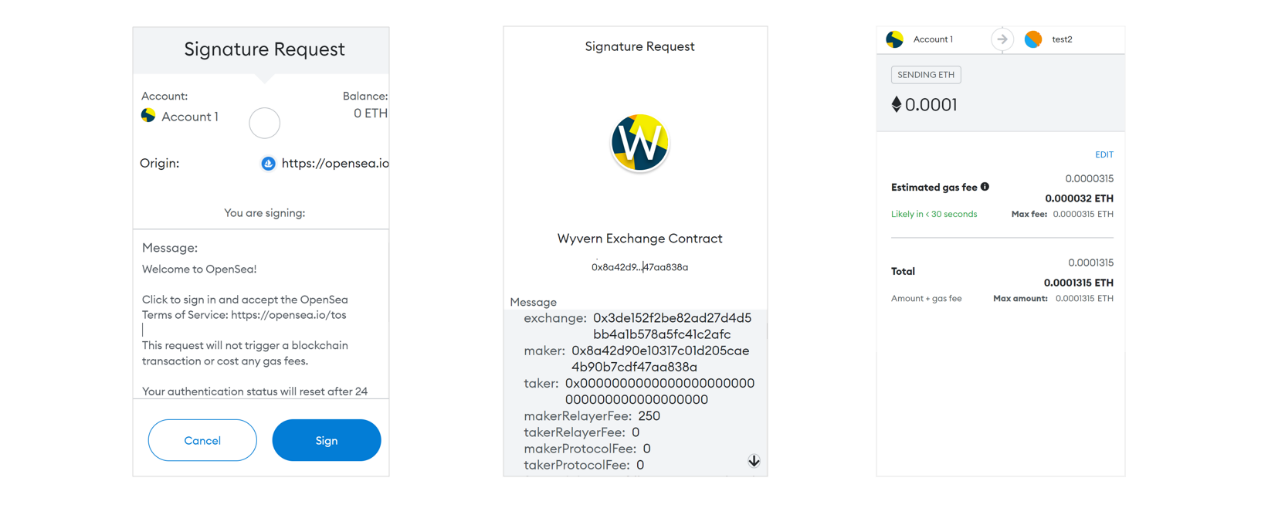
**EIP712 & EIP1271**

[2022-06-21](https://yuegs.com/2022/06/21/solidity-signature/)



签名，证明事件是在钱包的实际持有者授权的情况下执行的；验证，是通过加密算法对签名有效性的检测。签名并验证通过，是交易和授权的基础。  
本文，主要围绕着结构化数据的签名/验证以及合约签名/验证展开。其对应着 EIP712 以及 EIP1271 协议。

**1. 数据签名和验证**

The set of signable messages is extended from transactions and bytestrings 𝕋 ∪ 𝔹⁸ⁿ to also include structured data 𝕊. The new set of signable messages is thus 𝕋 ∪ 𝔹⁸ⁿ ∪ 𝕊. They are encoded to bytestrings suitable for hashing and signing as follows:  
**encode(transaction : 𝕋)** = RLP\_encode(transaction)  
**encode(message : 𝔹⁸ⁿ)** = “\x19Ethereum Signed Message:\n” ‖ len(message) ‖ message where len(message) is the non-zero-padded ascii-decimal encoding of the number of bytes in message.  
**encode(domainSeparator : 𝔹²⁵⁶, message : 𝕊)** = “\x19\x01” ‖ domainSeparator ‖ hashStruct(message) where domainSeparator and hashStruct(message) are defined below：  
**hashStruct(s : 𝕊)** = keccak256(typeHash ‖ encodeData(s)) where typeHash = keccak256(encodeType(typeOf(s)))  
**domainSeparator** = hashStruct(eip712Domain)  
[引用来源](https://eips.ethereum.org/EIPS/eip-712)

可以看到，基于非结构化数据 byte 的签名和基于结构化数据的签名，都包含一些特殊前缀，这种设计主要是为了**降低不同签名方式的碰撞风险，避免重放攻击。**

对于交易的签名和非结构化数据的签名，这里不再赘述，下述重点关注结构化数据签名。

**2. EIP712**

对结构化数据的签名，将遵循 EIP712 协议。该协议将使得签名过程中数据具有更好的可读性和透明性。  
以下，是在 Wyvern 协议中，构建卖单的过程。将调用 MetaMask 进行签名。

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 | const domain = {  name: 'Wyvern Exchange Contract',  version: '2.3',  chainId: 4,  verifyingContract: MARKETPLACE\_ADDR, }  *// The named list of all type definitions* const types = {  Order: [  { name: 'exchange', type: 'address'   { name: 'maker', type: 'address' },  { name: 'taker', type: 'address' },  { name: 'makerRelayerFee', type: 'uint256' },  { name: 'takerRelayerFee', type: 'uint256' },  { name: 'makerProtocolFee', type: 'uint256' },  { name: 'takerProtocolFee', type: 'uint256' },  { name: 'feeRecipient', type: 'address' },  { name: 'feeMethod', type: 'uint8' },  { name: 'side', type: 'uint8' },  { name: 'saleKind', type: 'uint8' },  { name: 'target', type: 'address' },  { name: 'howToCall', type: 'uint8' },  { name: 'calldata', type: 'bytes' },  { name: 'replacementPattern', type: 'bytes' },  { name: 'staticTarget', type: 'address' },  { name: 'staticExtradata', type: 'bytes' },  { name: 'paymentToken', type: 'address' },  { name: 'basePrice', type: 'uint256' },  { name: 'extra', type: 'uint256' },  { name: 'listingTime', type: 'uint256' },  { name: 'expirationTime', type: 'uint256' },  { name: 'salt', type: 'uint256' },  { name: 'nonce', type: 'uint256' },  ], };  *// The data to sign* const value = {  exchange: MARKETPLACE\_ADDR,  maker: maker,  taker: taker,  makerRelayerFee: MAKER\_RELAYER\_FEE,  takerRelayerFee: TAKER\_RELAYER\_FEE,  makerProtocolFee: MAKER\_PROTOCOL\_FEE,  takerProtocolFee: TAKER\_PROTOCOL\_FEE,  feeRecipient: FEE\_RECIPIENT,  feeMethod: feeMethod,  side: side,  saleKind: saleKind,  target: ORDER\_TARGET,  howToCall: howToCall,  calldata: calldata,  replacementPattern: SELL\_REPLACEMENT\_PATTERN,  staticTarget: ZEOR\_ADDRESS,  staticExtradata: EMPTY\_DATA,  paymentToken: paymentToken,  basePrice: basePrice,  extra: extra,  listingTime: Web3.utils.toBN(listingTime).toString(),  expirationTime: Web3.utils.toBN(expirationTime).toString(),  salt: generatePseudoRandomSalt(),  nonce: 0,  };  signature = await signer.\_signTypedData(domain, types, value); |

合约中的验证过程如下所示：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 | */\*\**  *\* @dev Derive the domain separator for EIP-712 signatures.*  *\* @return The domain separator.*  *\*/* function \_deriveDomainSeparator() private view returns (bytes32) {  return keccak256(  abi.encode(  \_EIP\_712\_DOMAIN\_TYPEHASH, *// keccak256("EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)")*  \_NAME\_HASH, *// keccak256("Wyvern Exchange Contract")*  \_VERSION\_HASH, *// keccak256(bytes("2.3"))*  \_CHAIN\_ID, *// NOTE: this is fixed, need to use solidity 0.5+ or make external call to support!*  address(this)  )  ); }  */\*\**  *\* @dev Hash an order, returning the hash that a client must sign via EIP-712 including the message prefix*  *\* @param order Order to hash*  *\* @param nonce Nonce to hash*  *\* @return Hash of message prefix and order hash per Ethereum format*  *\*/* function hashToSign(Order memory order, uint nonce) internal view returns (bytes32) {  return keccak256(  abi.encodePacked("\x19\x01", DOMAIN\_SEPARATOR, hashOrder(order, nonce))  ); }  */\*\**  *\* @dev Assert an order is valid and return its hash*  *\* @param order Order to validate*  *\* @param nonce Nonce to validate*  *\* @param sig ECDSA signature*  *\*/* function requireValidOrder(Order memory order, Sig memory sig, uint nonce) internal view returns (bytes32) {  bytes32 hash = hashToSign(order, nonce);  require(validateOrder(hash, order, sig));  return hash; }  */\*\**  *\* @dev Validate a provided previously approved / signed order, hash, and signature.*  *\* @param hash Order hash (already calculated, passed to avoid recalculation)*  *\* @param order Order to validate*  *\* @param sig ECDSA signature*  *\*/* function validateOrder(bytes32 hash, Order memory order, Sig memory sig) internal view returns (bool) {  ...  */\* recover via ECDSA, signed by maker (already verified as non-zero). \*/*  if (ecrecover(hash, sig.v, sig.r, sig.s) == order.maker) {  return true;  }  ... } |

外部账户签名、第三方验证签名的有效性这个过程本身是非常简单的。对于合约账户而言，并不具备有私钥，也无法完成数据的常规签名。

**3. EIP1271**

[**EIP1271**](https://eips.ethereum.org/EIPS/eip-1271)**就是为合约提供签名和验证的标准。** 它的标准接口：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 | pragma solidity ^0.5.0;  contract ERC1271 {   *// bytes4(keccak256("isValidSignature(bytes32,bytes)")*  bytes4 constant internal MAGICVALUE = 0x1626ba7e;   */\*\**  *\* @dev Should return whether the signature provided is valid for the provided hash*  *\* @param \_hash Hash of the data to be signed*  *\* @param \_signature Signature byte array associated with \_hash*  *\**  *\* MUST return the bytes4 magic value 0x1626ba7e when function passes.*  *\* MUST NOT modify state (using STATICCALL for solc < 0.5, view modifier for solc > 0.5)*  *\* MUST allow external calls*  *\*/*   function isValidSignature(  bytes32 \_hash,   bytes memory \_signature)  public  view   returns (bytes4 magicValue); } |

可知，具体的验证过程，实际上是在 isValidSignature 中完成的，如果验证过程通过，最终应当返回这个魔数 0x1626ba7e。该方法的实现者，通常是签名的合约本身，即 **谁签名谁验证**。

接下来，我们找两个合约实例看。  
第一个来自于爱死机的[合约](https://etherscan.io/address/0xfd43d1da000558473822302e1d44d81da2e4cc0d#code)：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 | function isSignatureValid(uint256 \_category, bytes memory \_signature) internal view returns (bool) {  bytes32 result = keccak256(abi.encodePacked(msg.sender, \_category));  bytes32 hash = keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", result));  return signer.isValidSignatureNow(hash, \_signature); } |

当然，它并不是一个标准的 EIP1271 实现，但是思路上具有一致性，均是通过线下由第三方签名，线上由第三方验证的过程。  
用户在线下，（通过 http 访问）向第三方获取到签名，并向合约提交该签名，最终，在合约中验证签名的有效性。  
这里，线下签名和签署者和线上签名的验证者，应是同一个账户。

接下来，再看一个来自 Gonisis 中 [CompatibilityFallbackHandler](https://github.com/safe-global/safe-contracts/blob/v1.3.0/contracts/handler/CompatibilityFallbackHandler.sol#L28)

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 | */\*\**  *\* Implementation of ISignatureValidator (see `interfaces/ISignatureValidator.sol`)*  *\* @dev Should return whether the signature provided is valid for the provided data.*  *\* @param \_data Arbitrary length data signed on the behalf of address(msg.sender)*  *\* @param \_signature Signature byte array associated with \_data*  *\* @return a bool upon valid or invalid signature with corresponding \_data*  *\*/* function isValidSignature(bytes calldata \_data, bytes calldata \_signature) public view override returns (bytes4) {  *// Caller should be a Safe*  GnosisSafe safe = GnosisSafe(payable(msg.sender));  bytes32 messageHash = getMessageHashForSafe(safe, \_data);  if (\_signature.length == 0) {  require(safe.signedMessages(messageHash) != 0, "Hash not approved");  } else {  safe.checkSignatures(messageHash, \_data, \_signature);  }  return EIP1271\_MAGIC\_VALUE; } |

这里，调用者需要是 Gnosis，将会检查 Gnosis 中是否已经记录了特定消息的签名，如果是，则表示验证通过，否则不通过。