Mantle Stealth Payments – Technical Documentation

1. Overview

This document describes how to design and implement an Umbra-style stealth payments protocol on Mantle Network using the ERC-5564 stealth address standard.[web:14][web:22][web:5] The goal is to enable private, non-interactive transfers of MNT and ERC-20 tokens where only the sender and recipient can link a payment to a recipient identity, while all on-chain activity remains publicly verifiable.[web:14][web:25]

The system comprises:

- A **stealth address layer** based on ERC-5564
- Optional **shielded pool layer** for hiding transfer amounts
- A **frontend + SDK** to generate keys, send payments, and scan for receipts
- Off-chain **indexing services** to make UX practical

This documentation is intended for Solidity and TypeScript developers targeting Mantle's EVM-compatible environment.[web:26][web:31]

---

2. Background: Stealth Addresses & Umbra

2.1 ERC-5564: Stealth Addresses

ERC-5564 standardizes how to generate and announce stealth addresses on EVM chains.[web:14][web:22][web:23] It defines:

- A **stealth meta-address** published by the recipient (derived from viewing and spending keys)
- A **mechanism for senders** to generate one-time stealth addresses using elliptic-curve operations and an ephemeral keypair
- A **Messenger/Announcer contract** that emits standardized `Announcement` events for recipients to scan and discover incoming payments[web:14][web:22]

Core ideas:

1. Recipient publishes a **stealth meta-address** (e.g., via ENS or your dapp profile).[web:22][web:5]
2. Sender uses meta-address + random secret to compute a unique stealth address.
3. Sender sends funds to that stealth address and posts an on-chain announcement containing encrypted data and an ephemeral public key.
4. Recipient scans announcements, derives the stealth private key using their viewing/spending keys, and withdraws funds.[web:22][web:5]

2.2 Umbra Protocol

Umbra is a production implementation of stealth payments on Ethereum and rollups, built before ERC-5564 was finalized.[web:25][web:5] It uses:

- A singleton smart contract for announcements
- Off-chain key derivation using ECDH on secp256k1
- A UX where recipients scan events and reconstruct stealth private keys[web:25][web:30]

Academic analysis shows that naïve use of stealth schemes can leak recipient identities through wallet behavior and on-chain heuristics, motivating stronger privacy features and careful UX design.[web:10]

Our Mantle implementation follows the ERC-5564 standard while borrowing practical ideas from Umbra's design and UX.[web:14][web:22][web:25]

---

3. System Architecture

3.1 High-Level Components

1. **Smart Contracts (Mantle EVM)**[web:26][web:31]
   - `ERC5564Announcer` – emits standardized announcement events
   - `StealthPay` – helper contract to send MNT/ERC-20 tokens to stealth addresses
   - (Optional) `ShieldedPool` – ZK-based pool to hide amounts and break linkability further
2. **Off-Chain Crypto & SDK (TypeScript)**
   - Key management: viewing key, spending key, stealth meta-address
   - ECDH shared-secret derivation (secp256k1)
   - Stealth address derivation from meta-address + shared secret[web:22][web:29]
   - Event scanning and stealth private key reconstruction[web:5][web:30]
3. **Frontend Dapp (Next.js/React)**
   - Sender flow: input recipient meta-address → generate stealth address → send payment
   - Recipient flow: connect wallet → scan announcements → list and withdraw received payments
4. **Indexing/Infrastructure**
   - A small service or The Graph-style indexer that tails `Announcement` events on Mantle and exposes them via an API for faster scanning.[web:28][web:31]

3.2 Interaction Flow

3.2.1 Sender

1. Fetch recipient's stealth meta-address.
2. Generate ephemeral keypair (`eph_priv`, `eph_pub`).
3. Compute ECDH shared secret with recipient's view public key.
4. Derive stealth address from shared secret + spending public key.[web:22][web:5]
5. Send MNT/ERC-20 to stealth address.
6. Call `announce()` on `ERC5564Announcer` with:
   - `schemeId` (e.g., `1` for secp256k1)
   - `stealthAddress`

- ephemeralPubKey
- metadata (encrypted shared secret or related data)[web:14][web:22]

3.2.2 Recipient

1. Periodically scan Announcement events.
2. For each event, use viewing private key + ephemeralPubKey to derive shared secret. [web:22][web:30]
3. Derive candidate stealth private key; compute its address.
4. If address matches stealthAddress in event, mark as payment for recipient.
5. Use stealth private key to sign a withdraw transaction to the main wallet (directly or via relayer).[web:25][web:30]

---

4. Smart Contracts on Mantle

4.1 ERC-5564 Announcer Contract

ERC-5564 defines a standard event and function for announcements.[web:14][web:22] [web:23] A minimal announcer:

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

/// @title ERC-5564 Announcer for Stealth Addresses on Mantle
/// @notice Emits standardized events for stealth payments
contract ERC5564Announcer {
/// @dev Emitted whenever a stealth payment is announced
event Announcement(
uint256 indexed schemeId,
address indexed stealthAddress,
address indexed caller,
bytes ephemeralPubKey,
bytes metadata
);

```
/// @notice Announce a stealth payment
/// @param schemeId Identifier of the cryptographic scheme (e.g., 1 = secp256k1
/// @param stealthAddress Recipient's one-time stealth address
/// @param ephemeralPubKey Ephemeral public key used to derive shared secr
/// @param metadata Encrypted data needed for recipient to reconstruct keys
function announce(
    uint256 schemeId,
    address stealthAddress,
    bytes calldata ephemeralPubKey,
    bytes calldata metadata
) external {
    emit Announcement(
```

```
        schemeId,
        stealthAddress,
        msg.sender,
        ephemeralPubKey,
        metadata
    );
}
```

}

Deployment notes:

- Deploy a **singleton** instance and hard-code its address in your SDK for deterministic scanning.[web:22][web:25]
- Use a fixed `schemeId` for your chosen curve (e.g., `1` for secp256k1).

4.2 Stealth Payment Helper

A helper contract can simplify token transfers to stealth addresses:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "./ERC5564Announcer.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract StealthPay {
ERC5564Announcer public immutable announcer;

    constructor(address _announcer) {
        announcer = ERC5564Announcer(_announcer);
    }

    function sendEtherStealth(
        uint256 schemeId,
        address stealthAddress,
        bytes calldata ephemeralPubKey,
        bytes calldata metadata
    ) external payable {
        require(msg.value > 0, "No value");
        (bool ok, ) = stealthAddress.call{value: msg.value}("");
        require(ok, "Transfer failed");

        announcer.announce(schemeId, stealthAddress, ephemeralPubKey, metadata
```

```
    }

    function sendTokenStealth(
        uint256 schemeId,
        address token,
        uint256 amount,
        address stealthAddress,
        bytes calldata ephemeralPubKey,
        bytes calldata metadata
    ) external {
        require(amount > 0, "No amount");
        IERC20(token).transferFrom(msg.sender, stealthAddress, amount);

        announcer.announce(schemeId, stealthAddress, ephemeralPubKey, metadata
    }
```

}

On Mantle, this works as on other EVM chains since Mantle is Ethereum-compatible at the bytecode level.[web:26][web:31]

4.3 Optional Shielded Pool (ZK Layer)

To protect **amounts** and mitigate deanonymization heuristics observed for Umbra,[web:10] you can add a ZK shielded pool:

- Users deposit to `ShieldedPool` to receive a commitment.
- A Merkle tree tracks commitments.
- Withdrawals require a ZK proof and a nullifier to prevent double-spending.

Typical interface:

interface IShieldedPool {
function deposit(bytes32 commitment) external payable;
function withdraw(
uint256 amount,
address recipient,
bytes32 nullifierHash,
bytes calldata proof
) external;
}

Circuit design includes:

- Private inputs: deposit secret, Merkle path, etc.
- Public inputs: `merkleRoot`, `nullifierHash`, `amount`, `recipient`.
- Constraints:

    - Commitment is in tree (valid path to root).
    - `nullifierHash` derived from secret.
    - Prevent reuse of `nullifierHash`.

This ZK layer is optional but significantly improves privacy guarantees.[web:10][web:30]

---

5. Cryptography & Key Management

5.1 Key Types

A typical implementation uses:[web:22][web:25][web:5]

- **Viewing keypair** (view_priv, view_pub) – used to derive shared secrets and identify payments.
- **Spending keypair** (spend_priv, spend_pub) – used to control resulting stealth addresses.

The **stealth meta-address** can be defined as a combination of these public keys, e.g.:

meta = (view_pub, spend_pub)

You may pack this into a single byte array or encode as an ENS text record.[web:22][web:5]

5.2 Sender Derivation (Off-Chain)

Given recipient meta-address:

1. Generate ephemeral keypair:
    - eph_priv = random_scalar()
    - eph_pub = eph_priv * G
2. Compute shared secret using ECDH on secp256k1:
    - shared = H(eph_priv * view_pub)
3. Derive stealth public key and address:
    - stealth_pub = spend_pub + shared * G
    - stealth_address = keccak256(stealth_pub)[12:] (standard Ethereum address derivation)[web:22][web:29]
4. Encrypt necessary data into `metadata`: e.g., encode `shared` or auxiliary info under `view_pub`.
5. Send funds to `stealth_address` and call `announce(schemeId, stealthAddress, eph_pub, metadata)`.[web:22][web:5]

5.3 Recipient Derivation (Off-Chain)

For each `Announcement`:

1. Extract `ephemeralPubKey` and `metadata`.
2. Compute shared secret with viewing private key:
    - `shared = H(view_priv * eph_pub)`
3. Compute candidate stealth private key:
    - `stealth_priv = spend_priv + shared (mod n)`
4. Compute address:
    - `addr = address_from_priv(stealth_priv)`

5. If `addr == stealthAddress` from event, this payment belongs to the recipient; store `(stealth_priv, stealthAddress, amount)` in local wallet for later withdrawal.[web:22][web:25][web:30]

This logic typically lives in your TypeScript SDK.

---

6. Off-Chain SDK Design (TypeScript)

6.1 Module Structure

A suggested structure:

- `keys.ts` – generate and serialize viewing/spending keypairs and meta-addresses.
- `sender.ts` – create stealth payments.
- `receiver.ts` – scan announcements and derive stealth keys.
- `contracts.ts` – ABIs and Mantle network configuration.[web:26][web:31]

6.2 Example: Sender Flow

```
import { ethers } from "ethers";
import { ERC5564Announcer__factory, StealthPay__factory } from "./typechain";
import { deriveStealthAddress } from "./crypto";

export async function sendStealthPayment(params: {
provider: ethers.JsonRpcProvider;
signer: ethers.Signer;
recipientMeta: {
viewPub: Uint8Array;
spendPub: Uint8Array;
};
amountWei: bigint;
schemeId: bigint;
announcerAddress: string;
stealthPayAddress: string;
}) {
const { signer, recipientMeta, amountWei, schemeId, announcerAddress, stealthPayAddress
} = params;

// 1. Derive stealth address and ephemeral key
const { stealthAddress, ephPub, metadata } = deriveStealthAddress(recipientMeta);

// 2. Call StealthPay contract on Mantle
const contract = StealthPay__factory.connect(stealthPayAddress, signer);

const tx = await contract.sendEtherStealth(
schemeId,
stealthAddress,
ephPub,
metadata,
{ value: amountWei }
);
```

return tx.wait();
}

6.3 Example: Receiver Scanner

import { ethers } from "ethers";
import { ERC5564Announcer__factory } from "./typechain";
import { deriveStealthPrivateKey } from "./crypto";

export async function scanAnnouncements(params: {
provider: ethers.JsonRpcProvider;
announcerAddress: string;
viewPriv: Uint8Array;
spendPriv: Uint8Array;
fromBlock: number;
toBlock: number;
}) {
const { provider, announcerAddress, viewPriv, spendPriv, fromBlock, toBlock } = params;

const announcer = ERC5564Announcer__factory.connect(announcerAddress, provider);

const filter = announcer.filters.Announcement(null, null, null);
const events = await announcer.queryFilter(filter, fromBlock, toBlock);

const results: Array<{
stealthAddress: string;
stealthPriv: string;
txHash: string;
}> = [];

for (const ev of events) {
const { schemeId, stealthAddress, caller, ephemeralPubKey, metadata } = ev.args;
// Only handle known schemeId
if (schemeId.toString() !== "1") continue;

```
const stealthPriv = deriveStealthPrivateKey({
  viewPriv,
  spendPriv,
  ephemeralPubKey: ephemeralPubKey as Uint8Array,
  metadata: metadata as Uint8Array,
});

const derivedAddress = new ethers.Wallet(ethers.hexlify(stealthPriv)).address;

if (derivedAddress.toLowerCase() === stealthAddress.toLowerCase()) {
  results.push({
    stealthAddress,
```

```
      stealthPriv: ethers.hexlify(stealthPriv),
      txHash: ev.transactionHash,
    });
  }
```

}

return results;
}

---

7. Frontend Dapp Flows

7.1 Sender UI

Key screens:

1. **Home / Send Payment**
   - Inputs: recipient handle or meta-address, amount, asset (MNT or ERC-20)
   - Button: “Send Private Payment”
   - Under the hood:
     - Parse/resolve meta-address
     - Call SDK to create stealth payment
     - Show transaction status and link to Mantle explorer.[web:28][web:31]
2. **Sent History (Optional)**
   - Show list of stealth payments made, using local state only (do not break privacy).

7.2 Recipient UI

Key screens:

1. **Connect Wallet & Set Up Keys**
   - If no stealth keys stored, generate viewing/spending keypairs and prompt user to back them up.
   - Optionally publish meta-address to a profile or ENS-like system.
2. **Incoming Payments**
   - “Scan Payments” button to query announcements (or connect to indexer).
   - Show list of claimable stealth balances (stealth address + approximate timestamp + asset).
3. **Withdraw Flow**
   - For each payment, allow “Withdraw to main wallet” (direct) or “Withdraw via relayer” (for extra privacy).
   - Uses stealth private key to sign a transaction moving funds from the stealth address to the main wallet.

---

8. Deployment on Mantle

8.1 Network Configuration

Use Mantle testnet/mainnet RPC endpoints and chain IDs from Mantle developer docs.[web:26][web:28][web:31]

Example configuration:

```
export const MANTLE_CONFIG = {
chainId: 5000, // Example, verify actual ID
rpcUrl: "https://rpc.mantle.xyz", // Check latest endpoint
explorerUrl: "https://explorer.mantle.xyz",
announcerAddress: "0x...", // Deployed ERC5564Announcer
stealthPayAddress: "0x...", // Deployed StealthPay
};
```

Refer to Mantle's official docs and GitHub repository for up-to-date network details and tooling.[web:26][web:31]

8.2 Steps

1. Compile contracts with Hardhat/Foundry.
2. Deploy `ERC5564Announcer` to Mantle testnet.
3. Deploy `StealthPay` with announcer address.
4. Configure SDK/Frontend with deployed addresses and chain ID.
5. Run end-to-end tests on testnet:
    - Generate recipient meta-address
    - Send stealth payment
    - Scan and withdraw

---

9. Security & Privacy Considerations

9.1 Cryptographic Security

- Use well-audited ECDH libraries and secp256k1 implementations.
- Ensure secure randomness for ephemeral private keys.
- Avoid reusing ephemeral keys across payments.

9.2 Deanonymization Risks

Research has shown that naive use of Umbra can allow deanonymization of 25–65% of recipients using on-chain heuristics.[web:10] Common pitfalls include:

- Spending from stealth addresses directly to a known address without delay.
- Reusing patterns (gas tokens, transaction timing) that link identities.

Mitigations:

- Encourage users to withdraw via relayers.
- Introduce random delays and intermediate hops.
- Use shielded pools to hide amounts and transaction graph.[web:10][web:30]

9.3 Key Management

- Viewing and spending keys must be backed up securely by the user; losing them means losing access to stealth payments.
- Consider integrating with hardware wallets or secure enclaves.

9.4 Contract Security

- Follow best practices from OpenZeppelin and audit guidelines.
- Protect against re-entrancy, integer overflows (using ^0.8+), and misconfigured approvals.[web:26]
- Document trust assumptions clearly.

---

10. Hackathon Deliverables Checklist

To present this project in a Mantle hackathon context:[web:15][web:31]

1. **Smart Contracts**
    - ERC-5564 Announcer on Mantle testnet
    - Stealth payment helper contracts
    - (Optional) ShieldedPool with ZK circuits
2. **SDK**
    - Functions for key generation, stealth address derivation, scanning, and withdrawal.
    - NPM package or inline library in the repo.
3. **Frontend Dapp**
    - Intuitive sender and receiver flows
    - Clear explanation of privacy guarantees and limitations
4. **Documentation (this document + README)**
    - High-level architecture
    - Cryptographic design and trade-offs
    - Setup instructions for developers and users
5. **Demo**
    - Recorded video showing:
        - Generating a meta-address
        - Sending a stealth payment
        - Scanning and withdrawing on Mantle testnet

---

11. References

[1] ERC-5564: Stealth Addresses – Ethereum Improvement Proposal. https://eips.ethereum.org/EIPS/eip-5564[web:14]

[2] “ERC-5564 – Stealth Addresses” – Intro & utils. https://nerolation.github.io/stealth-utils/[web:22]

[3] QuickNode – “Private Transactions on Ethereum using Stealth Addresses (ERC-5564).” https://www.quicknode.com/guides/ethereum-development/wallets/how-to-use-stealth-addresses-on-ethereum-eip-5564[web:5]

[4] ERC-5564 spec in ERCs repository. https://github.com/ethereum/ERCs/blob/master/ERCS/erc-5564.md[web:23]

[5] EIP-5564 directory entry. https://eip.directory/ercs/erc-5564[web:24]

[6] ScopeLift – “Introducing Umbra – Privacy Preserving Stealth Payments.” https://scopelift.co/blog/introducing-umbra[web:25]

[7] Umbra-style implementation notes and PoCs. https://github.com/stealthpadxyz/stealth-payments[web:30]

[8] “Anonymity Analysis of the Umbra Stealth Address Scheme.” arXiv:2308.01703. https://arxiv.org/abs/2308.01703[web:10]

[9] Ethereum Magicians thread on ERC-5564. https://ethereum-magicians.org/t/erc-5564-stealth-addresses/10614[web:27]

[10] Mantle Network documentation and developer resources. https://github.com/mantlenetworkio/documents[web:26]

[11] Mantle Network developer portal – “Building the Liquidity Chain of the Future.” https://www.mantle.xyz/developers[web:31]

[12] Mantle data overview (for event/log analysis). https://docs.dune.com/data-catalog/evm/mantle/overview[web:28]