining hash rate, and even then only at enormous capital cost.

The result is a draw where:

- Win probability is exactly proportional to stake (Sybil-resistant, no strategy)
- Expected value equals stake: zero house edge, proven algebraically
- Outcomes are verifiable on-chain by any observer
- Prize custody is held by the chain, not by any company
- Falcon Draw LLC cannot alter outcomes, freeze prizes, or take fees off-chain

2. Protocol Overview

FalconDraw inherits Bitcoin's core architecture and modifies three things:

1. **Block parameters** — 5MB blocks, 36-block draw cadence, 1000 FDRW base reward
2. **Transaction types** — adds TxVersionPoolStake (v6) for draw registration and TxVersionVDFResult (v7) for VDF proofs
3. **Draw consensus** — nodes run winner selection when a VDF result is posted; settlement is enforced as a consensus rule in the same block as the VDF proof

Everything else — P2P networking, UTXO set, mempool, mining, SegWit, difficulty adjustment, halving — is unchanged from btcd.

PARAMETER	VALUE
PoW algorithm	SHA256d (double SHA256)
Target block time	10 minutes
Block size	Variable, up to 5 MB base / 20 M weight
SegWit	Active from block 1 (AlwaysActiveHeight: 1)
Addresses	Bech32, prefix `fdrw` (mainnet)
Draw frequency	Every 36 blocks (~6 hours) = 4 draws/day

Base block reward	1000 FDRW
Halving interval	210,000 blocks (~4 years)
Floor reward	1 FDRW/block (never drops below)
Asymptotic supply	~430 million FDRW
Difficulty adjustment	Every 2,016 blocks

3. The Falcon Draw

3.1 Staking Phase

During the draw window — the 36 blocks of the interval leading up to draw block N — any participant may register by broadcasting a **pool stake transaction** (TxVersionPoolStake = 6). The staking transaction:

- Spends one or more P2WPKH UTXOs from the participant's wallet
- Carries a zero-value OP_RETURN output with 34 bytes of draw registration data
- Records: `walletHash (20)` + `amount (8)` + `drawHeight (4)`

The staked coins leave the wallet permanently at confirmation — there is no refund for losing stakes. An unconfirmed stake transaction may be cancelled by broadcasting a replacement spending the same inputs at a higher fee (RBF).

Staking cutoff: The protocol enforces that a stake transaction mined at block H is only valid when `drawHeight − H ≥ 3` (`StakeCutoffBlocks = 3`). This ensures every stake has at least 3 confirmations before the draw fires, making it prohibitively expensive to double-spend a stake via a chain reorganisation. A stake submitted in the final 3 blocks before the draw is rejected at the consensus level — the effective staking window is 33 blocks (blocks N−36 through N−3 inclusive). The web UI independently enforces the same threshold by skipping to the following draw whenever fewer than 4 blocks of clearance remain from the current tip.

Only whole FDRW can play.

3.2 Winner Selection — The Tape Mechanic

At draw block N, every node independently begins computing the draw outcome. No off-chain communication is required. The algorithm:

Step 1 — VDF output

After draw block N is mined, every node immediately begins computing a Sloth VDF (see §4) over the block hash — approximately 18–30 minutes of sequential work at steady state. The first node to finish broadcasts a TxVersionVDFResult (v7) transaction. Any block that includes this proof must also include the draw settlement as a consensus rule. The VDF

output is the entropy source for the draw; no miner can know the outcome before committing to the block.

Step 2 — Draw seed

```
seed = SHA256d(blockHash_N || vdf_output)
```

Where `vdf_output` is the 256-byte Sloth result. This seed is unpredictable by any miner until ~30 minutes of sequential computation completes.

Step 3 — Extended position

```
extended_pos = bigint(seed) mod (WinMultiplier × totalTokens)
```

Where `WinMultiplier = 10` (chain constant) and `totalTokens` is the sum of all staked token counts.

Step 3 — Jackpot check

```
if extended_pos < totalTokens:
    winner = wallet at position extended_pos (binary search on sorted stake list)
    jackpot = true
else:
    jackpot = false   // 90% of draws — pool rolls over
```

Visualised as a tape:

```
|<--- totalTokens --->|<---------- 9 × totalTokens (blanks) ---------->| [A tokens][B tokens]...[N tokens][ . . . . . . . . . . . . . . . . . . ] ^ lands here → jackpot (binary search → winner) ^ lands here → rollover
```

The tape has exactly `WinMultiplier × totalTokens` positions. Each token held by a participant occupies one position. The remaining `(WinMultiplier - 1) × totalTokens` positions are blanks. Win frequency = exactly `1 / WinMultiplier` = 10%, regardless of pool size.

Verified by simulation against the production Go implementation (`RunDraw`), 3.5 million draws across 7 scenarios:

SCENARIO	DRAWS	WINS	WIN%
1 staker, 1 token	500,000	50,423	10.085%
1 staker, 1,000 tokens	500,000	50,048	10.010%
5 stakers, 1 token each	500,000	50,168	10.034%
5 stakers, 200 tokens each	500,000	50,282	10.056%
50 stakers, 10 tokens each	500,000	49,350	9.870%

100 stakers, 1 token each	500,000	49,684	9.937%
3 stakers: 1, 10, 100 tokens	500,000	49,961	9.992%
All scenarios combined	**3,500,000**	**349,916**	**9.997%**

All results within 0.15% of the theoretical 10.000%. At 500,000 draws the standard deviation is ≈ 0.042%, so every scenario falls within 3σ of expectation.

- Expected value: exactly 1 FDRW per token staked (algebraic proof; zero house edge)
- No Sybil benefit: 10 tokens in one wallet = same odds as 10 wallets with 1 token each
- No tie possible: `extended_pos` maps to exactly one wallet position

3.3 Settlement

After draw block N, nodes compute the Sloth VDF in a background goroutine (~18–30 minutes). When complete, the node broadcasts a **TxVersionVDFResult (v7)** transaction. Any block N+k that includes a valid VDFResult transaction for draw N is a **VDF result block** — it must also contain the settlement transaction as a consensus rule. The VDF result block is identified by the presence of the v7 transaction, not by a fixed height offset. The draw result (jackpot flag and `winnerWalletHash`) is computed at that time and stored in the draw index; settlement validation reads two database keys — O(1) regardless of pool size or number of entrants.

Jackpot: the settlement transaction mints the full pool balance as a new P2WPKH output directly to the winning wallet address (`walletHash`). The coins appear in the winner's wallet immediately — no further action required. No claim transaction, no interaction with any server.

Rollover (no jackpot at draw N):

- The accumulated pool balance is carried forward: `poolbal[N]` added to `poolbal[N+DrawInterval]`
- `poolbal[N]` is preserved so historical queries return the correct total pool for that draw
- Single database write, no in-memory state, survives node restarts
- Pool accumulates across successive rollovers until a jackpot draw hits

4. Draw Entropy — Sloth VDF

FalconDraw uses a **Sloth Verifiable Delay Function** to derive draw entropy. This eliminates miner grinding: a miner cannot evaluate the draw outcome before committing to a block.

4.1 The Sloth VDF

Sloth computes T sequential modular square roots over a 2048-bit prime `p ≡ 3 (mod 4)`. Each step depends on the previous — the computation is strictly sequential. GPUs give zero speedup (unlike SHA256d). Verification takes T squarings and completes in ~3.5 seconds.

```
SlothEval(seed, T, p):
    x = toQR(seed mod p)   // make x a quadratic residue
    repeat T times: x = x^((p+1)/4) mod p   // modular square root
    return x

SlothVerify(seed, y, T, p):
    x = toQR(seed mod p)
    repeat T times: y = y² mod p   // squarings
    return y == x
```

The prime is RFC 3526 Group 14 (2048-bit, verified prime, `p ≡ 3 mod 4`). No external dependencies — pure Go `math/big`. T starts at 100,000 steps (~18 minutes on the reference node) and self-adjusts.

4.2 Draw Seed Derivation

```
vdf_output   = SlothEval(blockHash_N, T, p)      // ~18–30 min sequential
seed         = SHA256d(blockHash_N || vdf_output)
extended_pos = bigint(seed) mod (WinMultiplier × totalTokens)
```

A miner who controls block N must run SlothEval — ~30 minutes — before knowing `extended_pos`. Since block production is ~10 minutes, the miner must commit to the block long before the draw outcome is computable. Grinding is economically irrational.

4.3 VDF Result Transaction (v7)

Any node that completes SlothEval broadcasts a **TxVersionVDFResult (v7)** transaction. The OP_RETURN payload (324 bytes) carries:

FIELD	BYTES	DESCRIPTION
magic `FDRV`	4	Version tag
drawHeight	4	Draw block height N
drawBlockHash	32	Hash of block N (VDF input)
T_used	8	Steps computed (uint64 BE)
vdf_output	256	SlothEval result, zero-padded
submitterWalletHash	20	HASH160 of the confirmer's public key

The `submitterWalletHash` field identifies the node that performed the VDF computation and is used to direct the confirmer bounty (§4.4). Any block N+k that includes a valid VDFResult for draw N must also include the settlement transaction — VDF and settlement land in the same block. The first valid VDFResult for draw N is accepted; subsequent duplicates are

rejected.

4.4 VDF Confirmer Bounty

Running the Sloth VDF is worthwhile for stakers in that draw — they have a financial interest in the settlement proceeding. For draws with no fresh stakes, or for node operators who are not currently staked, there is no direct pool incentive. The protocol therefore pays an explicit **VDF confirmer bounty** from the coinbase of every VDF settlement block that has a non-zero pool balance:

ERA	BLOCKS	BOUNTY PER DRAW SETTLED
0	0 – 839,999	100 FDRW
1+	840,000+	50 FDRW (permanent)

The bounty halves exactly once, at block 840,000 — the epoch where the block reward first drops below 100 FDRW (to 62 FDRW at halving 4). It does not halve again — 50 FDRW per settled draw is the permanent floor. By the time block rewards reach 1 FDRW (block ~1.89M), the bounty is 50× the block reward, a strong long-term incentive for full-node participation.

How it works: the node that finishes SlothEval embeds its `submitterWalletHash` in the v7 transaction (§4.3). When a miner includes that v7 transaction in a VDF result block, they add an extra coinbase output paying the bounty amount to `submitterWalletHash`. Consensus rules verify this output is present — a VDF result block with a non-zero pool that omits the bounty output is rejected by all honest nodes. The bounty is additional inflation outside the standard block subsidy, so it does not reduce the miner's own reward.

The bounty is **only paid for draws with a positive pool balance**. VDF computation for draws with no stakers (empty pool) produces no bounty. Any full node operator, staker, or pool participant may race to post the VDF result; the first valid proof wins.

Sizing rationale — the 10% bounty

The Sloth VDF occupies one CPU core for 18–30 minutes per draw. Over a 6-hour draw window (360 minutes), that is 5–8% of one core's time. On a typical multi-core VPS (4–8 cores), the effective mining hashrate decrement is approximately **1–2%**. The 100 FDRW bounty equals **10% of the era-0 block reward** — roughly five to ten times the expected opportunity cost. This over-compensation is intentional: the goal is not merely to reimburse node operators for lost hash power, but to actively encourage a population of VDF-enabled full nodes to exist and run the draw game, independent of whether those nodes are mining at all. At mainnet with ASICs dominating hash rate, a CPU node's mining contribution is negligible regardless; the bounty then functions purely as a **full-node participation reward** that keeps the draw mechanism operating even in the absence of stakers with a direct pool incentive.

4.5 Dynamic T Adjustment

T is recalibrated every 2,016 settled draws using a rolling window. For each settled draw in the window, the node measures `elapsed = vdfResultBlockHeight − drawBlockHeight`. The new T is:

```
avgElapsed = mean(elapsed) over window
newT       = lastT × VDFTargetBlocks / avgElapsed
             clamped to [lastT/4, lastT×4], floor at VDFMinT
```

`VDFTargetBlocks = 3`. At steady-state 10-minute blocks, 3 blocks = 30 minutes. T self-adjusts so the VDFResult arrives ~3 blocks after the draw block regardless of current block time. The grinding disincentive is always proportional to 3 blocks of mining revenue — structurally identical at every stage of the chain. This measurement is objective (all nodes agree on block heights) and requires no external oracle.

VDF ASIC resistance: A custom ASIC could accelerate Sloth by approximately 2–5×. Any speedup is automatically detected within one adjustment window and T is raised accordingly, restoring the 3-block target.

5. Proof of Work

FalconDraw uses SHA256d — the same double-SHA256 algorithm as Bitcoin. ASIC miners are required at any significant scale. This choice is deliberate:

- **ASICs as a security model:** Mining ASICs represent billions of dollars of capital that can only profitably mine SHA256d chains. An attacker mounting a 51% attack must acquire or divert that hardware — cost scales with network hash rate and chain value.
- **Rental attacks rejected:** A CPU-friendly algorithm (e.g., RandomX) would allow renting cloud compute for 51% attacks at near-zero sunk cost. For a chain where block hashes directly determine draw winners, miner security must be backed by real capital destruction.
- **No CGo, no VM, no epoch machinery:** The SHA256d implementation is pure Go — no external C library, no random-access memory requirement, no per-epoch initialization.

6. Transaction Format

All FalconDraw transactions use SegWit (P2WPKH inputs, BIP143 sighash). SegWit is enforced from block 1 (`AlwaysActiveHeight: 1`). Output scripts are 22-byte `OP_0 <hash20>`. Addresses are bech32 with prefix `fdrw` (mainnet).

6.1 Regular Transfer (~141 vbytes, 1-in 2-out)

COMPONENT	BASE BYTES	WITNESS BYTES

version (4)	4	—
segwit marker + flag	2	—
input count	1	—
prevout (32+4) + sequence (4)	41	—
output count	1	—
recipient P2WPKH output (8+22+1)	31	—
change P2WPKH output	31	—
locktime	4	—
witness: ECDSA sig (~72) + pubkey (33)	—	~109
vbytes = (base × 3 + total) / 4	**115**	**~27**
Total: ~141 vbytes (vs ~226 P2PKH — 37% smaller)		

6.2 Stake Transaction (~151 vbytes)

OP_RETURN payload (34 bytes, zero value):

FIELD	BYTES	DESCRIPTION
OP_RETURN + push	2	`0x6a 0x20`
walletHash	20	HASH160 of public key
amount	8	Staked atoms (big-endian int64)
drawHeight	4	Target draw block (big-endian uint32)

Full stake tx: 125 stripped bytes × 4 + 102 witness bytes = **602 weight units ≈ 151 vbytes** (with change output; Schnorr witness).

6.3 Winner Claim Transaction

Custom Schnorr scriptSig (99 bytes): `OP_DATA64 <sig[64]> OP_DATA33 <pubkey[33]>`. The winning walletHash is the HASH160 of the corresponding public key — identity verified on-chain. Fee: ~200 atom/byte × ~200 bytes ≈ 20,000 atoms.

7. Token Economics

7.1 FDRW Supply Schedule

```
block_reward(H) = max(1, floor(1000 / 2^(floor(H / 210000))))
```

ERA	BLOCKS	REWARD	ERA SUPPLY
0	0–209,999	1000 FDRW	210M
1	210,000–419,999	500 FDRW	105M
2	420,000–629,999	250 FDRW	52.5M
3	630,000–839,999	125 FDRW	26.25M
…	…	…	…
∞	—	1 FDRW (floor)	∞

Asymptotic supply: ~430 million FDRW. This figure includes both block rewards (~418.5M from the geometric halving series) and VDF confirmer bounties (~3.8M through the block reward floor era, then ongoing at 50 FDRW per draw permanently). The 1 FDRW block reward floor and the 50 FDRW VDF bounty floor ensure the chain never reaches zero inflation; miners and VDF confirmers remain incentivized indefinitely.

7.2 Draw Prize Structure

```
pool(N) = Σ(all stakes for draw N) + poolbal[N]
```

- **Single winner:** 100% of pool. No tiers, no split.
- **No jackpot (90% of draws):** pool carries forward to next draw.
- **Win frequency:** exactly 10% per draw (`WinMultiplier = 10`).

7.3 Expected Value

For a participant staking `s` tokens in a draw with `T` total tokens:

```
P(win) = (1/WinMultiplier) × (s/T) = s / (10T)

E[prize | jackpot] = pool = 10T × 1 FDRW/token

E[prize] = P(win) × E[prize | win] = (s/10T) × 10T = s FDRW
```

Expected value equals stake. Zero house edge. This holds regardless of pool size, rollover depth, or number of participants — it is a consequence of the tape mechanic structure, not an approximation.

ROLLOVER DEPTH	PROBABILITY	PRIZE SIZE
0 (no rollover)	10%	1× draw pool
1 rollover	9%	2× draw pool
2 rollovers	8.1%	3× draw pool

k rollovers	10% × (90%)^k	(k+1)× draw pool

Average prize size: 10× per-draw pool contributions. Because the distribution is memoryless, each draw is fully independent of prior results — a long streak of rollovers does not make the next win more likely.

Consecutive no-win probability (10% win chance per draw, independent):

CONSECUTIVE MISSES	PROBABILITY
1	90.000%
5	59.049%
10	34.868%
15	20.589%
20	12.158%
25	7.179%
30	4.239%

A run of 15 consecutive no-win draws has a 20.6% probability — expected to occur roughly once in every five 15-draw stretches. A run of 30 misses has a 4.2% probability, still well within normal variance for a chain running 4 draws per day.

8. Block Capacity and Scalability

PARAMETER	VALUE
Max block weight	20,000,000 (5 MB blocks)
Stake tx size	~151 vbytes (602 weight units, with change)
Stake txs per block	20,000,000 / 602 ≈ **33,000**
Blocks per draw window	36
Stake txs per draw	**~1.2 million**
Draws per day	4
Stake txs per day	**~4.8 million**

At launch, 1.2 million entrants per draw far exceeds any realistic near-term demand. The 36-block staking window distributes load across six hours of block space rather than requiring a single burst block. If demand exceeds on-chain capacity in the long term, off-chain aggregation (ZK proof batch registration) is compatible with the protocol architecture and deferred to a future upgrade.

9. Security Model

9.1 Winner Selection Integrity

```
vdf_output = SlothEval(blockHash_N, T, p)
seed       = SHA256d(blockHash_N || vdf_output)
winner     = tape_position(seed, totalTokens, WinMultiplier)
```

Any node can recompute this from the VDFResult tx and verify it matches the stored draw result. There is no trusted party in the computation path.

9.2 Miner Grinding

A miner controlling block N could try alternative nonces, producing different block hashes, hoping to find one that produces a favourable draw outcome. The Sloth VDF eliminates this:

- To evaluate the draw outcome for a candidate block hash, the miner must run SlothEval — ~30 minutes of strictly sequential computation at steady state.
- Since block production averages 10 minutes, a new block will arrive before the miner finishes even a single grind trial.
- **Economic cost:** Withholding a valid block to compute the VDF and see the result forfeits ~3 block rewards — a proportional cost maintained at every stage of the chain by the T adjustment mechanism (§4.4).

9.3 51% Attack

A miner controlling 51% of hash rate can guarantee they mine block N, choosing whichever block hash maximises their draw outcome. However:

- Mining hardware for a 51% attack on a mature SHA256d chain costs hundreds of millions to billions of dollars
- Diverted hash rate means lost mining revenue on the existing Bitcoin chain
- The attack wins at most a few large prizes before community response
- Attack cost scales with network value — the chain is self-protecting at scale

9.4 Node Security

The `fdrw` CLI wallet is a **hot wallet** — it stores the signing key on disk so it can sign transactions without user interaction. The key is protected only by filesystem permissions. Operators are advised to treat it as permanently compromisable and to sweep any significant balance (including jackpot prizes, which land automatically) to cold storage promptly using `fdrw send`.

Mining rewards are configured separately: the `miningaddr=` field in `btcd.conf` accepts a bech32 address only — the corresponding private key never needs to be on the mining

server. This allows mining operators to direct block rewards to a cold wallet with no server-side key exposure.

The pprof profiling server (if enabled via `--profile`) is bound exclusively to `127.0.0.1`. RPC connections use TLS with the node's self-signed certificate; `fdrw` loads the certificate from `~/.btcd/rpc.cert` automatically and reads credentials from `btcd.conf`, so no credentials need to be passed on the command line.

9.5 Anti-DDoS: No Winner Interaction Required

Traditional lottery systems require the winner to contact the operator, creating a DDoS target. FalconDraw has no such endpoint:

- Settlement is computed entirely by the node from on-chain data
- The winning wallet receives a standard P2WPKH output in the VDF result block — coins arrive automatically
- No claim transaction, no winner communication, no server interaction required
- There is no endpoint to flood; prize delivery is a consensus-enforced block event

9.6 Conservation of Value

Every atom that enters the draw pool (via staking inputs) must appear in either the settlement prize UTXO (jackpot draw) or the rollover pool balance for the next draw. Settlement consensus rules enforce this at block connection time — an invalid VDF result block that omits the settlement transaction or mints the wrong amount is rejected by all honest nodes.

10. Node Operation

Operators run the FalconDraw full node (`btcd`) and optionally `fdrw` (the CLI wallet). Full node source code is published at github.com/FalconDraw/VPS — anyone may compile and run a node independently. Falcon Draw LLC also produces a web UI and Telegram Mini App as two ways to interact with the protocol, but neither is the exclusive interface — participants may use the CLI wallet, build their own UI against the JSON-RPC API, or interact with the chain directly.

Node income streams:

1. **Block rewards** — same halving schedule as FDRW emission
2. **Transaction fees** — all fees from transactions mined in the node's blocks
3. **VDF confirmer bounty** — 100 FDRW per draw settled in era 0 (blocks 0–839,999); 50 FDRW permanently after block 840,000 — paid to whichever node first posts a valid VDF result for a draw with a non-zero pool (§4.4)
4. **Draw participation** (optional) — operators may stake like any participant

Mining and the VDF bounty — a deliberate trade-off: A node spending CPU cycles on VDF computation is not simultaneously contributing that CPU to PoW mining. The protocol treats these two activities as complementary rather than competing. Block security comes from dedicated SHA256d miners (ASIC hardware). VDF confirmation comes from full nodes running the sequential Sloth computation in a background thread. The confirmer bounty is sized to make running VDF computation economically rational for a node that is *not* a dedicated miner — it is specifically designed to attract full-node participation separate from the mining pool.

Stratum mining pool: Node operators who also mine may do so solo (directing hash power at their own node's `getblocktemplate`) or by connecting to a Stratum pool. FalconDraw is compatible with ckpool — an open-source, zero-fee Stratum pool server. External SHA256d miners connect to the pool via the standard Stratum protocol, submit shares, and receive rewards proportional to their contributed hash rate.

Falcon Draw LLC operates a public hosted Stratum pool for SHA256d miners who do not wish to run their own node. Connect any standard SHA256d ASIC or CPU miner to `stratum+tcp://pool.falcondraw.com:3333` using your `fdrw1q…` wallet address as the username. Solo node operators may continue to mine directly against their own `btcd` instance.

Future direction: The current architecture keeps VDF computation on the node because existing Stratum pool infrastructure has no mechanism to distribute sequential, state-dependent work to remote workers. Future pool protocol extensions could change this — a Stratum-like protocol that coordinated both SHA256d mining and Sloth VDF computation would allow pool participants to contribute hash rate and VDF compute simultaneously, removing the trade-off entirely.

INTERFACE	ACCESS	KEY STORAGE
Web UI (falcondraw.com)	Browser, desktop or mobile	WebAuthn passkey (device-bound); PRF→HKDF→private key
Telegram Mini App (@FalconDrawBot)	Telegram app, any platform	Telegram CloudStorage (synced to Telegram account)
CLI wallet (`fdrw`)	Terminal	Local key file
Discord (discord.gg/hbM8vnaTu)	Discord app, any platform	N/A — community and announcements

Both the web UI and Telegram Mini App derive the wallet private key entirely client-side; no private key material is transmitted to or stored by any server. The Telegram Mini App uses Telegram's CloudStorage API to persist the key across reinstalls and devices within the same Telegram account. Users may export a BIP39 recovery phrase (derived from the raw 256-bit private key) for backup or cross-platform recovery.

Address resolution: The web UI and Telegram Mini App both support sending to a human-readable handle in addition to a raw `fdrw1q…` address. Resolution order: (1) FalconDraw handle (`@handle` , registered via passkey wallet); (2) Telegram username (`@telegramusername` ,

registered automatically on first Mini App open); (3) raw bech32 address.

Winner Board: The web UI and Telegram Mini App both display a live Winner Board — an on-chain record of all jackpot wins, sorted by prize size descending. Each entry shows the draw number, prize amount, and the winner's registered handle or Telegram username (if any). The board is populated entirely from chain data and requires no off-chain reporting.

Community: The FalconDraw Discord server (discord.gg/hbM8vnaTu) is the primary community hub, providing draw result announcements, jackpot notifications, mining pool updates, and general discussion. Future protocol upgrade proposals will be posted there first.

10.1 Passkey Wallet Design

The web UI wallet is derived entirely from a **WebAuthn passkey** using the PRF extension — no password, no seed phrase required at setup, and no private key material ever leaves the device.

Key derivation

1. The user authenticates with a passkey (platform authenticator: fingerprint, face unlock, or hardware key).
2. The WebAuthn PRF extension evaluates a fixed, well-known salt against the passkey's internal PRF, producing a 32-byte pseudo-random output. This output is deterministic: the same passkey always produces the same bytes.
3. HKDF-SHA256 derives a 32-byte secp256k1 private key from the PRF output.
4. The wallet address and `walletHash` (HASH160 of compressed public key) are derived from that private key using standard P2WPKH derivation.

The server stores only the WebAuthn credential ID and a server-side user handle — never the PRF output, the private key, or any key material. The private key is re-derived in the browser on every login and held in memory for the session only.

Cross-device access

Because the derivation is deterministic, any device where the passkey is synced produces the same wallet. Passkeys sync automatically via the user's password manager (Google Password Manager, iCloud Keychain, etc.). A user who signs in on a new phone immediately has full access to the same wallet with no manual key transfer.

Recovery

If access to all synced passkey devices is lost, the wallet can be recovered from a BIP39 mnemonic phrase (derived from the raw 256-bit private key and displayable from wallet settings). This phrase is compatible with any standard BIP39 wallet tool and provides a cross-platform escape hatch independent of the WebAuthn infrastructure.

One passkey — one wallet

Each WebAuthn credential maps to exactly one wallet address. There is no multi-wallet support per passkey and no server-side key storage. Switching to a different passkey provider (e.g., migrating from Google to Apple) creates a new credential and therefore a new wallet; the BIP39 recovery phrase from the original wallet can be used to import the old wallet into the new session.

Handle system

A human-readable handle (formatted as `adjective-animal-NNN`) is derived deterministically from the wallet address and registered with the node on first login. Handles are stable — the same passkey always produces the same handle. An optional public directory allows handle-to-address resolution for payment links and `@handle` transfers.

COMMAND	DESCRIPTION
`fdrw init`	Generate a new wallet key
`fdrw info`	Show wallet address and pubkey hash
`fdrw balance`	Show spendable and staked balance
`fdrw send <address> <amount>`	Send FDRW (~141 vbytes, P2WPKH)
`fdrw pool-stake <amount>`	Stake FDRW for the next draw (~151 vbytes)
`fdrw check-draw [height]`	Show draw state (default: next draw)
`fdrw draw-history [n]`	Show last N draw results (default: 10)

11. Legal and Governance

Falcon Draw LLC (Wyoming) is a software services firm. Its role is limited to publishing the FalconDraw software (open source), operating seed nodes and web infrastructure, and proposing protocol upgrades via software releases.

Falcon Draw LLC does **not**: operate the draw; hold participant funds; take off-chain fees; have special network authority (no admin keys, no emergency stop).

Legal profile is modeled on Bitcoin at launch: software firm ships open-source code; the network is decentralized; participants transact directly with the chain.

- Genesis parameters are immutable once mainnet launches
- Protocol upgrades require miner adoption (identical to Bitcoin's BIP process)
- No token presale; no ICO; genesis bootstrapped identically to Satoshi's early mining
- Draw participation is pseudonymous (wallet address only); no KYC collected on-chain
- **Sweepstakes classification:** Because every participant is offered a free stake on each draw (no purchase necessary), FalconDraw may qualify as a sweepstakes rather than a

lottery under U.S. law. The lottery definition requires consideration (payment); a mandatory free-entry alternative eliminates that element. The protocol implements this via an on-chain faucet: new wallets (web and Telegram) receive an initial FDRW grant, providing a genuine no-purchase entry path. Participants should consult applicable local law.

- **No monetary value — online game:** FDRW tokens are not listed or traded on any exchange and have no established market price. Because the tokens carry no monetary value, staking them does not constitute "consideration" in the legal sense. As such, FalconDraw is presently an online game rather than a lottery or gambling product under most legal frameworks that define gambling by the presence of money or monetary-equivalent stakes.

12. Comparison

FEATURE	FALCONDRAW	TRADITIONAL LOTTERY	CASINO
House edge	**0%**	40–50%	2–15%
Outcome verifiable?	**Yes (on-chain)**	No (operator)	No (RNG)
Custodial?	**No**	Yes	Yes
Prize tied up?	**No (UTXO claimable anytime)**	Often (claim deadline)	No
Entry size	Any whole FDRW	Fixed ticket price	Variable
Jackpot frequency	**10% of draws produce a jackpot**	Infrequent; jackpot rolls over when no ticket matches	Rare (major jackpots); frequent small payouts
Individual win odds	Proportional to stake share	~1:300M per ticket (Powerball)	~90% per spin (slot)
Sybil resistance	**Yes (proportional odds)**	No (per-ticket)	N/A

13. Conclusion

FalconDraw is Bitcoin with one addition: a provably fair draw that runs every six hours, funded entirely by participant stakes, with zero house edge and fully on-chain verifiable outcomes.

The protocol makes no promises that cannot be verified on-chain. It requires no trust in any company, server, or individual. Prizes are held by the chain; winners claim them like any UTXO.

Falcon Draw LLC exists to publish software and operate infrastructure. The draw runs whether or not Falcon Draw LLC continues to exist.

One sentence: FalconDraw is Bitcoin plus a provably fair, zero-edge, fully on-chain draw.

---

14. Glossary

Bech32

> The address encoding format used by FalconDraw. Produces human-readable addresses with a prefix (`fdrw` on mainnet) and built-in error detection. Example: `fdrw1qxyz…`

VDF (Verifiable Delay Function)

> A function requiring exactly T sequential computation steps to evaluate (cannot be parallelised), with output verifiable in milliseconds. FalconDraw uses Sloth — modular square roots over a 2048-bit safe prime, implemented in pure Go with no external dependencies. The VDF is live and determines draw entropy on every draw, eliminating miner grinding. See §4.

Draw height

> The block number at which a draw is resolved. Draw heights are multiples of `DrawInterval` (36). Stakes must be confirmed before this block to be included.

Extended position (`extended_pos`)

> The integer position sampled from the tape. Computed as `bigint(stochastic_value) mod (WinMultiplier × totalTokens)`. If it falls within the token range it maps to a winner; otherwise it is a blank (rollover).

HASH160

> SHA256 followed by RIPEMD160. Produces a 20-byte fingerprint of a public key. Used as the `walletHash` throughout the protocol.

Jackpot

> A draw in which `extended_pos < totalTokens`. The full accumulated pool is paid to the winning wallet in the settlement block. Occurs with probability 1/WinMultiplier = 10% per draw.

OP_RETURN

> A Bitcoin script opcode that marks an output as unspendable and allows arbitrary data to be embedded in the blockchain. FalconDraw pool stake transactions use a 34-byte OP_RETURN output to record draw registration data (`walletHash`, amount, draw height).

OutPoint

> A globally unique reference to a specific output in a previous transaction, consisting of a

> 32-byte transaction ID (`txid`) and a 4-byte output index (`vout`). Inputs spend outpoints.

P2WPKH (Pay to Witness Public Key Hash)

> A SegWit output type that locks coins to the HASH160 of a public key. The locking script is 22 bytes: `OP_0 <hash20>`. Spending requires an ECDSA signature and public key placed in the witness field rather than the scriptSig. All FalconDraw addresses are P2WPKH.

Passkey

> A WebAuthn credential stored in the user's platform authenticator (fingerprint sensor, Face ID, or hardware key) and synced via the user's password manager. FalconDraw uses the WebAuthn PRF extension to derive a deterministic wallet private key from the passkey — same passkey always produces the same wallet, with no key material ever sent to a server.

PRF (Pseudo-Random Function extension)

> A WebAuthn extension that allows an authenticator to evaluate a keyed pseudo-random function using the passkey's internal secret as the key and a caller-supplied salt as the input. The output is deterministic per credential and salt. FalconDraw uses this to derive a wallet private key without storing any secret server-side.

Pool stake transaction (TxVersionPoolStake, v6)

> The custom transaction type used to enter a draw. It spends one or more P2WPKH UTXOs, carries a 34-byte OP_RETURN draw registration output, and uses a Schnorr signature in the witness. The staked coins are permanently consumed — there is no UTXO created for the staked amount.

RBF (Replace-by-Fee)

> A mempool policy that allows an unconfirmed transaction to be replaced by a new version spending the same inputs at a higher fee. An unconfirmed stake transaction may be cancelled this way before it confirms; once confirmed the stake is final.

Rollover

> When a draw produces no jackpot (`extended_pos ≥ totalTokens`), the accumulated pool balance is carried forward to the next draw. The balance compounds across successive rollovers until a jackpot draw hits.

Schnorr signature

> A digital signature scheme producing compact 64-byte signatures (vs ~72 bytes for DER-encoded ECDSA). Used in FalconDraw pool stake transactions (v6) and winner claim transactions (v4).

SegWit (Segregated Witness)

> A Bitcoin protocol upgrade that moves signature data (the "witness") out of the transaction body into a separate structure. This reduces the effective byte weight of transactions and fixes transaction malleability. Active from block 1 on FalconDraw.

Settlement block

> The first block N+k that contains a valid VDFResult transaction (v7) for draw block N. Consensus rules require that block to also contain exactly one settlement transaction that pays the jackpot prize (or records a rollover). The VDF result block is identified by the presence of the v7 transaction, not by a fixed height offset. An invalid VDF result block is rejected by all honest nodes.

SHA256d

> Double SHA256: `SHA256(SHA256(data))`. The proof-of-work and hashing algorithm used by Bitcoin and FalconDraw. Requires dedicated ASIC hardware at any significant scale.

Draw entropy

> The deterministically verifiable value used to resolve draw outcomes. Computed as `SHA256d(blockHash_N || vdf_output)`, where `vdf_output` is the result of running the Sloth VDF over the draw block hash. The VDF requires ~18–30 minutes of sequential computation, making it impossible for miners to evaluate draw outcomes before committing to a block. See §4.

Tape mechanic

> The draw algorithm. Conceptually, a tape of length `WinMultiplier × totalTokens` positions is laid out: the first `totalTokens` positions are assigned to stakers proportionally, the remainder are blanks. A single position is sampled; landing on a staker's region selects that staker as winner, landing on a blank is a rollover.

UTXO (Unspent Transaction Output)

> A discrete unit of spendable coin on the blockchain. Every transfer creates one or more UTXOs; spending a UTXO destroys it and creates new ones. A wallet's balance is the sum of its UTXOs. FalconDraw pool stakes permanently consume UTXOs without creating a staking UTXO in return.

vbytes (virtual bytes)

> The SegWit-adjusted transaction size used for fee calculation. Computed as `ceil((base_size × 3 + total_size) / 4)`, where base_size excludes witness data. Witness bytes count at ¼ weight, so a P2WPKH transfer (~141 vbytes) is significantly cheaper than a legacy P2PKH transaction (~226 bytes).

walletHash

> The HASH160 (SHA256 + RIPEMD160) of a participant's compressed public key. Used as the 20-byte wallet identifier in OP_RETURN stake data, draw records, and winner selection. It is the same value encoded in a bech32 P2WPKH address.

WinMultiplier

> Chain constant set to 10. Determines the win frequency (1/WinMultiplier = 10% per draw) and the tape length (`WinMultiplier × totalTokens`). It also fixes the expected value: a winner receives WinMultiplier × their stake, yielding exactly 1× stake as expected value across all draws.

↓ Download PDF ← Back to Dashboard