ChatGPT

Signing an EIP-1559 (Type-2) Ethereum Transaction

Introduction: This guide provides a detailed, implementation-focused walkthrough for signing a post-London **EIP-1559** transaction (transaction type **0x02**). We will assemble the transaction, encode it, hash it, sign it with ECDSA (secp256k1), and serialize the final output. All steps follow the official **EIP-1559** specification [1]. We focus exclusively on EIP-1559 (type-2) rules (no legacy transaction support) and explain how **EIP-155** replay protection is applied.

EIP-1559 Transaction Format (Type 0x02)

EIP-1559 defines a new **type-2** transaction format under the EIP-2718 typed envelope scheme. A type-2 transaction is identified by a leading **transaction type byte** `0x02`, followed by an RLP-encoded payload of the transaction fields [2]. The fields included (in order) are:

- `chain_id` – The chain ID for replay protection (only valid on this chain) [3].
- `nonce` – Sender's account nonce (transaction sequence number).
- `max_priority_fee_per_gas` – Max tip (priority fee) per gas the sender is willing to pay.
- `max_fee_per_gas` – Max total fee (base fee + priority fee) per gas the sender will pay.
- `gas_limit` – Gas limit for this transaction.
- `to` (destination) – Recipient address (20 bytes) or `0x0` for contract creation.
- `value` – Amount of ETH to transfer (in wei).
- `data` – Transaction data payload (bytecode or call data; can be empty).
- `access_list` – List of accessed addresses and storage keys (can be an empty list).
- `signature_y_parity` – **Y-parity** of the signature (1 bit, 0 or 1).
- `signature_r` – ECDSA signature *r* value (32 bytes as integer).
- `signature_s` – ECDSA signature *s* value (32 bytes as integer).

These fields correspond to the EIP-1559 `TransactionPayload` structure [1]. The **Y-parity** (`signature_y_parity`) is essentially the ECDSA recovery identifier bit, replacing the old `v` value used in legacy transactions. (Since the `chain_id` is explicitly included in the payload, only a 0/1 parity is needed for the signature [3].)

EIP-155 Replay Protection: Including the `chain_id` in the signed payload ensures the transaction is only valid on that chain, implementing EIP-155 replay protection. In legacy (pre-2718) transactions, EIP-155 achieved this by encoding the chain ID into the signature `v` value (e.g. `v = chain_id * 2 + 35` or `36`) [4]. With EIP-1559 type-2 transactions, the `chain_id` is a dedicated field that is hashed as part of the signature, and the signature's recovery ID is stored separately as `y_parity` [3]. This means the signed transaction cannot be replayed on a different chain or interpreted as a different type, since the type byte `0x02` is also part of the hash [5].

Step-by-Step Signing Process (Pseudocode)

Below are the steps to construct, sign, and serialize an EIP-1559 transaction. We use a generic pseudocode style for clarity:

1. **Prepare the Transaction Fields:** Gather all required fields in the correct order defined by EIP-1559. For example:

```
fields = [
    chain_id,       // E.g. 1 for mainnet
    nonce,          // Sender's nonce
    max_priority_fee_per_gas,
    max_fee_per_gas,
    gas_limit,
    to,             // Recipient address (as integer or hex)
    value,          // Amount in wei
    data,           // Byte array of payload
    access_list     // List of [address, storageKeys] or [] if none
]
```

Make sure to use the exact order and types as specified. The `chain_id` comes first (providing replay protection), and `access_list` is included even if empty [1]. (No legacy `gasPrice` or legacy `v` field is used in type-2 transactions.)

1. **RLP-encode the Unsigned Transaction:** Use RLP (Recursive Length Prefix) encoding to serialize the list of fields **excluding** the signature. At this stage, do not include `y_parity`, `r`, or `s` (since we haven't signed yet). For example:

```
unsigned_rlp = RLP.encode(fields)
```

This produces the RLP byte array of the transaction payload without any signature. According to the spec, the transaction payload is defined as an RLP list of the fields (with signature fields to be added later) [1].

1. **Compute the Message Hash:** Calculate the Keccak-256 hash of the **transaction type prefix** concatenated with the RLP-encoded payload from step 2. EIP-1559 requires the type byte `0x02` to be included in the hash to sign [5]. Pseudocode:

```
tx_hash = keccak256( 0x02 || unsigned_rlp )
```

Here `||` denotes byte concatenation. This hash (32 bytes) is the message that will be signed. Including the `0x02` ensures the signature cannot be applied to a different transaction type (it "domains" the signature to type-2 transactions) [5] .

1. **Sign with ECDSA (secp256k1):** Using the sender's private key, produce a secp256k1 signature of `tx_hash`. The result will be the standard ECDSA components `(r, s)` and a recovery identifier (often called `v` in legacy, but here we'll derive parity separately). For example:

```
(r, s, rec_id) = secp256k1_sign(private_key, tx_hash)
```

The `secp256k1_sign` function returns the signature values. The `rec_id` is the recovery ID (0 or 1 in this context) which indicates which elliptic curve point was used in the signature. (If using a library that returns `v` as 27/28, subtract 27 to get the parity bit.)

1. **Determine the `y_parity`:** Convert the recovery ID into the **y-parity bit**. In EIP-1559, this is simply `0` or `1` corresponding to the even/odd parity of the curve point's y-coordinate [3] . For example:

```
y_parity = rec_id   // rec_id is 0 or 1 for secp256k1 signatures
```

There is no additional adjustment needed for chain ID here – the chain ID is already part of the hashed payload. (In other words, **v = 27/28 or chainId formula is not used**; we use the raw parity bit.)

1. **Add Signature Fields to the Payload:** Append the signature components to the field list in order to form the complete signed transaction payload. According to EIP-1559, the fields `signature_y_parity`, `signature_r`, `signature_s` come last [1] . For example:

```
signed_fields = fields + [ y_parity, r, s ]
```

Now `signed_fields` represents `[chain_id, nonce, max_priority_fee_per_gas, max_fee_per_gas, gas_limit, to, value, data, access_list, y_parity, r, s]`. This matches the full EIP-1559 transaction format with all signature data included [1] .

1. **RLP-encode the Signed Transaction:** RLP-encode the `signed_fields` list to get the serialized transaction payload with the signature. For example:

```
signed_tx_rlp = RLP.encode(signed_fields)
```

This produces the byte array encoding of the entire transaction (except the type prefix). At this point, the RLP includes the signature values. (If decoded, it would match the spec's transaction payload structure exactly.)

1. **Serialize with Type Byte:** Prepend the transaction type byte `0x02` to the RLP from step 7. This yields the final raw transaction byte stream. For example:

```
raw_transaction = 0x02 || signed_tx_rlp
```

The `raw_transaction` is the complete signed transaction in binary form, ready to be broadcast to the Ethereum network or included in a block. In hex notation, it will start with `0x02` to indicate an EIP-1559 transaction. *(The type byte is not part of the RLP itself but is the prefix as per EIP-2718's format [6] .)*

Each step above follows the official EIP-1559 specification for type-2 transactions. Notably, the signature is defined as over the RLP-encoded fields plus the type byte, and the signed output is encoded back into the same structure with the signature attached [5]. By including the `chain_id` in the signed content, EIP-1559 inherently provides replay protection across chains (fulfilling EIP-155) without needing the old `v`-value formula [4] [3].

References:

- Ethereum Improvement Proposal **1559** – *"Fee market change for ETH 1.0 chain"* (London hardfork) [1]
- Ethereum Improvement Proposal **2930** – *"Optional access lists"* (defines `chainId` and `yParity` in transaction format) [3]
- Ethereum Improvement Proposal **155** – *"Simple replay attack protection"* (legacy chain ID in signatures) [4]

---

[1] [2] [5] EIP-1559: Fee market change for ETH 1.0 chain
https://eips.ethereum.org/EIPS/eip-1559

[3] [6] EIP-2930: Optional access lists
https://eips.ethereum.org/EIPS/eip-2930

[4] EIP-155: Simple replay attack protection
https://eips.ethereum.org/EIPS/eip-155