ISSN 2085-4552

Implementation of ECDSA and MLDSA Digital Signature Schemes for Transaction Authentication on Blockchain

Ridwan Muhammad Raihan[1], Alam Rahmatulloh[2]*
[1,2]Department of Informatics, Faculty of Engineering, Siliwangi University, Tasikmalaya, West Java
[1] 227006079@student.unsil.ac.id, [2] alam@unsil.ac.id

Accepted 25 May 2026
Approved 28 June 2026

***Abstract*— Blockchain transaction authentication commonly relies on ECDSA because it is efficient and compatible with Ethereum smart contracts. However, ECDSA may be affected by future quantum attacks, while direct adoption of post-quantum signatures increases transaction payload and execution cost. This study implements and evaluates an oracle-assisted hybrid ECDSA and ML-DSA signature scheme for Ethereum-compatible transaction authentication. The proposed scheme uses ECDSA to maintain EVM compatibility and ML-DSA to provide post-quantum authentication. The system is implemented using a Client Oracle Blockchain architecture, where the oracle performs ML-DSA signing and verification off-chain, while the smart contract verifies the ECDSA component on-chain and stores the post-quantum signature hash as an integrity anchor. The experiment was conducted on a private Ethereum-compatible network using Foundry Anvil and evaluated ECDSA, standalone ML-DSA, and hybrid ML-DSA configurations. The hybrid ML-DSA schemes produced signature sizes from 2,531 to 4,738 bytes, signing times from 0.237 to 0.350 ms, verification times from 0.118 to 0.173 ms, round-trip latencies from 0.355 to 0.523 ms, and estimated gas from 64,260 to 99,344. Security testing showed that the scheme rejected signature stripping, message tampering, replay attempts, forged signatures, and invalid TLV formats. These results indicate that the proposed hybrid scheme is feasible as a transition approach for post-quantum enhanced Ethereum transaction authentication, although it introduces larger payload and gas overhead than ECDSA only authentication.**

***Index Terms*— Ethereum; ECDSA; hybrid signature; ML-DSA; transaction authentication.**

I. INTRODUCTION

Blockchain technology has become an important foundation for decentralized transaction systems because it provides transparency, data integrity, and transaction authentication without relying on a centralized authority [1]. In Ethereum-compatible blockchains, transaction ownership is commonly proven through digital signatures, where ECDSA (Elliptic Curve Digital Signature Algorithm) over the Secp256k1 curve is used to identify the sender and authorize transaction execution [2]. This mechanism is efficient and compatible with the EVM (Ethereum Virtual Machine), but it also means that transaction security strongly depends on the long-term strength of elliptic-curve cryptography [3].

The main security concern of ECDSA is its vulnerability to future quantum attacks. Shor's algorithm can theoretically solve the discrete logarithm problem efficiently on a sufficiently powerful quantum computer, which would weaken elliptic-curve-based signatures [4]. In the blockchain context, this risk is significant because exposed public keys and historical transaction data may become targets for private-key recovery or signature forgery [5]. Recent studies [6], [7], [8] on quantum threats to blockchain emphasize that Bitcoin, Ethereum, and other systems using public-key signatures require migration strategies toward quantum-resistant authentication.

Post-quantum cryptography provides one possible direction to address this issue. Among the standardized post-quantum signature schemes, ML-DSA is a lattice-based digital signature algorithm standardized by NIST in FIPS 204 [9] and is intended for secure signature generation and verification against large-scale quantum adversaries. However, direct adoption of ML-DSA in blockchain systems is not straightforward because post-quantum signatures generally introduce larger public keys, larger signatures, and additional computation compared with ECDSA [10], [11]. These factors affect transaction size, gas cost, and verification design in Ethereum-compatible environments.

Recent studies have shown that blockchain systems require post-quantum protection because classical signature schemes such as ECDSA remain vulnerable to future quantum attacks [12]. Several works have proposed lattice-based and post-quantum blockchain schemes to improve transaction security and signature resistance against quantum threats [13], [14]. Other studies focused on post-quantum transaction efficiency and consensus-level protection

in blockchain environments [15], [16]. However, these works have not fully addressed practical Ethereum-compatible hybrid verification that combines on-chain ECDSA validation with off-chain post-quantum signature verification.

This study proposes an oracle-based hybrid ECDSA and ML-DSA signature scheme for Ethereum-compatible blockchain transactions. The proposed model uses Secp256k1 ECDSA to maintain compatibility with EVM-based verification, while ML-DSA is added as a post-quantum signature component. Instead of replacing Ethereum's native signature mechanism, the scheme applies independent dual signing over the same canonical transaction hash. The oracle performs off-chain hybrid signing and ML-DSA verification, while the smart contract verifies the ECDSA component on-chain and stores hashed post-quantum signature as an integrity anchor for the ML-DSA signature.

The implementation is evaluated as a proof-of-concept using a Client–Oracle–Blockchain architecture. The experiment compares standalone ECDSA, standalone ML-DSA, and hybrid ECDSA + ML-DSA under the same benchmarking procedure, including signing time, verification time, signature size, and estimated calldata gas. This study does not claim to provide full on-chain ML-DSA verification. It focuses on demonstrating a practical transition approach that keeps compatibility with Ethereum-compatible smart contracts while introducing post-quantum-enhanced transaction authentication through an oracle-assisted design.

II. METHOD

This study proposes a hybrid digital signature scheme for Ethereum-compatible blockchain environments by integrating Secp256k1 ECDSA with ML-DSA. The scheme is designed to maintain compatibility with existing EVM signature verification while introducing a post-quantum authentication layer. For each transaction, a canonical message hash is constructed from chainId, nonce, sender, receiver, algorithm, signing mode, message, and amount. This hash is signed using both ECDSA and ML-DSA, and verification follows a strict Boolean AND logic, meaning that the transaction is accepted only if both signature components are valid.

The system adopts a Client–Oracle–Blockchain architecture. The oracle performs ML-DSA signing and verification off-chain to avoid the high computational cost of post-quantum verification inside the EVM. Once both signature components are verified, the oracle submits the transaction to the smart contract. The contract verifies the ECDSA signature on-chain using ecrecover, applies nonce-based replay protection, and records keccak256(pqSig) as an integrity anchor for the ML-DSA signature. This workflow enables post-quantum-enhanced transaction authentication while preserving practical deployment compatibility with Ethereum-based smart contracts. The flow of the proposed hybrid scheme is shown in Figure 1 and Figure 2.

A. Signing Transaction Flow

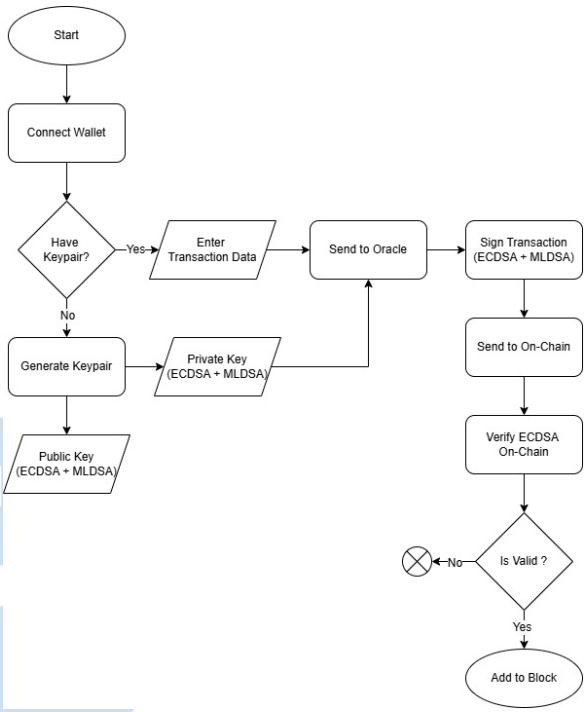

Fig 1. Hybrid scheme signing flow

The transaction signing workflow is illustrated in Figure 1. The proposed scheme follows a Client–Oracle–Blockchain architecture, where the client initiates the transaction, the oracle performs hybrid signature processing, and the blockchain smart contract validates the ECDSA component on-chain while recording the post-quantum signature anchor. The detailed workflow is described as follows.

- Wallet Connection

  The process begins when the user connects an Ethereum wallet, which represents the sender identity through a Secp256k1 ECDSA key pair. The wallet address is then used as the transaction sender and reference for on-chain signature validation.

- Hybrid Keypair Generation

  After the wallet is connected, the system checks whether a valid hybrid key pair already exists. The hybrid key combines ECDSA for Ethereum-compatible authentication and ML-DSA for post-quantum authentication. Both components are encoded in a TLV format with magic bytes, version, algorithm identifier, and key fields, making the structure easier to parse and verify. If no valid key is found, a new hybrid key pair is generated before the user enters transaction data.

- Transaction Data Input

  After the hybrid key is available, the user enters the sender address, receiver address, amount, and optional message. The oracle constructs a canonical payload from these transaction fields and the selected signing configuration on Table I.

TABLE I. TRANSACTION PAYLOAD FIELDS

Field	Description
chainId	Identifies the blockchain network
nonce	Ensures uniqueness for each transaction
sender	Address of the transaction sender
receiver	Address of the transaction recipient
algorithm	Selected cryptographic algorithm
mode	Single or hybrid signing mode
message	Optional transaction message
amount	Transaction value

  The encoded payload is then hashed using Keccak-256 to generate. Next, the oracle applies an Ethereum-oracle signing prefix, defined as *prefix = "\x19ETHEREUM-ORACLE-SIGNED:\n32"*, to produce the final signing hash *signedHash = keccak256(prefix ‖ messageHash)*. This domain-separated hash binds the signature to the intended oracle signing context. The *chainId* and *nonce* fields reduce replay risk, while *algorithm* and *mode* bind the signature to the selected signing configuration.

- Oracle Communication

  The transaction request, selected algorithm, signing mode, and hybrid private key are sent to the oracle module. The oracle serves as an off-chain layer that standardizes message formatting, generates the canonical hash, performs hybrid signing, verifies the signatures, and relays the transaction to the smart contract. This architecture is adopted because EVM smart contracts mainly support efficient ECDSA verification, while ML-DSA verification is handled off-chain. The blockchain then stores a cryptographic anchor of the ML-DSA signature for integrity reference.

- Hybrid Signature Generation

  After the canonical *signedHash* is produced, the oracle signs the same hash using both signature algorithms. First, the ECDSA private key signs the hash and produces the ECDSA signature:

$$\sigma_{\mathrm{ECDSA}} = \mathrm{Sign}_{\mathrm{ECDSA}}(\mathrm{sk}_{\mathrm{ECDSA}}, \mathrm{SignedHash}) \quad (1)$$

  Second, the ML-DSA private key signs the same hash and produces the post-quantum signature:

$$\sigma_{\mathrm{ML-DSA}} = \mathrm{Sign}_{\mathrm{ML-DSA}}(\mathrm{sk}_{\mathrm{ML-DSA}}, \mathrm{SignedHash}) \quad (2)$$

  The hybrid signature is represented as a structured combination of both components:

$$\sigma_{\mathrm{hybrid}} = \sigma_{\mathrm{ECDSA}} || \sigma_{\mathrm{ML-DSA}} \quad (3)$$

  In the API response and off-chain storage, the hybrid signature is encoded in TLV format with algorithm and signature fields. For smart contract submission, *ecdsaSig* and *pqSig* are sent separately because the contract requires the ECDSA signature in raw 65-byte format.

B. *Transaction Verification Flow*

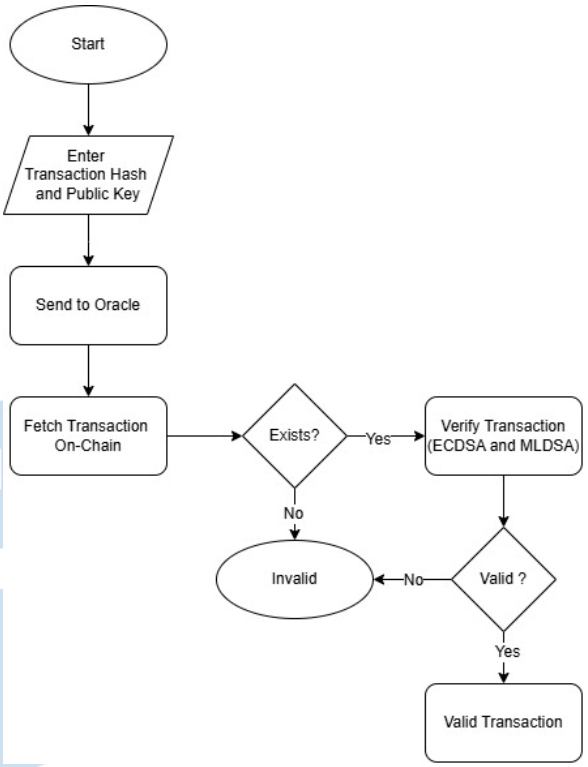

Fig 2. Hybrid scheme verifying flow

The transaction verification workflow is illustrated in Figure 2. This process is used to confirm whether a submitted blockchain transaction is valid based on the stored transaction data and the hybrid signature components.

- Input Transaction Hash and Public Key

  The verification process begins when the user or verifier enters the transaction hash and the corresponding hybrid public key. The transaction hash is used to retrieve the submitted transaction data from the blockchain, while the hybrid public key is used to verify the ECDSA and ML-DSA signature components.

- Send Verification Request to Oracle

  The transaction hash and hybrid public key are sent to the oracle. In this stage, the oracle acts as an off-chain verifier that reconstructs the transaction data and performs hybrid signature validation.

- Fetch Transaction Data On-Chain

  The oracle fetches the transaction data from the blockchain based on the provided transaction hash.

The retrieved data includes the sender address, receiver address, transaction amount, message, nonce, signing algorithm, signing mode, ECDSA signature, and the stored hash of the ML-DSA signature.

- Check Transaction Existence

  The oracle checks whether the transaction exists on-chain. If the transaction hash does not correspond to any recorded transaction, the verification process stops and the transaction is marked as invalid. If the transaction exists, the oracle continues to the signature verification stage.

- Hybrid Signature Verification

  The oracle reconstructs the canonical transaction hash using the same fields used during signing. The transaction data is first encoded as *data = abi.encode(chainId, nonce, sender, receiver, algorithm, mode, message, amount)* and then hashed with Keccak-256 to *produce messageHash = keccak256(data)*. The final signing hash is generated by applying the oracle-specific prefix, resulting in *signedHash = keccak256(prefix || messageHash)*. This ensures that verification is performed on the same deterministic payload used in the signing process.

  The oracle then verifies the two signature components:

$$\text{Verify}_{\text{ECDSA}}(\text{pk}_{\text{ECDSA}}, \text{SignedHash}, \sigma_{\text{ECDSA}}) \quad (4)$$

$$\text{Verify}_{\text{ML-DSA}}(\text{pk}_{\text{ML-DSA}}, \text{SignedHash}, \sigma_{\text{ML-DSA}}) \quad (5)$$

  The hybrid verification uses Boolean AND logic:

$$\text{Valid} = \text{Verify}_{\text{ECDSA}} \wedge \text{Verify}_{\text{ML-DSA}} \quad (6)$$

  Therefore, the transaction is valid only when both the ECDSA and ML-DSA signatures are successfully verified.

- ML-DSA Signature Hash Checking

  Because the smart contract stores the ML-DSA signature in hash form, the oracle also checks whether the hash of the provided ML-DSA signature matches the value stored on-chain:

$$\text{keccak256}(\sigma_{\text{ML-DSA}}) = \text{pqSigHash} \quad (7)$$

  This step ensures that the ML-DSA signature being verified is the same signature that was anchored during transaction submission.

- Verification Decision

If the transaction exists and both signature components are valid, the transaction is marked as a valid transaction. If either the ECDSA verification or ML-DSA verification fails, the transaction is marked as invalid. Thus, the verification flow ensures that a transaction is accepted only when it exists on-chain, matches the reconstructed message hash, satisfies ECDSA verification, satisfies ML-DSA verification, and matches the stored post-quantum signature hash anchor.

III. Result and Discussions

A. Experimental Setup

The experiment evaluates the proposed hybrid signature scheme by comparing standalone ECDSA, standalone PQC, and hybrid ECDSA + PQC under the same conditions. Secp256k1 is used for Ethereum-compatible ECDSA, while ML-DSA is used as the main post-quantum algorithm in the proposed scheme. In addition, Falcon, MAYO, and SNOVA are also included as comparative PQC algorithms to observe how different post-quantum signature families affect signature size, signing time, verification time, and estimated calldata gas. The tested PQC variants include ML-DSA-44/65/87, Falcon-512/1024, MAYO-1/3/5, and SNOVA_24_5_4/SNOVA_37_8_4/SNOVA_60_10_4.

The client side is developed using React as the transaction interface, while Wagmi and Viem are used to interact with the Web3 API and MetaMask wallet. The oracle server is implemented in Go using the Fiber framework and integrates liboqs for post-quantum signature operations. The blockchain layer runs on a private Ethereum-compatible network using Foundry Anvil. Each benchmark consists of 3 warm-up iterations and 50 measured iterations, executed on an Intel Core i5-6300U 2.4 GHz device with 16 GB RAM and Fedora 44. The reported metrics include key generation time, signing time, verification time, total signature size, and estimated calldata gas for both single and hybrid configurations.

B. Implementation Interface

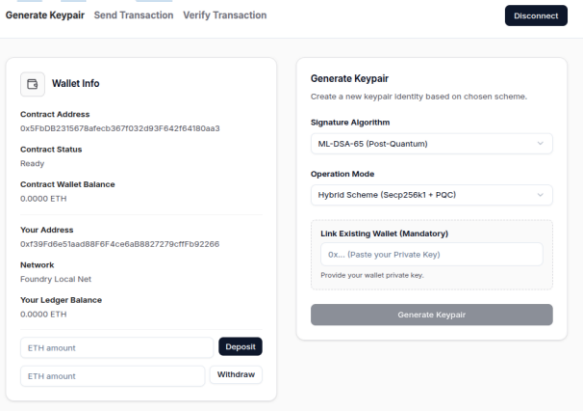

Fig. 1. Generate keypair page interface

The interface shown in Figure 1 illustrates the implementation interface for the key generation process. The interface is divided into two main panels: wallet information and hybrid key generation. The wallet information panel displays the smart contract

address, contract status, connected user address, blockchain network, and ledger balance. This information helps users verify that the system is connected to the correct Ethereum-compatible environment before performing cryptographic operations.

The key generation panel allows users to select the signature algorithm and operation mode. In this example, the system uses ML-DSA-65 as the post-quantum algorithm and combines it with Secp256k1 ECDSA in hybrid mode. The interface also requires the user to link an existing wallet private key, which is used to associate the generated hybrid key with the connected Ethereum identity. This design supports the proposed Client–Oracle–Blockchain architecture by preparing the cryptographic identity before transaction signing and verification.

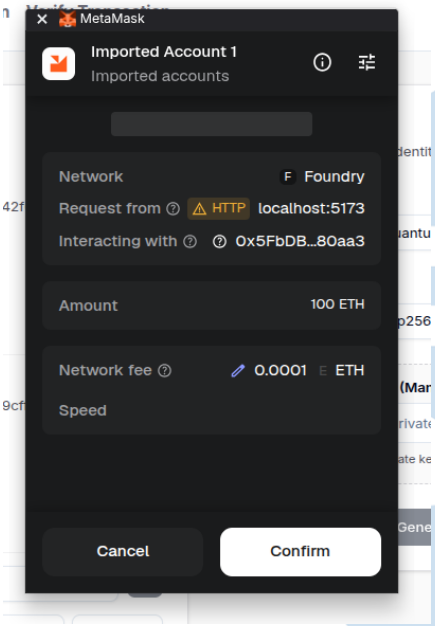

Fig. 2. Deposit ethereum token interface

The interface shown in Figure 2 presents the deposit process through MetaMask on the local Foundry network. Before the transaction is submitted, MetaMask displays important transaction details, including the connected account, target smart contract address, deposit amount, and estimated network fee. In this example, the user deposits 100 ETH into the smart contract ledger. This confirmation step ensures that the user can review and approve the transaction before it is recorded on the Ethereum-compatible local blockchain.

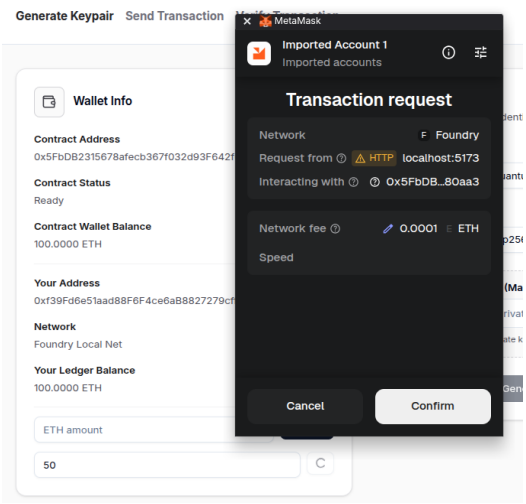

Fig. 3. Withdraw ethereum token interface

The interface shown in Figure 3 presents the withdrawal process from the smart contract ledger through MetaMask on the local Foundry network. The page displays the connected wallet information, contract balance, ledger balance, and withdrawal amount entered by the user. Before execution, MetaMask shows a transaction request containing the network, requesting application, target contract address, and estimated network fee. This confirmation process ensures that the withdrawal transaction is reviewed and approved by the user before being submitted to the Ethereum-compatible local blockchain.

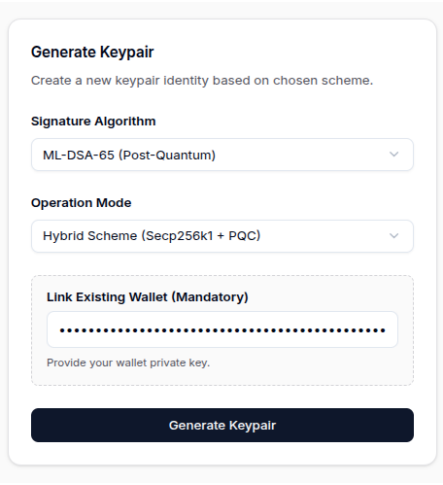

Fig. 4. Generate keypair form interface

The interface shown in Figure 4 presents the keypair generation form used to create a cryptographic identity for the transaction system. Users can select the post-quantum signature algorithm and operation mode before generating the keypair. In this example, ML-DSA-65 is selected and combined with Secp256k1 ECDSA in hybrid mode. The form also requires the existing wallet private key to link the generated hybrid key with the user's Ethereum identity. This interface

supports the initial setup phase before the user performs transaction signing and verification.

Fig. 5. Key generation result interface

The interface shown in Figure 5 presents the keypair generation result after the user completes the hybrid key generation process. The system displays the generated private key and public key in hexadecimal format, with copy and save options to support further transaction signing and verification. The private key is hidden to protect sensitive information, while a warning message reminds users not to share it with anyone. This interface emphasizes secure key management as an important part of the proposed hybrid ECDSA and ML-DSA transaction authentication system.

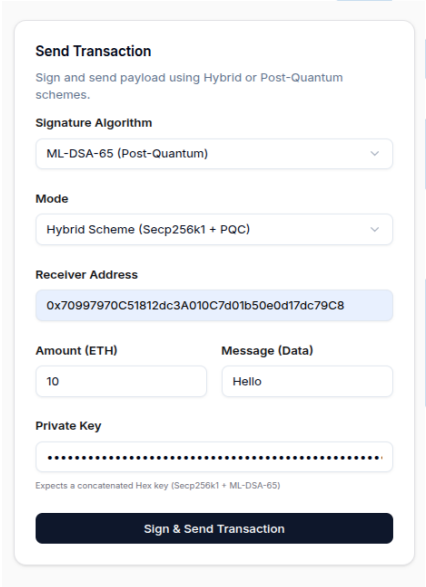

Fig. 6. Sign and send transaction interface

The interface shown in Figure 6 presents the transaction submission form used to sign and send payload data through the proposed hybrid signature scheme. The user selects the signature algorithm, operation mode, receiver address, transaction amount, message data, and private key before executing the transaction. In this example, ML-DSA-65 is combined with Secp256k1 ECDSA in hybrid mode to generate a signed transaction. This interface supports the signing process by ensuring that the transaction data and cryptographic parameters are prepared before being submitted to the blockchain system.

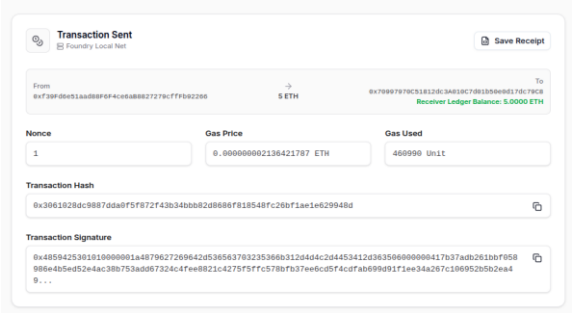

Fig. 7. Send transaction result interface

The interface shown in Figure 7 presents the transaction result after the signing and submission process is successfully completed. The system displays important transaction information, including sender address, receiver address, transferred amount, nonce, gas price, estimated gas used, transaction hash, and transaction signature. In this example, the transaction transfers 5 ETH and records the generated hybrid signature as cryptographic proof of authentication. This output confirms that the signed transaction has been processed and stored in the local Ethereum-compatible blockchain environment.

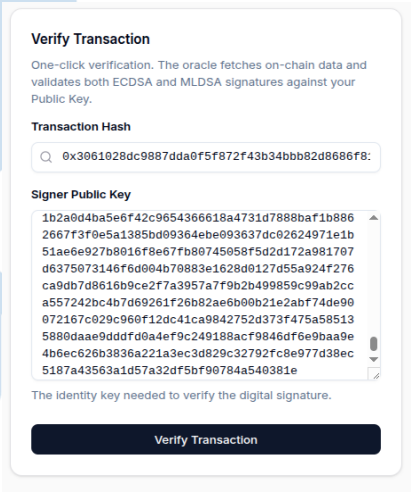

Fig. 8. Verify transaction form interface

The interface shown in Figure 8 presents the transaction verification form used to validate a submitted blockchain transaction. Users are required to input the transaction hash and signer public key before starting the verification process. The oracle then retrieves the on-chain transaction data and verifies both the ECDSA and ML-DSA signature components against the provided public key. This interface supports one-step verification, ensuring that the transaction authenticity and signature validity can be checked after the transaction has been recorded on the local Ethereum-compatible blockchain.

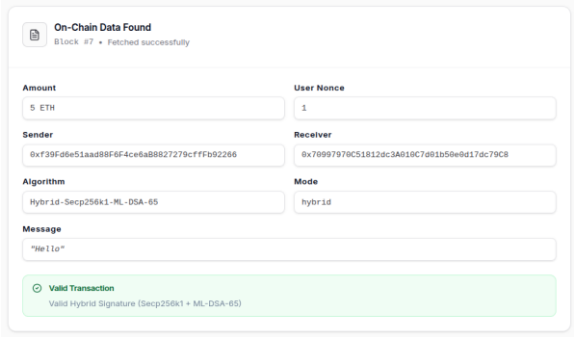

Fig. 9. Verify transaction result interface

The interface shown in Figure 9 presents the verification result after the transaction data is successfully retrieved from the blockchain. The system displays the on-chain information, including block number, amount, user nonce, sender address, receiver address, selected algorithm, operation mode, and message content. The green validation status indicates that the transaction is valid because the hybrid signature has been successfully verified. This result confirms that the oracle can retrieve on-chain data and validate both ECDSA and ML-DSA signature components correctly.

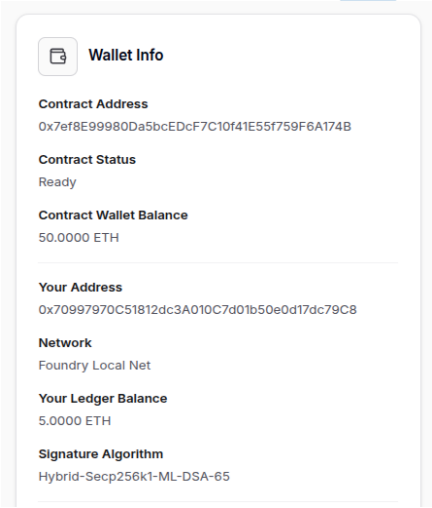

Fig. 10. Receiver wallet information

The interface shown in Figure 10 presents the receiver wallet information after the transaction process. The system displays the smart contract address, contract status, contract wallet balance, receiver address, network, ledger balance, and selected signature algorithm. In this example, the receiver ledger balance shows 5 ETH, indicating that the transferred amount has been successfully recorded in the smart contract-based ledger. This interface helps verify that the receiver account and transaction result are correctly synchronized with the local Ethereum-compatible blockchain.

C. Performance Evaluation

The performance evaluation compares three categories of signature schemes: single classical, single post-quantum cryptography (PQC), and hybrid classical–PQC schemes. The classical baseline is represented by ECDSA, while the PQC schemes include ML-DSA, Falcon, MAYO, and SNOVA at different security levels. The hybrid schemes combine ECDSA with each PQC algorithm to evaluate the additional overhead introduced when classical blockchain compatibility and post-quantum resistance are used together. The evaluated metrics include:

- Key Generation Time (ms)

  Measures the time required to generate the cryptographic key pair. This metric shows the computational cost of preparing public and private keys for each signature scheme. A lower KeyGen value indicates faster key generation and better initialization efficiency.

- Total Signature Size (Bytes)

  Measures the total signature size in bytes. For single schemes, this value refers to the size of one signature. For hybrid schemes, it includes the ECDSA signature, the PQC signature, and any additional encoding or metadata overhead. This metric is important because larger signatures increase transaction payload size and blockchain storage or calldata cost.

- Signing Time (ms)

  Measures the time required to generate a digital signature for a given message or transaction hash. A lower signing time indicates that the scheme can produce signatures more quickly, which is important for transaction submission and user-side responsiveness.

- Verification Time (ms)

  Measures the time required to validate a signature. In the hybrid scheme, verification includes checking both the ECDSA and PQC signature components. This metric is important because verification affects transaction validation performance and the workload of the oracle or verifier.

- Round-trip Time (ms)

  Measures the total time required to complete the signing and verification process in one test cycle. In this benchmark, it represents the combined latency of signing and verifying a message, not the full end-to-end blockchain transaction confirmation time. This metric is used to observe the practical execution delay of each signature scheme.

- Estimated Gas Used

  Measures the estimated gas cost related to storing or submitting the signature payload on an Ethereum-

compatible blockchain. Since larger signatures require more calldata or storage interaction, schemes with larger signature sizes usually produce higher estimated gas values. This metric is important for evaluating the blockchain deployment cost of each scheme.

Before presenting the evaluation metrics, the performance benchmarking results are first organized into two categories: single schemes and hybrid schemes. This separation helps clarify the comparison between standalone signature algorithms and combined classical–post-quantum configurations. The following figures illustrate the performance trends of each category based on signature size, signing time, verification time, round-trip latency, and estimated gas usage.

1. Single Scheme Benchmarking Result

The single scheme benchmarking evaluates the standalone performance of each digital signature algorithm before being combined into a hybrid scheme. This evaluation is important to identify the baseline cost of classical and post-quantum signature schemes in terms of key generation time, signature size, signing time, verification time, round-trip latency, and estimated calldata gas. The benchmarking results are presented in Table II.

TABLE II. SINGLE SCHEME BENCHMARKING RESULT

Algorithm	NIST Security Level	KeyGen (ms)	Total Sig (B)	Sign (ms)	Verify (ms)	Round-trip (ms)	Est. Gas
ECDSA	Insecure	0.068	65	0.096	0.062	0.158	25,828
Falcon-512	1	7.916	655	0.293	0.068	0.362	35,268
MAYO-1	1	0.098	454	0.268	0.113	0.382	32,064
SNOVA_24_5_4	1	0.139	248	0.575	0.121	0.696	28,768
ML-DSA-44	**2**	**0.046**	**2,420**	**0.119**	**0.038**	**0.158**	**63,220**
ML-DSA-65	**3**	**0.071**	**3,309**	**0.187**	**0.063**	**0.250**	**77,504**
MAYO-3	3	0.209	681	0.546	0.241	0.788	35,660
SNOVA_37_8_4	3	0.583	376	1.640	0.365	2.005	30,816
ML-DSA-87	**5**	**0.108**	**4,627**	**0.220**	**0.098**	**0.318**	**98,304**
Falcon-1024	5	25.514	1,274	0.587	0.131	0.718	45,172
MAYO-5	5	0.554	964	1.295	0.519	1.814	40,164
SNOVA_60_10_4	5	2.101	576	4.372	1.000	5.372	33,980

As shown in Table II, ECDSA has the smallest signature size and the lowest estimated gas usage, with a 65-byte signature and 25,828 estimated gas. It also achieves low signing and verification latency, with a round-trip time of 0.158 ms. However, ECDSA does not provide post-quantum security, so in this research it is used only as the classical baseline for Ethereum-compatible authentication.

For post-quantum schemes, the results show different trade-offs between signature size, latency, and estimated gas. At NIST security level 1, SNOVA_24_5_4 produces the smallest signature size, only 248 bytes, and the lowest gas usage among PQC schemes, but its signing time is higher than Falcon-512 and MAYO-1. Falcon-512 provides fast verification, while MAYO-1 offers faster key generation and moderate overall latency. This indicates that compact signature size does not always produce the fastest execution time.

At NIST security level 3, ML-DSA-65 provides the fastest round-trip time compared with MAYO-3 and SNOVA_37_8_4, reaching 0.250 ms. However, this efficiency comes with a larger signature size of 3,309 bytes and higher estimated gas consumption of 77,504. In contrast, SNOVA_37_8_4 has a much smaller signature size of 376 bytes and lower gas usage, but it has higher signing and verification latency. MAYO-3 provides a middle position, with smaller signature size than ML-DSA-65 but slower execution time.

At NIST security level 5, ML-DSA-87 achieves the fastest round-trip time among level-5 PQC schemes, with 0.318 ms. However, it also produces the largest signature size, 4,627 bytes, and the highest estimated gas usage, 98,304. Falcon-1024 has a smaller signature size than ML-DSA-87 but requires significantly higher key generation time. Meanwhile, MAYO-5 and SNOVA_60_10_4 produce smaller signatures and lower gas costs, but their signing and round-trip times are higher.

Overall, Table II shows that each signature scheme has different performance characteristics. ML-DSA is efficient in signing and verification time, but produces larger signatures and higher gas usage. SNOVA is more compact and gas-efficient, but its latency increases at higher security levels. MAYO

provides moderate signature size and gas usage, while Falcon offers fast verification but higher key generation time. These baseline results are important because they provide the foundation for evaluating the additional overhead introduced by the proposed hybrid ECDSA–PQC signature scheme in the next benchmarking stage.

2. Hybrid Scheme Benchmarking Result

Hybrid scheme benchmarking is conducted to evaluate the performance overhead introduced when ECDSA is combined with each post-quantum signature scheme. In this study, the hybrid scheme maintains Ethereum-compatible authentication through ECDSA while adding post-quantum protection through PQC signatures. Therefore, the hybrid benchmark is important to measure whether the additional security component increases signature size, signing time, verification time, round-trip latency, and estimated calldata gas. The hybrid benchmarking results are presented in Table III.

TABLE III. HYBRID SCHEME BENCHMARKING RESULT

Algorithm	NIST Security Level	KeyGen (ms)	Total Sig (B)	Sign (ms)	Verify (ms)	Round-trip (ms)	Est. Gas
Hybrid + Falcon-512	1	8.465	767	0.414	0.143	0.557	36,284
Hybrid + MAYO-1	1	0.240	562	0.383	0.188	0.572	33,056
Hybrid + SNOVA_24_5_4	1	0.281	362	0.647	0.196	0.844	29,796
Hybrid + ML-DSA-44	**2**	**0.177**	**2,531**	**0.237**	**0.118**	**0.355**	**64,260**
Hybrid + ML-DSA-65	**3**	**0.213**	**3,420**	**0.313**	**0.144**	**0.458**	**78,448**
Hybrid + MAYO-3	3	0.350	789	0.653	0.317	0.970	36,640
Hybrid + SNOVA_37_8_4	3	0.722	490	1.742	0.432	2.175	31,844
Hybrid + ML-DSA-87	**5**	**0.250**	**4,738**	**0.350**	**0.173**	**0.523**	**99,344**
Hybrid + Falcon-1024	5	23.975	1,381	0.734	0.211	0.945	46,080
Hybrid + MAYO-5	5	0.660	1,072	1.494	0.716	2.210	41,216
Hybrid + SNOVA_60_10_4	5	2.248	691	4.474	1.079	5.554	35,008

Based on Table III, the hybrid configurations generally produce larger signatures and higher gas consumption than single schemes because they include both ECDSA and PQC signature components. The amount of overhead, however, is not the same for all algorithms. Each PQC algorithm introduces different trade-offs between compactness, computation time, and gas efficiency.

For NIST security levels 1 and 2, the hybrid schemes show different trade-offs between compact signature size, latency, and gas usage. At level 1, Hybrid + SNOVA_24_5_4 produces the smallest total signature size, with 362 bytes, and the lowest estimated gas usage, 29,796. However, its round-trip time reaches 0.844 ms, which is slower than Hybrid + Falcon-512 and Hybrid + MAYO-1. Hybrid + Falcon-512 achieves the fastest round-trip time among level-1 schemes, at 0.557 ms, while Hybrid + MAYO-1 provides a close result at 0.572 ms with a smaller signature size than Falcon-512. Meanwhile, at level 2, Hybrid + ML-DSA-44 achieves the fastest overall round-trip time among all hybrid schemes, at 0.355 ms, with efficient signing and verification times of 0.237 ms and 0.118 ms. However, its total signature size reaches 2,531 bytes, increasing the estimated gas usage to 64,260. These results indicate that level-1 schemes are more compact and gas-efficient, while ML-DSA-44 at level 2 provides better computational efficiency but with higher calldata cost.

For NIST security level 3, Hybrid + ML-DSA-65 achieves the best latency result, with a round-trip time of 0.458 ms. Nevertheless, it produces a large total signature size of 3,420 bytes and an estimated gas usage of 78,448. On the other hand, Hybrid + SNOVA_37_8_4 produces a much smaller signature size of 490 bytes and lower gas usage of 31,844, but its round-trip time increases to 2.175 ms. Hybrid + MAYO-3 stands between these two schemes, offering lower signature size than ML-DSA-65 but slower execution time.

For NIST security level 5, Hybrid + ML-DSA-87 provides the fastest latency in its security level, with a round-trip time of 0.523 ms. However, it also has the largest total signature size, 4,738 bytes, and the highest estimated gas usage, 99,344. Hybrid + Falcon-1024 reduces the signature size and gas cost compared with ML-DSA-87, but it requires a much higher key generation time. Meanwhile, Hybrid + MAYO-5 and

Hybrid + SNOVA_60_10_4 provide smaller signatures and lower gas usage, although their signing and round-trip times are higher.

Overall, Table III indicates that the hybrid construction adds performance overhead, but the level of overhead varies depending on the PQC algorithm used. Hybrid schemes based on ML-DSA are more effective for systems that prioritize fast signing and verification, although they produce larger signatures and higher gas consumption. In contrast, SNOVA-based hybrid schemes offer smaller signatures and lower gas usage, but require longer execution time, particularly at higher security levels. MAYO provides a balanced option in terms of signature size and gas cost, while Falcon shows competitive verification performance but requires higher key generation time. Therefore, the selection of a hybrid scheme should consider the main system requirement, whether it focuses on latency, calldata efficiency, signature compactness, or post-quantum security level.

D. Security Analysis

Security testing was conducted to evaluate whether the proposed hybrid ECDSA and ML-DSA scheme can reject invalid signatures, modified messages, replay attempts, and malformed signature formats. As shown in Table IV, the test scenarios cover attacks against both signature components and the verification structure, including signature stripping, key or algorithm substitution, message tampering, mode confusion, replay attacks, front-running attempts, forged signatures, and TLV format manipulation.

TABLE IV. SECURITY TESTING

Threat Name	Attack Scenario	Result	Mitigation
Signature stripping	Attacker drops one component from hybrid sig	Rejected	Verifier requires both components; TLV length validation
Key or algorithm substitution	Wrong public key or algorithm downgrade	Rejected	Public key binding; algorithm_id check in TLV header
Message tampering	Tampered signature bits or wrong hash	Rejected	Cryptographic hash binding; signature malleability checks
Mode confusion	Single sig presented as hybrid or vice-versa	Rejected	Mode flag in verifier; TLV magic/version validation
Cross-key mixing	ECDSA from key A + PQC from key B	Accepted (Known Limitation)	*Documented*; mitigated by on-chain nonce + future PQC key commitment
Replay attack	Reuse of valid (hash, sig) pair with different nonce	Rejected	Hash includes nonce; on-chain nonce enforcement
Cross-sender replay	Sig for sender A reused as sender B	Rejected	Hash includes sender address; public key binding
Front-running attempt	Tampered receiver/amount in hash	Rejected	Full transaction context in hash; atomic oracle signing
Malicious relayer	Oracle signs wrong hash or uses own key	Rejected	Hash binding; public key verification; nonce uniqueness
Signature forgery	All-zero, garbage, truncated, wrong-mode sigs	Rejected	TLV magic/version validation; length checks; format parsing
TLV format manipulation	Magic, version, algorithm mismatch	Rejected	Strict TLV parser; round-trip encoding/decoding tests

The results show that all invalid cases were rejected, while valid single-signature and hybrid-signature cases were accepted. This indicates that the implementation correctly enforces the verification rules through canonical message hashing, nonce binding, public key validation, algorithm identification, and strict TLV parsing. The signed hash includes important transaction fields such as nonce, sender, receiver, amount, algorithm, and signing mode, preventing valid signatures from being reused in a different transaction context.

At the implementation level, classical signature validation is performed in the blockchain layer, while ML-DSA verification is handled by the oracle layer. The blockchain stores an integrity reference for the post-quantum signature component, allowing the system to maintain Ethereum-compatible transaction

validation while adding post-quantum authentication support.

However, the cross-key mixing scenario remains a documented limitation, as shown in Table IV. This occurs because independently valid ECDSA and ML-DSA signatures may still be combined if the signed payload does not explicitly bind both public key components. Future work should address this by adding a key commitment mechanism, such as including a hash of the post-quantum public key in the signed payload. Overall, the security results confirm that the proposed implementation can mitigate common transaction-level manipulation attacks, while still requiring stronger key binding in future development.

IV. CONCLUSION

This study implemented an oracle-assisted hybrid ECDSA and ML-DSA signature scheme for Ethereum-compatible transaction authentication. The results show that ML-DSA can be combined with ECDSA to provide post-quantum-enhanced authentication while maintaining compatibility with EVM-based smart contracts. Across the evaluated ML-DSA hybrid variants, the increase in security level also increases signature size and estimated gas cost. However, the signing and verification times remain low in the tested private Ethereum-compatible environment. These results indicate that the proposed hybrid scheme is feasible as a transition approach, where ECDSA supports blockchain compatibility and ML-DSA adds post-quantum authentication.

Security testing showed that the scheme rejected signature stripping, message tampering, replay attempts, forged signatures, and malformed TLV formats. Future work should improve key binding between ECDSA and ML-DSA components, evaluate the scheme on public Ethereum testnets, and explore EIP-712 domain separation or multi-oracle verification to reduce reliance on a single oracle.

REFERENCES

[1] N. R. Reddy, S. Suryadevara, K. G. R. Reddy, R. Umamaheswari, R. Guttula, and R. Kotoju, "Quantum secured blockchain framework for enhancing post quantum data security," *Sci. Rep.*, vol. 15, no. 1, Dec. 2025, doi: 10.1038/s41598-025-16315-8.

[2] R. Campbell and S. Fbba, "Hybrid Post-Quantum Signatures for Bitcoin and Ethereum: A Protocol-Level Integration Strategy," *JBBA*, vol. 9, no. 1, Dec. 2025, doi: 10.31585/jbba-9-1-(2)2026.

[3] R. Kysil, I. A. Seres, P. Kutas, and N. Kelecsényi, "poqeth: Efficient, post-quantum signature verification on Ethereum," in *Proceedings of the ACM Conference on Computer and Communications Security*, Association for Computing Machinery, Aug. 2025, pp. 327–343. doi: 10.1145/3708821.3736193.

[4] H. Y. Kwon, I. Bajuna, and M. K. Lee, "Compact Hybrid Signature for Secure Transition to Post-Quantum Era," *IEEE Access*, vol. 12, pp. 39417–39429, 2024, doi: 10.1109/ACCESS.2024.3374645.

[5] A. C. H. Chen, "The Security Performance Analysis of Blockchain System Based on Post-Quantum Cryptography — A Case of Cryptocurrency Exchanges," *Vietnam Journal of Computer Science*, pp. 1–30, Sep. 2025, doi: 10.1142/S2196888825500204.

[6] P. Zhang, L. Wang, W. Wang, K. Fu, and J. Wang, "A Blockchain System Based on Quantum-Resistant Digital Signature," *Security and Communication Networks*, vol. 2021, 2021, doi: 10.1155/2021/6671648.

[7] M. Seck and A. Roux-Langlois, "Towards Post-Quantum Bitcoin Blockchain using Dilithium Signature," *IACR Communications in Cryptology*, vol. 2, no. 3, pp. 1–30, Oct. 2025, doi: 10.62056/ak5wom2hd.

[8] Z. Zhang, Z. Cao, and Y. Wang, "Forensics System for Internet of Vehicles Based on Post-Quantum Blockchain," *Sensors*, vol. 25, no. 19, Oct. 2025, doi: 10.3390/s25196038.

[9] G. Raimondo and L. Locascio, "Module-Lattice-Based Digital Signature Standard," Aug. 2024. doi: 10.6028/NIST.FIPS.204.

[10] D. Marchsreiter, "Towards quantum-safe blockchain: Exploration of PQC and public-key recovery on embedded systems," *IET Blockchain*, vol. 5, no. 1, Jan. 2025, doi: 10.1049/blc2.12094.

[11] K. S. Roy, S. Singh, P. Srivathsa, R. A. Hazarika, S. M. Hassan, and K. S. Kumar, "Post-Quantum Digital Signatures for Enhanced Medical Image Security," *IET Quantum Communication*, vol. 6, no. 1, pp. 1–13, Jul. 2025, doi: 10.1049/qtc2.70006.

[12] T. M. Fernandez-Carames and P. Fraga-Lamas, "Towards Post-Quantum Blockchain: A Review on Blockchain Cryptography Resistant to Quantum Computing Attacks," *IEEE Access*, vol. 8, pp. 21091–21116, 2020, doi: 10.1109/ACCESS.2020.2968985.

[13] Y. Quan, "Improving Bitcoin's Post-Quantum Transaction Efficiency with a Novel Lattice-Based Aggregate Signature Scheme Based on CRYSTALS-Dilithium and a STARK Protocol," *IEEE Access*, vol. 10, pp. 132472–132482, 2022, doi: 10.1109/ACCESS.2022.3227394.

[14] A. Holcomb, G. Pereira, B. Das, and M. Mosca, "PQFabric: A permissioned blockchain secure from both classical and quantum attacks," in *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2021*, Institute of Electrical and Electronics Engineers Inc., May 2021. doi: 10.1109/ICBC51069.2021.9461070.

[15] S. Alsubai, A. Alqahtani, H. Garg, M. Sha, and A. Gumaei, "A blockchain-based hybrid encryption technique with anti-quantum signature for securing electronic health records," *Complex and Intelligent Systems*, vol. 10, no. 5, pp. 6117–6141, Oct. 2024, doi: 10.1007/s40747-024-01477-1.

[16] M. A. R. Javaid, M. Ashraf, T. Rehman, and N. Tariq, "Impact of Post Quantum Digital Signatures On Block Chain: Comparative Analysis," *The Asian Bulletin of Big Data Management*, vol. 4, no. 1, Mar. 2024, doi: 10.62019/abbdm.v4i1.133