Shadow Swap – MVP Build Plan

This is a **developer-detailed implementation guide** for building the MVP of Shadow Swap — a privacy-preserving DEX on Solana that uses:

- **Arcium** for encrypted order matching
- **Sanctum Gateway** (JITO-only routing) for private, MEV-free execution
- **Anchor** smart contract for order state + escrow
- **No Jupiter fallback, no multi-token complexity** in MVP — only SOL/USDC

Built by **2 full-stack engineers**, fast and focused.

🔁 MVP Workflow Summary (High-Level Flow)

graph TD

A[Trader UI: Encrypted order form] --> B[Arcium SDK: Encrypt order]

B --> C[Submit to Anchor program]

C --> D[Order stored encrypted on-chain]

D --> E[Arcium MPC cluster polls & matches]

E --> F[Threshold decryption triggers callback]

F --> G[Anchor: callback executes token transfers]

G --> H[Transaction routed via Sanctum → Jito]

H --> I[Settlement confirmed on Solana]

🗓️ 7-Day MVP Plan (No Fluff, Fast Build)

👥 Team Roles

- **Dev A**: Solana smart contract (Anchor) + callback + Gateway tx logic
- **Dev B**: Frontend + Arcium encryption flow + MPC logic + devops

🧭 Day 1: Environment, Architecture, and Arcium Boot

- Set up: Rust, Anchor CLI, Solana CLI (Devnet), Node.js, Next.js
- Clone Arcium's dark pool demo repo as starting point (modularize it)
- Setup .env (Arcium API, Jito RPC, wallet keypair)
- Register MXE and MPC cluster on Arcium Testnet via CLI:
- arc-cli init-comp-def --cluster testnet --name mvp-shadow
- arc-cli register-cluster --url <cluster_url>
- Dev A starts base Anchor program with:
  - OrderBook PDA account
  - EncryptedOrder PDA account per user order

    - Escrow token accounts for SOL and USDC
- Dev B sets up frontend skeleton (Next.js) with wallet connection

🧱 Day 2: Smart Contract + Arcium Instruction Hook

- Define order struct:
- #[derive(ArcisEncryptable)]
- pub struct Order {
- pub token: mu64,
- pub amount: mu64,
- pub side: mbool, // buy = true
- }
- submit_encrypted_order instruction:
    - Validates escrowed funds
    - Pushes encrypted order into ArcisArray in OrderBook
    - Queues Arcium computation via CPI to queue_computation
- MPC matching logic:
    - Price-time priority matching in Arcis DSL
    - Only matches opposite sides with compatible price
- Dev B compiles Arcis instruction → .arc bytecode → registers it

🔒 Day 3: Encrypted Order Submission + MPC Matching

- Frontend Arcium SDK setup:
    - Fetch MXE pubkey
    - Encrypt form input using Rescue + X25519
- const encrypted = encryptOrder({ side, token, amount }, mxePubkey);
    - Submit to Solana via Anchor client
- Dev B tests:
    - Order is stored encrypted in OrderBook
    - No plaintext on-chain
- Dev A writes Arcium callback instruction:
- #[arcium_callback(confidential_ix = "match_order")]
- pub fn match_callback(ctx, output: Vec<u8>) {...}
    - Output decoded → matched buyer/seller → token transfers executed

⚙️ Day 4: Execution Flow + Sanctum Gateway + Jito

- Dev A writes transaction builder (in callback):
    - Pull matched buyer/seller order
    - Construct atomic transaction:
        - Close orders
        - Transfer USDC and SOL between escrow PDAs
    - Serialize and pass to Sanctum Gateway SDK
- Dev A sets routing config:
- strategy: 'private_only'
- gateway.send({ tx, strategy })
- Dev A tests: Confirm transaction goes via JITO bundle → validator → on-chain

🧪 Day 5: Dev B Builds Live Frontend Flow

- Connect wallet (Phantom)
- Order form (SOL/USDC): side, amount, price
- Encrypt via Arcium SDK
- Call program instruction (via Anchor client)
- Live status updates (order submitted, matched, executed)
- WebSocket or polling: listen for confirmation event (TradeMatched)

🔁 Day 6: Callback Server + Large Payload Handling (Optional)

- Arcium handles small outputs natively
- If output > 1232 bytes:
  - Dev B writes Express.js callback server:
    - Validates MPC node signatures
    - Calls finalize_computation on-chain with output
- Deploy server to Vercel or Railway
- Secure with HMAC or Arcium node IP safelist

🚀 Day 7: Hardening, Testing, Demo Prep

- Manual test cases:
  - Buy > balance
  - Price mismatch
  - 2 identical orders → one fills, one queues
- Log MPC events: match latency, confirmation
- Record demo: order → encrypted → match → execution (Jito path)
- Add success metric logs:
  - % orders matched
  - Jito-only confirmation success

🧩 Tips & Patterns

- Use ArcisArray for encrypted orderbook (capped length)
- Use PDA as authority for all token transfers
- Store escrow token balances per user (prevent double use)
- MPC logic returns *only* minimal match info to avoid info leaks
- Transactions submitted only after threshold decrypt

🔗 Final Architecture Summary

Layer	Tech	Role
Frontend	Next.js + Arcium SDK	Encrypt orders, connect wallet
Smart Contract	Anchor	Order state, MPC hook, callback execution
MPC	Arcium MXE + Arcis DSL	Encrypted matching
Execution	Sanctum Gateway + Jito	Private delivery, no mempool leaks

🔐 Configuration

- Arcium Testnet cluster + MXE registered at start
- Sanctum Gateway strategy = private_only
- Fallbacks = disabled
- Token: only SOL/USDC (USDC mint hardcoded for MVP)

🧪 After MVP: What's Next

- ✅ Add Jupiter fallback (if no match after timeout)
- ✅ Add multi-token support (dynamic order struct)
- ✅ Improve UX (quotes, live orderbook, gas estimate)
- ✅ Add test coverage, CI/CD pipeline, audit routines