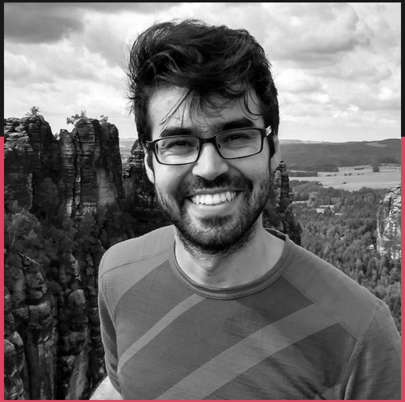
Unstoppable CoWs: How to Leverage ERC-1271 to Place All Sorts of Smart Orders on CoW Protocol



Nicholas Rodrigues Lordello
CoW Protocol

Unstoppable CoWs:

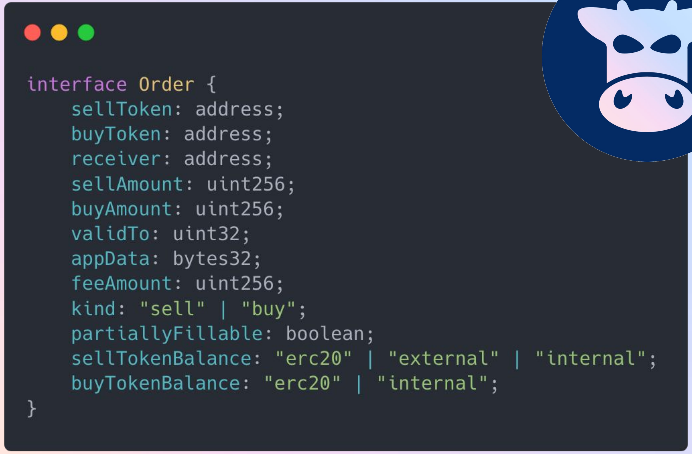
How to Leverage ERC-1271 to Place All Sorts of Smart Orders on CoW Protocol

Nicholas Rodrigues Lordello

Overview

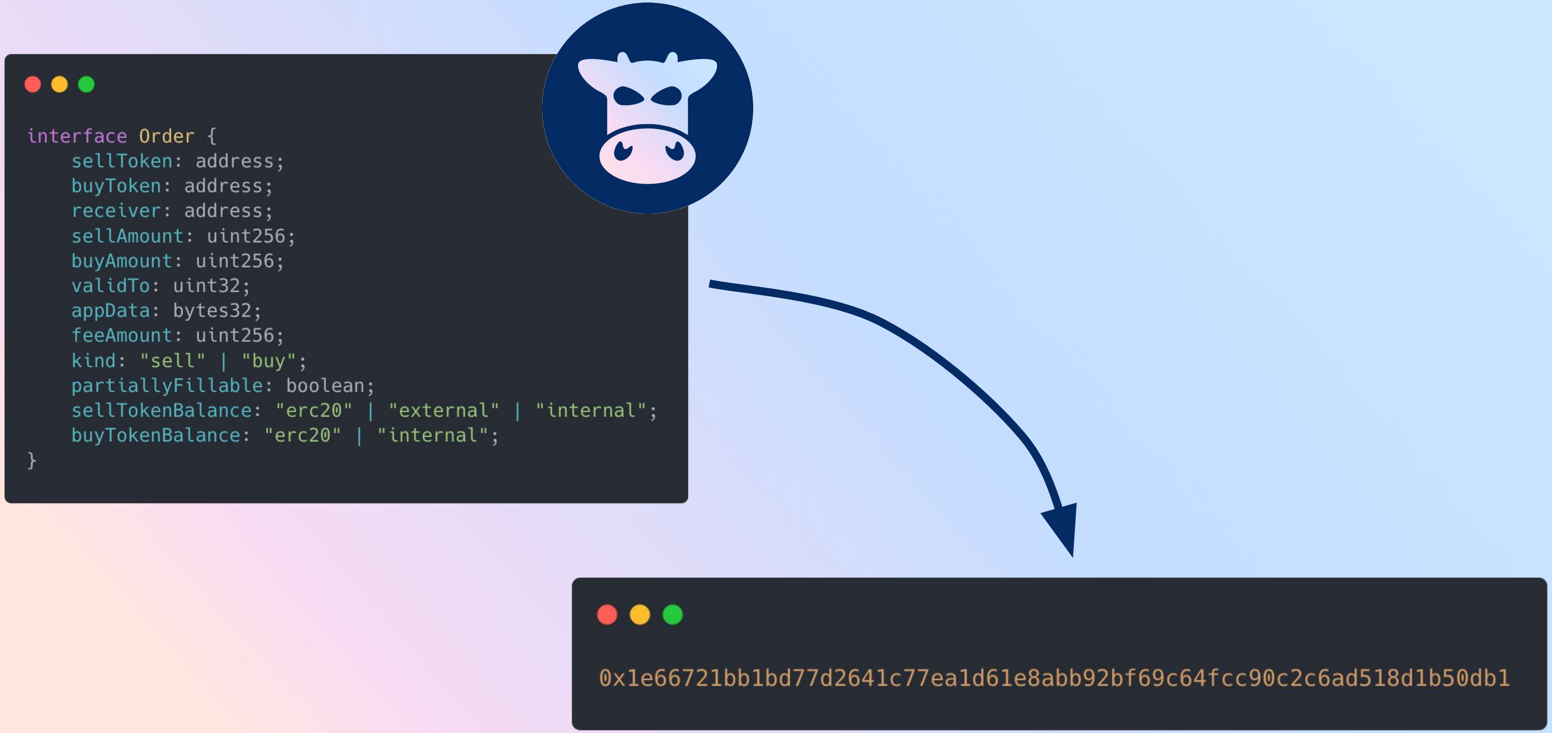
- The Basics
  - CoW Protocol orders: signing of typed data
  - ~~EIP~~ ERC-1271: signature validation standard
- Smart Contract Wallet Orders
  - Safe: How to trade without gas fees
- Smart Orders!

The Basics: CoW Protocol Orders



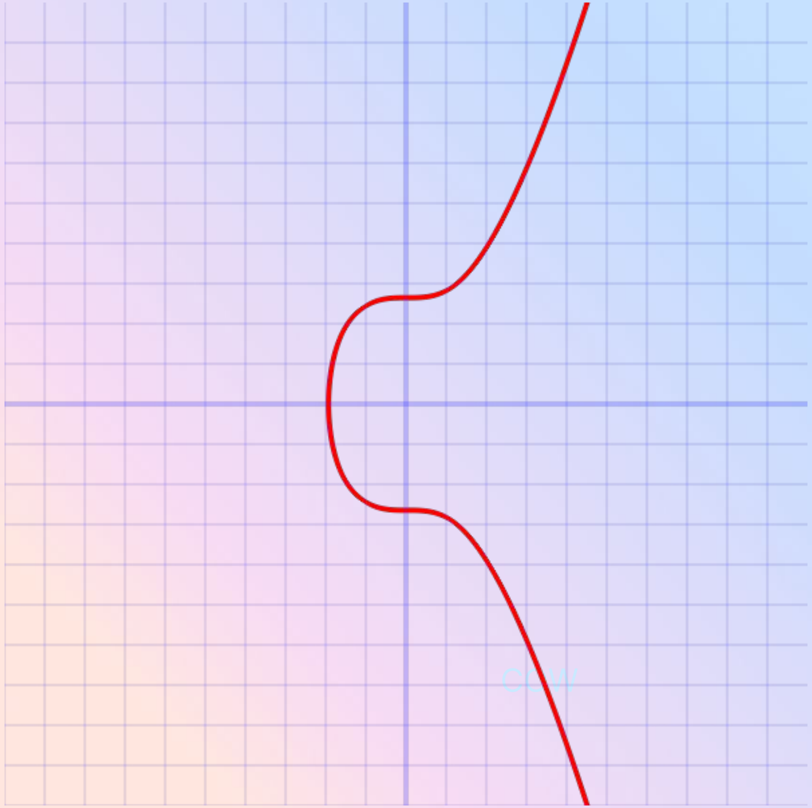
```
interface Order {
    sellToken: address;
    buyToken: address;
    receiver: address;
    sellAmount: uint256;
    buyAmount: uint256;
    validTo: uint32;
    appData: bytes32;
    feeAmount: uint256;
    kind: "sell" | "buy";
    partiallyFillable: boolean;
    sellTokenBalance: "erc20" | "external" | "internal";
    buyTokenBalance: "erc20" | "internal";
}
```

The Basics: CoW Protocol Orders

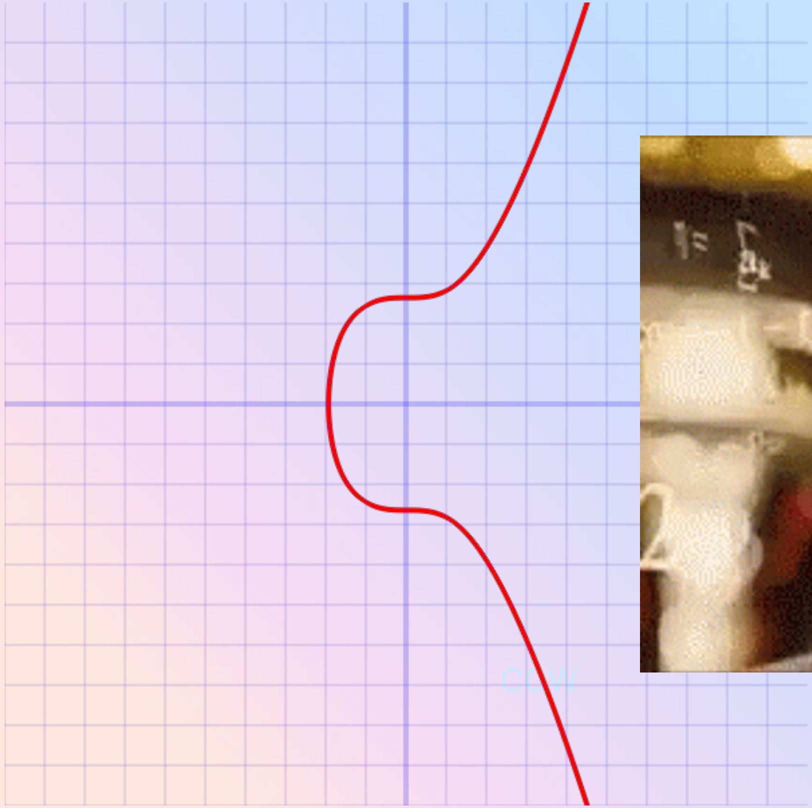
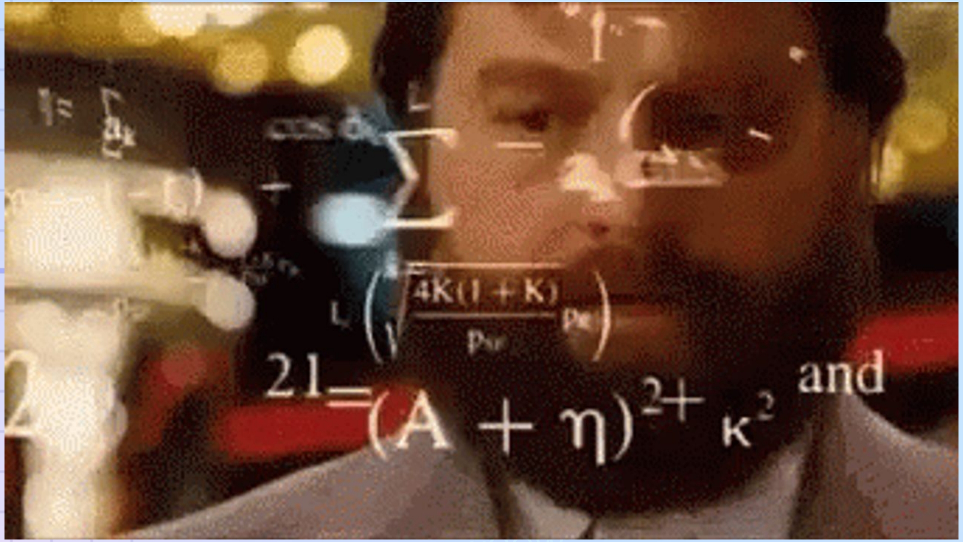


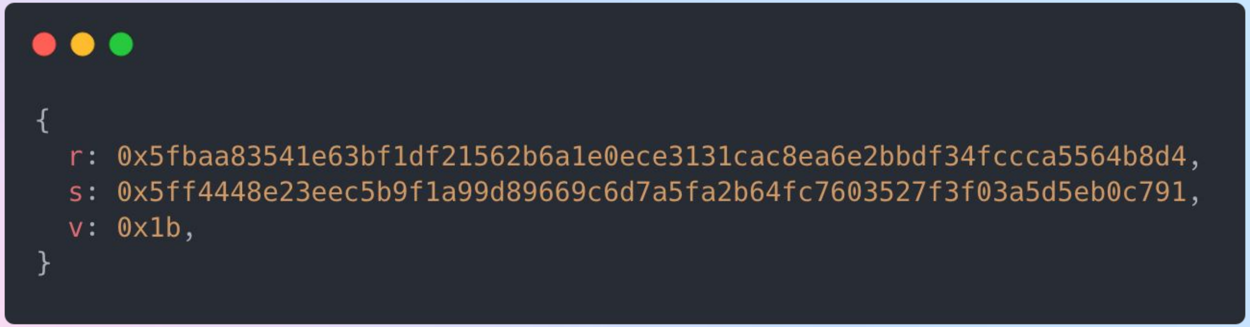
```
interface Order {
    sellToken: address;
    buyToken: address;
    receiver: address;
    sellAmount: uint256;
    buyAmount: uint256;
    validTo: uint32;
    appData: bytes32;
    feeAmount: uint256;
    kind: "sell" | "buy";
    partiallyFillable: boolean;
    sellTokenBalance: "erc20" | "external" | "internal";
    buyTokenBalance: "erc20" | "internal";
}
```

```
0x1e66721bb1bd77d2641c77ea1d61e8abb92bf69c64fcc90c2c6ad518d1b50db1
```



vCOW

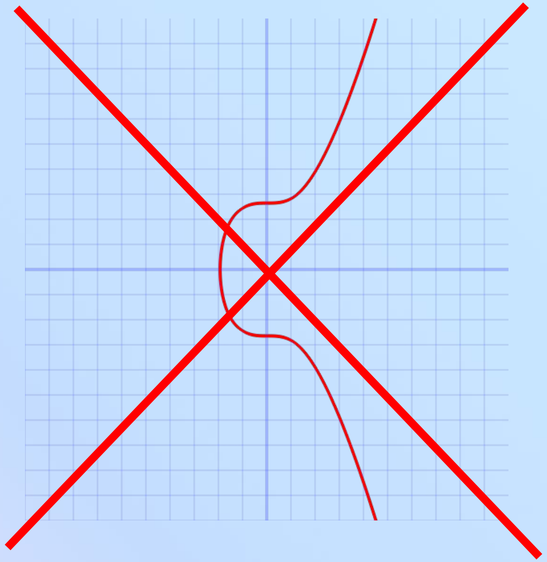


cos
4K(1 + K)
21
(A + η)^2+ κ^2 and



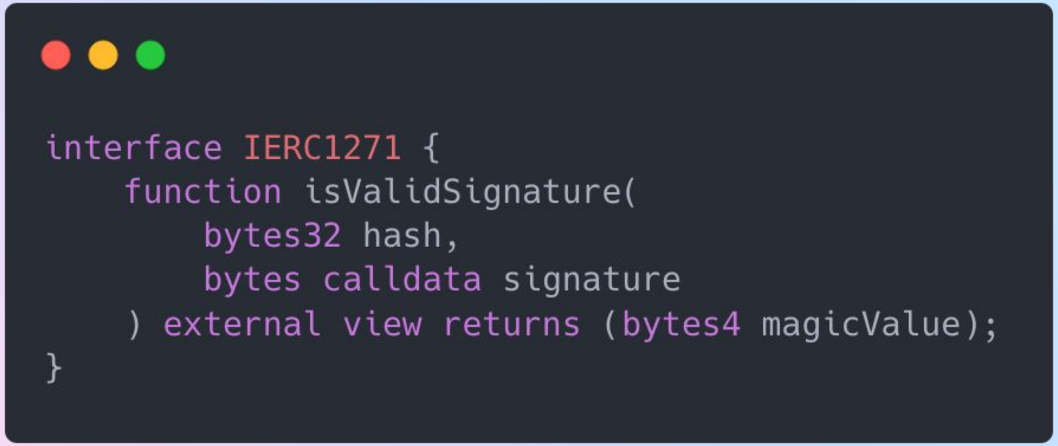
```
{
  r: 0x5fbaa83541e63bf1df21562b6a1e0ece3131cac8ea6e2bbdf34fccca5564b8d4,
  s: 0x5ff4448e23eec5b9f1a99d89669c6d7a5fa2b64fc7603527f3f03a5d5eb0c791,
  v: 0x1b,
}
```

The Basics: ERC-1271

- Smart Contracts have no private key
- They can’t sign things with ECDSA!
- Another signature scheme is needed…

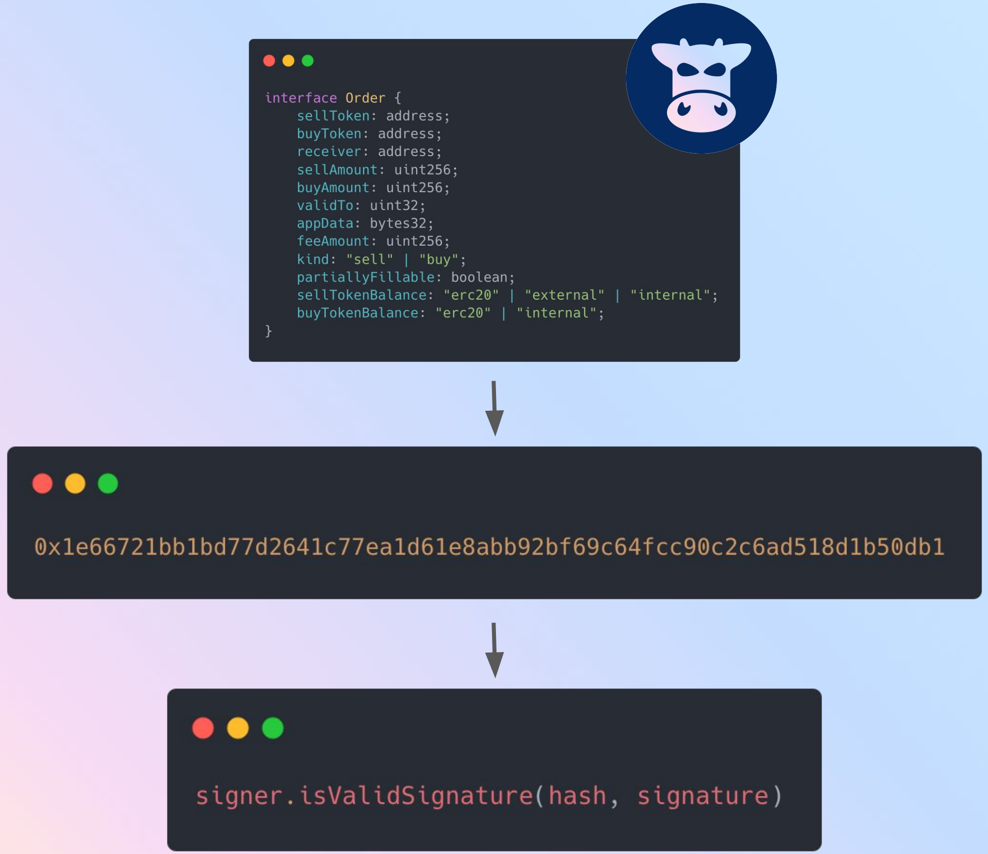


The Basics: ERC-1271



```
interface IERC1271 {
    function isValidSignature(
        bytes32 hash,
        bytes calldata signature
    ) external view returns (bytes4 magicValue);
}
```

The Basics: ERC-1271



```
interface Order {
    sellToken: address;
    buyToken: address;
    receiver: address;
    sellAmount: uint256;
    buyAmount: uint256;
    validTo: uint32;
    appData: bytes32;
    feeAmount: uint256;
    kind: "sell" | "buy";
    partiallyFillable: boolean;
    sellTokenBalance: "erc20" | "external" | "internal";
    buyTokenBalance: "erc20" | "internal";
}
```

```
0x1e66721bb1bd77d2641c77ea1d61e8abb92bf69c64fcc90c2c6ad518d1b50db1
```

```
signer.isValidSignature(hash, signature)
```

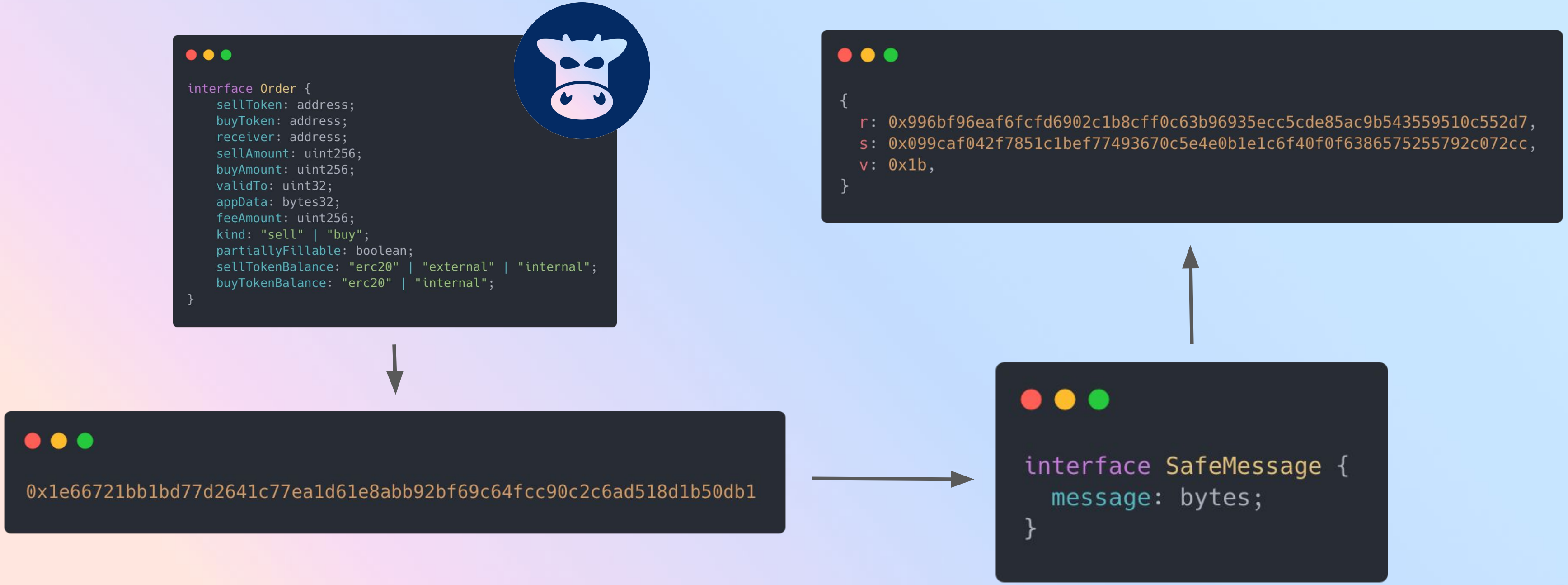
Smart Contract Wallet Orders

- Smart Contract wallet implements ERC-1271
- Implementation specific verification scheme
  - Controller uses transaction to indicate hash is trusted
  - Accept signatures from certain domains
  - Controller off-chain signatures

Smart Contract Wallet Orders

- Smart Contract wallet implements ERC-1271
- Implementation specific verification scheme
  - Controller uses transaction to indicate hash is trusted
  - Accept signatures from certain domains
  - Controller off-chain signatures

Safe

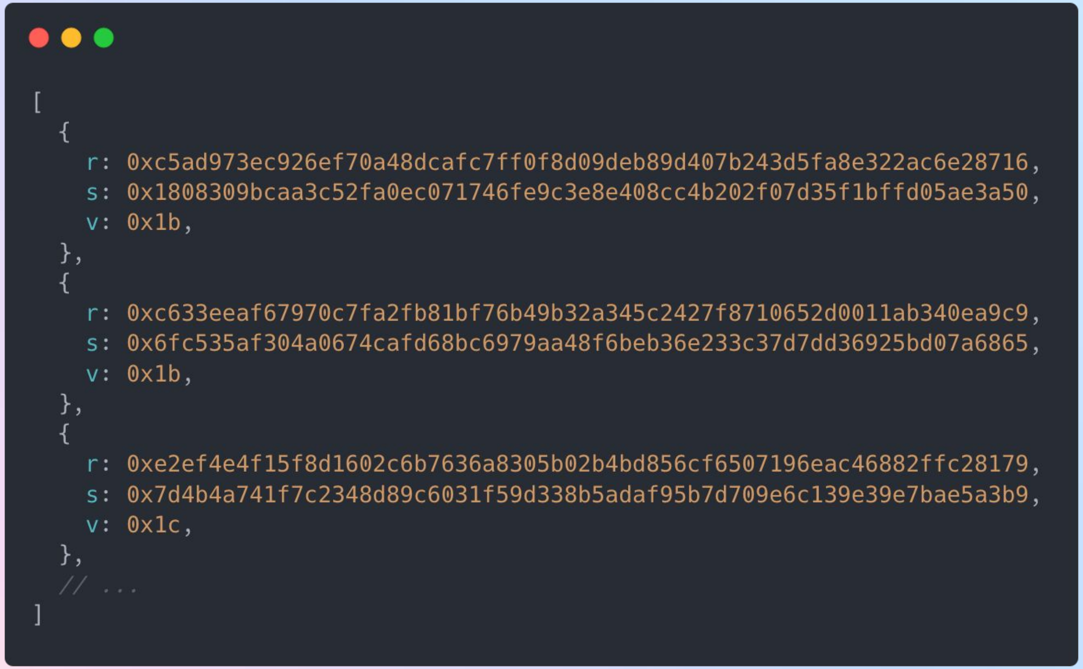


```
interface Order {
    sellToken: address;
    buyToken: address;
    receiver: address;
    sellAmount: uint256;
    buyAmount: uint256;
    validTo: uint32;
    appData: bytes32;
    feeAmount: uint256;
    kind: "sell" | "buy";
    partiallyFillable: boolean;
    sellTokenBalance: "erc20" | "external" | "internal";
    buyTokenBalance: "erc20" | "internal";
}
```

```
0x1e66721bb1bd77d2641c77ea1d61e8abb92bf69c64fcc90c2c6ad518d1b50db1
```

```
interface SafeMessage {
  message: bytes;
}
```

```
{
  r: 0x996bf96eaf6fcfd6902c1b8cff0c63b96935ecc5cde85ac9b543559510c552d7,
  s: 0x099caf042f7851c1bef77493670c5e4e0b1e1c6f40f0f6386575255792c072cc,
  v: 0x1b,
}
```

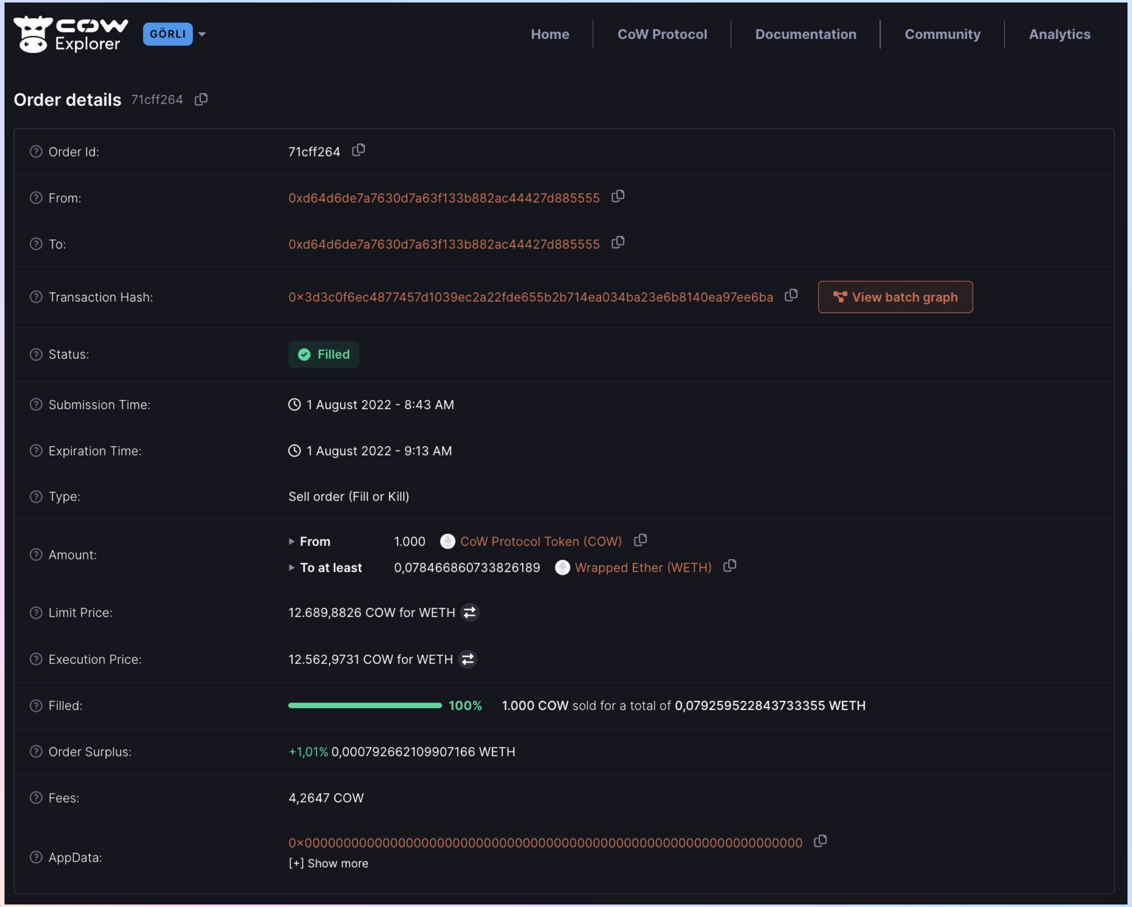


```
[
  {
    r: 0xc5ad973ec926ef70a48dcafc7ff0f8d09deb89d407b243d5fa8e322ac6e28716,
    s: 0x1808309bcaa3c52fa0ec071746fe9c3e8e408cc4b202f07d35f1bffd05ae3a50,
    v: 0x1b,
  },
  {
    r: 0xc633eeaf67970c7fa2fb81bf76b49b32a345c2427f8710652d0011ab340ea9c9,
    s: 0x6fc535af304a0674cafd68bc6979aa48f6beb36e233c37d7dd36925bd07a6865,
    v: 0x1b,
  },
  {
    r: 0xe2ef4e4f15f8d1602c6b7636a8305b02b4bd856cf6507196eac46882ffc28179,
    s: 0x7d4b4a741f7c2348d89c6031f59d338b5adaf95b7d709e6c139e39e7bae5a3b9,
    v: 0x1c,
  },
  // ...
]
```

Safe

1. Collected signatures from Safe owners encoded into bytes
2. Foreach signature in decoded from `signature` bytes:
   a. ECDSA recover the signer
   b. Verify that it is indeed a Safe owner
3. Check that # of signatures is greater than owner threshold

https://barn.explorer.cow.fi/goerli/orders/0x71cff2646c6ca7b26844fdada874db8f20ff10cc831ffc8ba381b77dc185279fd64d6de7a7630d7a63f133b882ac44427d88555562e77d0e



Order details 71cff264

Order Id:	71cff264
From:	0xd64d6de7a7630d7a63f133b882ac44427d885555
To:	0xd64d6de7a7630d7a63f133b882ac44427d885555
Transaction Hash:	0×3d3c0f6ec4877457d1039ec2a22fde655b2b714ea034ba23e6b8140ea97ee6ba
Status:	Filled
Submission Time:	1 August 2022 - 8:43 AM
Expiration Time:	1 August 2022 - 9:13 AM
Type:	Sell order (Fill or Kill)
Amount:	**From** 1.000 CoW Protocol Token (COW) **To at least** 0,078466860733826189 Wrapped Ether (WETH)
Limit Price:	12.689,8826 COW for WETH
Execution Price:	12.562,9731 COW for WETH
Filled:	100% **1.000 COW** sold for a total of **0,079259522843733355 WETH**
Order Surplus:	+1,01% 0,000792662109907166 WETH
Fees:	4,2647 COW
AppData:	0×0000000000000000000000000000000000000000000000000000000000000000 [+] Show more

Smart Orders

Smart Orders

Work with ERC-1271 just like the Safe:

1. Deposit some tokens into a contract
2. Implement ERC-1271 and validate some order hash

Smart Orders

Work with ERC-1271 just like the Safe:

1. Deposit some tokens into a contract
2. Implement ERC-1271 and validate some order hash

BUT add custom on-chain validation logic

Smart Orders

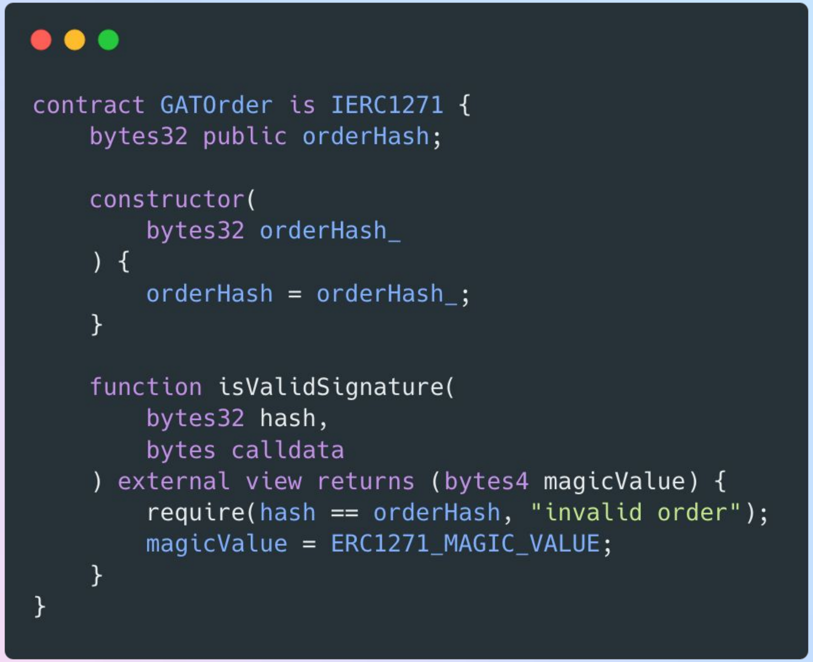
Work with ERC-1271 just like the Safe:

1. Deposit some tokens into a contract
2. Implement ERC-1271 and validate some order hash

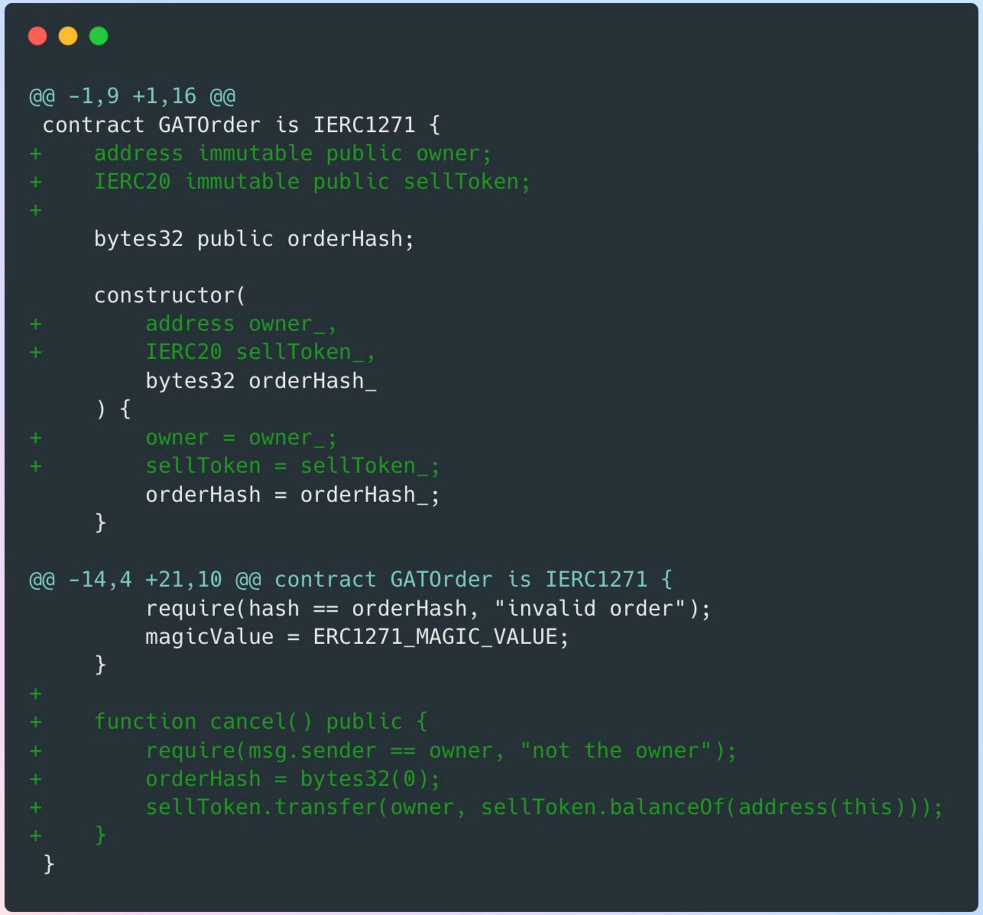
BUT add custom on-chain validation logic - **THAT'S IT!**

Good After Time (GAT) Orders

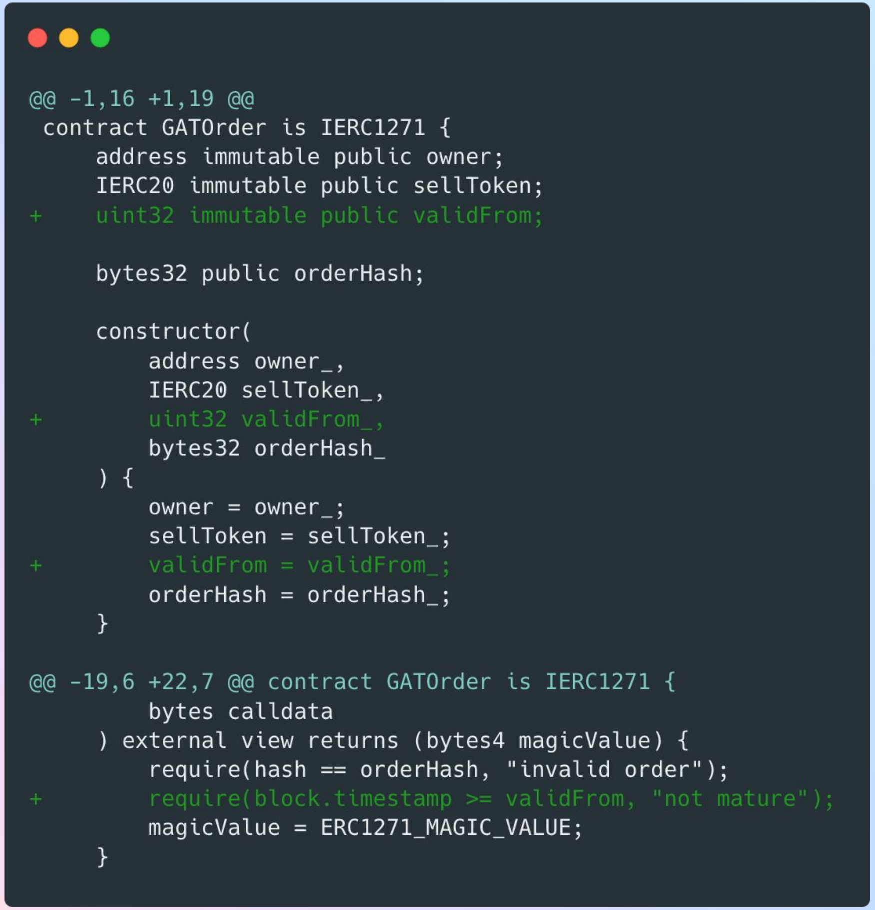
- Only become valid after a certain timestamp
- Not supported by CoW Protocol orders
  - Only `validTo` field is supported in “native” orders
- Check block timestamp in `isValidSignature` call
  - Revert if the block is too recent
- Order would automatically “get picked up” once it becomes valid



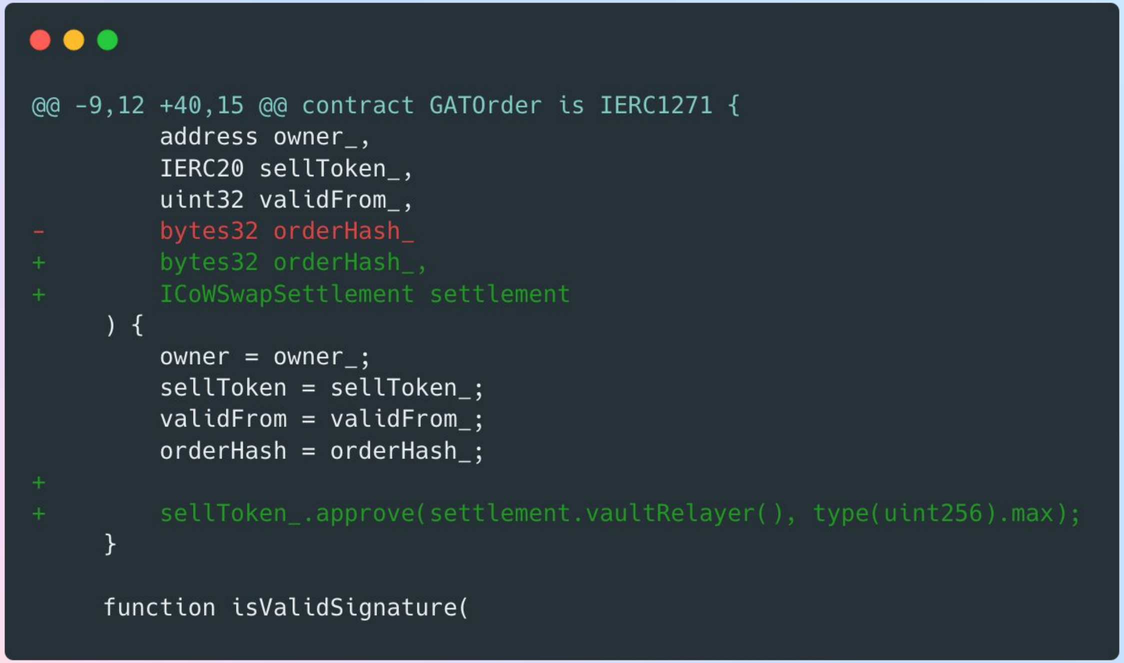
```
contract GATOrder is IERC1271 {
    bytes32 public orderHash;

    constructor(
        bytes32 orderHash_
    ) {
        orderHash = orderHash_;
    }

    function isValidSignature(
        bytes32 hash,
        bytes calldata
    ) external view returns (bytes4 magicValue) {
        require(hash == orderHash, "invalid order");
        magicValue = ERC1271_MAGIC_VALUE;
    }
}
```



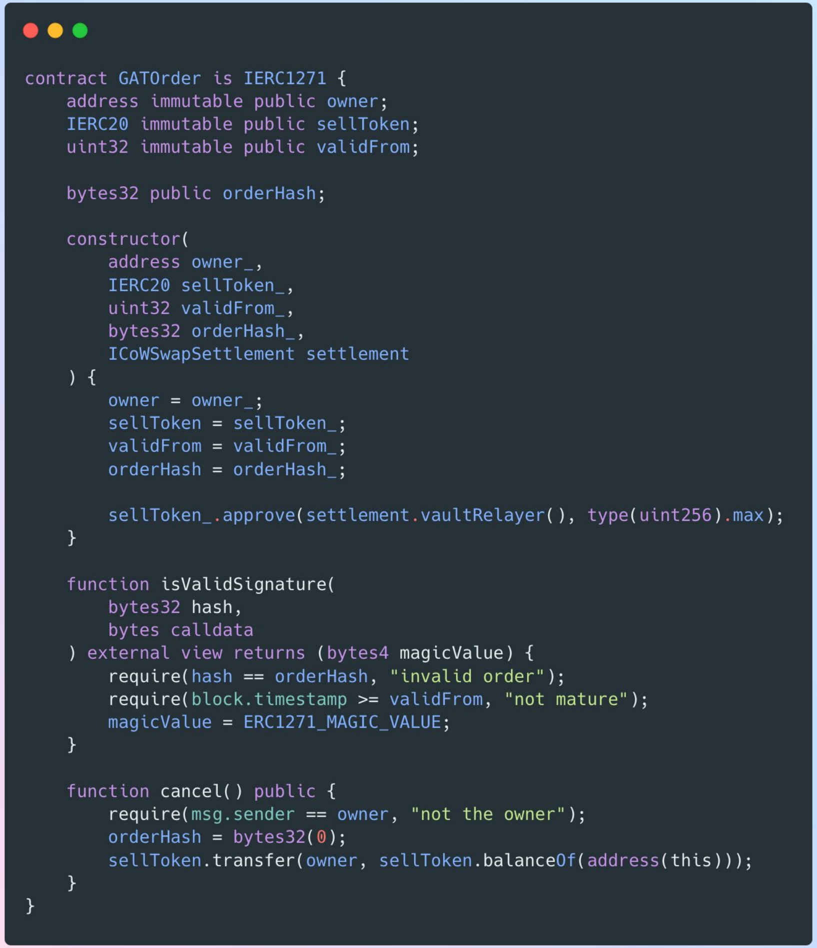
```diff
@@ -1,9 +1,16 @@
 contract GATOrder is IERC1271 {
+    address immutable public owner;
+    IERC20 immutable public sellToken;
+
     bytes32 public orderHash;

     constructor(
+        address owner_,
+        IERC20 sellToken_,
         bytes32 orderHash_
     ) {
+        owner = owner_;
+        sellToken = sellToken_;
         orderHash = orderHash_;
     }

@@ -14,4 +21,10 @@ contract GATOrder is IERC1271 {
         require(hash == orderHash, "invalid order");
         magicValue = ERC1271_MAGIC_VALUE;
     }
+
+    function cancel() public {
+        require(msg.sender == owner, "not the owner");
+        orderHash = bytes32(0);
+        sellToken.transfer(owner, sellToken.balanceOf(address(this)));
+    }
 }
```



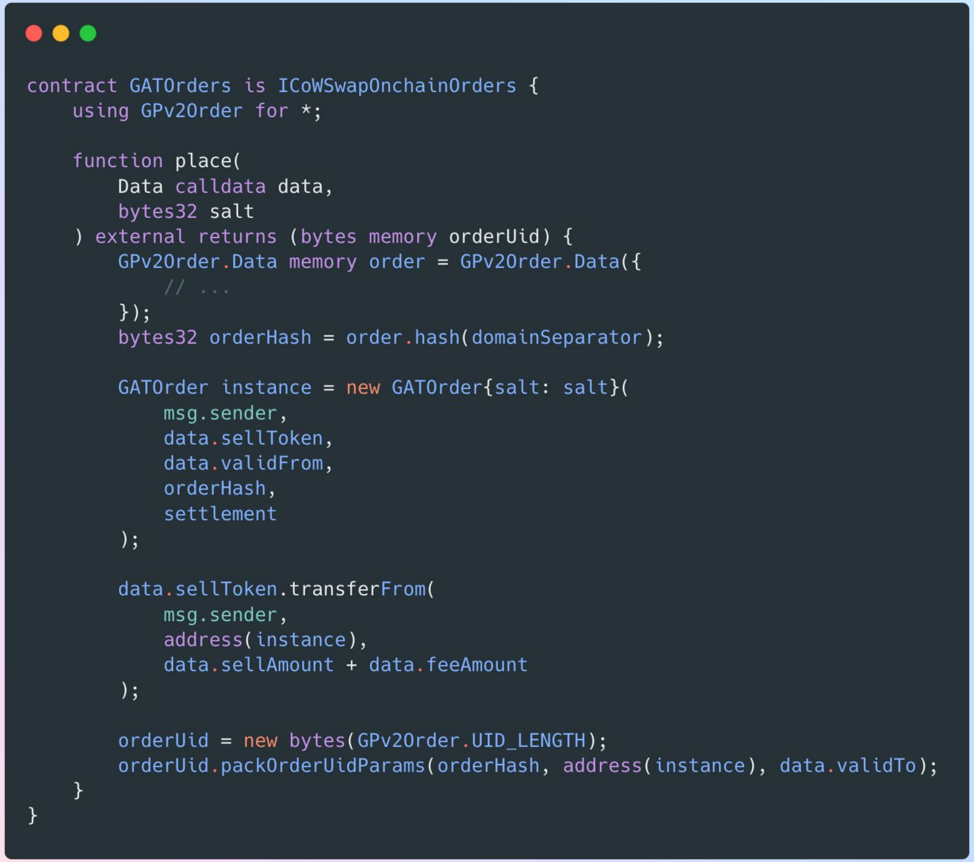
```diff
@@ -1,16 +1,19 @@
 contract GATOrder is IERC1271 {
     address immutable public owner;
     IERC20 immutable public sellToken;
+    uint32 immutable public validFrom;

     bytes32 public orderHash;

     constructor(
         address owner_,
         IERC20 sellToken_,
+        uint32 validFrom_,
         bytes32 orderHash_
     ) {
         owner = owner_;
         sellToken = sellToken_;
+        validFrom = validFrom_;
         orderHash = orderHash_;
     }

@@ -19,6 +22,7 @@ contract GATOrder is IERC1271 {
         bytes calldata
     ) external view returns (bytes4 magicValue) {
         require(hash == orderHash, "invalid order");
+        require(block.timestamp >= validFrom, "not mature");
         magicValue = ERC1271_MAGIC_VALUE;
     }
```



```
@@ -9,12 +40,15 @@ contract GATOrder is IERC1271 {
         address owner_,
         IERC20 sellToken_,
         uint32 validFrom_,
-        bytes32 orderHash_
+        bytes32 orderHash_,
+        ICoWSwapSettlement settlement
     ) {
         owner = owner_;
         sellToken = sellToken_;
         validFrom = validFrom_;
         orderHash = orderHash_;
+
+        sellToken_.approve(settlement.vaultRelayer(), type(uint256).max);
     }

     function isValidSignature(
```

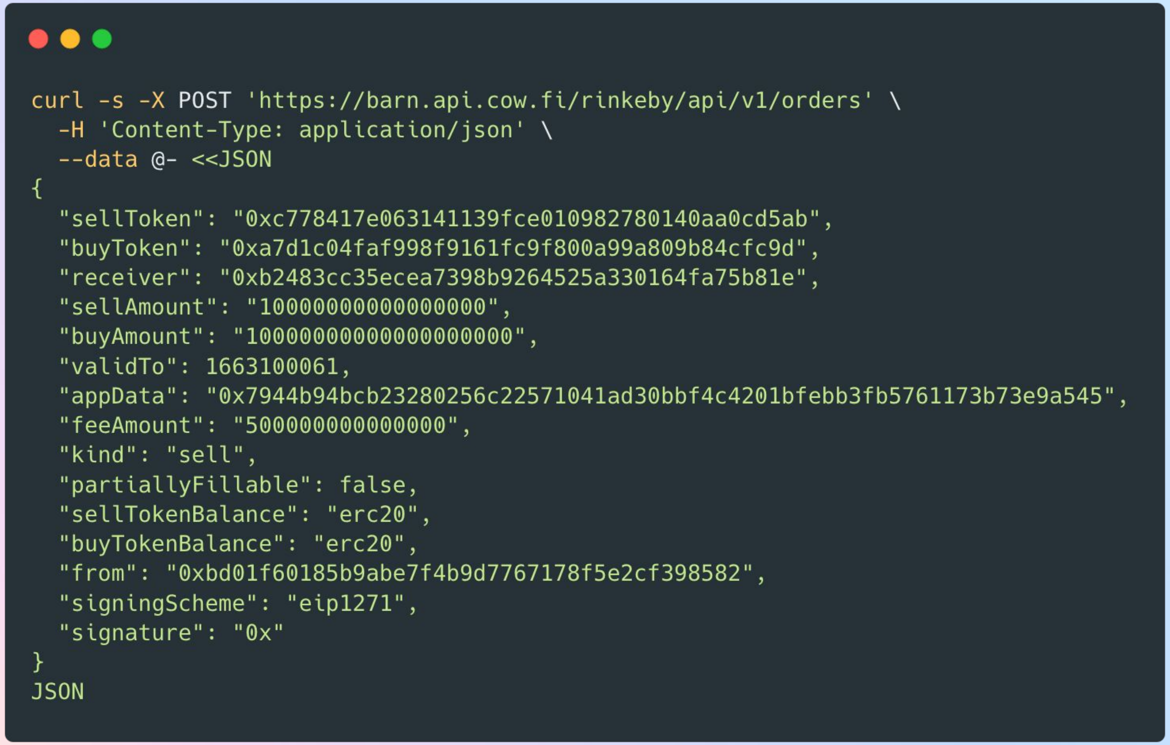


```
contract GATOrder is IERC1271 {
    address immutable public owner;
    IERC20 immutable public sellToken;
    uint32 immutable public validFrom;

    bytes32 public orderHash;

    constructor(
        address owner_,
        IERC20 sellToken_,
        uint32 validFrom_,
        bytes32 orderHash_,
        ICoWSwapSettlement settlement
    ) {
        owner = owner_;
        sellToken = sellToken_;
        validFrom = validFrom_;
        orderHash = orderHash_;

        sellToken_.approve(settlement.vaultRelayer(), type(uint256).max);
    }

    function isValidSignature(
        bytes32 hash,
        bytes calldata
    ) external view returns (bytes4 magicValue) {
        require(hash == orderHash, "invalid order");
        require(block.timestamp >= validFrom, "not mature");
        magicValue = ERC1271_MAGIC_VALUE;
    }

    function cancel() public {
        require(msg.sender == owner, "not the owner");
        orderHash = bytes32(0);
        sellToken.transfer(owner, sellToken.balanceOf(address(this)));
    }
}
```


```
contract GATOrders is ICoWSwapOnchainOrders {
    using GPv2Order for *;

    function place(
        Data calldata data,
        bytes32 salt
    ) external returns (bytes memory orderUid) {
        GPv2Order.Data memory order = GPv2Order.Data({
            // ...
        });
        bytes32 orderHash = order.hash(domainSeparator);

        GATOrder instance = new GATOrder{salt: salt}(
            msg.sender,
            data.sellToken,
            data.validFrom,
            orderHash,
            settlement
        );

        data.sellToken.transferFrom(
            msg.sender,
            address(instance),
            data.sellAmount + data.feeAmount
        );

        orderUid = new bytes(GPv2Order.UID_LENGTH);
        orderUid.packOrderUidParams(orderHash, address(instance), data.validTo);
    }
}
```

Good After Time (GAT) Orders

1. Trader approves `GATOrders` factory contract
2. Calls `place` function:
   a. Specify additional validFrom parameter
   b. Create a GATOrder instance
   c. Transfer sell tokens to the “smart order” and sets approval to settlement vaultRelayer contract
3. Order is ready!

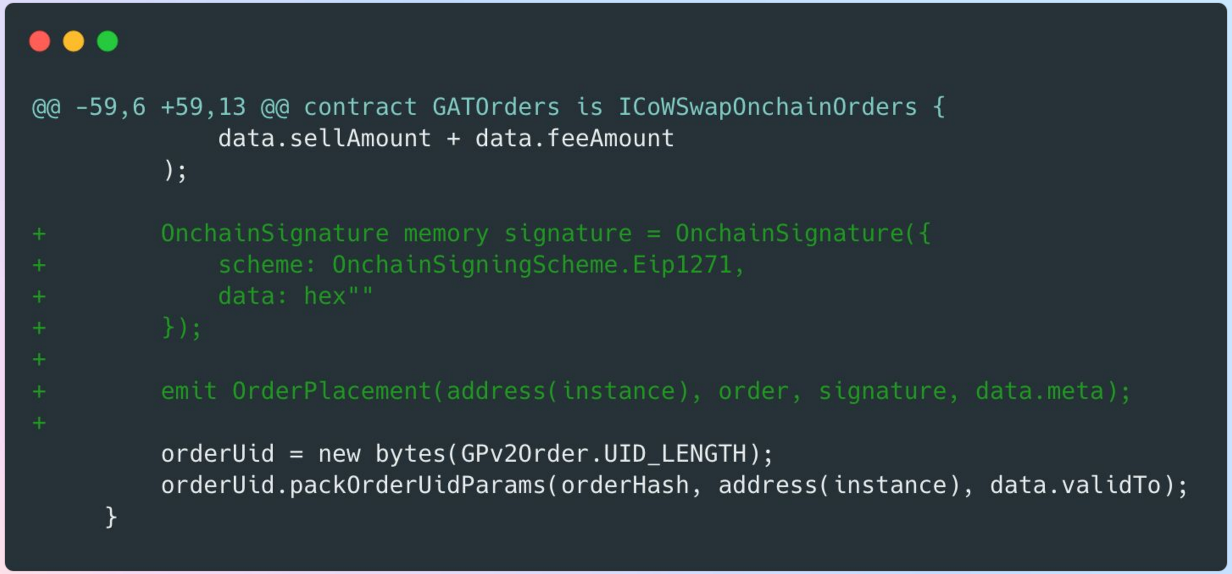


```
curl -s -X POST 'https://barn.api.cow.fi/rinkeby/api/v1/orders' \
  -H 'Content-Type: application/json' \
  --data @- <<JSON
{
  "sellToken": "0xc778417e063141139fce010982780140aa0cd5ab",
  "buyToken": "0xa7d1c04faf998f9161fc9f800a99a809b84cfc9d",
  "receiver": "0xb2483cc35ecea7398b9264525a330164fa75b81e",
  "sellAmount": "10000000000000000",
  "buyAmount": "10000000000000000000",
  "validTo": 1663100061,
  "appData": "0x7944b94bcb23280256c22571041ad30bbf4c4201bfebb3fb5761173b73e9a545",
  "feeAmount": "500000000000000",
  "kind": "sell",
  "partiallyFillable": false,
  "sellTokenBalance": "erc20",
  "buyTokenBalance": "erc20",
  "from": "0xbd01f60185b9abe7f4b9d7767178f5e2cf398582",
  "signingScheme": "eip1271",
  "signature": "0x"
}
JSON
```

Good After Time (GAT) Orders

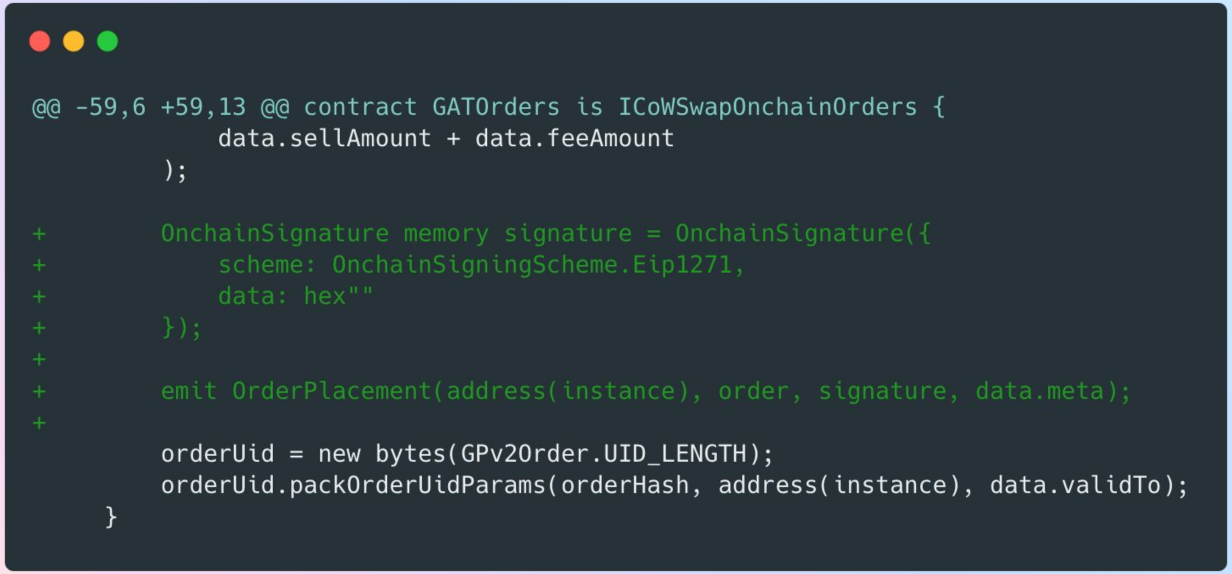
1. Submit smart order to the API
2. Signature is validated before every auction
   a. Checks that `block.timestamp >= validFrom`
3. Once order matures, it automatically gets included in the next auction
4. CoW Protocol calls `isValidSignature` on-chain
   a. Trustless guarantees that the order won't get matched before its actually mature

Getting Rid of the API Call



```diff
@@ -59,6 +59,13 @@ contract GATOrders is ICoWSwapOnchainOrders {
                 data.sellAmount + data.feeAmount
             );

+            OnchainSignature memory signature = OnchainSignature({
+                scheme: OnchainSigningScheme.Eip1271,
+                data: hex""
+            });
+
+            emit OrderPlacement(address(instance), order, signature, data.meta);
+
             orderUid = new bytes(GPv2Order.UID_LENGTH);
             orderUid.packOrderUidParams(orderHash, address(instance), data.validTo);
         }
```

Getting Rid of the API Call*



```
@@ -59,6 +59,13 @@ contract GATOrders is ICoWSwapOnchainOrders {
             data.sellAmount + data.feeAmount
         );

+        OnchainSignature memory signature = OnchainSignature({
+            scheme: OnchainSigningScheme.Eip1271,
+            data: hex""
+        });
+
+        emit OrderPlacement(address(instance), order, signature, data.meta);
+
         orderUid = new bytes(GPv2Order.UID_LENGTH);
         orderUid.packOrderUidParams(orderHash, address(instance), data.validTo);
     }
```

Smart Orders

1. Stop-loss orders
    - Order that becomes valid once an on-chain oracle says your sell token goes below some price
2. Advanced GAT use-cases
    - Large order that becomes available a little at a time - such as a DAO selling some token little-by-little over a month
3. ...whatever else you can think of!

Smart Orders

1. Stop-loss orders
    - Order that becomes valid once an on-chain oracle says your sell token goes below some price
2. Advanced GAT use-cases
    - Large order that becomes available a little at a time - such as a DAO selling some token little-by-little over a month
3. ...whatever else you can think of!
    - Smart orders don't require special integration - **just an on-chain contract that follows the ERC-1271 standard**

Thank you

Stop searching **for better prices**

Github

@cowprotocol

Open positions

cow.fi/careers

Twitter

@CoWSwap

REFERENCES

1. https://github.com/nlordell/safe-cow-order
2. https://github.com/nlordell/dappcon-2022-smart-orders