# EELS

## The future of Ethereum Execution Layer Specifications

**Peter Davies**

EELS Team

# What do I mean by “Execution Layer”?

- We only care about the “state transition function”.
  - Can a new block be added to the end of a chain?
  - What happens to the chain state when we add a block?
- Everything else is out of scope.
  - Fork choice
  - Reorgs
  - Networking
  - APIs (e.g. JSON RPC)
  - Performance
  - Transaction Gossip

# Current Sources of Information

- Yellow Paper
- EIPs
- Testsuite
- Client source code

Thus we are able to define the block header validity function $V(H)$:

$$
\begin{aligned}
(56) \quad V(H) \quad \equiv \quad & n \leqslant \frac{2^{256}}{H_{\mathrm{d}}} \wedge m = H_{\mathrm{m}} \quad \wedge \\
& H_{\mathrm{d}} = D(H) \quad \wedge \\
& H_{\mathrm{g}} \leq H_{\mathrm{l}} \quad \wedge \\
& H_{\mathrm{l}} < P(H)_{\mathrm{H_l}} + \left\lfloor \frac{P(H)_{\mathrm{H_l}}}{1024} \right\rfloor \quad \wedge \\
& H_{\mathrm{l}} > P(H)_{\mathrm{H_l}} - \left\lfloor \frac{P(H)_{\mathrm{H_l}}}{1024} \right\rfloor \quad \wedge \\
& H_{\mathrm{l}} \geqslant 5000 \quad \wedge \\
& H_{\mathrm{s}} > P(H)_{\mathrm{H_s}} \quad \wedge \\
& H_{\mathrm{i}} = P(H)_{\mathrm{H_i}} + 1 \quad \wedge \\
& \|H_{\mathrm{x}}\| \leq 32
\end{aligned}
$$

where $(n, m) = \mathtt{PoW}(H_{\not n}, H_{\mathrm{n}}, \mathbf{d})$

Noting additionally that **extraData** must be at most 32 bytes.

Defines the gas cost of the `ModExp` ( `0x00..05` ) precompile.

## Abstract

To accurately reflect the real world operational cost of the `ModExp` precompile, this EIP specifies an algorithm for calculating the gas cost. This algorithm approximates the multiplication complexity cost and multiplies that by an approximation of the iterations required to execute the exponentiation.

## Motivation

Modular exponentiation is a foundational arithmetic operation for many cryptographic functions including signatures, VDFs, SNARKs, accumulators, and more. Unfortunately, the ModExp precompile is currently over-priced, making these operations inefficient and expensive. By reducing the cost of this precompile, these cryptographic functions become more practical, enabling improved security, stronger randomness (VDFs), and more.

## Specification

As of `FORK_BLOCK_NUMBER`, the gas cost of calling the precompile at address `0x0000000000000000000000000000000000000005` will be calculated as follows:

```
def calculate_multiplication_complexity(base_length, modulus_length):
    max_length = max(base_length, modulus_length)
    words = math.ceil(max_length / 8)
    return words**2

def calculate_iteration_count(exponent_length, exponent):
    iteration_count = 0
    if exponent_length <= 32 and exponent == 0: iteration_count = 0
    elif exponent_length <= 32: iteration_count = exponent.bit_length() - 1
    elif exponent_length > 32: iteration_count = (8 * (exponent_length - 32)) + ((exp
    return max(iteration_count, 1)

def calculate_gas_cost(base_length, modulus_length, exponent_length, exponent):
    multiplication_complexity = calculate_multiplication_complexity(base_length, modu
    iteration_count = calculate_iteration_count(exponent_length, exponent)
    return max(200, math.floor(multiplication_complexity * iteration_count / 3))
```

## Rationale

# Specifications need to be part of standards processes

- Updating standards can't be an afterthought
- Code that isn't tested isn't worth anything

*"Beware of bugs in the above code; I have only proved it correct, not tried it."*
*– Donald Knuth*

# Our approach

- Specifications are written in code
  - Python without classes/methods (basically pseudocode)
  - Common language of all programmers
  - Can be executed
- Focus solely on readability
  - Performance is for real clients
- Keep forks separate rather than lots of conditionals
  - Horrendous for code duplication, great for the casual reader
  - Specialist diff tools for comparing hardforks

```python
def sload(evm: Evm) -> None:
    """
    Loads to the stack, the value corresponding to a certain key from the
    storage of the current account.

    Parameters
    ----------
    evm :
        The current EVM frame.

    """
    # STACK
    key = pop(evm.stack).to_be_bytes32()

    # GAS
    if (evm.message.current_target, key) in evm.accessed_storage_keys:
        charge_gas(evm, GAS_WARM_ACCESS)
    else:
        evm.accessed_storage_keys.add((evm.message.current_target, key))
        charge_gas(evm, GAS_COLD_SLOAD)

    # OPERATION
    value = get_storage(evm.env.state, evm.message.current_target, key)

    push(evm.stack, value)

    # PROGRAM COUNTER
    evm.pc += 1
```

# The two sides of development

- R&D people (e.g. Vitalik Buterin)
  - Interested in theoretical concerns
  - Don't care about performance complexities
  - Want a flexible playground
- Implementers (e.g. Péter Szilágyi)
  - Care about precise details
  - Want to focus on complicated performance issues (DB structure, etc...)

EELS provides a common framework for these two sides to talk to each other.

# Development stages in an EELS world

R&D:

1. Develop an idea to improve the execution layer
2. Prototype the idea in EELS

EELS:

3. Integrate with other proposals to make a hardfork in EELS
4. Fill tests, start ephemeral testnets?

Implementers:

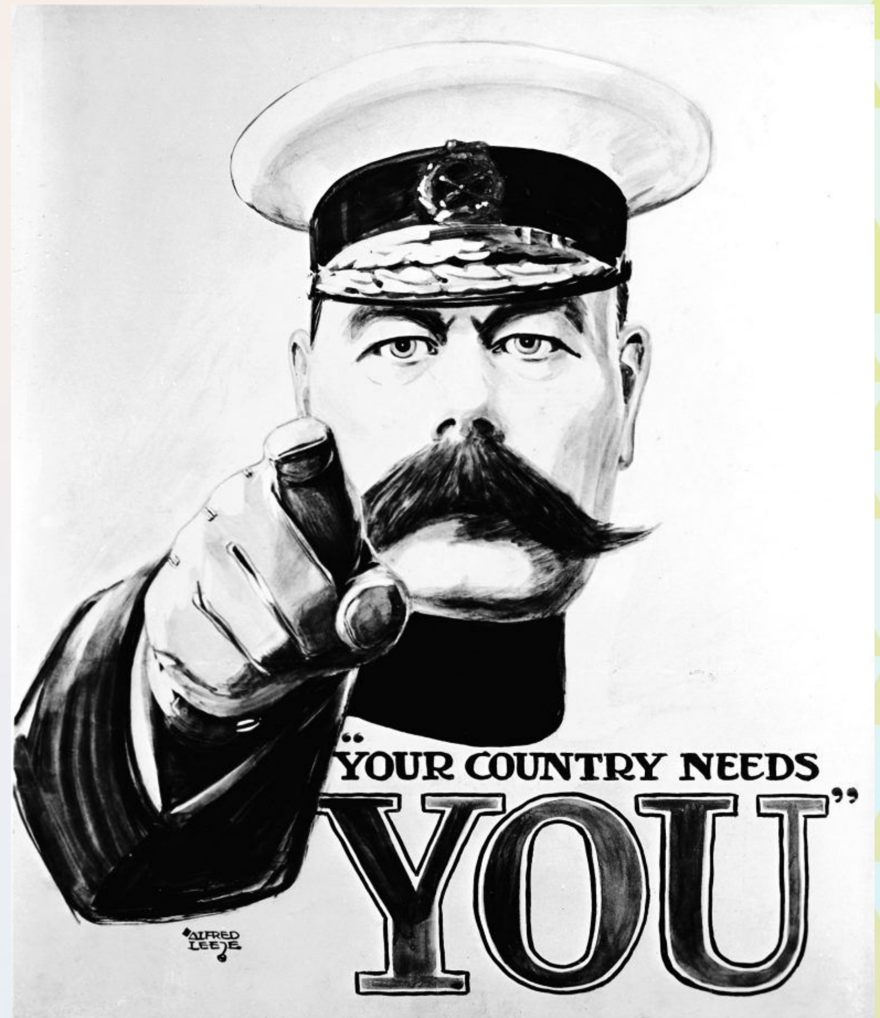
5. Implement in production clients
6. Deploy on testnets and mainnet

# Current Status

- All hardforks are implemented (The Merge is still a PR)
- Refactoring complete and code freeze in November (hopefully)
- Shanghai governance shadowing

# How you can help!

- We don’t need your help until we’ve finished coding
- Implement your favourite EIP and give us feedback



# Questions?

ethereum.github.io/execution-specs

github.com/ethereum/execution-specs

# Thanks for listening!

ethereum.github.io/execution-specs

github.com/ethereum/execution-specs