Ethereum token standard catalog

in-house seminar

Jeongho Jeon

2025-09-09

DSRV

version 20250909

Quick Reference

- Ethereum token
- ERC-20
- receive function
- refund (escrew)
- gasless with Permit
    - ERC-2612 (ERC20Permit)
    - ERC-3009 Authorization
- memo
- ERC-3643 Permissioned Token
- ERC-7291 PBM
- ERC-4626 Vault
- rebase token
- Fireblocks ERC20F
- privacy
- ZKPassport
- token checklist

Ethereum token

- native token **ETH** vs smart contract token **ERC-20**
- **WETH** (wrapped ETH) makes ETH usable like an ERC-20.
- Protocols (e.g. Uniswap) auto-wrap ETH for simpler implementation.
- ERC-7528 defines ETH's virtual token address as `0xeeee…ee`.

Token Model

- Each token smart contract keeps a balance sheet in its own state (storage).
- To know an account's holdings, you must query all token contracts.
- Some chains like Aptos, Sui, and NEAR store all token information under the account.

Ethereum token

Token Storage

- Token contracts store a `account → balance` mapping.
- Because accounts are hashed (see “Layout of State Variables in Storage”), you cannot read a balance table from contract storage directly.
- Only ERC-20 transfers emit `Transfer` events, and ETH can be moved via internal transactions.
- You need EVM execution traces or high-level APIs to track both ETH and ERC-20 transfers.

Standards

EIP	(Ethereum Improvement Proposal)
ERC	(Ethereum Request for Comment) subset of EIPs for token and contract interoperability

Beginning of all 20s: ERC-20

`name()`	
`symbol()`	
`decimals()`	
`totalSupply()`	
`balanceOf(owner)`	
`transfer(to,value)`	emit a `Transfer` event
`transferFrom(from,to,value)`	indirect transfers via swap smart contracts and so on
`approve(spender,value)`	maximum amount that a contract can transferFrom() on behalf of a user, emit an `Approval` event
`allowance(owner,spender)`	how much approval is left

receive function (transfer hook)

When sending ETH to a contract, `payable receive()` is called. This function can execute code or reject transfer. There are experimental proposals for ERC-20, inspired by ERC-721's `safeTransferFrom()` + `onERC721Received()`. `safeTransferFrom()` triggers `onERC721Received()` of the recipient contract.

ERC-223	`transfer(to,value,data)` + `tokenReceived()`, not ERC-20 compatible
ERC-677	`transferAndCall(to,amount,data)` + optional `onTokenTransfer()`, example: LINK
ERC-777	`tokensToSend()` + `tokensReceived()` with hook registry and authorized operators
ERC-1363	`transferAndCall/transferFromAndCall(…,data)` + `onTransferReceived()`, `approveAndCall(…,data)` + `onApprovalReceived()`

refund (escrew)

Example: Circle Refund Protocol circlefin/refund-protocol

`constructor(arbiter,token,…)`

`pay(to,amount,refundTo)` manages balances separately from the internal token

`refundByRecipient(paymentID)` callable only by recipient

`refundByArbiter(paymentID)` if recipient balance is insufficient, refund by borrowing from arbiter

`withdraw(paymentIDs)` repays arbiter first (`settleDebt()`), updates internal token balances

`earlyWithdrawByArbiter(…)` withdraw with an optional fee before lockup, requires recipient signature

`updateRefundTo(paymentID,newRefundTo)` changes refund target

gasless transfer with Permit

`approve()` and `transferFrom()` in ERC-20 are not atomic, which can lead to race condition or reentrancy attacks. For convenience, many users grant an infinite allowance, which has been exploited in hacks. Then the **Permit** mechanism was introduced.

(see Abstract Account slide for Coinbase SpendPermission and MagicSpend.)

ERC-2612 (aka ERC20Permit)

- `permit(owner,spender,value,deadline,v,r,s)` serves the same role as `approve()`.
- `v,r,s` is an owner's signature of `{owner, spender, value, nonce, deadline}`.
- `nonce` (not in parameters) comes from `nonces(owner)`, and is incremented by one per `permit()` to prevent replay attacks.
- Someone else may pay the transaction fee and send the transaction with the owner's off-chain signature.
- `deadline` specifies the validity of the signature as a Unix timestamp (seconds since epoch, 1970-01-01). `permit()` must be called before this deadline.

Two usage patterns

- a user calls the smart contract that `permit()` a contract and `transferFrom()` within the same transaction.
- a user `permit()`s a recipient contract. Someone else calls the recipient contract that calls `transferFrom()` later.

ERC-3009: Transfer With Authorization

- No reliance on ERC-20 allowance. All transfers require authorization.
- `transfer/receiveWithAuthorization(from,to, value,validAfter,validBefore,nonce,v,r,s)`
- `v,r,s` is an owner's signature of `{from, to, value, validAfter, validBefore, nonce}`.
- A third party can submit the transaction and pay gas with the signature.
- Differ from ERC-2612: nonce need not be sequential, only unique
- ERC-2612 examples: USDC, USDe, USDS, ~~DAI~~
- ERC-3009 example: USDC (written by Peter Jihoon Kim at Coinbase)

memo

Example: Circle Recibo circlefin/recibo

- Attach a message to a transfer, example: invoice or encrypted Travel Rule identity
- A user call Recibo contact's `transferFromWithMsg(to,amount,msg)`. Recibo contract emits an event and calls the token contract's `transferFrom()`.
- A user must `approve()` a Recibo contract in advance.
- Events: `TransferWithMsg`, `ApproveWithMsg`, `SentMsg` without a transfer

If the token contract supports **Permit**,

ERC-2612	`permit[AndTransferFrom]WithMsg(…,`**`permit`**`,`**`msg`**`)`
ERC-3009	`transferWithAuthorizationWithMsg(…,`**`authorization`**`,msg)`

ERC-3643 Permissioned Tokens

- T-REX, Token for Regulated EXchanges, by Tokeny
- On 2025-03-20, DTCC (Depository Trust & Clearing Corporation) joined the ERC-3643 Association.
  On 2025-07-16, ERC-3643 Association met SEC and requested to incorporate ERC-3643 into SEC-sponsored sandboxes.
- Transfer checks
  - KYC `tokenIdentityRegistry.isVerified(to)`
  - `tokenCompliance.canTransfer(from,to,amount)`
    (ERC-1404 Simple Restricted Token and Polymath ST-20 require only a similar function)
    (`tokenCompliance.transferred(from,to,amount)`)
- mint/burn, forcedTransfer, full/partial freeze, pause, batch*
- `recoveryAddress(lostWallet,newWallet,investorOnchainID)`
- ERC-1400 Security Token Standard by Consensys is similar.

ONCHAINID (on-chain identity) with ERC-3643

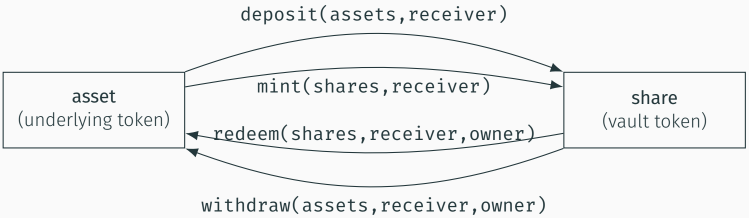
- `IIdentityRegistry.registerIdentity(userAddress,` **`identity`**`,country)`
- identity = ERC-734 Key Manager + ERC-735 (extended ERC-725) Claim Holder, proposed by Fabian Vogelsteller, the creator of ERC-20 and web3.js
- Each address has one identity contract, and an identity holds multiple claims.
- ERC-734 manages keys by type and purpose, and allows `execute(to,value,data)` with `approve(id,bool)`.
- Claims can be verified with claim issuers using `isClaimValid(identity,claimTopic,sig,data)`.

ERC-7291: Purpose Bounded Money (PBM)

- ERC-1155 (multi-token/NFT container) token that wraps underlying tokens
- `safeMint()` - PBM contract takes ownership of the underlying token, which requires approve()
- $\Leftrightarrow$ `unwrap()` - burn PBM token and redeem underlying token
- on-redemption checks: idenity (`isBlacklisted(address)`, `isMerchant(address)`), expiry, and so on
- supports ERC-721 `safeTransferFrom()`
- use cases: government vouchers, conditional disbursements, escrow-style payments
- In 2023, the Monetary Authority of Singapore (MAS) released the PBM whitepaper through the CBDC research, Project Orchid.
- Examples: Singapore-based StraitsX PBM, Sooho PBM (Bank of Korea's CBDC pilot 'Project Hangang')

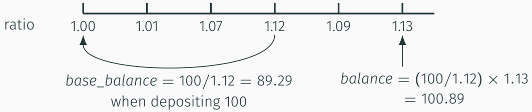
ERC-4626 Tokenized Vault Standard

- A yield-bearing token that changes in value based on yield or interest.
- EIP-4626 defines a minimal interface for exchanging assets (underlying tokens, ERC20) and shares (vault tokens, ERC-4626).
- The exchange rate is determined by the asset/share ratio, adjusted by asset deposits or withdrawals (beyond the scope of EIP) as the vault's value changes.
- Example: sUSDe (Staked USDe)



rebase token

- Rebase tokens change token quantity, while EIP-4626 tokens change token value.
- Don't use share tokens. Balances are calculated by
  $balance = base\_balance \times ratio$
  On mint: $base\_balance = balance/ratio$
- The operator periodically adjusts the ratio to reflect value changes (e.g., staking rewards).
- Because balances change continuously, rebase tokens are hard to use in DeFi.
- Can wrap rebase tokens into EIP-4626-style shares (e.g., stETH $\rightarrow$ wstETH, wrapped stETH).


ratio
1.00
1.01
1.07
1.12
1.09
1.13
$base\_balance = 100/1.12 = 89.29$
when depositing 100
$balance = (100/1.12) \times 1.13$
$= 100.89$

Fireblocks ERC20F

- fireblocks/fireblocks-smart-contracts
- `grant/revokeRole()` and fine-grained roles like `UPGRADER_ROLE`, `BURNER_ROLE` and `SALVAGE_ROLE`
- Both sender and recipient must be in the allowlist.
- ERC-2612 Permit and safer `increase/descreaseAllowance(spender,value)`
- `recoverTokens(address,amount)` revokes tokens from unlisted addresses
- rescue ERC-20 (`salvageERC20()`) or ETH (`salvageGas()`) accidentally sent to the token contract
- `mint/burn()`, `pause/unpause()`
- ERC-2771 meta transaction friendly gasless version, ERC20FGasless
- Example: WUSD (Worldwide USD), FRNT (Wyoming's Frontier)

privacy

Economic privacy is important for B2B contracts, salaries, spending details, and so on. Unlike cash or e-cash, blockchains are inherently transparent. Privacy solutions have been proposed using techniques such as encryption, ZKP, and FHE.

encryption	J.P. Morgan’s Quorum blockchain separates public state from privacy state and provides privacy transactions.
ZKP	Zerocash (the basis of Zcash and Tornado Cash) hides the sender, recipient, and amount by building a shielded pool. Various zkERC20 proposals, ERC-1724 Confidential Token, and EY Nightfall L2 fall into this category. Since Zerocash’s UTXO model is unsuitable for DeFi, there have also been attempts to hide the account state in the account-based model, such as Zether.
FHE	Several solutions have emerged leveraging Zama’s FHE smart contract library.

*Passport

- ZKPassport and Self (OpenPassport, by Celo Marek)
- NFC reader extracts e-passport data and signature, that are encrypted with MRZ info to prevent unauthorized reading.

Machine Readable Zone (MRZ)
P<D<<MUSTERMANN<<ERIKA<<<<<<<<<<<<<<<<<<<<<<
C01XYCCG91D<<6408125F2702283<<<<<<<<<<<<<<<8

- Present ZKP proving age, nationality, and issuing country conditions.
  - age $\geq$ 18
  - nationality not in SANCTIONED_COUNTRIES
- Limitations: ~~forged signatures~~, identity cloning, passport possession rate, passport revocation

Our token checklist

- mint/burn, allowlist/denylist, freeze
- refund, escrow
- AA-friendly (gas sponsorship)
- KYC, AML, compliance, proof of reserve, trade record
- upgradeable contract with storage slot (EIP-1967)
- address and QR code generation
- social login and recovery
- on/off-ramp, oracle services
- monitoring, emergency action, pause, rescue

- ERC-3643 needs pre-arranged identity info. How about on-demand indentity? Also, overhead of one contract per identity.
- scalability: ERC-7291 `Blacklist(…,addresses,…)` event with 8 billion people?
- scalability: 22,100 gas (32 bytes) $\times$ 8b $\times$ 1 gwei/gas $=$ \$707m at \$4000/ETH $\longrightarrow$ accumulator-based, interface with proof input