MoneroUSD Protocol

Whitepaper v1.2 — Protocol v1.2.182 · Adds Bulletproofs++ + DSOL canonical syscalls

April 2026

CONTENTS

1. Abstract

MoneroUSD (USDm) is a privacy-preserving, overcollateralized USD stablecoin built on a Monero technology fork. It combines Monero's battle-tested privacy infrastructure with stablecoin mechanics to create a digital dollar that respects user sovereignty. USDm is backed by a basket of BTC and XMR reserves maintained at a minimum 150% collateralization ratio, with adaptive fees and usage-proportional emission that protect reserve solvency under all market conditions.

The protocol employs FCMP++ (Full-Chain Membership Proofs) for transaction privacy, providing an anonymity set equal to the entire UTXO set — a significant improvement over traditional ring signature schemes. All wallet operations are performed locally; private keys never leave the user's machine.

---

2. Motivation

The cryptocurrency ecosystem lacks a stablecoin that genuinely respects user privacy. Existing stablecoins (USDT, USDC, DAI) operate on transparent blockchains where every transaction is publicly traceable. This creates a surveillance infrastructure incompatible with financial privacy.

Monero has proven that privacy-preserving cryptocurrency is viable at scale. However, Monero's price volatility limits its utility for everyday transactions, savings, and commerce. A merchant accepting XMR faces exchange rate risk; a user holding XMR for rent money faces the same.

MoneroUSD bridges this gap: the stability of a dollar-pegged asset with the privacy guarantees Monero users expect. It is not a replacement for XMR — it is a complement to it. Where XMR serves as private digital cash, USDm serves as a private digital dollar.

Design Principles

- **Privacy by default** — FCMP++ on all transactions, no opt-in required
- **Overcollateralization** — every USDm backed by more than $1 of real crypto assets
- **Transparency of protocol, not of users** — protocol rules are public and verifiable; user transactions are private
- **Sustainable economics** — no unsustainable yield promises, no inflationary token dumps
- **Local-first** — wallet keys stay on your machine, always

---

3. Protocol Architecture

MoneroUSD is a standalone blockchain forked from Haven Protocol (itself a Monero fork), extended with FCMP++ privacy proofs and a stablecoin reserve system. It runs its own network, consensus, and mining — separate from Monero mainnet.

CONSENSUS

RandomX

CPU-friendly PoW, same as Monero

BLOCK TIME

~120s

Matching Monero's 2-minute target

ADDRESS PREFIX

M

Mainnet addresses start with M

PRIVACY

FCMP++

Full-chain membership proofs

System Components

Desktop wallet architecture — keys never leave the local machine

The desktop wallet operates in a CakeWallet-style model: a local wallet-RPC process handles all cryptographic operations on the user's machine, communicating with a remote daemon for blockchain synchronization only. No relay servers, no session tokens, no cookies. The wallet is a direct peer of the network.

---

4. Privacy Model: FCMP++

MoneroUSD implements FCMP++ (Full-Chain Membership Proofs with Forward Secrecy), representing the next generation of Monero privacy research. Unlike ring signatures, which provide a fixed anonymity set of 16 decoys, FCMP++ proves that a spent output belongs to the *entire* UTXO set without revealing which specific output is being spent.

Ring Signatures vs FCMP++

PROPERTY	RING SIGNATURES (MONERO)	FCMP++ (MONEROUSD)
Anonymity set	16 decoys per ring	Entire UTXO set
Statistical analysis resistance	Vulnerable to intersection attacks over time	Every output equally likely
ring_size parameter	16 (configurable)	1 (FCMP++ replaces rings)
Proof system	CLSAG	Curve tree membership proofs

Implementation note: ring_size=1 is enforced unconditionally in the wallet and consensus code. The FCMP++ proof replaces the ring signature entirely — the single "ring member" is the real spend, with the membership proof providing privacy. This is not a weaker setting; it is a fundamentally different (and stronger) privacy model.

---

5. Reserve System & Collateralization

The USDm peg is maintained by a crypto reserve consisting of BTC and XMR, held in protocol-controlled wallets. The reserve system enforces a minimum 150% collateralization ratio — for every $1 of USDm in circulation, at least $1.50 of crypto assets back it.

Reserve Health Tiers

Normal	Reserve ratio > 120% — all operations enabled, base fee 0.5%	`>120%`
Warning	Reserve ratio 100%–120% — elevated fees, reduced mining rewards	`100-120%`
Stress	Reserve ratio 80%–100% — high fees, mining rewards at floor	`80-100%`
Emergency	Reserve ratio < 80% — maximum fees, mining rewards at minimum floor	`<80%`

The reserve health directly controls three protocol parameters: swap fees (Section 7), mining emission (Section 8), and lending capacity. This creates a self-correcting feedback loop — stress conditions increase fee revenue while reducing outflows, naturally restoring reserve health.

Why 150% Overcollateralization?

The 1.5:1 ratio provides a substantial buffer against crypto market volatility. Even a 33% decline in BTC/XMR prices keeps the reserve at 100% coverage. The combination of overcollateralization + adaptive fees + emission gating creates three independent defense layers. For comparison, MakerDAO requires 150-175% collateralization for DAI — MoneroUSD's 150% floor is within established DeFi norms.

---

6. Swap Protocol

MoneroUSD supports bidirectional swaps between USDm and two collateral assets: BTC and XMR. The swap protocol uses on-chain verification to ensure atomicity — no trust in third parties required.

Mint Direction (Crypto → USDm)

MINT FORMULA

```
receive_usdm = (deposit_amount × price_usd) / COLLATERAL_RATIO - fee
```

The user deposits BTC or XMR to a protocol-controlled address. Once the deposit is confirmed on-chain, the protocol mints USDm. The 1.5x collateral ratio means depositing $150 of BTC yields 100 USDm (minus the fee). The surplus grows the reserve.

Redeem Direction (USDm → Crypto)

REDEEM FORMULA

```
payout_crypto = (burn_amount × (1 - adaptive_fee)) / price_usd
```

The user burns USDm on the MoneroUSD blockchain. Once the burn transaction is confirmed, the protocol pays out the equivalent crypto. The collateral ratio does NOT apply to redemptions — only the adaptive fee is deducted. This ensures users can always exit at close to face value.

> **Critical design choice:** The collateral ratio applies only to minting, never to redemption. This asymmetry is intentional — it means every mint grows the reserve (bullish for the protocol), while every redemption is fair to the user (they get face value minus a small fee). Applying the collateral ratio to both directions would make redemption punitive and destroy user trust.

Burn Verification

Burns are verified at the daemon level using on-chain transaction confirmation via the `/get_transactions` RPC endpoint. This is more reliable than wallet-RPC methods (which require an open wallet) and works regardless of wallet state. The protocol also supports `check_tx_key` as a supplementary verification when available.

Automatic Recovery

If a swap fails due to transient issues (network timeout, service restart, temporary resource unavailability), the protocol automatically detects and recovers the swap on the next service restart. Failed burns are reset to their pre-failure state and retried. Users never lose funds due to transient infrastructure issues.

---

7. Adaptive Fee System

Swap fees dynamically adjust based on reserve health. The base fee rate is 0.5%, competitive with major decentralized exchanges. Under reserve stress, fees increase in predefined tiers to create a natural circuit breaker.

TIER	RESERVE RATIO	FEE RATE	EFFECT
Normal	> 120%	0.5%	Standard operation, competitive rates
Warning	100% – 120%	1.0%	Mildly discourages outflows
Stress	80% – 100%	2.0%	Strongly discourages outflows, doubles fee revenue
Emergency	< 80%	3.0%	Maximum protection, high fee revenue per swap

Fee Distribution

RESERVES

80%

Directly strengthens the backing

YIELD POOL

15%

Funds staking rewards

OPERATIONS

5%

Infrastructure and development

The adaptive fee system creates a self-correcting feedback loop. During stress, higher fees simultaneously discourage outflows (reducing selling pressure) and generate more revenue per swap (rebuilding reserves faster). This is fundamentally different from algorithmic stablecoins (like UST) that relied on arbitrage incentives that collapsed under stress. MoneroUSD's fees are a transparent, deterministic circuit breaker — users always know the current rate.

---

8. Adaptive Mining Emission

MoneroUSD uses a usage-proportional emission model rather than a fixed block reward. Mining rewards scale with network activity, preventing reserve dilution while rewarding miners proportionally to the value they secure.

EMISSION FORMULA

```
block_reward = max(FLOOR, avg_fee_per_block × 0.25)

where FLOOR = 0.00001 USDm, CAP = 0.1 USDm
```

Emission Parameters

REWARD RATE	FLOOR
25%	0.00001
Of average fee per block	Minimum USDm per block

CAP	GATE
0.1	Reserve
Maximum USDm per block	Rewards drop to floor under reserve stress

Why Adaptive Emission?

A fixed emission rate (e.g., 30 USDm/block) creates constant inflation regardless of network usage. In a stablecoin, uncontrolled inflation directly threatens the peg by diluting reserve backing. Adaptive emission solves this: high network activity generates fees, which justify proportionally larger rewards. Low activity means low fees and minimal emission. Under reserve stress, rewards drop to the floor (0.00001 USDm) — mining never stops, because the blockchain's security depends on continuous block production, but emission becomes negligible to protect reserve backing.

Transaction-Triggered Mining

Blocks are only mined by the primary node when the mempool contains transactions. A peer node handles keepalive (empty) blocks to maintain chain liveness. This design minimizes unnecessary emission and ensures miner rewards are correlated with actual economic activity on the network.

---

9. Price Oracle Architecture

Accurate, manipulation-resistant price data is essential for a stablecoin swap protocol. MoneroUSD implements a multi-source median oracle with redundancy, caching, and deduplication designed to serve thousands of concurrent users reliably.

Oracle Sources

#	SOURCE	TYPE	ASSETS
1	CoinGecko	Aggregator	BTC, XMR
2	CryptoCompare	Aggregator	BTC, XMR
3	Coinbase	Exchange	BTC, XMR
4	Kraken	Exchange	BTC, XMR
5	Binance	Exchange	BTC, XMR

#	SOURCE	TYPE	ASSETS
6	OKX	Exchange	BTC, XMR
7	Bitfinex	Exchange	BTC, XMR

Median Pricing

All 7 sources are queried in parallel. The median price is used — not the mean. Median pricing is resistant to manipulation: even if 3 out of 7 sources are compromised or return outlier data, the median remains accurate. Any source reporting a price more than 10% off the median is flagged and logged for investigation.

Reliability Design

- **Request deduplication** — Concurrent price requests from thousands of users are coalesced into a single external API call. 10,000 users hitting the price endpoint simultaneously results in 1 external query, not 10,000.
- **Background pre-fetching** — Prices are refreshed every 20 seconds regardless of user activity. The cache is always warm; user requests never trigger cold external API calls.
- **Stale-while-revalidate** — If all 7 sources temporarily fail, the last known good price is served for up to 5 minutes. Users never see "price unavailable" due to transient network issues.
- **Tiered cache TTL** — Fresh cache: 30 seconds. Error retry: 5 seconds. Stale fallback: 5 minutes. This balances accuracy with availability.
- **Per-IP rate limiting** — 60 requests per minute per IP, with proper 429 responses and Retry-After headers.

---

10. Lending Protocol

MoneroUSD supports collateralized lending, allowing users to borrow USDm against their BTC or XMR holdings without selling them. This preserves crypto exposure while providing stablecoin liquidity.

PARAMETER	BTC COLLATERAL	XMR COLLATERAL
Max Loan-to-Value	60%	55%
Liquidation Threshold	65%	65%
Interest	Variable, market-driven	Variable, market-driven
Collateral Ratio	> 150% at all times	> 150% at all times

The lower LTV for XMR reflects its higher volatility relative to BTC. Liquidation triggers at 65% LTV, giving borrowers a 5-10% buffer to manage their positions. All loan interest flows into the yield pool, which funds staking rewards — creating a closed economic loop where borrower demand generates staker yield.

11. Staking & Yield

USDm holders can stake their tokens for yield, with higher rates for longer lock periods. Critically, yield is paid exclusively from lending interest — never from reserves or newly minted tokens.

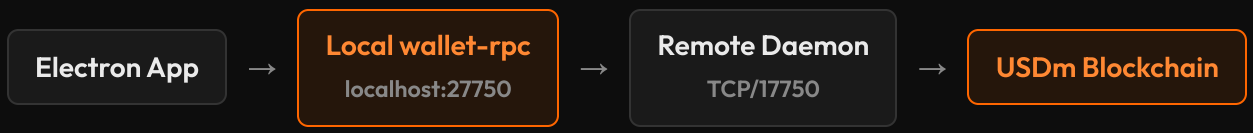
LOCK PERIOD	APR	WITHDRAWAL
Flexible (no lock)	1.5%	Anytime
30 days	3.0%	After lock expires
90 days	4.5%	After lock expires
180 days	6.0%	After lock expires

> **Sustainable yield guarantee:** Yield is only distributed when the lending interest pool has sufficient balance. If loan demand drops, yield payouts slow rather than drawing from reserves. This prevents the protocol from promising unsustainable returns — a failure mode that has destroyed many DeFi protocols (Anchor/UST, Celsius, etc.).

---

12. Wallet Architecture

The MoneroUSD desktop wallet follows a local-first, CakeWallet-style architecture. Private keys are generated and stored locally on the user's machine. All cryptographic operations (signing, proof generation) happen locally via a bundled wallet-RPC process.



Desktop wallet: keys stay local, only blockchain sync data crosses the network

Key Properties

- **No relay, no sessions, no cookies** — the desktop wallet communicates directly with the daemon. No intermediary server sees your transactions.
- **Auto-restart on crash** — if wallet-RPC dies (network error, daemon unreachable), the app automatically restarts it with exponential backoff.
- **OTA updates** — the wallet checks for updates on launch and applies them automatically, ensuring users always run the latest security patches.
- **Cross-platform** — macOS (ARM64 and x64) with Electron. Same codebase serves both desktop and browser wallet paths.

Node Infrastructure

MoneroUSD is designed for easy node deployment. The desktop wallet includes a one-click node manager that downloads the daemon binary, configures the data directory, starts syncing, and automatically connects the wallet to the local node upon successful verification.

Tri-Mode Daemon Source

The wallet supports three daemon connection modes, selectable at runtime with no restart required:

- **Public node (default)** — connects to the MoneroUSD public daemon. Zero setup; ideal for new users. A green status indicator confirms the connection.
- **Local node** — runs a full USDmd daemon on the user's machine. One-click download, sync, and wallet attachment. An orange status indicator distinguishes it from public mode. Mining starts automatically upon connection.
- **Custom VPS node** — connects to a user-operated remote daemon at any address. Available under Advanced Options. Input is validated (regex format check, 253-character limit, port range 1–65535) both in the renderer and the main process for defense-in-depth. Before switching, a JSON-RPC `get_info` probe confirms the remote daemon is reachable and running USDmd. Non-USDm endpoints are naturally rejected by response validation.

Switching between modes uses a fast path (`set_daemon` RPC) when only the daemon address changes, avoiding a full wallet-RPC restart. The wallet auto-refreshes balances and sync state after every switch.

Auto-Mining

When a local node is running, the wallet automatically starts CPU mining on three code paths to ensure blocks are always produced:

- **Button click** — user manually starts mining via the Local Node panel.
- **App load** — if a local daemon is detected on startup, mining begins without user interaction.
- **Daemon restart** — if the daemon crashes and auto-restarts, mining resumes immediately via the `onLocalNodeRestarted` event.

Mining uses background mode with configurable thread count (default 2). This ensures the blockchain always advances, even when only a single local node is running.

Block Explorer Integration

The block height displayed in the Local Node panel is interactive. Clicking it navigates the in-app dApp browser to the MoneroUSD block explorer at that specific height, providing instant visibility into block contents, transactions, and miner rewards without leaving the wallet.

Advanced Node Options

An expandable Advanced Options panel provides power-user controls: daemon address input, VPS connectivity testing, and custom node configuration. The panel is styled for discoverability (visible accent border) but collapsed by default to keep the primary UI clean. All inputs use `.textContent` assignment (never `innerHTML`) to prevent XSS, and daemon processes are spawned with array-based arguments to prevent shell injection.

Node Setup Methods

For VPS operators and terminal users, a single-command installer handles platform detection, binary download with SHA256 verification, directory setup, and optional systemd service creation. Both desktop and terminal methods use checkpoint-based fast-sync: hardcoded block hashes at known heights allow new nodes to verify chain integrity via hash comparison rather than re-validating every historical

transaction. This is cryptographically sound — a block hash commits to the entire block content including all transactions, so a matching hash at height N transitively proves all blocks 0..N are correct. Blocks above the checkpoint height receive full validation: adaptive LWMA difficulty, PoW verification, miner transaction validation, FCMP++ proof verification, and double-spend checks.

NODE SETUP METHODS

```
Desktop: Local Node tab → Create → auto-download + sync + wallet connect

Terminal: curl -sSL https://monerousd.org/install-node.sh | bash

VPS:     bash install-node.sh --prune --public --systemd

Custom:  Advanced Options → enter daemon address → Connect
```

Network ports: P2P sync on 17749, daemon RPC on 17750, restricted RPC on 17751 (public nodes), wallet-RPC on 27750. Seed node discovery via `seed.monerousd.org` with hardcoded IP fallback for DNS failure resilience. For full setup instructions, see Run a Node.

Ion Swap Developers

---

13. Comparison with Existing Stablecoins

The following table compares MoneroUSD against major stablecoin designs across dimensions that matter most to users who value privacy and sovereignty.

FEATURE	USDT / USDC	DAI	UST (FAILED)	MONEROUSD
Privacy	None (transparent ledger)	None (Ethereum)	None (Terra)	FCMP++ (full-chain privacy)
Backing	Fiat reserves (trust required)	Overcollateralized (150%+)	Algorithmic (failed)	Overcollateralized (150%+ BTC/XMR)
Centralization	Centralized issuer	Semi-decentralized (MKR governance)	Decentralized (failed)	Decentralized (protocol-enforced)
Censorship Resistance	Can freeze addresses	Moderate	N/A	Full (Monero-level)
Mining	None	None	None	RandomX CPU mining

FEATURE	USDT / USDC	DAI	UST (FAILED)	MONEROUSD
Auditability	Attestations only	Full (Ethereum)	N/A	View key verification
Self-Custody	Optional (exchange-dependent)	Yes (Ethereum wallet)	N/A	Required (25-word seed)

DAI is the closest comparable system in terms of collateralization. The key differences are privacy (DAI is fully transparent on Ethereum, MoneroUSD uses FCMP++) and governance (DAI relies on MKR holder votes, MoneroUSD uses deterministic protocol rules). The tradeoff is that MoneroUSD does not benefit from Ethereum's DeFi composability, operating instead as an independent chain optimized for private value transfer.

> **No company or foundation mints or redeems USDm.** Mint and redeem are protocol operations executed by consensus rules. No entity can freeze, blacklist, or confiscate USDm. This is fundamentally different from USDT/USDC, where the issuer maintains a centralized blacklist.

---

14. Security Considerations

Reserve Solvency

The three-layer defense system (150% overcollateralization + adaptive fees + emission gating) provides defense-in-depth against reserve depletion. Unlike algorithmic stablecoins that rely on market confidence, MoneroUSD's reserves are real crypto assets held in protocol wallets.

Oracle Manipulation

Median pricing across 7 independent sources makes oracle manipulation economically infeasible — an attacker would need to compromise 4+ sources simultaneously to move the price. Outlier detection provides real-time alerting for attempted manipulation.

Service Reliability

All protocol services use systemd process management with automatic restart, port cleanup on startup (preventing stale process conflicts), and rate limiting. Transient failures trigger automatic recovery rather than permanent failure states.

Privacy Guarantees

FCMP++ provides unconditional privacy — all transactions use full-chain membership proofs regardless of transaction type. There is no "transparent mode" or privacy opt-out. The protocol treats privacy as a fundamental right, not a feature toggle.

15. Ion Swap — Private Decentralized Exchange

Ion Swap is MoneroUSD's native decentralized exchange (DEX), operating at ion.monerousd.org. It provides private automated market-making (AMM), dark-pool limit orders, a PoW mining pool, and multi-asset bridging — all inheriting FCMP++ privacy at the consensus layer. Every swap, every liquidity position, and every limit order is private by default. There is no transparent mode.

15.1 Private Batch AMM

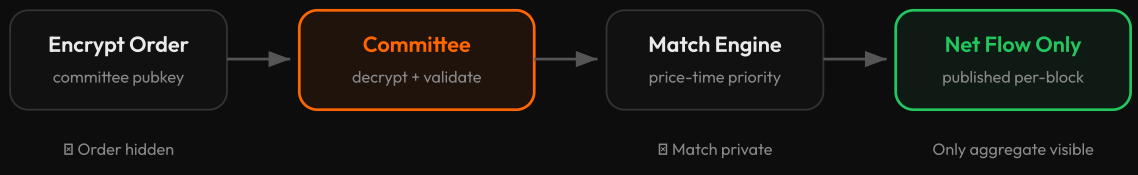
Ion Swap uses a sealed-bid batch AMM based on the Uniswap v2 constant-product formula ($x \times y = k$). Unlike conventional DEXs where trades execute immediately and are visible in the mempool, every Ion Swap trade goes through a mandatory commit-reveal cycle that makes front-running and MEV extraction cryptographically impossible.

Three-block commit-reveal lifecycle — every swap, no exceptions. MEV is cryptographically impossible.

> **No transparent swap path exists in the codebase.** The only swap ingress is the commit-reveal batch endpoint. During the commit phase, the trader submits a hash commitment that reveals nothing about trade direction or size. All reveals in a block clear at a uniform price computed over net order flow.

15.2 Dark-Pool Limit Orders

Large trades can be placed as encrypted limit orders on Ion Swap's dark-pool book. Orders are encrypted with the committee's public key. The committee decrypts and matches orders internally, publishing only per-block net flow — individual order details remain hidden from all observers.



Dark-pool order lifecycle — individual orders never revealed publicly

15.3 Protocol Fee Split

Every fee collected anywhere in Ion Swap routes through a non-configurable split, enforced at the settlement layer. The split is frozen at release time and cannot be adjusted at runtime — it is a protocol constant, not an operator parameter.

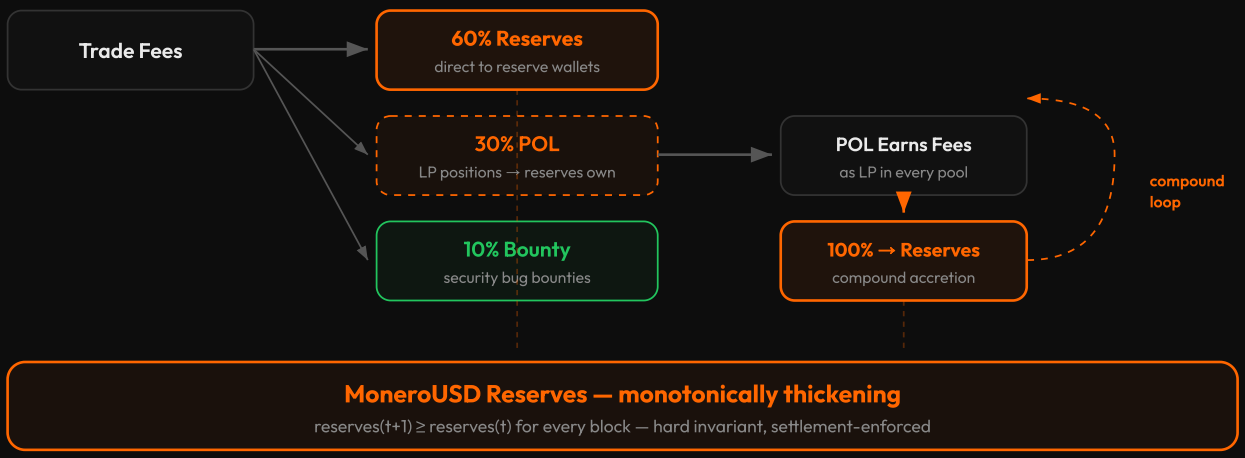
SOURCE	RATE	RESERVES	POL	BOUNTY
Standard swap	30 bps	60%	30%	10%
Stable-pair swap	5 bps	60%	30%	10%
Dark-pool match	20 bps	60%	30%	10%
Flash-loan	9 bps	70%	20%	10%
PoW mining pool	1%	80%	—	20%
Merchant invoicing	30 bps	60%	30%	10%
Bridge wrap/unwrap	10 bps	70%	20%	10%
Token creation bond	Flat	60%	30%	10%
NFT royalty	200 bps	50%	30%	20%

15.4 Reserve Accretion Model

Every feature in Ion Swap is designed to be reserve-accretive or reserve-neutral. No feature can drain MoneroUSD reserves. LP rewards never come from minted USDm — only from accumulated fee revenue, hard-capped by the reserves slice collected each block.

DAILY RESERVE ACCRETION

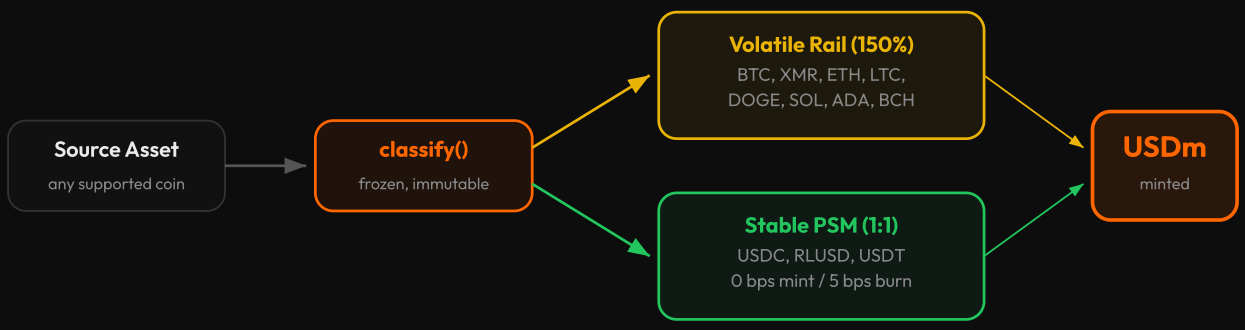
$$\Delta\text{reserves} = \Sigma(\text{fee}_i \times \text{reserve\_split}_i) + \Sigma(\text{POL\_earnings}) + \Sigma(\text{mint\_inflows})$$



Three-lane fee split with POL compound accretion loop — reserves grow every block

15.5 Dual-Rail Mint/Burn

USDm issuance operates on two rails, selected automatically based on the source asset's class. Users see a unified interface; the protocol dispatches internally.



Asset classification is immutable per launch list — no runtime reclassification. User sees a unified interface.

15.6 Multi-Asset Support & Zero-Deposit Bridging

Ion Swap supports bridging from 12+ home chains. Every bridge operation follows a zero-deposit model — the user's wallet only needs the asset being bridged, never a separate gas token. Bridge relayers pay all source-chain gas via chain-appropriate fee-delegation mechanisms.

VOLATILE ASSETS (150%)

8

BTC, XMR, ETH, LTC, DOGE, SOL, ADA, BCH

STABLECOIN ASSETS (1:1)

7

USDC (Stellar, Algo, Hedera, ETH, SOL), RLUSD, USDT

CUSTOM TOKENS

USDm-T1

Private fungible tokens with FCMP++ privacy

SHADOW NFTS

USDm-N1

Hidden ownership, encrypted content, view-key reveals

Bridge reserves are strictly segregated from USDm reserves. Each wrapped asset has its own reserve wallet on its home chain. A 50% BTC crash cannot impair USDm's collateralization ratio. Per-block invariant: home_chain_reserve ≥ circulating_wrapped_supply.

15.7 Token Addresses (ion1_)

Every asset on the MoneroUSD network is identified by a unique `ion1_` token address — a 69-character identifier derived from a BLAKE2b-256 cryptographic hash. Token addresses provide a universal, privacy-preserving identification scheme for both protocol-native verified assets and user-created custom tokens.

TOKEN ADDRESS FORMAT

```
ion1_ + BLAKE2b-256(input) → 69 characters (5-char prefix + 64 hex digits)
```

For protocol-native verified assets (USDm, wBTC, wXMR, wETH, etc.), addresses are derived deterministically from the protocol identifier: `BLAKE2b("MoneroUSD_protocol_asset:" + symbol)`. These addresses are permanent, reproducible, and independently verifiable. For user-created custom tokens, addresses are derived from the creator's stealth key and a random salt: `BLAKE2b(creatorStealth + randomSalt)`. This ensures that custom token addresses are unique and privacy-preserving — the address reveals neither the creator's identity nor the token's name.

PREFIX

ion1_

Version byte 1 — distinguishes token addresses from wallet addresses and tx hashes

HASH

BLAKE2b

CryptoNote-native hash function — same as key derivation and tx hashing

LENGTH

69 chars

5-char prefix + 64 hex digits (256-bit security)

PRIVACY

Stealth

Custom token addresses hide creator identity via stealth key + random salt

Why BLAKE2b? BLAKE2b is the hash function used natively throughout CryptoNote and Monero for key derivation, transaction hashing, and PoW. Using it for token addresses maintains cryptographic consistency across the entire protocol stack. The `ion1_` prefix (version 1) enables future address format upgrades without ambiguity.

15.8 Privacy Features

Ion Swap includes several privacy-first features that are impossible on transparent blockchains. Each generates protocol revenue and strengthens reserves.

PRIVATE DCA

Scheduled recurring swaps with time jitter to break correlation patterns

STEALTH INVOICING

Merchants receive any asset, settled privately via stealth subaddresses

INCOGNITO MODE

Enforced .onion routing, fresh subaddresses, no localStorage

FLASH LOANS

Borrow + repay in one batch window. 9 bps fee to reserves

IUSD VAULT

Pedersen-committed savings vault earning fee-split yield

CMAP

Confidential asset tags hide even the asset type from observers

15.9 PoW Mining Pool

Ion Swap operates a Stratum V2 + RandomX mining pool for USDm hashrate. PPLNS payout with per-miner variable difficulty. Username is a USDm wallet address — no account creation required. A Tor (.onion) stratum endpoint provides IP privacy for miners.

POOL FEE

1%

80% to reserves, 20% to bounty

PAYOUT SCHEME

PPLNS

Sliding window share accounting

DIFFICULTY

Vardiff

Per-miner adaptive adjustment

15.10 Circuit Breaker Recovery

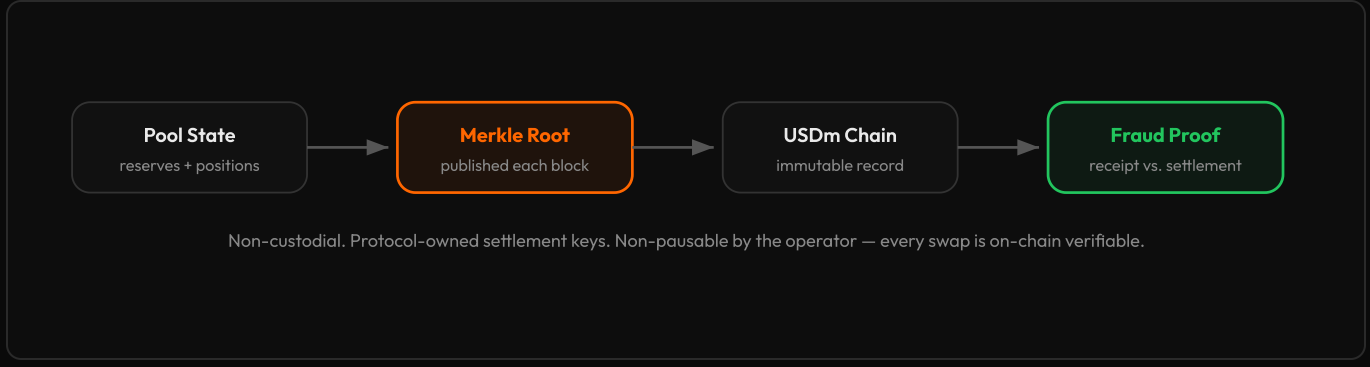
The protocol includes automated reserve-recovery mechanisms that activate during market stress. Critically, no feature is ever halted or disabled — the Monero ethos requires recovery through increased fees, not restrictions.

HEALTHY	ratio ≥ 150% — Normal operations	1× fees
WARNING	130–149% — Increased fees, reserve-priority split	1.5× fees
STRESS	110–129% — Double fees, 175% collateral requirement	2× fees
EMERGENCY	< 110% — Triple fees, accelerated liquidation	3× fees

Recovery, never restriction. Every feature stays fully operational at every tier. The system rebuilds reserves by extracting more revenue from usage during high-volatility periods — when volume is typically highest and reserves need it most. Circuit breaker tiers operate backend-only; no reserve status is exposed to any frontend.

15.11 Protocol-Owned Settlement

Ion Swap is non-custodial. The operator does not hold user funds — every reserve wallet, pool wallet, and swap-signing key is bound to the protocol itself and governed by the settlement rules encoded on the USDm chain. The backend publishes a Merkle root of all pool state (reserves, positions, balances) to the USDm chain each block. Every swap is signed by the protocol's settlement key under rules the operator cannot modify, pause, or freeze at runtime. A discrepancy between the signed receipt and on-chain settlement constitutes a cryptographic fraud proof.



---

16. Dark Contracts — Programmable Privacy

The MoneroUSD base layer defines a fixed set of protocol operations — token create, LP mint/burn, bridge wrap/unwrap, NFT mint, limit order, DCA, savings — each dispatched through a hardcoded opcode inside the attestation monitor. This scales cleanly through v1, but every new protocol feature requires a coordinated change across the backend monitor, the wallet's approval-modal renderer, and the indexer's state-derivation logic. Past that point, most of the interesting things developers want to build on Monero-style economics do not belong in the core backend — they belong in user-deployed contracts.

Dark Contracts extend the attestation dispatcher into a programmable virtual machine. Developers write in **DarkSolidity (DSOL)**, a Solidity-like language with privacy-native types and commit-reveal semantics. The compiler emits `.dc` bytecode that runs in a deterministic off-chain VM, **DarkVM**, hosted identically by every Ion Swap indexer operator. Every state mutation lands in the same append-only `pool_events` log that the existing Merkle publisher anchors to the USDm chain each block — Dark Contracts inherit the base layer's consensus-by-convention discipline without adding a new publisher path.

The goal is to match Solidity's developer UX while surpassing it on privacy. Five capabilities that transparent smart-contract chains fundamentally cannot offer are native here:

- **Private state** — Pedersen commitments over ed25519 with Bulletproofs++ range proofs (see §17). Private fields are never plaintext in any public API response; only commitment hex and a range-proof hash are exposed.
- **Front-run-free execution** — every private mutation is forced through a commit-reveal cycle by the compiler's `@batch` attribute. There is no public mempool race window.
- **Stealth-scoped identity** — `msg.sender` resolves to a caller's stealth address, not a persistent account. The same user signs successive calls from different stealth identities with no on-chain linkability.
- **Cross-chain privacy bridge as a language primitive** — the `syscall(BRIDGE_WRAP)` / `syscall(BRIDGE_UNWRAP)` host-call forms let DSOL contracts pull wBTC, wXMR, wETH, and other wrapped assets into contract logic without revealing amounts.
- **No state-inspection attacks** — contract storage is Merkle-committed to the chain for replay determinism, but never publicly readable. View access requires the contract's or caller's viewkey.

16.1 The DarkVM

The DarkVM is a pure-JavaScript stack machine with approximately 60 opcodes. It uses `BigInt` throughout — no `Number`, no `Date`, no `Math.random` — so every indexer operator running the same bytecode converges on byte-identical state. The interpreter is auditable end-to-end in roughly a week of review. A WASM backend is planned for Phase 3 performance work; the bytecode semantics do not change.

Resource discipline is intentionally flat, not market-priced. Every call pays a fixed 0.1 USDm fee and is capped at 1,000,000 instructions and 256 KB of combined stack + heap memory. The limits are constants, not parameters, so no two indexers can ever disagree on whether a given call halted successfully or ran out of budget. This trades the flexibility of per-opcode gas for a stronger consensus guarantee — in line with the broader Monero ethos of avoiding fee auctions wherever possible.

16.2 DarkSolidity (DSOL)

DSOL is a Solidity-dialect source language. The compiler lowers DSOL source through an abstract syntax tree, a typed intermediate representation, and a code generator that produces the `.dc` bytecode artifact plus a companion `abi.json` descriptor. Source is advisory; bytecode is the consensus artifact. The compiler ships inside the desktop wallet so every user can compile and sign a contract deployment locally, without uploading source anywhere.

A production-ready private ERC-20 analogue in DSOL:

DSOL — ERC20PRIVATE (PRODUCTION-READY)

```
dark contract Erc20Private is Ownable {
  private mapping(stealth ⇒ uint64) balances;
  public string symbol;
  public uint64 totalSupply;

  constructor(string sym, uint64 supply) {
    symbol = sym;
    totalSupply = supply;
    balances[msg.sender] = supply;
  }

  @batch
  entry transfer(stealth to, uint64 amount) {
    require(amount > 0, "AMOUNT_ZERO");
    require(to ≠ msg.sender, "SELF_TRANSFER");
    require(balances[msg.sender] ⩾ amount, "INSUFFICIENT");
    balances[msg.sender] = balances[msg.sender] - amount;
    balances[to] = balances[to] + amount;
    emit encrypted Transfer(msg.sender, to, amount);
  }

  @direct
  entry balanceOf(stealth who) returns (uint64 when revealed) {
    return balances[who];
```

```
    }
  }
```

Three `require` guards on `transfer` are mandatory across every DSOL transfer entrypoint — `amount > 0` rejects no-op gas-burn calls, `to ≠ msg.sender` rejects no-op self-transfers, and the balance check rejects overdrafts. The compiler does not insert these for you. v1.2.182's production-readiness sweep made them mandatory across the standard library; any contract that ships without them is a regression.

Four language constructs do most of the privacy work:

- `private` fields are Pedersen-committed on every write; range proofs are emitted at reveal time.
- `@batch` on an entrypoint forces the commit-reveal cycle so the call cannot be front-run — the compiler emits a `REQUIRE_COMMIT_REVEAL` preamble that the runtime enforces (invariant DC-4).
- `emit encrypted` AES-256-GCM-wraps the event payload using a key derived via HKDF-SHA256 from the caller's viewkey, the contract id, and the event name — only the caller can decrypt.
- `when revealed` on a return type compiles to an explicit open-and-prove: the compiler auto-generates the range proof and encrypts the opened value to `msg.sender`, so a public getter is actually a per-caller private disclosure.

Phase 3 of the language adds Solidity-familiar constructs: `is` inheritance with C3 linearization, `modifier` declarations inlined per-entrypoint, and `EXT_CALL_TAIL` — an inter-contract call restricted to tail position. Tail-only calling eliminates reentrancy by construction, because there is no caller frame to return into. Upgrades are expressed through a nullifier-gated proxy pattern rather than mutable bytecode, preserving invariant DC-13.

16.3 Life of a call

Deploy once, then commit at block N and reveal at block N+2. The indexer's `deriveDarkState` pass picks up every state diff deterministically; the explorer decodes arguments against `abi.json`.

Scheduled logic is handled through a permissionless keeper model. Contracts expose a public `tick()` entrypoint that anyone can call for a per-call bounty; there is no per-block "run every contract" loop that would force every indexer to do work the operator may not care about. Contracts the operator is not interested in can be replayed lazily on demand — the `/v1/contracts/:id` endpoint surfaces a `lastComputedBlock` so clients can tell whether they are looking at up-to-date state.

16.4 Invariants

Dark Contracts layer fifteen invariants on top of the existing base-layer rules. Each one has a single enforcement point so the audit surface stays small.

#	INVARIANT	ENFORCEMENT
DC-1	Deterministic execution	Pure-JS `BigInt` VM; no `Number` / `Date` / `Math.random`; typecheck rejects non-deterministic sources.
DC-2	Step-limit termination	1,000,000-opcode counter in `vm.js::execute`; overrun aborts the call, rolls back state, keeps the fee.
DC-3	Memory-limit termination	Per-call 256 KB cap tracked across stack + heap.
DC-4	Commit-reveal for private mutations	Compiler emits `REQUIRE_COMMIT_REVEAL` preamble; runtime rejects direct calls.
DC-5	Nullifier uniqueness	`contract_nullifiers` primary-key enforcement; duplicate insert aborts the tx.
DC-6	No plaintext private state in public APIs	`/v1/contracts` returns commitment hex + range-proof hash only.
DC-7	Syscall allowlist	Frozen table in `host.js`; unknown codes yield `DC_SYSCALL_UNKNOWN`.
DC-8	Syscall invariants propagate	Host forwards to existing base-layer handlers; same rules apply.
DC-9	State anchored every block	Existing `pool_events` + Merkle publisher pipeline; zero new publisher code.
DC-10	Deployment bond non-refundable	Re-uses the same bond path as TOKEN_CREATE.
DC-11	No transparent call path	No HTTP route mutates contract state; only attestations can.
DC-12	No `pool_events` bypass	Single writer always emits the event inside the mutation's transaction.
DC-13	Bytecode immutable	`code_hash` update trigger denies mutation; upgrades flow through the proxy pattern only.
DC-14	Compile-time privacy analysis	Typecheck runs information-flow analysis: private → public without `when revealed` is a compile error.
DC-15	Wallet decodes before approval	The approval modal decodes `DC_CALL` arguments against ABI; loud warning if the ABI is missing.

16.5 Composition with the base layer

Dark Contracts are strictly additive. The hardcoded fast path for TOKEN_TRANSFER, LP_MINT, BRIDGE_WRAP, and the other v1 opcodes remains unchanged — any asset the base layer already tracks keeps its fast-path guarantees. New DSOL contracts that need those primitives reach them through the

syscall interface (`syscall(TOKEN_TRANSFER_EMIT, {...})`), which forwards into the exact same backend handler that processes the hardcoded path. The on-chain `token_transfer` , `bridge_mint` , and `bridge_burn` events are byte-identical whether a base-layer op or a Dark Contract produced them, so indexers require no special-casing.

Inter-contract calls are restricted to tail position through the `EXT_CALL_TAIL` opcode. This removes the entire class of reentrancy vulnerabilities that have driven most historical smart-contract exploits on transparent chains — there is no caller frame to return into, so there is no window in which a callee can re-enter a mid-execution caller. Arbitrary call graphs are intentionally deferred; they will return only alongside a formal reentrancy discipline.

16.6 Production-readiness security model (v1.2.182)

Two security rules are now binding on every DSOL contract that lands in the wallet bundle:

- **Caller-authorization on every state-mutating entrypoint.** Nobody other than the owner, the scoped stealth address, or the explicit `operatorStealth` set at deploy may call any function that mutates state attributed to a specific actor, acts as a privileged operator, or modifies another actor's resource. Comments-as-authorization (e.g. `// operator-only` without the matching `require()`) are forbidden — that bug class produced two critical findings in the v1.2.182 audit.
- **Mandatory security audit before bundling.** Every contract change runs through a checklist before it's compiled into the wallet binary: documented authorization model, `msg.sender` guards on every privileged path, positivity checks on numeric inputs, self-transfer + zero-destination checks on transfer paths, unique nullifiers per authorized intent, and parent-guard replication across every override. The full DSOL test suite must pass.

> **Layered defense.** For mints, supply changes, and bridge unwraps, the on-chain DSOL `require()` chain is one layer; the backend canonical-handler check (e.g. `tokens/mint.js::mintSupply` validating `creator_stealth === ctx.callerStealth`) is another. Both must hold for the operation to land. A DSOL bug that bypasses one layer is caught by the other.

---

17. Bulletproofs++ — Real Range-Proof Soundness

Pedersen commitments hide values, but a commitment alone cannot prove the value is in a sensible range. Without a range proof, a malicious party could commit to a negative balance, an absurdly large overdraft, or any other field-element value the verifier cannot directly inspect. **Bulletproofs++** closes that gap: it lets a prover demonstrate that a committed value lies in $[0, 2^{64})$ without revealing the value itself, in **544 bytes** per proof, with no trusted setup.

The MoneroUSD verifier is a clean-room implementation in approximately 2,660 lines of pure-JavaScript `BigInt` arithmetic — auditable end-to-end, deterministic across every indexer operator, and live across the protocol since v1.2.182. Soundness defenses were validated against the twelve-item attack-surface review in `BPP-AUDIT.md` , with 250 adversarial tests covering torsion-point rejection, non-canonical scalar refusal, transcript-replay resistance, and prover/verifier fold-round consistency.

17.1 Why Bulletproofs++, not Bulletproofs+

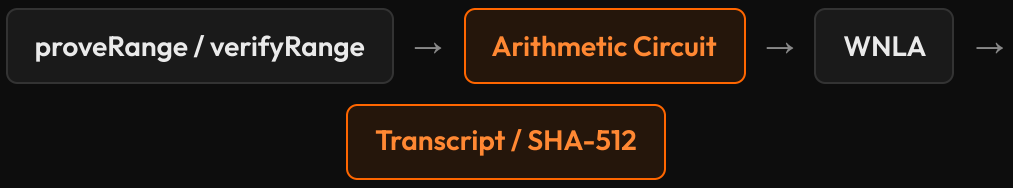
Bulletproofs+ (BP+, single-plus) is the proof system shipping today inside Monero's base layer. Bulletproofs++ (BP++, double-plus) is the next-generation construction by Eagen, Kanjalkar, Ruffing, and Nick (IACR ePrint 2022/510, EUROCRYPT 2024) that swaps BP+'s inner-product argument for the **weighted norm linear argument (WNLA)** built on top of a **reciprocal set-membership argument**. Practical wins:

PROOF SIZE	VERIFIER TIME
544 B	**~800ms**
vs ~672 B for BP+ on the same 64-bit range	End-to-end on commodity hardware

TRUSTED SETUP	AGGREGATION
None	**Native**
Pure DL hardness on ed25519	WNLA composes for batch range proofs

17.2 Construction in three layers

The Bulletproofs++ verifier is implemented as a stack of four well-defined pieces, each independently testable. The combined module is approximately 2,660 lines of JavaScript, audited end-to-end (see `BPP-AUDIT.md`) against twelve attack-surface items.



Reciprocal range proof at the top reduces to a circuit of multiplication + linear gates, which reduces to a single WNLA invocation, which reduces to a Fiat-Shamir transcript of point + scalar appends.

- **Reciprocal range argument (top layer).** Decomposes the value X into k digits in base b, then proves digit-validity via the identity $r_i \cdot (d_i + e) = 1$ where e is a Fiat-Shamir challenge. The trick: this identity holds iff every d_i is in $[0, b)$. Canonical parameterization for uint64 is $k = b = 16$.
- **Arithmetic circuit (middle layer).** Translates the digit-decomposition + multiplicity vectors into a circuit of multiplication and linear gates, builds the eight partition-routed coefficient matrices, and constructs the polynomial commitments that feed the WNLA.
- **Weighted Norm Linear Argument (WNLA, inner layer).** Replaces BP+'s inner-product argument with a recursion over even-odd halving of length-n vectors, emitting one cross-term commitment (R, X) pair per fold round. For canonical parameters, four fold rounds shrink the proof to two final scalars per side.

- **Fiat-Shamir transcript.** SHA-512 over a domain-prefixed (`BPPP-V1-`) sequence of length-prefixed appends. Every challenge is non-zero-checked. The same transcript shape is replayed by the verifier — any prover/verifier divergence on a single byte produces a different challenge and the proof is rejected.

17.3 Wire format (canonical 64-bit range)

The serialized proof is fixed-length for given $(N_d, N_p) = (16, 16)$:

WIRE LAYOUT — 544 BYTES TOTAL

```
offset  size  field
──────  ────  ──────────────────────────────
     0    32  V        — commitment to reciprocals
    32    32  CL       — left-wire blinding
    64    32  CR       — right-wire blinding
    96    32  CO       — output-wire blinding
   128    32  CS       — cross-term blinding
   160   256  WNLA fold rounds (4 × 64 B)
   416    64  WNLA final L vector (2 × 32 B)
   480    64  WNLA final N vector (2 × 32 B)
```

The byte layout has no internal length prefixes — the deserializer parses by offset. Two byte sequences of the SAME length cannot decode to the same proof because every field is bijective at its offset, so malleable-serialization attacks (A4 in the audit) are foreclosed by construction.

17.4 The DSOL syscall

Dark Contracts call the verifier through a single syscall:

DSOL — VERIFY A RANGE PROOF

```
// argv: { commitmentHex, proofHex, Nd?, Np? }
// returns: { valid: bool, soundnessMode: "real"|"soft", Nd, Np }
let result = syscall(VERIFY_RANGE_PROOF_BPPP_V1, argv_bytes);
require(result.valid, "RANGE_PROOF_INVALID");
```

The syscall charges **100,000 instruction steps** (10% of the per-call 1,000,000 budget) — heavy enough to deter contracts that loop on verification, cheap enough that ~10 verifications per call are economically reasonable. Step-cost is consensus-relevant: every indexer running the same bytecode on the same input consumes the same budget.

17.5 Soundness audit

Twelve attack-surface items (A1–A12) were enumerated in the original spec and covered by adversarial tests before activation. Highlights:

#	ATTACK	DEFENSE
A1	Small-subgroup point input	Every deserialized point routes through `assertPrimeOrder` backed by noble's `isTorsionFree()`. Tested against all 7 real ed25519 torsion points + a contamination case.
A4	Malleable serialization	Fixed-length wire format (no internal prefixes); length mismatch throws `BPPP_WIRE_LENGTH_MISMATCH`.
A5	Zero / edge-case challenge	`challenge()` loops on zero; every consumed challenge is in `[1, L)`.
A8	Non-canonical scalar (≥ L)	`scalarFromBytes32` rejects encodings whose value reduces to a non-canonical representative.
A9	Compressed-point ambiguity	noble's canonical `fromHex` rejects `y ≥ p` + off-curve encodings.
A12	WNLA fold inconsistency	Pure-function fold-count derivation from (N_d, N_p) guarantees prover/verifier agreement.

Out-of-scope items, declared explicitly: side-channel timing leaks from the prover (the verifier runs in trusted contexts; the prover's wallet is not the threat model), and quantum resistance (BP++ relies on DL hardness, broken by Shor — same posture as the rest of the MoneroUSD stack).

The complete defense matrix lives in `BPP-AUDIT.md`. Total tests: 250 (229 in `bulletproofs-plus-plus.test.js` + 14 in `bppp-integration.test.js` + 7 in `pedersen.test.js`).

17.6 What contracts get

Every Dark Contract has direct access to the verifier via `syscall(VERIFY_RANGE_PROOF_BPPP_V1, …)`, and every Pedersen-committed value the protocol emits — LP positions, custom-token balances, NFT rarity commitments — is range-bounded by a 544-byte BP++ proof. `pedersen.js::rangeProof()` emits the canonical (commitment, proof) pair as 576 bytes; consumers parse by fixed offsets, and any byte tampering produces an immediate length-mismatch reject before the cryptographic check runs.

---

18. Future Work

Several areas are under active development or consideration for future protocol upgrades:

- **Multi-node infrastructure:** The one-click desktop node and terminal installer now make it easy for anyone to run a node. The next step is deploying geographically distributed seed nodes to eliminate single-VPS dependency, improve sync latency, and strengthen network resilience through community-operated infrastructure.
- **Sovereign web hosting on the MoneroUSD blockchain:** Bringing the entire monerousd.org site, Ion Swap, the block explorer, and every protocol dashboard inside the desktop wallet as a locally hostable, censorship-proof service. Chain-anchored content addressing ensures that if any domain is taken down, every user's wallet continues to serve a byte-identical copy peer-to-peer. Adds a launchpad tile for self-hosting these sites locally.

- **Additional collateral types:** Expanding reserve assets beyond BTC and XMR. Candidates include other major proof-of-work assets. Any addition must meet strict liquidity and decentralization requirements.
- **Mobile wallet:** Native iOS and Android wallet applications that perform all cryptographic operations locally on the device, matching the desktop wallet's local-first architecture.
- **Governance exploration:** On-chain governance for protocol parameter updates (fee schedule, collateral ratios) that preserves the deterministic nature of the current design while allowing community input. This is approached cautiously — governance capture is a real risk.

---

19. Conclusion

MoneroUSD proves privacy and stability are not a tradeoff. Built on Monero's battle-tested privacy stack and extended with FCMP++, overcollateralized BTC + XMR reserves, adaptive fees, and usage-proportional emission, USDm delivers a digital dollar that respects user sovereignty without compromising on the privacy guarantees Monero users expect.

Ion Swap, Dark Contracts, and the cross-chain bridge turn that foundation into a complete private DeFi ecosystem — sealed-bid AMM, programmable smart contracts in DarkSolidity, custom `ion1_` tokens, hidden-ownership NFTs, multi-chain wrapped assets, and CPU-friendly RandomX mining. Every contract inherits privacy by construction: Pedersen commitments backed by Bulletproofs++ range proofs, commit-reveal that defeats front-running, AES-GCM-encrypted events sealed to the caller's viewkey, and tail-only inter-contract calls that make reentrancy impossible.

The protocol is non-custodial, deterministic, and open-source from the daemon to the contract layer. Settlement keys are bound to the chain, not to any operator. Fee revenue routes through a frozen three-lane split (reserves, protocol-owned liquidity, bounty) that monotonically strengthens the peg with every trade and every bridge. Tradeoffs are acknowledged honestly: standalone privacy chain (no Ethereum DeFi composition), ~20-second sealed-bid batch latency (MEV is cryptographically impossible), flat per-call resource ceilings (deterministic over auction-priced).

MoneroUSD is not a replacement for Monero — it's a complement, built by the same principles: privacy by default, keys on your machine, no surveillance, no compromises. For users who refuse to choose between financial privacy and a working DeFi stack, this is the answer.

The MoneroUSD protocol is open source. Verify everything. Trust nothing.

— Cas