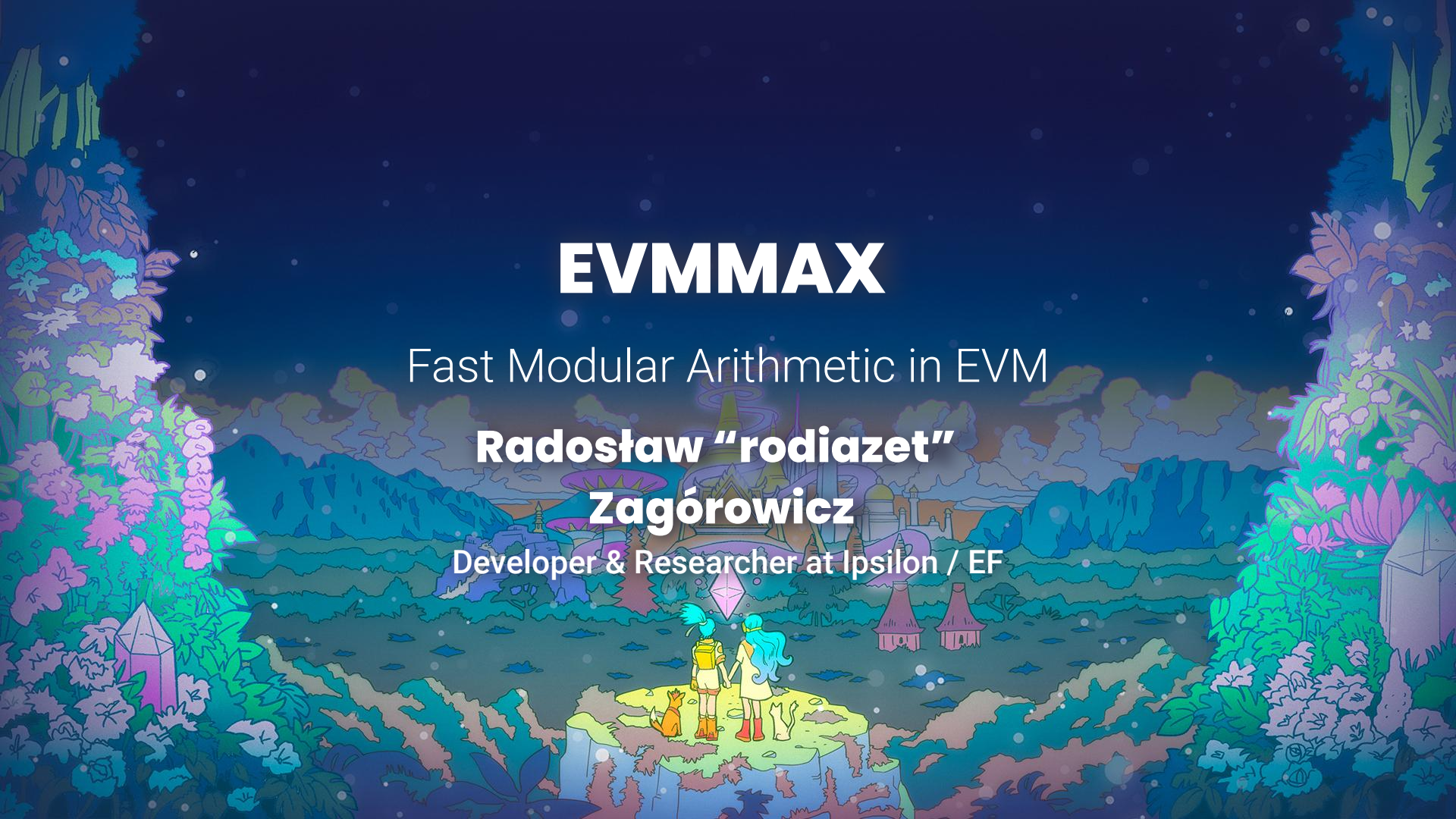
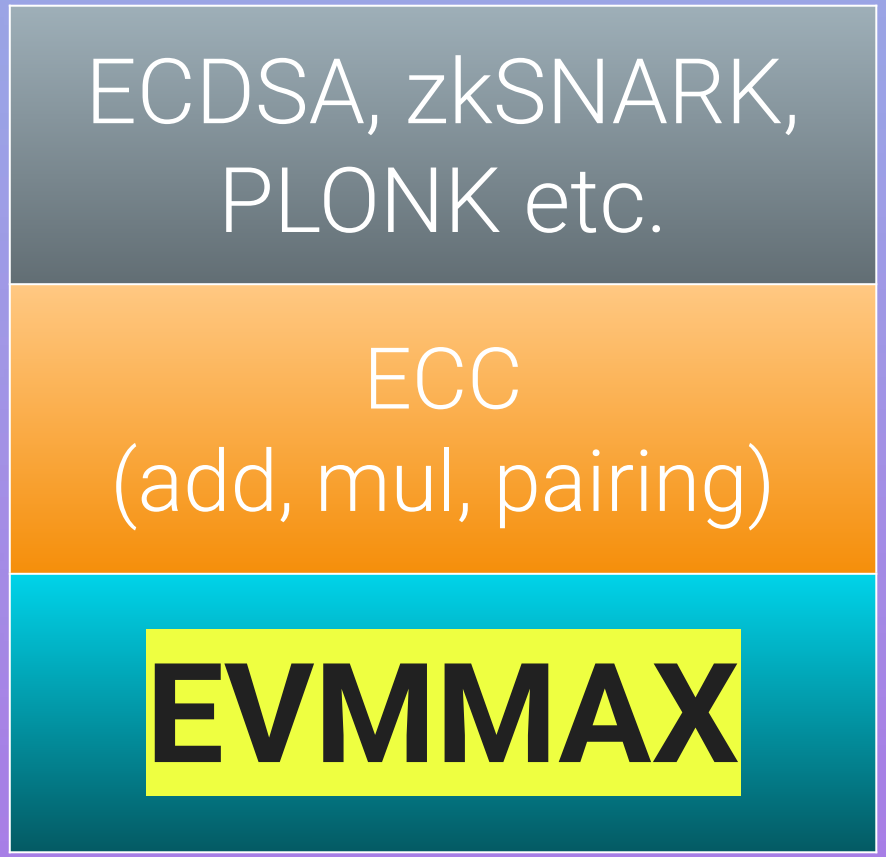

EVMMAX
Fast Modular Arithmetic in EVM
Radosław “rodiazet” Zagórowicz
Developer & Researcher at Ipsilon / EF

Section 1

Introduction

What is EVMMAX

- EVMMAX = EVM Modular Arithmetic Extension
- New set of modular arithmetic instructions for odd modulus
  - Addition
  - Subtraction
  - Multiplication
  - ?Exponentiation
- Built on the top of EOF
  - Utilizes EOF immediate arguments and validation



Where EVMMAX lives

- **Modular arithmetic operations.**
- They are used to implemented basic operations on elliptic curves points like point addition, multiplication or multi scalar multiplications. Or more complex like pairing verification.
- The above are used to implement more advanced ECC based algorithms like ECDSA or ZK related algorithms.

Why EVMMAX

- Convenient way to implement cryptography related functions efficiently in the EVM.
- No need to wait for precompile which implements this.
- Avoid implementation specific problems when adding new precompile.
- Get rid of adding so many new precompiles
- EVMMAX as a tool to define reference precompile implementation if still needed
- Custom cryptography functions for specific problem

Section 2

Instructions

EVMMAX instructions overview

1. EVMMAX Context/modulus setup
2. Modular arithmetic operations
3. EVM <-> EVMMAX communication

```
setupx[id, modulus, vals_used]
```

- Arguments
  - Context identifier
  - Modulus value
  - Number value slots being used
- Creates new context
  - Sets field modulus
  - Initialises needed constants (i.e. r^2)
  - Initialises scratch buffer
- Switches context

```
addmodx
submodx [result, x, y]
mulmodx
```

- Arguments
  - Result value slot index
  - Left operand value slot index
  - Right operand value slot index
- Modular addition/subtraction/multiplication of the values in the operand slots

EVM → EVMMAX

```
storex[dest, src, num_vals]
```

EVMMAX → EVM

```
loadx[dest, idx, num_vals]
```

Stores values from EVM memory in EVMMAX value slots

- Destination value start slot
- Source memory pointer
- Number of values to store

Loads values back to EVM memory

- Destination memory pointer
- Value slot start index
- Number of values to load

Section 3

Precompiles

Precompiles issues

- Implemented by many different libraries across clients
- Potential consensus issue if different implementations return different result for the same input
- Precompiles functions details are not specced out
- The above results in redundant operations in the implementation

Curve equation*

$$y^2 = x^3 + 3$$

Points on curve

$$P_1 = (x_1, y_1)$$
$$P_2 = (x_2, y_2)$$

P_1, P_2 slope

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}$$

sub, sub, div

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_3 - x_1) - y_1$$

mul, sub, sub, mul, sub, sub

Result point

$$P_3 = (x_3, y_3) = P_1 \oplus P_2$$

BN254 point addition

1. Check if $x_1 \neq x_2$. If equal -> special case.
2. Calculate slope of a line intersecting two input points
3. Calculate result point coordinates
4. **6 subs + 2 muls + 1 div**

** over prime field F_p*

Division in modular arithmetic

- Very expensive and complicated code comparing to other basic modular operations
- Should be avoided if possible
- Single division is an equivalent of **~300 muls**.

How it's done in real

- Different algorithms (i.e. projective or Jacobian coordinates)
- No division when adding points $(P_1+P_2+P_3+P_3+...+P_n)$
- **At most one** division if really needed

Benefit EVMMAX over precompiles

- **Improve efficiancy:** No redundant computation
- **More flexibility:** Alibity to implement custom cryptography algorithms according exact project requirements
- **Security:**
  - No possible consensus issue
  - Much lower attack surface
  - Complexity moved from EVM to smart contract
- **Faster shipment:** No need to wait for a fork

Section 4

Summary

What has been done?

- Spec *in progress* (EIP-5843, EIP-6690, maybe more?)
- Experimental implementation of EVMMAX in evmone
- Precompiles implemented:
  - BN254 ecmul, ecadd (faster than libff, EVM overhead ~25%)
- Initial spec of vectorized EVMMAX version + experimental implementation in Go. Thanks to @jwasinger

What to do next

New use cases:

- More precompiles (i.e. bls curve)
- Poseidon hash function

Features:

- 2^k modulus support
- instruction set extension
- utilisation of SIMD (AVX)

Tools:

- Support in high level languages (yul, solidity, huff, etc.)
- Language for future precompiles specification
- [EVMMAX -> High level languages] transpiler

Our goals

- Finalize EVMMAX spec
- To equip EVM in a cryptography friendly toolchain
- To minimize need of adding new precompiles
- Make EVM more ZK friendly
- To introduce on mainnet in 2025/26

Work on EVMMAX

- @ipsilon
- @jwasinger (geth)
- @Kev (ARG)

Thank you for your attention

Q&A