OnChain Message

Immutable On-Chain Communication Protocol

Whitepaper V1.0

March 2026

Author: Textoshi

A permissionless, immutable smart contract for sending permanent messages on the blockchain. Deployed across 8 EVM chains at a single deterministic address. No owner. No admin. No upgrades.

Contract Address (all chains):

```
0x0000000000000000000000000000000000000000
```

To be updated upon deployment.

Supported Chains:

Chain	Chain ID	Native Token
Ethereum	1	ETH
Base	8453	ETH
BNB Smart Chain	56	BNB

Arbitrum One	42161	ETH
Polygon	137	POL
Monad	143	MON
Sonic	146	S
HyperEVM	999	HYPE

Table of Contents

1. Abstract

OnChain Message is an immutable smart contract protocol that enables anyone to send permanent, censorship-resistant messages on the blockchain. Messages can be broadcast publicly or sent directly to any Ethereum address. The protocol is deployed at a single deterministic contract address across 8 EVM-compatible chains: Ethereum, Base, BNB Smart Chain, Arbitrum, Polygon, Monad, Sonic, and HyperEVM.

The contract has no owner, no admin keys, no upgrade proxy, and no pause mechanism. Once deployed, it cannot be modified by anyone. This is a deliberate and permanent design decision. The protocol charges a flat, tiered fee per message based on length, with all fees forwarded to an immutable fee recipient address.

OnChain Message solves a fundamental gap in blockchain infrastructure: there is no simple, user-friendly way to send a message to an Ethereum address. Today, doing so requires encoding raw calldata, deploying custom contracts, or using cumbersome block explorer tools. This protocol makes on-chain communication as simple as typing a message and clicking send.

2. The Problem

Blockchains are excellent at transferring value but terrible at transferring information. There is no native mechanism in the EVM for one address to send a human-readable message to another address. This creates several real-world problems:

Whitehat communication. When a protocol is exploited, the only way to reach the attacker is to encode a message in a transaction's input data field. This requires technical knowledge most users and even most protocol teams do not possess. Critical negotiations are delayed by hours or days because of this friction.

Anonymous contact. Pseudonymous developers, artists, and operators have no public inbox. Twitter DMs can be disabled. Discord requires joining a server. Email requires doxxing. An on-chain message to their deployer address is the only truly permissionless way to reach them.

Permanent public statements. Tweets get deleted. Websites go down. Blog posts are edited. There is no widely-used tool for making permanent, immutable public statements on-chain. On-chain communication should be a public good, not a developer tool.

Existing solutions are inadequate. Etherscan's Input Data Messages (IDM) feature requires users to manually encode hex data. It has no recipient field, no inbox, no notification system, and no cross-chain support. It is a developer-facing tool, not a user-facing product.

3. Protocol Design

The protocol is governed by five immutable design principles:

Immutability. The contract has no owner, no admin, no upgrade proxy, no pause mechanism, and no selfdestruct. Once deployed, its behavior is fixed forever.

Permissionless. Any address can send a message to any other address. No registration, no approval, no KYC. If you have a wallet, you can write on-chain.

Simplicity. One function: sendMessage(address to, string message). The entire contract is under 80 lines of Solidity. Simplicity minimizes attack surface.

Deterministic deployment. The same contract is deployed at the same address on all 8 chains using CREATE2. Users interact with one address regardless of which chain they are on.

Transparent pricing. Fees are hardcoded as constants in the contract. They cannot be changed. Users always know exactly what a message costs before sending it.

4. Smart Contract Architecture

The contract exposes a single external function:

```
function sendMessage(address to, string calldata message) external payable
```

And emits a single event:

```
event MessageSent( address indexed sender, address indexed to, string message,
uint256 timestamp )
```

The **sender** and **to** fields are both indexed, enabling efficient filtering by either party. Frontend applications can query for all messages sent by a specific address, all messages sent to a specific address, or both, using standard EVM event log filtering.

Messages are stored as event logs, not in contract storage. Event logs are part of the blockchain's permanent record and can be read by any node, but they cost significantly less gas than SSTORE operations. This is the correct tradeoff: messages are write-once, read-many data that does not need to be accessed by other smart contracts.

4.1 Execution Flow

The contract follows the Checks-Effects-Interactions (CEI) pattern. When sendMessage is called: (1) the message byte length is validated against the maximum (200,000 bytes); (2) the required payment is determined by the tier; (3) msg.value is validated against the required amount; (4) the MessageSent event is emitted; (5) the payment is forwarded to the immutable fee recipient via a low-level call. The external call is always the last operation, preventing reentrancy.

4.2 Immutability Guarantees

The contract contains no owner state variable, no Ownable inheritance, no access control modifiers, no proxy pattern, no delegatecall, no selfdestruct (deprecated in Solidity 0.8.24+), and no pause mechanism. The fee recipient is declared as a constant (not immutable), meaning it is embedded directly in the bytecode at compile time. There is no function to change it. There is no function to withdraw stuck funds. The receive() function reverts to prevent accidental ETH transfers.

5. Tiered Pricing Model

Message fees are determined by the byte length of the message content. All thresholds and prices are hardcoded as constants in the contract.

Tier	Max Bytes	Approx. Characters	Price
Short	560	~280	0.005 ETH
Medium	2,000	~1,000	0.01 ETH
Long	200,000	~100,000	0.05 ETH

The byte-length approach (rather than character count) ensures consistent pricing across all character encodings. ASCII characters are 1 byte each, while multi-byte UTF-8 characters (emoji, CJK) consume 2-4 bytes. The exact msg.value must match the required tier price; overpayment and underpayment both revert.

6. Message Types

6.1 Public Broadcast

When the **to** parameter is set to address(0) (the zero address), the message is treated as a public broadcast. It appears in the Global Feed and is not directed at any specific recipient. Public broadcasts are suitable for announcements, statements, on-chain proofs, and any message intended for a general audience.

6.2 Direct Message

When the **to** parameter is set to any non-zero address, the message is a direct message to that recipient. The message appears in the recipient's Inbox on the frontend. Important: direct messages are still publicly visible on the blockchain. The "direct" designation affects routing and display, not privacy. All on-chain data is inherently public.

6.3 Self-Message

A user may set **to** equal to their own address (msg.sender). This functions as an on-chain note-to-self or personal timestamp. The message appears in both the user's Sent and Inbox views.

7. Multi-Chain Deployment

The contract is deployed at the same deterministic address across all 8 supported chains using the CREATE2 opcode. This means users interact with a single contract address regardless of which chain they are on.

Messages on different chains are independent. A message sent on Base cannot be read via Ethereum event logs, and vice versa. The frontend allows users to switch between chains to view messages on each chain separately.

Chain selection affects gas cost. L2 chains like Base, Arbitrum, and Polygon offer significantly lower gas fees than Ethereum mainnet, making messaging more affordable. The protocol fee (0.005 / 0.01 / 0.05 ETH) is the same across all chains; only the network gas fee varies.

8. Frontend Architecture

The frontend is a single static HTML file hosted on Cloudflare Pages (free tier). It has no backend, no database, no server-side logic, and no API keys. All blockchain interaction happens client-side via ethers.js and free public RPC endpoints.

Wallet connection uses the EIP-6963 standard for multi-wallet discovery. This allows users to select from all installed browser wallets (MetaMask, Rabby, Coinbase Wallet, etc.) via a modal, with a fallback to window.ethereum for legacy wallets.

The frontend implements a "Last 25" query pattern for all feed views. Rather than scanning a time range of blocks (which becomes expensive at high message volumes), the frontend queries backward from the latest block and stops as soon as 25 messages are found. This scales to infinite message volume with constant query cost.

ENS names are resolved against Ethereum mainnet and cached in-session. Resolved names are displayed alongside full 42-character addresses. Full addresses are always shown to prevent address poisoning attacks. All addresses link to the appropriate chain's block explorer.

9. Security Model

9.1 Contract Security

The contract follows the Checks-Effects-Interactions pattern. The external call (fee transfer) is always the last operation. Custom errors are used instead of require strings to reduce gas costs and improve debuggability. The contract contains no loops, no unbounded arrays, and no delegatecall. The receive() function reverts unconditionally.

9.2 Frontend Security

All user-generated content is escaped via textContent assignment before rendering to prevent XSS attacks. No API keys are exposed in the frontend. All RPC endpoints are free public endpoints requiring no authentication. Address inputs are validated using ethers.isAddress() before any contract interaction.

9.3 Anti-Spoofing

Full 42-character addresses are displayed everywhere on the site. Addresses are never truncated in message feeds to prevent address poisoning attacks where a malicious actor creates an address with matching first and last characters. All addresses link directly to the chain's block explorer for independent verification.

10. Use Cases

Whitehat negotiation. After a protocol exploit, anyone can send a message to the attacker's address requesting return of funds, offering a bounty, or establishing communication.

Anonymous contact. Reach pseudonymous developers, artists, or operators via their on-chain address. No social media account required.

On-chain proof. Timestamp a statement, prediction, or commitment permanently. The blockchain provides an immutable record with cryptographic proof of authorship.

DAO communication. Proposals, announcements, and governance messages can be recorded permanently on-chain for any address to read.

Public declarations. Open letters, manifestos, and public statements that cannot be edited, deleted, or censored after publication.

Cross-chain presence. Establish a message history on multiple chains. A single address can maintain separate on-chain presence on Ethereum, Base, BSC, and more.

11. Revenue Model

All message fees are forwarded to the immutable fee recipient address in the same transaction. There is no accumulated balance in the contract, no withdrawal function, and no way to change the fee recipient. Revenue flows directly and immediately.

The tiered pricing model incentivizes short messages (high volume, low friction) while capturing higher value from longer content. At scale, the revenue mix is expected to be approximately 75% short messages, 20% medium, and 5% long, yielding an effective average revenue of approximately 0.008 ETH per message.

12. Roadmap

Phase 1 — Launch (Q1 2026). Contract deployment and verification on all 8 chains. Static frontend launch on Cloudflare Pages. Etherscan/Basescan contract verification.

Phase 2 — Growth (Q2 2026). Dune Analytics dashboard for message volume tracking. Share-to-X viral loop optimization. Community outreach to whitehat and security communities.

Phase 3 — Scale (Q3-Q4 2026). Subgraph/indexer deployment if volume exceeds 5,000 messages per day per chain. Shareable OG link previews via Cloudflare Workers. Additional chain deployments as new EVM chains launch.

Phase 4 — Ecosystem (2027+). Third-party frontend integrations. Wallet-native messaging UX partnerships. Multi-language support. Reply threading via optional transaction hash references.

13. Conclusion

OnChain Message is a protocol, not a product. The contract is immutable. The frontend is a convenience layer that anyone can rebuild. The messages are permanent. Even if every frontend disappears, every message ever sent remains on-chain, readable by any node, any block explorer, any application, forever.

This is not a communication platform with terms of service, content moderation, or account management. It is a piece of immutable infrastructure — a public good for the EVM ecosystem. The contract charges a fee for sustainability, not for access. Anyone can read messages for free. Anyone can build a frontend. The protocol simply exists.

— Textoshi, March 2026

onchain-message.com