zkETHer + MoPro Implementation Roadmap

From Zero to Mobile Privacy Revolution in 20 Days

Phase 1: Foundation (Days 1-5)

Goal: Understand the core technologies

Box 1A: MoPro Basics (Day 1-2)

What to Learn:

- MoPro installation and setup
- Basic Noir circuit compilation
- Hello World proof generation on mobile
- WASM integration fundamentals

Hands-On Tasks:

- Install MoPro CLI
- Create simple Noir circuit (addition proof)
- Compile to WASM for mobile
- Generate proof on phone using React Native example

Success Metric: Generate your first mobile ZK proof

Box 1B: Poseidon Hash Deep Dive (Day 3)

What to Learn:

- Why Poseidon over SHA256 in ZK circuits
- Poseidon implementation in Noir
- Constraint counting for mobile optimization

Hands-On Tasks:

- Write Noir function using poseidon_bn254
- Compare constraints: Poseidon vs SHA256
- Benchmark hash performance on mobile

Success Metric: Mobile Poseidon hash under 100ms

Box 1C: Merkle Trees in Noir (Day 4-5)

What to Learn:

- Merkle tree inclusion proofs
- Path verification in circuits
- Mobile-optimized tree depth selection

Hands-On Tasks:

- Build Noir circuit for Merkle inclusion
- Test with different tree depths (16 vs 20 vs 24)
- Measure mobile performance impact

Success Metric: Merkle proof verification on mobile

Phase 2: Core zkETHer Components (Days 6-10)

Box 2A: Note System (Day 6)

What to Build:

- Note data structure implementation
- Commitment calculation
- Nullifier generation

Technical Focus:

typescript

```typescript
interface Note {
  amount: bigint;         // Wei amount
  owner_pubkey: string;   // Recipient's spending key
  nonce: Uint8Array;      // Random 32-byte salt
}

function calculateCommitment(note: Note): string {
  return Poseidon([note.amount, note.owner_pubkey, note.nonce]);
}
```

Success Metric: Create and verify notes programmatically

Box 2B: Basic Withdrawal Circuit (Day 7-8)

What to Build:

- Complete Noir circuit for withdrawal proof
- Private/public input handling
- Circuit constraint optimization for mobile

Circuit Structure:

```noir
fn main(
    // Private inputs
    note_amount: Field,
    owner_privkey: Field,
    note_nonce: Field,
    merkle_path: [Field; 20],

    // Public inputs
    pub merkle_root: Field,
    pub nullifier: Field,
    pub recipient: Field
) {
    // Implementation here
}
```

Success Metric: Generate withdrawal proofs under 5 seconds on mobile

Box 2C: Smart Contract Foundation (Day 9-10)

What to Build:

- Basic zkETHer smart contract
- Groth16 verifier integration
- Merkle tree management on-chain

Contract Core:

```solidity
```

```
contract zkETHer {
    mapping(bytes32 => bool) nullifierSet;
    IMerkleTree merkleTree;
    IVerifier verifier;

    function deposit(bytes32 commitment) external payable;
    function withdraw(bytes memory proof, bytes32 nullifier, address recipient) external;
}
```

Success Metric: Deploy on testnet, complete one deposit/withdraw cycle

Phase 3: Mobile Integration (Days 11-15)

Box 3A: React Native + MoPro Setup (Day 11-12)

What to Build:

- React Native app foundation
- MoPro FFI integration
- Basic wallet interface

Technical Setup:

- React Native 0.72+
- MoPro iOS/Android bindings
- Encrypted storage for keys
- Basic ETH wallet functionality

Success Metric: Generate ZK proofs from React Native app

Box 3B: Key Management System (Day 13)

What to Build:

- Separate zkETHer spending keys from ETH wallet keys
- Secure key storage (iOS Keychain, Android Keystore)
- Biometric authentication integration

Security Architecture:

```
typescript
```

```typescript
interface ZKEtherWallet {
  ethereumPrivkey: string;     // For gas, standard transactions
  spendingPrivkey: string;     // For zkETHer nullifiers (secure storage)
  spendingPubkey: string;      // For receiving notes
}
```

Success Metric: Secure key generation and storage on mobile

Box 3C: Note Scanner Implementation (Day 14-15)

What to Build:

- Background service monitoring deposit events
- Trial decryption of incoming notes
- Local encrypted note database

Technical Challenge:

```typescript
class NoteScanner {
  async scanForNotes() {
    // Monitor Deposit events
    // Try decrypt each commitment with user's spending key
    // Save owned notes to encrypted storage
    // Optimize for battery life
  }
}
```

Success Metric: Automatic detection of incoming notes

Phase 4: Complete Privacy Enhancement (Days 16-18)

Box 4A: Relayer for Deposits (Day 16)

What to Build:

- **THIS IS YOUR KEY INSIGHT**: Hide Alice's deposit address too
- Relayer infrastructure for deposit transactions
- Signature verification for relayed deposits

Enhanced Contract:

```solidity
function relayedDeposit(
    bytes32 commitment,
    address depositor,        // Alice pays fees from here
    bytes calldata signature    // Alice authorizes the relay
) external {
    // Verify signature, take fees, add commitment
    // Alice's address NEVER appears in zkETHer transaction
}
```

Success Metric: Complete deposit privacy through relayers

Box 4B: Dual-Relayer Integration (Day 17)

What to Build:

- Mobile app integration with deposit relayers
- Fee management and relayer selection
- Fallback mechanisms for relayer failures

Complete Privacy Flow:

```
1. Alice → Relayer A → zkETHer (deposit)
2. Bob → Relayer B → zkETHer (withdrawal)
3. Result: NO participant addresses visible on-chain
```

Success Metric: End-to-end private transfer with no participant traces

Box 4C: Multi-Relayer Network (Day 18)

What to Build:

- Multiple independent relayers
- Relayer discovery and selection
- Fee comparison and optimization

Decentralization Strategy:

- Deploy 3-5 relayers on different infrastructure
- Implement relayer health monitoring
- Random relayer selection for privacy

Success Metric: Resilient multi-relayer network operational

Phase 5: Demo Preparation (Days 19-20)

Box 5A: Live Demo Integration

What to Build:

- Two-phone demo flow
- Real ETH integration on testnet
- Etherscan integration for live privacy proof

Box 5B: Performance Optimization

What to Focus:

- Sub-5 second proof generation
- Reliable network handling
- Battery optimization

Critical Technical Questions for Yash:

1. Complete Privacy Architecture:

"Your spec achieves withdrawal privacy via relayers but leaves deposit privacy completely exposed. Why not use relayers for deposits too? This would hide Alice's address and create true financial invisibility."

2. MoPro Constraint Optimization:

"What's your target constraint count for mobile proving? Your Merkle depth 20 + Poseidon hashing could exceed mobile performance limits. Have you benchmarked the circuit on actual phones?"

3. Relayer Economics:

"How do you handle relayer centralization risks? If only one relayer exists, it becomes a central point of failure and surveillance."

4. Mobile-Specific Adaptations:

"Your spec doesn't address mobile-specific challenges: battery life for background scanning, secure key storage, offline proof generation. How do you handle these in MoPro integration?"

5. Fixed vs Variable Amounts:

"Your spec mentions fixed 1 ETH amounts but also variable amounts. For mobile privacy, shouldn't we stick to fixed denominations to prevent amount-based correlation attacks?"

Implementation Priority Order:

1. **Start with Box 1A** - Get MoPro working with simple proofs
2. **Master Box 2B** - This is your core technical achievement
3. **Focus on Box 4A** - This is your unique insight that enhances the protocol
4. **Perfect Box 5A** - This wins the hackathon

The Key Questions to Ask Yash:

"Why not complete privacy with relayers for both deposit and withdrawal?"

This single question could revolutionize his entire protocol design. You've seen something he missed.

Your technical instincts are sharp. Start with MoPro basics, then push toward the complete privacy model you've identified.