📋 Shadow Swap PRD: Atomic Swaps Between Starknet & Zcash Project Duration: Nov 15 - Nov 30, 2024 (16 days) Team Size: 3-5 developers Goal: Build first privacy-preserving atomic swap bridge between Starknet and Zcash

🎯 Executive Summary What We're Building: A decentralized exchange that enables trustless, atomic swaps between Starknet (STRK token) and Zcash (ZEC coin) with best-in-class UX powered by backend-managed Cartridge Controller and optional privacy features.

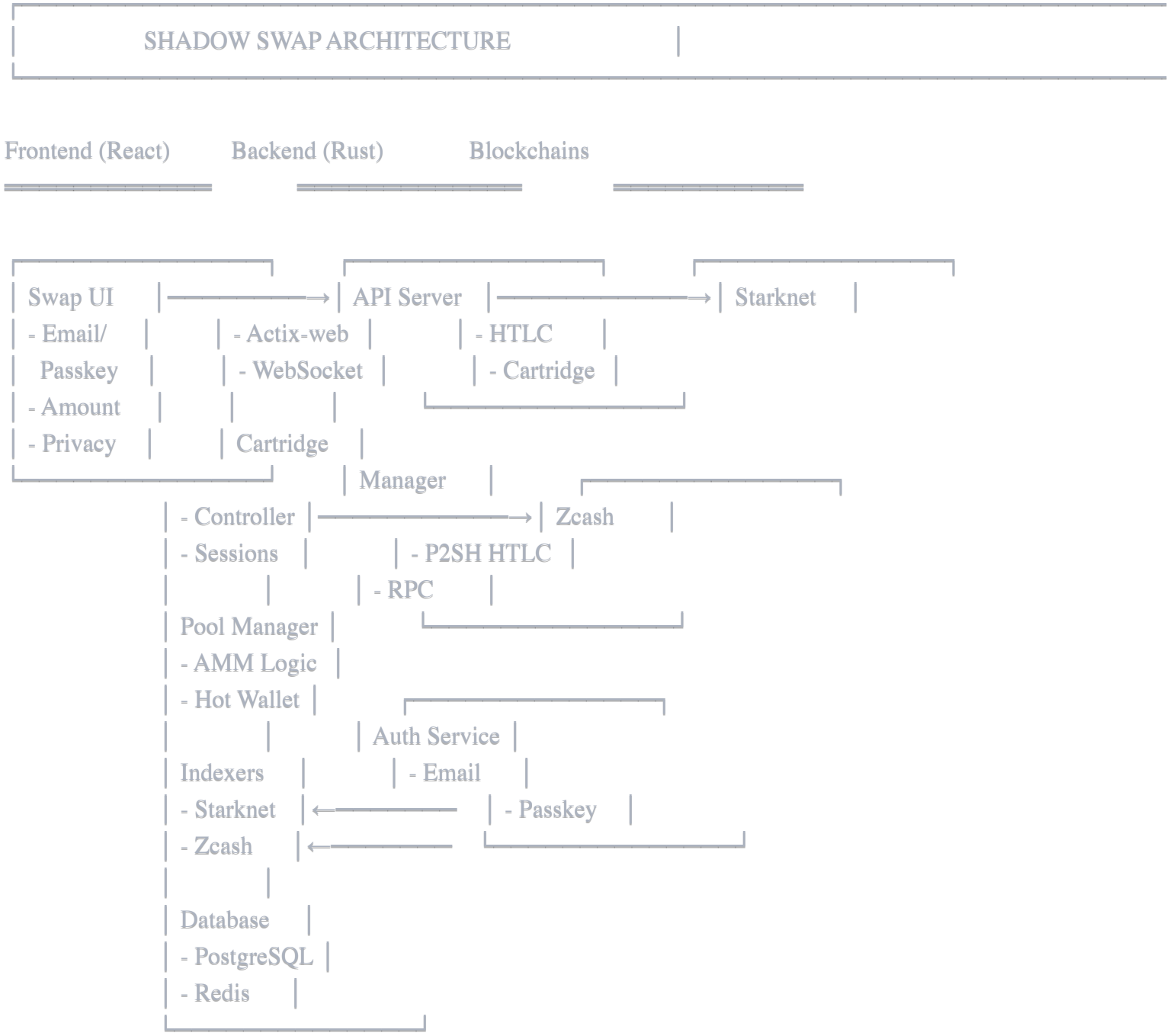
Why It Matters:

- No existing bridge between Starknet and Zcash
- Current atomic swaps require manual coordination (terrible UX)
- Backend-managed Cartridge Controller enables seamless, gasless swaps
- Privacy routing through Zcash shielded pool (unique feature)

Success Metrics: ✅ Complete 1 successful swap on testnet ✅ Demo video showing < 2 minute swap ✅ Working frontend with email/passkey authentication ✅ Privacy mode functional

---

📐 System Architecture



🔐 Authentication Flow (Backend-Managed Cartridge)

USER AUTHENTICATION & CONTROLLER CREATION

STEP 1: USER SIGN-IN (Frontend → Backend)

User chooses authentication method:
- Option A: Email (magic link)
- Option B: Passkey (WebAuthn)

Frontend → POST /api/auth/initiate

```
{
  "auth_type": "email",  // or "passkey"
  "identifier": "user@example.com"
}
```

Backend response:

```
{
  "session_token": "jwt-abc123...",
  "auth_challenge": "...",  // For passkey verification
  "user_id": "uuid-1234"
}
```

STEP 2: CONTROLLER CREATION (Backend)

On first authentication, backend automatically:

1. Creates Cartridge Controller for user:

```rust
let controller = cartridge_sdk::Controller::create(
    user_id,
    policies: SessionPolicies {
        contracts: vec![HTLC_CONTRACT],
        max_fee: "1000000000000000",  // 0.001 STRK
        expiry: "24h"
    }
).await?;
```

2. Stores controller credentials in database:

```sql
INSERT INTO user_controllers (
```

```
    user_id,
    controller_address,
    session_key_encrypted,
    policies,
    created_at
  ) VALUES (...);
```

3. Returns controller info to frontend:
```json
  {
    "controller_address": "0x123...",
    "session_active": true,
    "policies": {...}
  }
```

STEP 3: SWAP EXECUTION (Backend-Signed)

When user initiates swap:

Frontend → POST /api/swap/execute
  {
    "amount": 100,
    "destination_zcash": "z1abc..."
  }
  Headers: { Authorization: "Bearer jwt-abc123..." }

Backend (using stored session):
  1. Validates user session
  2. Retrieves user's controller
  3. Signs HTLC deployment transaction
  4. Broadcasts to Starknet
  5. Returns swap details

NO WALLET POPUPS! Backend handles all signing.

RESULT:
User: Seamless experience - just email/passkey, no wallet management
Backend: Full control over Cartridge sessions and transaction signing

---

🔢 Data Flow: Complete Swap Journey

ATOMIC SWAP FLOW (30-60 seconds)

STEP 1: USER INITIATES (Frontend → Backend)

User input:
- Amount: 100 STRK
- Destination: z1abc...xyz (Zcash address)
- Privacy: OFF

Frontend → POST /api/swap/initiate
Headers: { Authorization: "Bearer jwt-token..." }

```
{
  "strk_amount": 100,
  "user_zcash": "z1abc...",
  "privacy_mode": false
}
```

Backend response:

```
{
  "swap_id": "uuid-1234",
  "secret_hash_sha256": "0xabc...",
  "secret_hash_poseidon": "0xdef...",
  "zec_output": 0.95,
  "timelock_starknet": "2024-11-20T12:00:00Z",
  "timelock_zcash": "2024-11-19T12:00:00Z",
  "controller_address": "0x789..."
}
```

STEP 2: BACKEND LOCKS STRK (Backend → Starknet)

Backend (automatically, no user interaction):
1. Retrieves user's controller session
2. Signs HTLC deployment transaction
3. Deploys HTLC contract with user's 100 STRK

Starknet HTLC Contract:

```
constructor(
  participant: POOL_ADDRESS,
  secret_hash_poseidon: 0xdef...,
  secret_hash_sha256: 0xabc...,
```

timelock: 48h,
amount: 100 STRK
)

WebSocket → Frontend: "Starknet lock initiated ⏳ "

STEP 3: INDEXER DETECTS LOCK (Starknet → Backend)

Starknet Indexer (polling every 6 seconds):
✅ Detected HTLCInitiated event
✅ Verified secret_hash matches swap_id
✅ Updated database: swap.state = "starknet_locked"
✅ WebSocket → Frontend: "Starknet lock confirmed ✅ "

STEP 4: POOL LOCKS ZEC (Backend → Zcash)

Pool Manager (automated):
1. Calculate ZEC output: 0.95 ZEC
2. Create P2SH HTLC script on Zcash:
   - Recipient: user_zcash (z1abc...)
   - Hash: 0xabc... (SHA256)
   - Timelock: 24h
   - Amount: 0.95 ZEC
3. Sign with pool hot wallet
4. Broadcast to Zcash network

WebSocket → Frontend: "ZEC locked for you! ✅ "

STEP 5: USER CLAIMS ZEC (Frontend → Backend → Zcash)

Frontend:
- User clicks "Claim ZEC"

Frontend → POST /api/swap/{swap_id}/claim

Backend:
1. Retrieves secret for swap
2. Creates Zcash redemption tx

3. Signs with secret
4. Broadcasts to Zcash network

Zcash blockchain:
✅ Secret revealed: 0x123secret456
✅ User receives 0.95 ZEC

WebSocket → Frontend: "ZEC claimed! Check your wallet 🎉"

STEP 6: POOL CLAIMS STRK (Zcash → Starknet)

Zcash Indexer (monitoring):
✅ Detected secret reveal in claim tx
✅ Extracted secret: 0x123secret456
✅ Notifies Pool Manager

Pool Manager:
1. Use secret to claim 100 STRK from HTLC
2. Updates reserves: +100 STRK, -0.95 ZEC
3. Database: swap.state = "completed"
4. WebSocket → Frontend: "Swap completed! 🎉"

RESULT:
User: Traded 100 STRK → Received 0.95 ZEC (in 45 seconds)
NO WALLET POPUPS - seamless backend signing
Pool: Received 100 STRK → Paid 0.95 ZEC (earned 0.3% fee)

🗃️ Database Schema

sql

```sql
-- ============================================================
-- USER ACCOUNTS (Authentication & Controllers)
-- ============================================================
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),

    -- Authentication
    email VARCHAR(255) UNIQUE,
    passkey_credential_id VARCHAR(512),  -- WebAuthn credential
    passkey_public_key TEXT,             -- For passkey verification

    -- Cartridge Controller (managed by backend)
    controller_address VARCHAR(66) UNIQUE NOT NULL,
    session_key_encrypted TEXT NOT NULL,  -- Encrypted session private key
    session_policies JSONB NOT NULL,
    session_expires_at TIMESTAMPTZ NOT NULL,

    -- Metadata
    created_at TIMESTAMPTZ DEFAULT NOW(),
    last_login_at TIMESTAMPTZ,
    total_swaps INTEGER DEFAULT 0,

    CONSTRAINT valid_auth CHECK (
        email IS NOT NULL OR passkey_credential_id IS NOT NULL
    )
);

CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_controller ON users(controller_address);

-- ============================================================
-- AUTH SESSIONS (JWT tokens)
-- ============================================================
CREATE TABLE auth_sessions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    token_hash VARCHAR(64) NOT NULL UNIQUE,  -- SHA256 of JWT
    ip_address VARCHAR(45),
    user_agent TEXT,
    expires_at TIMESTAMPTZ NOT NULL,
    created_at TIMESTAMPTZ DEFAULT NOW(),
    last_used_at TIMESTAMPTZ DEFAULT NOW()
);
```

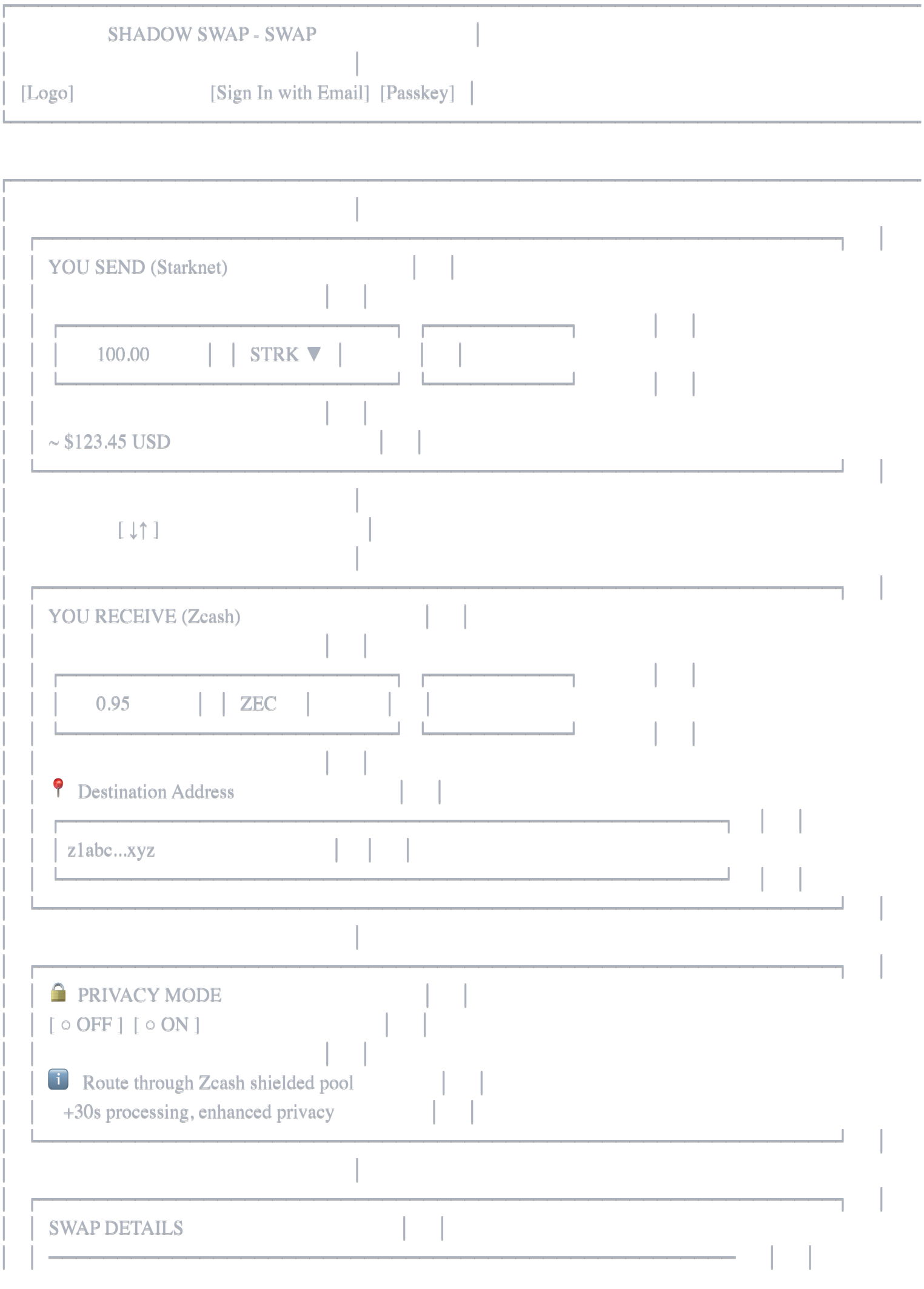
```sql
CREATE INDEX idx_sessions_user ON auth_sessions(user_id);
CREATE INDEX idx_sessions_token ON auth_sessions(token_hash);
CREATE INDEX idx_sessions_expiry ON auth_sessions(expires_at);

-- ============================================================
-- SWAPS TABLE (Core swap state tracking)
-- ============================================================
CREATE TABLE swaps (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),

    -- User reference
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,

    -- Participants
    user_starknet_address VARCHAR(66) NOT NULL,  -- Controller address
    user_zcash_address VARCHAR(95) NOT NULL,

    -- Amounts
    strk_amount BIGINT NOT NULL,            -- Amount in wei (10^18)
    zec_amount BIGINT NOT NULL,             -- Amount in zatoshis (10^8)
    fee_amount BIGINT NOT NULL DEFAULT 0,    -- Protocol fee

    -- Hashes (dual hash for cross-chain compatibility)
    secret_hash_sha256 VARCHAR(64) NOT NULL,
    secret_hash_poseidon VARCHAR(66) NOT NULL,
    secret_revealed VARCHAR(64),            -- Populated when claimed

    -- HTLC addresses/txids
    starknet_htlc_address VARCHAR(66),
    starknet_lock_tx VARCHAR(66),
    zcash_htlc_txid VARCHAR(64),
    zcash_claim_txid VARCHAR(64),
    starknet_claim_tx VARCHAR(66),

    -- Timelocks
    timelock_starknet TIMESTAMPTZ NOT NULL,  -- 48 hours
    timelock_zcash TIMESTAMPTZ NOT NULL,     -- 24 hours

    -- State machine
    state VARCHAR(20) NOT NULL DEFAULT 'initiated',
    -- States: initiated, starknet_locked, zcash_locked,
    --         claiming, completed, refunded, failed

    -- Privacy feature
```

```
    privacy_mode BOOLEAN DEFAULT FALSE,
    shielded_hops INTEGER DEFAULT 0,

    -- Metadata
    created_at TIMESTAMPTZ DEFAULT NOW(),
    starknet_locked_at TIMESTAMPTZ,
    zcash_locked_at TIMESTAMPTZ,
    completed_at TIMESTAMPTZ,

    -- Indexes for fast lookups
    CONSTRAINT unique_secret_hash UNIQUE(secret_hash_sha256)
);

CREATE INDEX idx_swaps_state ON swaps(state);
CREATE INDEX idx_swaps_user ON swaps(user_id);
CREATE INDEX idx_swaps_controller ON swaps(user_starknet_address);
CREATE INDEX idx_swaps_created ON swaps(created_at DESC);

-- ============================================================
-- SWAP EVENTS (Audit trail)
-- ============================================================
CREATE TABLE swap_events (
    id SERIAL PRIMARY KEY,
    swap_id UUID REFERENCES swaps(id) ON DELETE CASCADE,
    event_type VARCHAR(50) NOT NULL,
    -- Types: initiated, starknet_locked, zcash_locked,
    --        secret_revealed, claimed, refunded
    blockchain VARCHAR(20),              -- 'starknet' or 'zcash'
    tx_hash VARCHAR(66),
    block_number BIGINT,
    data JSONB,                          -- Additional event data
    created_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_events_swap ON swap_events(swap_id);
CREATE INDEX idx_events_type ON swap_events(event_type);

-- ============================================================
-- POOL STATE (AMM reserves)
-- ============================================================
CREATE TABLE pool_state (
    id SERIAL PRIMARY KEY,
    starknet_reserve BIGINT NOT NULL,    -- STRK in pool
    zcash_reserve BIGINT NOT NULL,       -- ZEC in pool
```

```
    total_swaps INTEGER DEFAULT 0,
    total_volume_strk BIGINT DEFAULT 0,
    total_volume_zec BIGINT DEFAULT 0,
    total_fees_collected BIGINT DEFAULT 0,
    last_updated TIMESTAMPTZ DEFAULT NOW()
);

-- Initialize pool
INSERT INTO pool_state (starknet_reserve, zcash_reserve)
VALUES (10000000000000000000, 1000000000);  -- 1000 STRK, 10 ZEC

-- ============================================================
-- INDEXER STATE (Track last synced block)
-- ============================================================
CREATE TABLE indexer_state (
    chain VARCHAR(20) PRIMARY KEY,        -- 'starknet' or 'zcash'
    last_block_number BIGINT NOT NULL,
    last_synced_at TIMESTAMPTZ DEFAULT NOW()
);

INSERT INTO indexer_state (chain, last_block_number) VALUES
('starknet', 0),
('zcash', 0);
```

🎨 Frontend Design Specification

Page: Swap Interface


SHADOW SWAP - SWAP
[Logo]
[Sign In with Email]
[Passkey]
YOU SEND (Starknet)
100.00
STRK ▼
~ $123.45 USD
[↓↑]
YOU RECEIVE (Zcash)
0.95
ZEC
Destination Address
z1abc...xyz
PRIVACY MODE
[○ OFF] [○ ON]
Route through Zcash shielded pool
+30s processing, enhanced privacy
SWAP DETAILS

Component Breakdown:

1. Header Component

tsx

```tsx
// src/components/Header.tsx

<Header>
  <Logo src="/shadowswap-logo.svg" />
  <Nav>
    <NavLink to="/swap">Swap</NavLink>
    <NavLink to="/history">History</NavLink>
    <NavLink to="/docs">Docs</NavLink>
  </Nav>
  <AuthButtons>
    {!isAuthenticated ? (
      <>
        <SignInButton onClick={handleEmailSignIn}>
          <MailIcon />
          Sign In with Email
        </SignInButton>
        <SignInButton onClick={handlePasskeySignIn}>
          <FingerprintIcon />
          Passkey
        </SignInButton>
      </>
    ) : (
      <UserProfile>
        <Avatar />
        <UserEmail>{user.email}</UserEmail>
        <ControllerBadge>
          Controller: {formatAddress(user.controllerAddress)}
        </ControllerBadge>
      </UserProfile>
    )}
  </AuthButtons>
</Header>
```

Styling:

- Height: 72px
- Background: Linear gradient (`#1a1a2e` → `#16213e`)
- Logo: 40px height, glow effect
- Font: Inter, 16px, white
- Auth buttons: Side-by-side, 120px width each

2. Swap Card Component

tsx

```
// src/components/SwapCard.tsx

<SwapCard>
  <InputSection type="send">
    <Label>YOU SEND (Starknet)</Label>
    <AmountInput
      value={amount}
      onChange={handleAmountChange}
      token="STRK"
    />
    <UsdValue>~ ${usdValue} USD</UsdValue>
  </InputSection>

  <SwapDirection>
    <IconButton onClick={handleFlip}>
      <ArrowUpDown />
    </IconButton>
  </SwapDirection>

  <InputSection type="receive">
    <Label>YOU RECEIVE (Zcash)</Label>
    <AmountDisplay value={outputAmount} token="ZEC" />
    <AddressInput
      placeholder="z1abc...xyz"
      value={zcashAddress}
      onChange={handleAddressChange}
    />
  </InputSection>

  <PrivacyToggle>
    <Label>🔒 PRIVACY MODE</Label>
    <Toggle
      checked={privacyMode}
      onChange={setPrivacyMode}
    />
    <InfoText>
      Route through Zcash shielded pool
      +30s processing, enhanced privacy
    </InfoText>
  </PrivacyToggle>

  <SwapDetails>
    <DetailRow>
      <span>Rate:</span>
```

```
      <span>1 STRK = {rate} ZEC</span>
    </DetailRow>
    <DetailRow>
      <span>Fee (0.3%):</span>
      <span>{fee} STRK</span>
    </DetailRow>
    <DetailRow highlight>
      <span>You receive:</span>
      <span>{outputAmount} ZEC</span>
    </DetailRow>
    <DetailRow>
      <span>Time:</span>
      <span>~{estimatedTime}s</span>
    </DetailRow>
  </SwapDetails>

  <SwapButton
    onClick={handleSwap}
    disabled={!canSwap}
    loading={isSwapping}
  >
    {buttonText}
  </SwapButton>
</SwapCard>
```

Styling:

- Width: 480px max
- Padding: 32px
- Border-radius: 24px
- Background: rgba(255, 255, 255, 0.05)
- Backdrop-filter: blur(10px)
- Border: 1px solid rgba(255, 255, 255, 0.1)

Colors:

- Primary: `#6366f1` (indigo)
- Secondary: `#8b5cf6` (purple)
- Success: `#10b981` (green)
- Background: `#0f172a` (dark blue)
- Text primary: `#f1f5f9` (white)
- Text secondary: `#94a3b8` (gray)

Fonts:

- Headings: Inter Bold, 18px
- Body: Inter Regular, 14px
- Numbers: IBM Plex Mono, 20px

3. Authentication Modal Component

tsx

```tsx
// src/components/AuthModal.tsx

<Modal open={showAuthModal}>
  {authType === 'email' ? (
    <EmailAuth>
      <Title>Sign in with Email</Title>
      <EmailInput
        type="email"
        placeholder="your@email.com"
        value={email}
        onChange={setEmail}
      />
      <SendButton onClick={handleSendMagicLink}>
        Send Magic Link
      </SendButton>
      <InfoText>
        We'll email you a secure sign-in link
      </InfoText>
    </EmailAuth>
  ) : (
    <PasskeyAuth>
      <Title>Sign in with Passkey</Title>
      <FingerprintIcon size={64} />
      <AuthButton onClick={handlePasskeyAuth}>
        Authenticate
      </AuthButton>
      <InfoText>
        Use your device's biometric authentication
      </InfoText>
    </PasskeyAuth>
  )}
</Modal>
```

4. Progress Modal Component

tsx

```
// src/components/ProgressModal.tsx

<Modal open={showProgress}>
  <ProgressTracker>
    <Step
      status={step >= 1 ? 'complete' : 'pending'}
      icon={<Lock />}
    >
      <Title>Locking STRK</Title>
      <Description>
        Backend deploying HTLC on Starknet...
      </Description>
      {step === 1 && <Spinner />}
      {step > 1 && <CheckCircle />}
    </Step>

    <Connector active={step >= 2} />

    <Step
      status={step >= 2 ? 'complete' : 'pending'}
      icon={<Zap />}
    >
      <Title>Pool Locking ZEC</Title>
      <Description>
        Creating Zcash HTLC...
      </Description>
      {step === 2 && <Spinner />}
      {step > 2 && <CheckCircle />}
    </Step>

    <Connector active={step >= 3} />

    <Step
      status={step >= 3 ? 'complete' : 'pending'}
      icon={<Gift />}
    >
      <Title>Claiming ZEC</Title>
      <Description>
        Revealing secret and claiming funds...
      </Description>
      {step === 3 && <Spinner />}
      {step > 3 && <CheckCircle />}
    </Step>
```

```
    <Connector active={step >= 4} />

    <Step
      status={step >= 4 ? 'complete' : 'pending'}
      icon={<PartyPopper />}
    >
      <Title>Swap Complete!</Title>
      <Description>
        You received {outputAmount} ZEC
      </Description>
      {step === 4 && <Confetti />}
    </Step>
  </ProgressTracker>

  <Actions>
    <Button onClick={viewOnExplorer}>
      View Transaction
    </Button>
    <Button onClick={closeModal}>
      Close
    </Button>
  </Actions>
</Modal>
```

Styling:

- Width: 600px
- Steps: Vertical timeline
- Active step: Pulsing glow animation
- Complete step: Green checkmark
- Pending step: Gray, dimmed

Responsive Design:

Desktop (> 1024px):

- Swap card: Centered, 480px width
- Two column layout available for stats

Tablet (768px - 1024px):

- Swap card: Full width, max 600px
- Single column layout

Mobile (< 768px):

- Swap card: Full width, 16px padding
- Font sizes reduced 10%
- Buttons full width

Animation Specifications:

Page Load:

css

```
.swap-card {
  animation: fadeInUp 0.6s ease-out;
}

@keyframes fadeInUp {
  from {
    opacity: 0;
    transform: translateY(20px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}
```

Button Hover:

css

```
.swap-button:hover {
  transform: translateY(-2px);
  box-shadow: 0 12px 32px rgba(99, 102, 241, 0.5);
  transition: all 0.3s ease;
}
```

Loading State:

css

```css
.swap-button.loading {
  background: linear-gradient(
    90deg,
    #6366f1 0%,
    #8b5cf6 50%,
    #6366f1 100%
  );
  background-size: 200% 100%;
  animation: shimmer 2s infinite;
}

@keyframes shimmer {
  0% { background-position: -200% 0; }
  100% { background-position: 200% 0; }
}
```

Success Confetti:

- Library: `react-confetti`
- Duration: 3 seconds
- Colors: `['#6366f1', '#8b5cf6', '#10b981', '#f59e0b']`

👨‍💻 Team Roles & Tasks

Role 1: Smart Contract Developer (Cairo + Zcash)

Task 1.1: Starknet HTLC Contract

File: `contracts/starknet/src/htlc.cairo`

Requirements:


cairo

```
#[starknet::contract]
mod AtomicSwapHTLC {
    // Storage
    - initiator: ContractAddress
    - participant: ContractAddress
    - amount: u256
    - secret_hash_poseidon: felt252
    - secret_hash_sha256: u256
    - timelock: u64
    - state: enum { Initiated, Redeemed, Refunded }

    // Functions
    - constructor() - Initialize HTLC with params
    - redeem(secret: felt252) - Claim funds with secret
    - refund() - Refund after timelock expires
    - get_state() - View current state
}
```

Acceptance Criteria: ✅ Compiles without errors ✅ Passes all unit tests (10+ test cases) ✅ Deploys to Sepolia testnet ✅ Verified on Starkscan ✅ Gas optimized (< 50k gas per function)

Testing Requirements:


bash

```
# Must pass all tests
scarb test

# Test cases:
1. Deploy HTLC successfully
2. Redeem with correct secret
3. Reject redemption with wrong secret
4. Reject redemption by non-participant
5. Refund after timelock expires
6. Reject refund before timelock
7. Reject refund by non-initiator
8. Prevent double redemption
9. Prevent double refund
10. Verify state transitions
```

Deliverables:

- `htlc.cairo` - Contract code

- `lib.cairo` - Module exports
- `test_htlc.cairo` - Test suite
- `Scarb.toml` - Dependencies
- `DEPLOYMENT.md` - Deployment guide
- Contract address on Sepolia

Estimated Time: 3-4 days

Task 1.2: Zcash HTLC Script

File: `contracts/zcash/htlc_script.py`

Requirements:

python

```
def create_htlc_script(
    secret_hash: bytes,          # SHA256 hash
    recipient_pubkey: bytes,     # Alice's pubkey
    refund_pubkey: bytes,        # Pool's pubkey
    timelock: int                # Unix timestamp
) -> CScript:
    """
    P2SH script:
    IF
        <secret> SHA256 <hash> EQUALVERIFY
        <recipient_pubkey> CHECKSIG
    ELSE
        <timelock> CHECKLOCKTIMEVERIFY DROP
        <refund_pubkey> CHECKSIG
    ENDIF
    """

def create_funding_tx(...) -> bytes:
    """Send ZEC to P2SH address"""

def create_redeem_tx(
    secret: bytes,
    recipient_sig: bytes
) -> bytes:
    """Claim ZEC by revealing secret"""

def create_refund_tx(...) -> bytes:
    """Refund ZEC after timelock"""
```

Acceptance Criteria: ✅ Script validates on Zcash testnet ✅ Funding transaction succeeds ✅ Redemption with correct secret succeeds ✅ Redemption with wrong secret fails ✅ Refund before timelock fails ✅ Refund after timelock succeeds

Testing Requirements:


bash

```
# Run on Zcash testnet
python test_htlc_script.py

# Test cases:
1. Create valid P2SH script
2. Fund P2SH address
3. Redeem with valid secret
4. Reject invalid secret
5. Reject early refund
6. Allow refund after timelock
```

Deliverables:

- `htlc_script.py` - Script generator
- `transaction_builder.py` - TX helpers
- `test_htlc_script.py` - Test suite
- `README.md` - Usage guide
- Example transactions on testnet

Estimated Time: 3-4 days

Role 2: Backend Developer (Rust)

Task 2.1: Authentication & Controller Management

Files:

- `backend/src/auth/mod.rs`
- `backend/src/cartridge/controller_manager.rs`

Requirements:


rust

```rust
// Auth Service
pub struct AuthService {
    db: PgPool,
    jwt_secret: String,
    email_service: EmailService,
}

impl AuthService {
    /// Initiate email authentication
    pub async fn initiate_email_auth(
        &self,
        email: String,
    ) -> Result<AuthChallenge, AuthError> {
        // 1. Generate magic link token
        // 2. Send email with link
        // 3. Return challenge ID
    }

    /// Verify magic link and create session
    pub async fn verify_magic_link(
        &self,
        token: String,
    ) -> Result<AuthResponse, AuthError> {
        // 1. Validate token
        // 2. Get or create user
        // 3. Create Cartridge controller if new user
        // 4. Generate JWT
        // 5. Return session
    }

    /// Initiate passkey authentication
    pub async fn initiate_passkey_auth(
        &self,
    ) -> Result<PasskeyChallenge, AuthError> {
        // Return WebAuthn challenge
    }

    /// Verify passkey and create session
    pub async fn verify_passkey(
        &self,
        credential: PasskeyCredential,
    ) -> Result<AuthResponse, AuthError> {
        // 1. Validate credential
        // 2. Get or create user
```

```
        // 3. Create Cartridge controller if new user
        // 4. Generate JWT
        // 5. Return session
    }
}

// Cartridge Controller Manager
pub struct CartridgeManager {
    cartridge_sdk: CartridgeSDK,
    db: PgPool,
    encryption_key: Vec<u8>,
}

impl CartridgeManager {
    /// Create new controller for user
    pub async fn create_controller(
        &self,
        user_id: Uuid,
    ) -> Result<ControllerInfo, CartridgeError> {
        // 1. Generate controller via Cartridge SDK
        // 2. Create session with policies
        // 3. Encrypt session key
        // 4. Store in database
        // 5. Return controller address
    }

    /// Get controller for user
    pub async fn get_controller(
        &self,
        user_id: Uuid,
    ) -> Result<Controller, CartridgeError> {
        // 1. Fetch from database
        // 2. Decrypt session key
        // 3. Validate session not expired
        // 4. Return controller instance
    }

    /// Sign transaction with user's controller
    pub async fn sign_transaction(
        &self,
        user_id: Uuid,
        tx: Transaction,
    ) -> Result<SignedTransaction, CartridgeError> {
        // 1. Get controller
```

```
        // 2. Validate against session policies
        // 3. Sign transaction
        // 4. Return signed tx
    }

    /// Refresh session if needed
    pub async fn refresh_session(
        &self,
        user_id: Uuid,
    ) -> Result<(), CartridgeError> {
        // Extend session if close to expiry
    }
}
```

Acceptance Criteria: ✅ Email magic links work ✅ Passkey authentication functional ✅ Controllers created automatically for new users ✅ Session keys encrypted at rest ✅ JWT tokens secure and validated ✅ Controller session policies enforced

Security Requirements:

- Session keys encrypted with AES-256-GCM
- JWT tokens signed with RS256
- Magic link tokens expire after 15 minutes
- Rate limiting on auth endpoints
- Secure passkey implementation (WebAuthn)

Deliverables:

- `auth/mod.rs` - Auth service
- `auth/email.rs` - Email magic links
- `auth/passkey.rs` - Passkey auth
- `cartridge/controller_manager.rs` - Controller management
- `tests/auth_tests.rs` - Auth test suite
- `SECURITY.md` - Security documentation

Estimated Time: 4-5 days

Task 2.2: API Server Setup

File: `backend/src/main.rs`

Requirements:

rust

```
// Endpoints to implement:

// Authentication
POST /api/auth/email/initiate
  - Input: { email }
  - Output: { challenge_id }

POST /api/auth/email/verify
  - Input: { token }
  - Output: { jwt, user, controller_address }

POST /api/auth/passkey/initiate
  - Output: { challenge }

POST /api/auth/passkey/verify
  - Input: { credential }
  - Output: { jwt, user, controller_address }

GET /api/auth/me
  - Headers: { Authorization: Bearer JWT }
  - Output: { user, controller_address, session_expiry }

// Swaps
POST /api/swap/initiate
  - Headers: { Authorization: Bearer JWT }
  - Input: { strk_amount, user_zcash, privacy_mode }
  - Output: { swap_id, secret_hashes, timelocks, zec_output }

GET /api/swap/{swap_id}/status
  - Output: { state, starknet_locked, zcash_locked, ... }

POST /api/swap/{swap_id}/claim
  - Headers: { Authorization: Bearer JWT }
  - Output: { success, tx_hash }

GET /api/user/swaps
  - Headers: { Authorization: Bearer JWT }
  - Output: { swaps: [...] }

// Pool
GET /api/pool/info
  - Output: { starknet_reserve, zcash_reserve, rate, tvl }

GET /api/rates
```

```
  - Output: { strk_zec_rate, usd_prices }

// WebSocket
WS /ws/swap/{swap_id}
  - Real-time updates for swap progress
```

Acceptance Criteria: ✅ Server runs on port 8080 ✅ All endpoints return correct responses ✅ CORS configured for frontend ✅ WebSocket connections stable ✅ Error handling with proper HTTP codes ✅ Request logging with tracing ✅ JWT middleware validates tokens ✅ Rate limiting on sensitive endpoints

Deliverables:

- `main.rs` - Server entry point
- `api/routes.rs` - Route definitions
- `api/handlers/` - Request handlers
- `middleware/auth.rs` - JWT validation
- `.env.example` - Environment template
- `README.md` - Setup instructions

Estimated Time: 3-4 days

Task 2.3: Hash Bridge Implementation

File: `backend/src/crypto/hash_bridge.rs`

Requirements:

rust

```rust
pub struct HashBridge;

impl HashBridge {
    /// Generate random 32-byte secret
    pub fn generate_secret() -> [u8; 32];

    /// Hash for Zcash (SHA256)
    pub fn hash_for_zcash(secret: &[u8; 32]) -> [u8; 32];

    /// Hash for Starknet (Poseidon via felt252)
    pub fn hash_for_starknet(secret: &[u8; 32]) -> String;

    /// Verify secret matches both hashes
    pub fn verify_dual_hash(
        secret: &[u8; 32],
        zcash_hash: &[u8; 32],
        starknet_hash: &str,
    ) -> bool;
}
```

Critical Requirements:

- SHA256 must match Zcash's implementation exactly
- Poseidon hash must match Cairo's `poseidon_hash_span`
- Secret generation must be cryptographically secure
- All functions must be deterministic (no randomness in hashing)

Acceptance Criteria: ✅ SHA256 hash verified against test vectors ✅ Poseidon hash matches Cairo output ✅ Secret generation passes randomness tests ✅ Dual verification works correctly ✅ 100% test coverage

Testing Requirements:


rust

```rust
#[test]
fn test_sha256_compatibility() {
    let secret = hex::decode("123...").unwrap();
    let hash = HashBridge::hash_for_zcash(&secret);
    // Verify against known Zcash hash
}

#[test]
fn test_poseidon_compatibility() {
    let secret = [1u8; 32];
    let hash = HashBridge::hash_for_starknet(&secret);
    // Verify against Cairo contract output
}

#[test]
fn test_dual_hash_verification() {
    let secret = HashBridge::generate_secret();
    let sha = HashBridge::hash_for_zcash(&secret);
    let poseidon = HashBridge::hash_for_starknet(&secret);
    assert!(HashBridge::verify_dual_hash(&secret, &sha, &poseidon));
}
```

Deliverables:

- `hash_bridge.rs` - Core implementation
- `tests/hash_bridge_tests.rs` - Test suite
- `docs/HASH_COMPATIBILITY.md` - Technical docs

Estimated Time: 2-3 days

Task 2.4: AMM Pool Manager

File: `backend/src/amm/pool_manager.rs`

Requirements:


rust

```rust
pub struct PoolManager {
    starknet_client: StarknetClient,
    zcash_client: ZcashClient,
    cartridge_manager: CartridgeManager,
    pool_controller_user_id: Uuid,  // Pool's user ID for controller access
    db: PgPool,
}

impl PoolManager {
    /// Calculate output amount using constant product formula
    pub fn calculate_output(
        &self,
        input_amount: u64,
        input_reserve: u64,
        output_reserve: u64,
    ) -> u64 {
        // (x + dx) * (y - dy) = x * y
        // Take 0.3% fee
    }

    /// Execute pool-side swap
    pub async fn execute_pool_swap(
        &self,
        swap_id: Uuid,
    ) -> Result<(), PoolError> {
        // 1. Get swap details from DB
        // 2. Calculate ZEC output
        // 3. Lock ZEC in HTLC for user
        // 4. Wait for user to claim (reveal secret)
        // 5. Use secret to claim user's STRK via controller
        // 6. Update reserves
    }

    /// Claim STRK from user's HTLC using revealed secret
    async fn claim_strk_from_htlc(
        &self,
        swap_id: Uuid,
        secret: [u8; 32],
    ) -> Result<String, PoolError> {
        // 1. Get user's HTLC address
        // 2. Use pool's controller to call redeem()
        // 3. Return transaction hash
    }
```

```rust
    /// Get current pool state
    pub async fn get_pool_info(&self) -> PoolInfo {
        // Return reserves, rate, TVL, etc.
    }
}
```

Acceptance Criteria: ✅ Constant product formula correct ✅ 0.3% fee calculated properly ✅ Pool locks ZEC automatically when user locks STRK ✅ Pool claims STRK after secret revealed using controller ✅ Reserves updated correctly ✅ Rate calculation accurate

Security Requirements:

- Controller credentials loaded from secure storage
- Never log sensitive data
- Use testnet for development
- Implement rate limiting (prevent drain attacks)

Deliverables:

- `pool_manager.rs` - Core logic
- `amm.rs` - AMM formulas
- `tests/pool_tests.rs` - Test suite
- `SECURITY.md` - Security considerations

Estimated Time: 3-4 days

Task 2.5: Blockchain Indexers

Files:

- `backend/src/monitoring/starknet_indexer.rs`
- `backend/src/monitoring/zcash_indexer.rs`

Requirements:

Starknet Indexer:


rust

```rust
pub struct StarknetIndexer {
    rpc_client: StarknetClient,
    db: PgPool,
    websocket: WebSocketServer,
}

impl StarknetIndexer {
    /// Poll Starknet every 6 seconds
    pub async fn start_monitoring(&self) {
        loop {
            sleep(Duration::from_secs(6)).await;
            self.check_for_htlc_events().await?;
        }
    }

    async fn check_for_htlc_events(&self) -> Result<()> {
        // 1. Get latest block
        // 2. Query events from HTLC contracts
        // 3. Parse HTLCInitiated, HTLCRedeemed, HTLCRefunded
        // 4. Update database
        // 5. Notify WebSocket clients
    }
}
```

Zcash Indexer:

rust

```rust
pub struct ZcashIndexer {
    rpc_client: ZcashRpcClient,
    db: PgPool,
}

impl ZcashIndexer {
    /// Poll Zcash mempool every 75 seconds
    pub async fn start_monitoring(&self) {
        loop {
            sleep(Duration::from_secs(75)).await;
            self.check_for_htlc_transactions().await?;
        }
    }

    async fn check_for_htlc_transactions(&self) -> Result<()> {
        // 1. Get pending swaps from DB
        // 2. Check if P2SH addresses have been funded
        // 3. Monitor for secret reveals in claim txs
        // 4. Update database
        // 5. Trigger pool claims when secret revealed
    }

    fn extract_secret_from_tx(&self, tx: &Transaction) -> Option<[u8; 32]> {
        // Parse witness data to extract secret
    }
}
```

Acceptance Criteria: ✅ Starknet indexer detects HTLCs within 12 seconds ✅ Zcash indexer detects funding within 2.5 minutes ✅ Secret extraction from Zcash tx works correctly ✅ Database updates are atomic ✅ WebSocket notifications sent immediately ✅ Handles RPC failures gracefully (retry logic)

Deliverables:

- `starknet_indexer.rs` - Starknet monitoring
- `zcash_indexer.rs` - Zcash monitoring
- `tests/indexer_tests.rs` - Test suite
- `MONITORING.md` - Architecture docs

Estimated Time: 4-5 days

Role 3: Frontend Developer (React + TypeScript)

Task 3.1: Authentication UI

Files:

- `frontend/src/components/auth/AuthModal.tsx`
- `frontend/src/hooks/useAuth.ts`

Requirements:

typescript

```
// useAuth Hook
export function useAuth() {
  const [isAuthenticated, setIsAuthenticated] = useState(false);
  const [user, setUser] = useState<User | null>(null);
  const [loading, setLoading] = useState(true);

  // Initialize from localStorage JWT
  useEffect(() => {
    const token = localStorage.getItem('jwt');
    if (token) {
      validateAndSetUser(token);
    } else {
      setLoading(false);
    }
  }, []);

  // Email authentication
  const signInWithEmail = async (email: string) => {
    const response = await fetch('/api/auth/email/initiate', {
      method: 'POST',
      body: JSON.stringify({ email }),
    });
    // Show "check your email" message
  };

  const verifyEmailToken = async (token: string) => {
    const response = await fetch('/api/auth/email/verify', {
      method: 'POST',
      body: JSON.stringify({ token }),
    });
    const data = await response.json();

    // Store JWT
    localStorage.setItem('jwt', data.jwt);
    setUser(data.user);
    setIsAuthenticated(true);
  };

  // Passkey authentication
  const signInWithPasskey = async () => {
    // 1. Get challenge from backend
    const challengeRes = await fetch('/api/auth/passkey/initiate');
    const challenge = await challengeRes.json();
```

```
    // 2. Use WebAuthn API
    const credential = await navigator.credentials.get({
      publicKey: challenge,
    });

    // 3. Verify with backend
    const verifyRes = await fetch('/api/auth/passkey/verify', {
      method: 'POST',
      body: JSON.stringify({ credential }),
    });
    const data = await verifyRes.json();

    // Store JWT
    localStorage.setItem('jwt', data.jwt);
    setUser(data.user);
    setIsAuthenticated(true);
  };

  const signOut = () => {
    localStorage.removeItem('jwt');
    setUser(null);
    setIsAuthenticated(false);
  };

  return {
    isAuthenticated,
    user,
    loading,
    signInWithEmail,
    verifyEmailToken,
    signInWithPasskey,
    signOut,
  };
}
```

AuthModal Component:

typescript

```tsx
export function AuthModal({
  open,
  onClose,
  initialTab = 'email'
}: AuthModalProps) {

  const [tab, setTab] = useState<'email' | 'passkey'>(initialTab);
  const [email, setEmail] = useState('');
  const [emailSent, setEmailSent] = useState(false);
  const { signInWithEmail, signInWithPasskey } = useAuth();

  const handleEmailSubmit = async () => {
    await signInWithEmail(email);
    setEmailSent(true);
  };

  const handlePasskeyAuth = async () => {
    try {
      await signInWithPasskey();
      onClose();
    } catch (error) {
      showError('Passkey authentication failed');
    }
  };

  return (
    <Modal open={open} onClose={onClose}>
      <Tabs value={tab} onChange={setTab}>
        <Tab value="email">Email</Tab>
        <Tab value="passkey">Passkey</Tab>
      </Tabs>

      {tab === 'email' ? (
        emailSent ? (
          <EmailSent>
            <CheckEmailIcon />
            <Title>Check your email</Title>
            <Description>
              We sent a magic link to {email}
            </Description>
          </EmailSent>
        ) : (
          <EmailForm>
            <Title>Sign in with Email</Title>
```

```
          <EmailInput
            type="email"
            placeholder="your@email.com"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
          />
          <SubmitButton onClick={handleEmailSubmit}>
            Send Magic Link
          </SubmitButton>
        </EmailForm>
      )
    ) : (
      <PasskeyForm>
        <Title>Sign in with Passkey</Title>
        <FingerprintIcon size={80} />
        <Description>
          Use your device's biometric authentication
        </Description>
        <AuthButton onClick={handlePasskeyAuth}>
          Authenticate
        </AuthButton>
      </PasskeyForm>
    )}
  </Modal>
 );
}
```

Acceptance Criteria: ✅ Email magic links sent successfully ✅ Passkey authentication functional ✅ JWT stored securely in localStorage ✅ Auto-login on page refresh ✅ Sign out clears session ✅ Error handling for failed auth

Deliverables:

- `AuthModal.tsx` - Auth modal component
- `useAuth.ts` - Auth hook
- `EmailVerification.tsx` - Magic link verification page
- `auth.module.css` - Styling

Estimated Time: 2-3 days

Task 3.2: Swap Interface UI

File: `frontend/src/components/swap/SwapInterface.tsx`

Requirements:

typescript

```
export function SwapInterface() {
  // State
  const [amount, setAmount] = useState('');
  const [zcashAddress, setZcashAddress] = useState('');
  const [privacyMode, setPrivacyMode] = useState(false);
  const [isSwapping, setIsSwapping] = useState(false);
  const [showProgress, setShowProgress] = useState(false);

  // Hooks
  const { isAuthenticated, user } = useAuth();
  const { initiateSwap, claimZEC } = useSwap();

  // Calculate output
  const outputAmount = useMemo(() => {
    if (!amount) return '0';
    return calculateSwapOutput(parseFloat(amount));
  }, [amount]);

  // Handle swap
  const handleSwap = async () => {
    if (!isAuthenticated) {
      showAuthModal();
      return;
    }

    try {
      setIsSwapping(true);

      // 1. Initiate swap on backend (backend creates HTLC)
      const swapDetails = await initiateSwap({
        strkAmount: parseFloat(amount),
        userZcash: zcashAddress,
        privacyMode
      });

      // 2. Show progress modal
      setShowProgress(true);

      // 3. Monitor via WebSocket
      subscribeToSwapUpdates(swapDetails.swapId);

    } catch (error) {
      showError(error.message);
    } finally {
```

```
    setIsSwapping(false);
  }
};

const canSwap = () => {
  return (
    isAuthenticated &&
    amount &&
    parseFloat(amount) > 0 &&
    zcashAddress &&
    validateZcashAddress(zcashAddress)
  );
};

const getButtonText = () => {
  if (!isAuthenticated) return 'Sign In to Swap';
  if (!amount) return 'Enter Amount';
  if (!zcashAddress) return 'Enter Zcash Address';
  if (!validateZcashAddress(zcashAddress)) return 'Invalid Zcash Address';
  return 'Initiate Swap';
};

return (
  <>
    <SwapCard>
      <InputSection type="send">
        <Label>YOU SEND (Starknet)</Label>
        <AmountInput
          value={amount}
          onChange={setAmount}
          token="STRK"
        />
        <UsdValue>~ ${calculateUsd(amount)} USD</UsdValue>
      </InputSection>

      <SwapDirection>
        <IconButton onClick={handleFlip}>
          <ArrowUpDown />
        </IconButton>
      </SwapDirection>

      <InputSection type="receive">
        <Label>YOU RECEIVE (Zcash)</Label>
        <AmountDisplay value={outputAmount} token="ZEC" />
```

```
  <AddressInput
    placeholder="z1abc...xyz"
    value={zcashAddress}
    onChange={setZcashAddress}
    error={zcashAddress && !validateZcashAddress(zcashAddress)}
  />
</InputSection>

<PrivacyToggle>
  <Label>🔒 PRIVACY MODE</Label>
  <Toggle
    checked={privacyMode}
    onChange={setPrivacyMode}
  />
  <InfoText>
    Route through Zcash shielded pool<br/>
    +30s processing, enhanced privacy
  </InfoText>
</PrivacyToggle>

<SwapDetails>
  <DetailRow>
    <span>Rate:</span>
    <span>1 STRK = {rate} ZEC</span>
  </DetailRow>
  <DetailRow>
    <span>Fee (0.3%):</span>
    <span>{fee} STRK</span>
  </DetailRow>
  <DetailRow highlight>
    <span>You receive:</span>
    <span>{outputAmount} ZEC</span>
  </DetailRow>
  <DetailRow>
    <span>Time:</span>
    <span>~{estimatedTime}s</span>
  </DetailRow>
</SwapDetails>

<SwapButton
  onClick={handleSwap}
  disabled={!canSwap()}
  loading={isSwapping}
>
```

```
        {getButtonText()}
      </SwapButton>
    </SwapCard>

    {showProgress && (
      <ProgressModal
        swapId={currentSwapId}
        onClose={() => setShowProgress(false)}
      />
    )}
  </>
 );
}
```

Component Structure:

```
SwapInterface/
├── SwapCard.tsx          # Main card container
├── InputSection.tsx      # Amount input
├── AddressInput.tsx      # Zcash address input
├── PrivacyToggle.tsx     # Privacy mode toggle
├── SwapDetails.tsx       # Rate, fee, time display
├── SwapButton.tsx        # CTA button
└── styles.module.css     # Component styles
```

Acceptance Criteria: ✅ Matches design exactly (pixel-perfect) ✅ Amount input validates (positive numbers) ✅ Zcash address validates (z-address format) ✅ Output amount calculates correctly ✅ Privacy toggle works ✅ Button shows correct state (disabled, loading, ready) ✅ Responsive design works on mobile ✅ Animations smooth (60fps) ✅ Shows auth modal if not signed in

Deliverables:

- `SwapInterface.tsx` + sub-components
- `swap.module.css` - Styling
- `SwapInterface.test.tsx` - Component tests

Estimated Time: 3-4 days

Task 3.3: Progress Tracking Modal

File: `frontend/src/components/ProgressModal.tsx`

Requirements:

typescript

```tsx
export function ProgressModal({
  swapId,
  onClose
}: ProgressModalProps) {

  const [step, setStep] = useState(1);
  const [txHashes, setTxHashes] = useState<TxHashes>({});
  const [error, setError] = useState<string | null>(null);

  // WebSocket connection for real-time updates
  useEffect(() => {
    if (!swapId) return;

    const ws = new WebSocket(`${WS_URL}/ws/swap/${swapId}`);

    ws.onmessage = (event) => {
      const update = JSON.parse(event.data);

      switch (update.event) {
        case 'starknet_locked':
          setStep(2);
          setTxHashes(prev => ({
            ...prev,
            starknetLock: update.txHash
          }));
          break;

        case 'zcash_locked':
          setStep(3);
          setTxHashes(prev => ({
            ...prev,
            zcashLock: update.txHash
          }));
          break;

        case 'secret_revealed':
          setStep(4);
          break;

        case 'completed':
          setStep(5);
          triggerConfetti();
          break;
```

```
      case 'error':
        setError(update.message);
        break;
    }
  };

  ws.onerror = () => {
    setError('Connection lost. Please refresh.');
  };

  return () => ws.close();
}, [swapId]);

return (
  <Modal open={!!swapId} onClose={onClose}>
    {error ? (
      <ErrorState>
        <ErrorIcon />
        <ErrorTitle>Swap Failed</ErrorTitle>
        <ErrorMessage>{error}</ErrorMessage>
        <Button onClick={onClose}>Close</Button>
      </ErrorState>
    ) : (
      <ProgressTracker>
        <Step
          number={1}
          title="Locking STRK"
          description="Backend deploying HTLC on Starknet..."
          status={getStepStatus(1, step)}
          txHash={txHashes.starknetLock}
        />

        <Connector active={step >= 2} />

        <Step
          number={2}
          title="Pool Locking ZEC"
          description="Creating Zcash HTLC..."
          status={getStepStatus(2, step)}
          txHash={txHashes.zcashLock}
        />

        <Connector active={step >= 3} />
```

```
        <Step
          number={3}
          title="Claiming ZEC"
          description="Revealing secret and claiming funds..."
          status={getStepStatus(3, step)}
        />

        <Connector active={step >= 4} />

        <Step
          number={4}
          title="Swap Complete!"
          description="ZEC delivered to your wallet"
          status={getStepStatus(4, step)}
        />
      </ProgressTracker>
    )}

    {step === 5 && <Confetti />}
  </Modal>
);
}
```

Step States:

- `pending` - Gray, dimmed
- `active` - Blue, pulsing animation
- `complete` - Green, checkmark icon
- `error` - Red, X icon

Acceptance Criteria: ✅ Steps update in real-time via WebSocket ✅ Transaction hashes are clickable (open explorer) ✅ Loading animations smooth ✅ Confetti animation on completion ✅ Error state handled gracefully ✅ Modal closes cleanly ✅ Reconnects WebSocket on disconnection

Deliverables:

- `ProgressModal.tsx` - Modal component
- `Step.tsx` - Individual step component
- `useWebSocket.ts` - WebSocket hook
- `progress.module.css` - Styling

Estimated Time: 2-3 days

Task 3.4: Swap History Page

File: `frontend/src/pages/HistoryPage.tsx`

Requirements:

typescript

```tsx
export function HistoryPage() {
  const { user, isAuthenticated } = useAuth();
  const [filter, setFilter] = useState<'all' | 'completed' | 'pending' | 'failed'>('all');

  const { data: swaps, isLoading } = useQuery({
    queryKey: ['swaps', user?.id, filter],
    queryFn: () => fetchUserSwaps(filter),
    enabled: isAuthenticated,
  });

  if (!isAuthenticated) {
    return <SignInPrompt />;
  }

  return (
    <Container>
      <Header>
        <Title>Swap History</Title>
        <FilterButtons>
          <FilterButton
            active={filter === 'all'}
            onClick={() => setFilter('all')}
          >
            All
          </FilterButton>
          <FilterButton
            active={filter === 'completed'}
            onClick={() => setFilter('completed')}
          >
            Completed
          </FilterButton>
          <FilterButton
            active={filter === 'pending'}
            onClick={() => setFilter('pending')}
          >
            Pending
          </FilterButton>
          <FilterButton
            active={filter === 'failed'}
            onClick={() => setFilter('failed')}
          >
            Failed
          </FilterButton>
        </FilterButtons>
```

```
      </Header>

      {isLoading ? (
        <Skeleton count={5} />
      ) : swaps.length === 0 ? (
        <EmptyState>
          <EmptyIcon />
          <EmptyTitle>No swaps yet</EmptyTitle>
          <EmptyDescription>
            Start your first swap to see your history
          </EmptyDescription>
          <Link to="/swap">
            <Button>Go to Swap</Button>
          </Link>
        </EmptyState>
      ) : (
        <SwapList>
          {swaps.map(swap => (
            <SwapCard key={swap.id}>
              <SwapInfo>
                <Amount>{formatAmount(swap.strkAmount)} STRK</Amount>
                <Arrow>→</Arrow>
                <Amount>{formatAmount(swap.zecAmount)} ZEC</Amount>
              </SwapInfo>
              <StatusBadge status={swap.state}>
                {formatStatus(swap.state)}
              </StatusBadge>
              <Timestamp>
                {formatTimestamp(swap.createdAt)}
              </Timestamp>
              <ViewButton onClick={() => viewDetails(swap.id)}>
                View Details
              </ViewButton>
            </SwapCard>
          ))}
        </SwapList>
      )}
    </Container>
  );
}
```

Acceptance Criteria: ✅ Fetches user's swap history from API ✅ Filters work correctly ✅ Shows loading state ✅ Empty state when no swaps ✅ Click to view swap details ✅ Status badges color-coded ✅ Pagination for large lists ✅ Redirects to sign-in if not authenticated

Deliverables:

- `HistoryPage.tsx` - Page component
- `SwapCard.tsx` - List item component
- `useSwapHistory.ts` - Data fetching hook
- `history.module.css` - Styling

Estimated Time: 2 days

Role 4: DevOps/Integration

Task 4.1: Database Setup & Migrations

Files:

- `backend/migrations/001_initial_schema.sql`
- `backend/migrations/002_indexes.sql`
- `backend/migrations/003_auth_tables.sql`

Requirements:

- Set up PostgreSQL database
- Create all tables from schema
- Add necessary indexes
- Seed initial pool state
- Create backup/restore scripts

Acceptance Criteria: ✅ Database runs in Docker ✅ Migrations run successfully ✅ Connection pooling configured ✅ Indexes improve query performance ✅ Backup script functional

Estimated Time: 1 day

Task 4.2: Environment Configuration

Files:

- `backend/.env.example`
- `frontend/.env.example`
- `docker-compose.yml`

Requirements:

Backend .env:

bash

```
# Database
DATABASE_URL=postgresql://postgres:password@localhost:5432/shadowswap
REDIS_URL=redis://localhost:6379

# Starknet
STARKNET_RPC_URL=https://starknet-sepolia.infura.io/v3/YOUR_KEY
STARKNET_HTLC_ADDRESS=0x...

# Zcash
ZCASH_RPC_URL=http://localhost:8232
ZCASH_RPC_USER=testuser
ZCASH_RPC_PASSWORD=testpass

# Pool
INITIAL_STRK_RESERVE=10000000000000000000  # 1000 STRK
INITIAL_ZEC_RESERVE=1000000000  # 10 ZEC

# Auth
JWT_SECRET=your-secret-key-min-32-chars
ENCRYPTION_KEY=your-encryption-key-32-bytes
EMAIL_API_KEY=your-sendgrid-api-key
EMAIL_FROM=noreply@shadowswap.io

# Cartridge
CARTRIDGE_API_KEY=your-cartridge-api-key
CARTRIDGE_RPC_URL=https://cartridge-rpc.example.com

# Server
PORT=8080
CORS_ORIGIN=http://localhost:5173
LOG_LEVEL=debug
```

Frontend .env:


bash

```
VITE_API_URL=http://localhost:8080
VITE_WS_URL=ws://localhost:8080
VITE_STARKNET_CHAIN_ID=SN_SEPOLIA
```

Docker Compose:

yaml

```yaml
version: '3.8'

services:
  postgres:
    image: postgres:15
    ports:
      - "5432:5432"
    environment:
      POSTGRES_DB: shadowswap
      POSTGRES_PASSWORD: password
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:7
    ports:
      - "6379:6379"

  zcash:
    image: electriccoinco/zcashd:latest
    ports:
      - "8232:8232"
    command:
      - "-testnet"
      - "-rpcuser=testuser"
      - "-rpcpassword=testpass"
      - "-rpcallowip=0.0.0.0/0"
    volumes:
      - zcash_data:/root/.zcash

  backend:
    build: ./backend
    ports:
      - "8080:8080"
    depends_on:
      - postgres
      - redis
      - zcash
    env_file:
      - ./backend/.env
    volumes:
      - ./backend:/app

  frontend:
```

```
    build: ./frontend
    ports:
      - "5173:5173"
    depends_on:
      - backend
    env_file:
      - ./frontend/.env
    volumes:
      - ./frontend:/app

volumes:
  postgres_data:
  zcash_data:
```

Acceptance Criteria: ✅ All services start with `docker-compose up` ✅ Environment variables load correctly ✅ No secrets committed to git ✅ Clear setup documentation

Estimated Time: 1 day

Task 4.3: Testing & CI/CD

Files:

- `.github/workflows/test.yml`
- `.github/workflows/deploy.yml`

Requirements:

GitHub Actions Workflow:

yaml

```yaml
name: Test

on: [push, pull_request]

jobs:
  test-backend:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions-rs/toolchain@v1
        with:
          toolchain: stable
      - name: Run tests
        run: cd backend && cargo test

  test-frontend:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: 18
      - name: Install dependencies
        run: cd frontend && npm install
      - name: Run tests
        run: cd frontend && npm test

  test-contracts:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Install Scarb
        run: curl --proto '=https' --tlsv1.2 -sSf https://docs.swmansion.com/scarb/install.sh | sh
      - name: Run Cairo tests
        run: cd contracts/starknet && scarb test
```

Acceptance Criteria: ✅ Tests run on every PR ✅ CI passes before merge ✅ Code coverage reports generated ✅ Linting enforced

Estimated Time: 1-2 days

📆 Project Timeline

Week 1 (Nov 15-21):
├── Smart Contracts (Tasks 1.1, 1.2)
├── Backend Auth & Controllers (Task 2.1)
├── Backend API Setup (Task 2.2)
├── Frontend Auth UI (Task 3.1)
└── DevOps Setup (Tasks 4.1, 4.2)

Week 2 (Nov 22-28):
├── Hash Bridge (Task 2.3)
├── Pool Manager (Task 2.4)
├── Blockchain Indexers (Task 2.5)
├── Swap Interface UI (Task 3.2, 3.3)
└── History Page (Task 3.4)

Final Days (Nov 29-30):
├── Integration testing
├── Bug fixes
├── Demo video recording
└── Documentation

✅ Definition of Done

A task is complete when:

1. **Code Quality:**
   - All tests passing
   - Code reviewed by at least one team member
   - No linting errors
   - Documented with comments
2. **Functionality:**
   - Meets all acceptance criteria
   - Works on testnet
   - Handles error cases gracefully
3. **Documentation:**
   - README updated
   - API endpoints documented
   - Deployment guide written
4. **Testing:**
   - Unit tests written
   - Integration tests passing
   - Manual testing completed

🚀 Success Criteria

Project is successful when:

✅ **Technical:**

- 1 successful swap completed on testnet
- < 60 second swap time
- No critical bugs
- All tests passing

✅ **UX:**

- Email/Passkey sign-in works seamlessly
- No wallet popups during swap
- Clear progress indicators
- Mobile-responsive

✅ **Demo:**

- 2-minute demo video recorded
- Shows complete swap flow
- Highlights privacy features
- Demonstrates backend-managed auth

📞 Support & Resources

- **Starknet Docs:** https://docs.starknet.io
- **Zcash Docs:** https://zcash.readthedocs.io
- **Cartridge Docs:** https://docs.cartridge.gg
- **Team Chat:** [Slack/Discord Channel]
- **Daily Standups:** 10 AM UTC

🎯 Post-Launch Roadmap

After successful testnet launch:

1. **Security Audit** (Week 3)
2. **Mainnet Deployment** (Week 4)
3. **Liquidity Bootstrap** (Week 5)
4. **Mobile App** (Month 2)
5. **Additional Chains** (Month 3+)

Let's build Shadow Swap! 🚀