Performance and Security Evaluation of Ethereum ERCs in an Industrial Environment

David Lema-Núñez, Esteban López-Lodeiro, Tiago M. Fernández-Caramés, and Paula Fraga-Lamas

Centro Mixto de Investigación UDC-Navantia, Universidade da Coruña, 15403 Ferrol, Spain
Centro de Investigación CITIC, Universidade da Coruña, 15071 A Coruña, Spain
Navantia S. A., Unidad de Producción de Ferrol, 15403 Ferrol, Spain
Correspondence: david.lema.nunez@udc.es

DOI: *https://doi.org/10.17979/spu.23.c58*

Abstract: Ethereum use as a decentralized platform for executing smart contracts has driven the adoption of standards that optimize interoperability in industrial environments. Ethereum Requests for Comments (ERC) establish uniform patterns for smart contracts, facilitating their integration and operation in network nodes, which are essential for industrial applications such as supply chain management or process automation. However, this standardization can propagate vulnerabilities or inefficiencies in critical systems if the contracts are not optimized, affecting the reliability of industrial processes. Since operations in Ethereum consume computational resources (measured in gas) with associated economic costs, poor design can lead to significant losses in industrial settings. This paper evaluates the efficiency and security of ERC standards by examining their functional diversity and technical complexity. Thus, it analyzes existing implementations to identify common errors and proposes improvements to enhance the robustness and optimization of three of the most popular standards: ERC-20, ERC-1400 and ERC-3643. The ultimate goal is to support the effective adoption of ERCs in industrial applications. Considering the Ethereum network incentives on lower complexity logic and the obtained results, it is advisable to use simple standards, which also reduce error risks and ease maintainability.

1 Introduction

Industrial asset management (e.g., machines, products, licenses) using blockchain technology enables advancements in automation, traceability, transparency and decentralized factory models. Ethereum, as a programmable decentralized network, requires standards to ensure interoperability across its nodes. Ethereum Requests for Comments (ERC) standards define smart contracts, ensuring compatibility and intercommunication. However, a non-optimized design may propagate vulnerabilities and inefficiencies. As Ethereum operations consume gas, which translates into economic costs, an inefficient design can lead to significant losses in industrial environments.

This paper analyzes the efficiency and security of three prominent ERC standards: ERC-20, ERC-1400 and ERC-3643, selected based on their relevance, functional diversity and technical complexity. The evaluated key metrics include gas consumption, which reflects computational complexity, and deployment and initialization latency. In some cases, achieving an operational state requires additional steps, potentially increasing costs. In addition, gas consumption and latency for commonly used operations are studied, with the transfer operation serving as the primary focus of analysis due to its universality across all ERC standards and its utility as a reference.

Regarding security, specific tests were designed for each standard, alongside general cases of interest, with the objective of identifying common vulnerabilities in smart contracts. Specifically, the main goal was to verify whether the evaluated implementations met minimal security criteria and to detect exploitable logic. To facilitate a deeper evaluation, some implementations incorporated common development issues arising from misinterpretations or ambiguities in the analyzed standards.

Thus, the proposed tests compare efficiency (gas consumption and latency) and security across the implementations of each standard, aiming for a balance between performance and robustness. This analysis provides a useful guidance for developers on how to minimize risks and maximize efficiency.

2 ERCs

2.1 ERC-20

The ERC-20 standard, defined in Vogelsteller and Buterin (2015), is arguably the most widely used protocol, primarily employed for creating fungible tokens on Ethereum. Its defined functions include basic operations for managing and transferring tokens, such as direct transfers, balance checking and token transfer delegation. Its widespread adoption ensures that tokens developed under this standard are supported by most wallets, exchanges and various smart contracts.

An ERC-20 token holder owns these assets and can transfer them without intermediaries. However, an erroneous transaction to an address cannot be reversed due to the immutable nature of the Ethereum blockchain. The only possible methods for retrieving these tokens rely entirely on the goodwill of the recipient or legal action against the address owner.

In contrast to Ether, the native cryptocurrency of Ethereum used to pay transaction fees (proportional to gas), ERC-20 tokens are digital assets created through a smart contract. These customized tokens may represent specific project coins or tokenized physical assets, such as national banknotes, similar to those deployed by Circle Internet Group, Inc (2022); Tether Operations (2017). It is important to note that, as all the performed computations occur on the Ethereum network, all operations with ERC-20 tokens require a fee paid in Ether.

2.2 ERC-1400

The ERC-1400 standard is a modular framework for security tokens on Ethereum, compatible with ERC-20, as it implements its basic functions (transfers and balance checking) while extending them to meet the financial and administrative requirements outlined in Stephane (2018). It incorporates specifications such as ERC-1410 (partitioned tokens) of Dossa, Adam (A) (2018), ERC-1594 (issuance and transfers) of Dossa, Adam (B) (2018), ERC-1643 (document management) of Dossa, Adam (C) (2018), and ERC-1644 (forced transfers) of Dossa, Adam (D) (2018), ensuring interoperability with wallets and platforms that support ERC-20 while including functionalities for regulated securities.

The main feature of ERC-1400 is its partitioned tokens, where tokens are divided into disjoint subsets with specific transfer rules and legal restrictions as specified in Dossa, Adam (A) (2018). Tokens within a partition are fungible within that domain but non-fungible across different partitions, enabling capabilities to simulate complex financial instruments.

Moreover, ERC-1400 defines operator and controller roles. Operators manage partitions on behalf of their owners, which is useful for delegating token custody, while controllers can enforce forced transfers in cases such as fraud or other legal requirements, as outlined in Dossa, Adam (D) (2018). In addition, ERC-1643, specified in Dossa, Adam (C) (2018), allows participants to register legal document metadata on the blockchain, including information such as name, URI or hash.

2.3 ERC-3643

The ERC-3643 standard, described in Lebrun et al. (2023), facilitates the tokenization of regulated securities, building upon ERC-20 to ensure compatibility with DeFi platforms and wallets while incorporating features for regulatory compliance as outlined in Lebrun et al. (2021); Vogelsteller and Buterin (2015). Unlike ERC-20 and ERC-1400, ERC-3643 tokens are permissioned, restricting transfers to verified addresses that meet the regulatory criteria specified by the contract deployer, such as Know Your Customer (KYC) and Anti-Money Laundering (AML) requirements. Its modular architecture is divided into three components: token transfer logic, identity management and compliance policies. The latter enables features such as token pausing, token freezing, forced transfers, asset recovery and batch operations.

Identity verification is conducted via ONCHAINID, as described in Tokeny Sàrl (2017), using ERC-734 and ERC-735 to manage keys and claims in accordance with Vogelsteller, Fabian (A) (2017); Vogelsteller, Fabian (B) (2017) specifications. Each user has a public key linked to an ONCHAINID, with claims signed by trusted issuers responsible for validating relevant attributes, such as jurisdiction or legal investor status, without exposing sensitive private data. An authorized agent verifies these claims in the Claim Topics Registry, confirms the presence of issuers in the Trusted Issuers Registry, and registers compliant identities in the Identity Registry. Additional transaction restrictions, such as geographical or temporal constraints, are specified by the Compliance Registry.

All transactions require mandatory ONCHAINID identification with validated claims from trusted issuers, ensuring automatic verification of criteria such as token retention limits or additional requirements discussed in Lebrun et al. (2021, 2023). Non-compliant operations are rejected, and contract deployers may expand, reduce or modify these rules as needed.

3 Test environment

Security tests were performed using the Hardhat test network developed by Nomic Foundation (2019). This network simulates a local blockchain without the latency, approval and propagation issues associated with nodes in real environments. The primary objective of creating such an environment is to detect logical errors in implementations to assess their potential impact on end users.

The ERC-20 tests consisted of a series of standard compliance verifications based on **OpenZeppelin** implementations in Zeppelin Group Ltd (2016), **USDT** from Tether Operations (2017), and **BNB** (Ethereum) from Binance Team (A) (2017). These three tokens are of significant interest because **OpenZeppelin** provides the recommended framework for developing ERC-20 tokens, **USDT** is the ERC-20 token with the most holders, and **BNB** is the second most capitalized token, despite lacking support since 2019, as noted in Binance Team (B) (2019). Figure 1 shows a screenshot of Etherscan with currently widely used ERC-20 tokens.

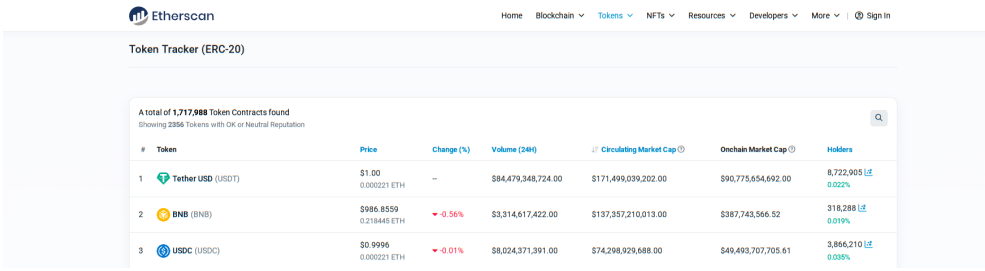

Figure 1: A total of 1,715,988 Token Contracts found in accordance to Etherscan (2025).

For ERC-1400 and ERC-3643, due to the absence of mature implementations, the tested ver-

sions were developed to incorporate common development errors derived from their specifications or ambiguities in critical areas. For ERC-1400, common errors stem from inadequate implementations of token partition mechanisms and incorrect privilege management in the controller role. In the case of ERC-3643, potential verification mechanisms and interactions with elements external to the blockchain were analyzed.

Regarding performance, each implementation was evaluated on the Ethereum testchains Sepolia and Holesky via an RPC (Remote Procedure Call) node provided by PublicNode (2022). The Sepolia network is a compact test environment for small and rapid tests, while Holesky aims to replicate the Ethereum mainchain environment. Latency and gas consumption are the primary metrics used to evaluate deployment, initialization and transfer operations or variations thereof. The focus on the transfer operation is due to it being probably the most frequently used operation across the evaluated ERC standards.

4 Results

4.1 Security analysis

A common security flaw across all implementations of the evaluated standards is the double allowance issue for token transfers, originating from the widely used ERC-20 standard. This flaw, illustrated in Figure 2, occurs when a third party is authorized to transfer tokens from a wallet. If the amount of tokens authorized for transfer is updated, the delegate may have already transferred tokens beyond the new limit. Consequently, the owner's action, intended to prevent further transfers, inadvertently authorizes additional transfers.

The only implementation that offers a solution (although an *ad hoc* one) is the ERC-20 USDT contract. Once an allowance is granted, it can only be redefined as zero, ensuring that the token holder is aware that no further transfers will occur after the operation is validated. In the case of OpenZeppelin, they acknowledged the existence of the issue and provided an example of a proposed solution. However, such a proposal requires the addition of non-standard operations while preserving the vulnerable logic inherent in ERC-20 compliance. This is the reason why OpenZeppelin decided not to adopt this proposal in their ERC-20 implementation.

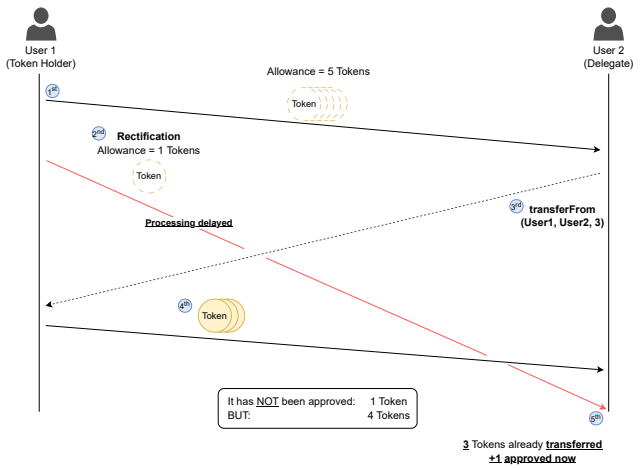

Figure 2: Token transfer deceiving allowance.

For ERC-1400, it is critical to properly implement and review all operations related to the controller role, as the standard lacks a generic interface, allowing this role to exercise significant

authority over token holders. This situation may lead to potential abuses and token freezing scenarios, particularly when user-created partitions are involved. Another important consideration is that incorrect implementation of ERC-20 compatibility may cause issues when retrieving information about wallet balances and the total token supply. The standard requires treating all partitions as non-existent to calculate the balance and total supply, which can result in overflow and underflow issues.

In ERC-3643, it is essential to thoroughly review each Compliance module (the "laws") added to the smart contract. These modules alter the standard functionality of the contract by imposing transfer restrictions between users. Even if the standard has been audited, added modules may not inherit the same security guarantees, increasing contract complexity, which leads to higher fees and reduced efficiency.

Another relevant feature of ERC-3643 is its attempt to incorporate all relevant security exchange logic within the blockchain. However, it is likely that these tokens will need to integrate off-chain interactions, potentially with regulatory authority endpoints. In such cases, trust anchors such as oracles described by Fernández-Blanco et al. (2025) are critical. It is essential to ensure their trustworthiness, robustness and resistance to unauthorized manipulation. Otherwise, the blockchain validation mechanisms may fail to maintain consistency.

4.2 Performance analysis

In order to evaluate performance, gas was selected as the metric, focusing on the latency between the operation request and its finalization. It is important to note that gas consumption should remain constant across all testchains; however, it is contingent upon the version of the Ethereum Virtual Machine (EVM) in use.

Deployment and initialization

In Figure 3 the average gas consumption per testchain is presented. ERC-20 implementations require no gas consumption for initialization, as contract deployment is the only necessary operation. Notably, **USDT** and **BNB** consume more gas than the **OpenZeppelin** implementation due to their higher code complexity.

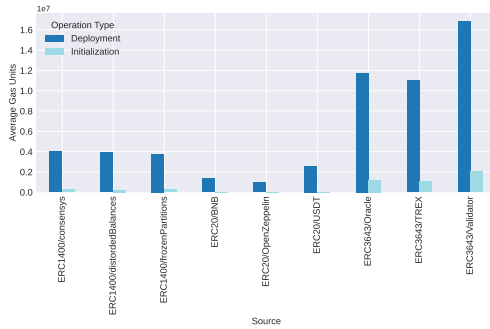

Figure 3: Average deployment and initialization gas consumption per ERC implementation.

For ERC-1400, there are minimal variations, as the code across implementations is quite similar. In contrast, ERC-3643 exhibits the most significant differences, particularly in the **Validator** version, where the identity verification logic for transfers has been shifted to the deployment and initialization phase. The higher gas consumption of the **Oracle** version can be attributed to the addition of an extra Compliance module, which increases complexity.

Regarding latency shown in Figure 4, the trends observed in deployment gas consumption persist, except for ERC-3643, where the initialization process results in considerably higher la-

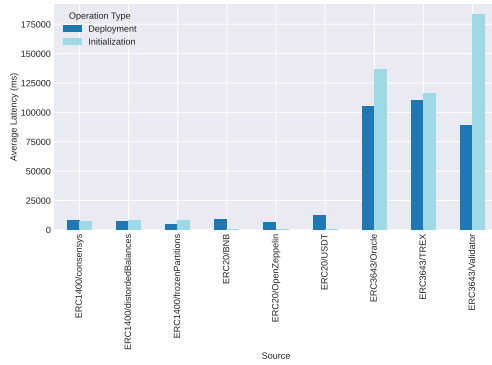

Figure 4: Average deployment and initialization latency per ERC implementation.

tency. This is due to the user registration process, which requires multiple verification steps before any transaction can occur. It is important to note that initialization is inherently slower than deployment, as it involves a series of sequential steps compared to a single standardized deployment request.

Token Transfers

The gas consumption for ERC-20 token transfers, as shown in Figure 5, follows the same pattern as contract deployment: higher deployment complexity leads to increased transfer complexity, resulting in greater gas consumption. In the `ERC-1400/distortedBalances` implementation, generic transfers are not supported due to its lack of ERC-20 compatibility. However, all `transferByPartition` operations across ERC-1400 implementations exhibit lower gas requirements. This is because the user specifies the working partition in these operations, relieving the system from computing the appropriate partition, thereby reducing operational complexity.

For ERC-3643, as presented in Figure 5, the `batchTransfer` operation, which enables multiple generic transfers simultaneously, is gas-intensive. In the **Oracle** version, the application of Compliance module rules increases transfer complexity compared to the standard **T-REX** implementation. This is particularly notable when compared to the **Validator** version, which has the lowest gas consumption per operation due to the relocation of identity verification logic to the initialization phase.

Regarding latency, as shown in Figure 6, the Sepolia testchain exhibits considerable variations, likely due to its small network size and lack of stability. In contrast, the Holesky testchain is significantly more stable, with fewer oscillations recorded across all ERC implementations. Variations in latency across the ERC standards are relatively similar, which can be attributed to network saturation, suggesting that there is no significant time difference in transfer invocations despite differences in their complexity.

5 Conclusions

In terms of security, reducing code complexity benefits the evaluated standards. ERC-20, in particular, exhibits fewer vulnerabilities in its operations and well-established implementations are available to address and mitigate known issues. Furthermore, as a well-established standard, many of ERC-20's issues are already documented.

ERC-1400 illustrates how advanced features can become problematic without precise specifications. The standard ambiguity creates numerous opportunities for exploitation due to pro-

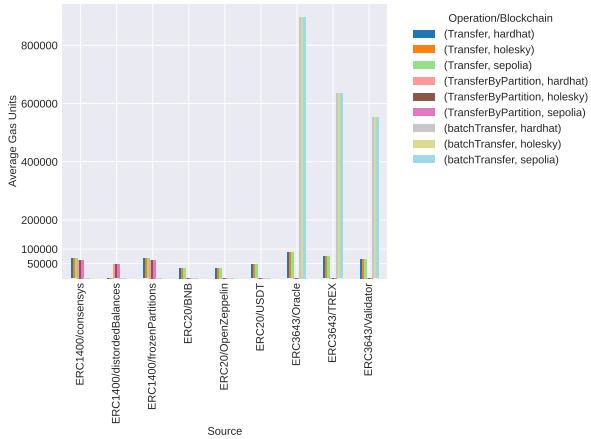

Figure 5: Average gas consumption per network transfer operation.

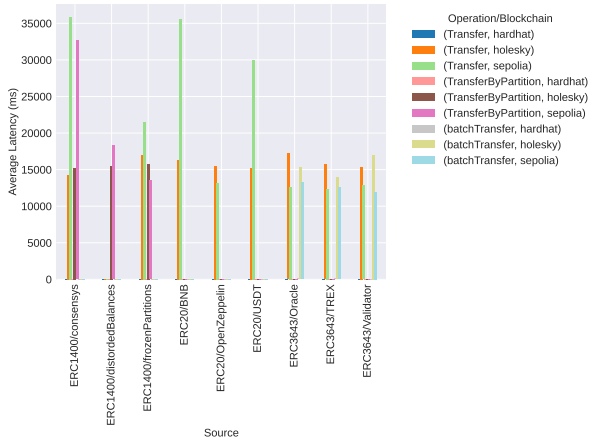

Figure 6: Average latency per network transfer operation.

grammer misimplementations, which is particularly concerning given its more complex codebase in comparison to ERC-20.

ERC-3643 has undergone a rigorous development and assessment process, including an external audit to ensure that its complex mechanisms function as intended without significant errors. However, the standard is notably complex, gas-intensive and prone to misconfiguration.

In conclusion, it is advisable to use the least complex standard, as it is easier to maintain and more cost-effective. Well-established standards are also preferable, as their known issues are documented, and best practices and reliable implementations are likely to have been developed.

6 Acknowledgments

This work has been supported by Centro Mixto de Investigación UDC-NAVANTIA (IN853C 2022/01), funded by GAIN (Xunta de Galicia) and ERDF Galicia 2021–2027.

Bibliography

Binance Team (A). BNB Ethereum ERC-20 Token Source Code, 2017. URL *https://vscode.blockscan.com/ethereum/0xB8c77482e45F1F44dE1745F52C74426C631bDD52*.

Binance Team (B). Binance Completes BNB Mainnet Swap and Opens Deposits / Withdrawals — Binance, 2019. URL *https://www.binance.com/en/support/announcement/detail/360027291331/*.

Circle Internet Group, Inc. EURC — A Euro-Backed Stablecoin, 2022. URL *https://www.circle.com/eurc/*.

Dossa, Adam (A). ERC 1410: Partially Fungible Token Standard, 2018. URL *https://github.com/ethereum/EIPs/issues/1410/*.

Dossa, Adam (B). ERC 1594: Core Security Token Standard, 2018. URL *https://github.com/ethereum/eips/issues/1594/*.

Dossa, Adam (C). ERC-1643: Document Management Standard, 2018. URL *https://github.com/ethereum/eips/issues/1643/*.

Dossa, Adam (D). ERC-1644: Controller Token Operation Standard, 2018. URL *https://github.com/ethereum/eips/issues/1644/*.

Etherscan. Token Tracker — Etherscan, 2025. URL *https://etherscan.io/tokens*.

G. Fernández-Blanco, P. García-Cereijo, T. M. Fernández-Caramés, and P. Fraga-Lamas. Hands-On Blockchain Teaching and Learning: Integrating IPFS and Oracles Through Open-Source Practical Use Cases. *Education Sciences*, 15(9), 2025. URL *https://www.mdpi.com/2227-7102/15/9/1229*.

J. Lebrun, T. Malghem, K. Thizy, L. Falempin, and A. Boudjemaa. ERC-3643: T-REX - Token for Regulated EXchanges, 2021. URL *https://eips.ethereum.org/EIPS/eip-3643/*.

J. Lebrun, L. Falempin, K. Thizy, T. Malghem, X. Aznal, T. Bousselin, and F. Croiseaux. Whitepaper ERC3643 The T-REX protocol, 2023. URL *https://tokeny.com/wp-content/uploads/2023/05/ERC3643-Whitepaper-T-REX-v4.pdf*.

Nomic Foundation. Hardhat — Ethereum development environment for professionals by Nomic Foundation, 2019. URL *https://hardhat.org/*.

PublicNode. PublicNode, 2022. URL *https://ethereum.publicnode.com/*.

Stephane. ERC-1400: Security Token Standard, 2018. URL *https://github.com/ethereum/eips/issues/1400/*.

Tether Operations. USDT Ethereum ERC-20 Token Source Code, 2017. URL *https://vscode.blockscan.com/ethereum/0xdac17f958d2ee523a2206206994597c13d831ec7*.

Tokeny Sàrl. ONCHAINID The identity system for compliant digital assets, 2017. URL *https://cdn.prod.website-files.com/60ed5607a0d4556dd864b950/619367fe1ca0963a9bcb2edf_OID%20Token%20-%20Whitepaper%20-%20V%201.0%20.pdf*.

F. Vogelsteller and V. Buterin. ERC-20: Token Standard, 2015. URL *https://eips.ethereum.org/EIPS/eip-20*.

Vogelsteller, Fabian (A). ERC: Key Manager, 2017. URL *https://github.com/ethereum/eips/issues/734/*.

Vogelsteller, Fabian (B). ERC: Claim Holder, 2017. URL *https://github.com/ethereum/eips/issues/735/*.

Zeppelin Group Ltd. OpenZeppelin Contracts is a library for secure smart contract development., 2016. URL *https://github.com/OpenZeppelin/openzeppelin-contracts/*.