ERC-1271: Standard Signature Validation Method for Contracts

Standard way to verify a signature when the account is a smart contract

|  |  |
| --- | --- |
| **Authors** | Francisco Giordano ([@frangio](https://github.com/frangio)), Matt Condon ([@shrugs](https://github.com/shrugs)), Philippe Castonguay ([@PhABC](https://github.com/PhABC)), Amir Bandeali ([@abandeali1](https://github.com/abandeali1)), Jorge Izquierdo ([@izqui](https://github.com/izqui)), Bertrand Masius ([@catageek](https://github.com/catageek)) |
| **Created** | 2018-07-25 |

Table of Contents

* [Abstract](https://eips.ethereum.org/EIPS/eip-1271#abstract)
* [Motivation](https://eips.ethereum.org/EIPS/eip-1271#motivation)
* [Specification](https://eips.ethereum.org/EIPS/eip-1271#specification)
* [Rationale](https://eips.ethereum.org/EIPS/eip-1271#rationale)
* [Backwards Compatibility](https://eips.ethereum.org/EIPS/eip-1271#backwards-compatibility)
* [Reference Implementation](https://eips.ethereum.org/EIPS/eip-1271#reference-implementation)
* [Security Considerations](https://eips.ethereum.org/EIPS/eip-1271#security-considerations)
* [Copyright](https://eips.ethereum.org/EIPS/eip-1271#copyright)

 Abstract

Externally Owned Accounts (EOA) can sign messages with their associated private keys, but currently contracts cannot. We propose a standard way for any contracts to verify whether a signature on a behalf of a given contract is valid. This is possible via the implementation of a isValidSignature(hash, signature) function on the signing contract, which can be called to validate a signature.

 Motivation

There are and will be many contracts that want to utilize signed messages for validation of rights-to-move assets or other purposes. In order for these contracts to be able to support non Externally Owned Accounts (i.e., contract owners), we need a standard mechanism by which a contract can indicate whether a given signature is valid or not on its behalf.

One example of an application that requires signatures to be provided would be decentralized exchanges with off-chain orderbook, where buy/sell orders are signed messages. In these applications, EOAs sign orders, signaling their desire to buy/sell a given asset and giving explicit permissions to the exchange smart contracts to conclude a trade via a signature. When it comes to contracts however, regular signatures are not possible since contracts do not possess a private key, hence this proposal.

 Specification

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC 2119](https://www.ietf.org/rfc/rfc2119.txt).

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);

}

isValidSignature can call arbitrary methods to validate a given signature, which could be context dependent (e.g. time based or state based), EOA dependent (e.g. signers authorization level within smart wallet), signature scheme Dependent (e.g. ECDSA, multisig, BLS), etc.

This function should be implemented by contracts which desire to sign messages (e.g. smart contract wallets, DAOs, multisignature wallets, etc.) Applications wanting to support contract signatures should call this method if the signer is a contract.

 Rationale

We believe the name of the proposed function to be appropriate considering that an *authorized* signers providing proper signatures for a given data would see their signature as “valid” by the signing contract. Hence, a signed action message is only valid when the signer is authorized to perform a given action on the behalf of a smart wallet.

Two arguments are provided for simplicity of separating the hash signed from the signature. A bytes32 hash is used instead of the unhashed message for simplicity, since contracts could expect a certain hashing function that is not standard, such as with [EIP-712](https://eips.ethereum.org/EIPS/eip-712).

isValidSignature() should not be able to modify states in order to prevent GasToken minting or similar attack vectors. Again, this is to simplify the implementation surface of the function for better standardization and to allow off-chain contract queries.

The specific return value is expected to be returned instead of a boolean in order to have stricter and simpler verification of a signature.

 Backwards Compatibility

This EIP is backward compatible with previous work on signature validation since this method is specific to contract based signatures and not EOA signatures.

 Reference Implementation

Example implementation of a signing contract:

*/\*\**

*\* @notice Verifies that the signer is the owner of the signing contract.*

*\*/*

**function** isValidSignature(

**bytes32** \_hash,

**bytes** **calldata** \_signature

) **external** **override** **view** **returns** (**bytes4**) {

*// Validate signatures*

**if** (recoverSigner(\_hash, \_signature) **==** owner) {

**return** 0x1626ba7e;

} **else** {

**return** 0xffffffff;

}

}

*/\*\**

*\* @notice Recover the signer of hash, assuming it's an EOA account*

*\* @dev Only for EthSign signatures*

*\* @param \_hash Hash of message that was signed*

*\* @param \_signature Signature encoded as (bytes32 r, bytes32 s, uint8 v)*

*\*/*

**function** recoverSigner(

**bytes32** \_hash,

**bytes** **memory** \_signature

) **internal** **pure** **returns** (**address** signer) {

require(\_signature.length **==** 65, "SignatureValidator#recoverSigner: invalid signature length");

*// Variables are not scoped in Solidity.*

**uint8** v **=** **uint8**(\_signature[64]);

**bytes32** r **=** \_signature.readBytes32(0);

**bytes32** s **=** \_signature.readBytes32(32);

*// EIP-2 still allows signature malleability for ecrecover(). Remove this possibility and make the signature*

*// unique. Appendix F in the Ethereum Yellow paper (https://ethereum.github.io/yellowpaper/paper.pdf), defines*

*// the valid range for s in (281): 0 < s < secp256k1n ÷ 2 + 1, and for v in (282): v ∈ {27, 28}. Most*

*// signatures from current libraries generate a unique signature with an s-value in the lower half order.*

*//*

*// If your library generates malleable signatures, such as s-values in the upper range, calculate a new s-value*

*// with 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141 - s1 and flip v from 27 to 28 or*

*// vice versa. If your library also generates signatures with 0/1 for v instead 27/28, add 27 to v to accept*

*// these malleable signatures as well.*

*//*

*// Source OpenZeppelin*

*// https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/cryptography/ECDSA.sol*

**if** (**uint256**(s) **>** 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0) {

revert("SignatureValidator#recoverSigner: invalid signature 's' value");

}

**if** (v **!=** 27 **&&** v **!=** 28) {

revert("SignatureValidator#recoverSigner: invalid signature 'v' value");

}

*// Recover ECDSA signer*

signer **=** ecrecover(\_hash, v, r, s);

*// Prevent signer from being 0x0*

require(

signer **!=** **address**(0x0),

"SignatureValidator#recoverSigner: INVALID\_SIGNER"

);

**return** signer;

}

Example implementation of a contract calling the isValidSignature() function on an external signing contract ;

**function** callERC1271isValidSignature(

**address** \_addr,

**bytes32** \_hash,

**bytes** **calldata** \_signature

) **external** **view** {

**bytes4** result **=** IERC1271Wallet(\_addr).isValidSignature(\_hash, \_signature);

require(result **==** 0x1626ba7e, "INVALID\_SIGNATURE");

}

 Security Considerations

Since there are no gas-limit expected for calling the isValidSignature() function, it is possible that some implementation will consume a large amount of gas. It is therefore important to not hardcode an amount of gas sent when calling this method on an external contract as it could prevent the validation of certain signatures.

Note also that each contract implementing this method is responsible to ensure that the signature passed is indeed valid, otherwise catastrophic outcomes are to be expected.

 Copyright

Copyright and related rights waived via [CC0](https://eips.ethereum.org/LICENSE).

Citation

Please cite this document as:

Francisco Giordano ([@frangio](https://github.com/frangio)), Matt Condon ([@shrugs](https://github.com/shrugs)), Philippe Castonguay ([@PhABC](https://github.com/PhABC)), Amir Bandeali ([@abandeali1](https://github.com/abandeali1)), Jorge Izquierdo ([@izqui](https://github.com/izqui)), Bertrand Masius ([@catageek](https://github.com/catageek)), "ERC-1271: Standard Signature Validation Method for Contracts," *Ethereum Improvement Proposals*, no. 1271, July 2018. [Online serial]. Available: https://eips.ethereum.org/EIPS/eip-1271.