Computational Cost Analysis of Poseidon as an EVM Precompile

Presenter: Marcos Villagra

August 2023

Draft | Standards Track: Core

EIP-5988: Add Poseidon hash function precompile

Add a precompiled contract which implements the hash function used in the Poseidon cryptographic hashing algorithm

Authors	Abdelhamid Bakhta (@abdelhamidbakhta), Eli Ben Sasson (@Elistark), Avihu Levy (@avihu28), David Levit Gurevich (@DavidLevitGurevich)

Created	2022-11-15
Discussion Link	https://ethereum-magicians.org/t/eip-5988-add-poseidon-hash-function-precompile/11772

Table of Contents

Outline

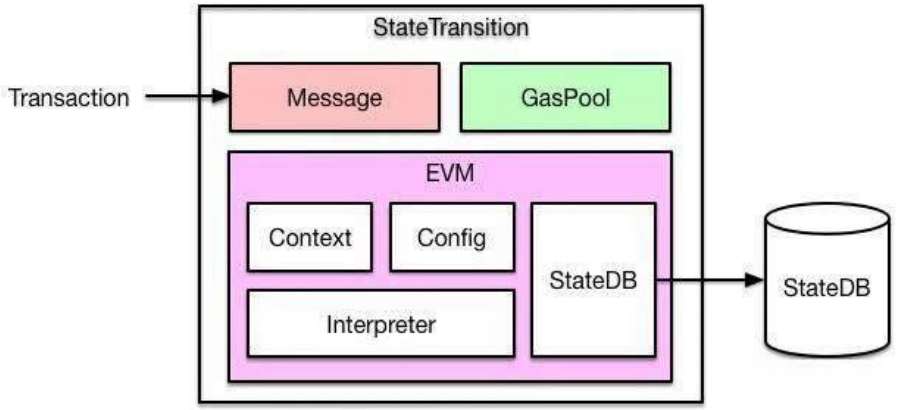
1. Precompiled contracts.
2. Poseidon Hash.
3. Motivation behind EIP-5988.
4. Project proposal.
    a. Goals
    b. Methodology.
    c. Expected results.
    d. Potential problems.

Outline

1. Precompiled contracts.
2. Poseidon Hash.
3. Motivation behind EIP-5988.
4. Project proposal.
    a. Goals
    b. Methodology.
    c. Expected results.
    d. Potential problems.

Precompiled Contracts

- A precompiled contract is code that is complex to execute on the EVM, for example, elliptic curve operations.
- Reserved for frequently used subroutines.
- Simple, but not suitable for EVM opcode.
- Runs on the client.
- Gas cost is fixed.

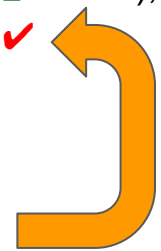


```go
// run runs the given contract and takes care of running precompiles with a fallback to the byte code interpreter.
func run(evm *EVM, contract *Contract, input []byte, readOnly bool) ([]byte, error) {
    if contract.CodeAddr != nil {
        precompiles := PrecompiledContractsHomestead
        if evm.ChainConfig().IsByzantium(evm.BlockNumber) {
            precompiles = PrecompiledContractsByzantium
        }
        if p := precompiles[*contract.CodeAddr]; p != nil {
            return RunPrecompiledContract(p, input, contract)
        }
    }
    for _, interpreter := range evm.interpreters {
        if interpreter.CanRun(contract.Code) {
            if evm.interpreter != interpreter {
                // Ensure that the interpreter pointer is set back
                // to its current value upon return.
                defer func(i Interpreter) {
                    evm.interpreter = i
                }(evm.interpreter)
                evm.interpreter = interpreter
            }
            return interpreter.Run(contract, input, readOnly)
        }
    }
    return nil, ErrNoCompatibleInterpreter
}
```

Precompiled Contracts (cont'd)

List of currently implemented precompiled contracts.

1. `ecRecover` (ECDSA public key recovery function),
2. `SHA2-256` (hash functon), ✔
3. `RIPEMD-160` (hash function), ✔
4. `identity` (identity function),
5. `modexp` (modular exponentiation),
6. `ecAdd` (point addition on the elliptic curve `alt_bn128`),
7. `ecMul` (scalar multiplication on the elliptic curve `alt_bn128`),
8. `ecPairing` (bilinear function on groups over the elliptic curve `alt_bn128`),
9. `blake2f` (compression function `F` in the `BLAKE2` hash algorithm). ✔



only compression function

Outline

1. Precompiled contracts.
2. Poseidon Hash.
3. Motivation behind EIP-5988.
4. Project proposal.
    a. Goals
    b. Methodology.
    c. Expected results.
    d. Potential problems.

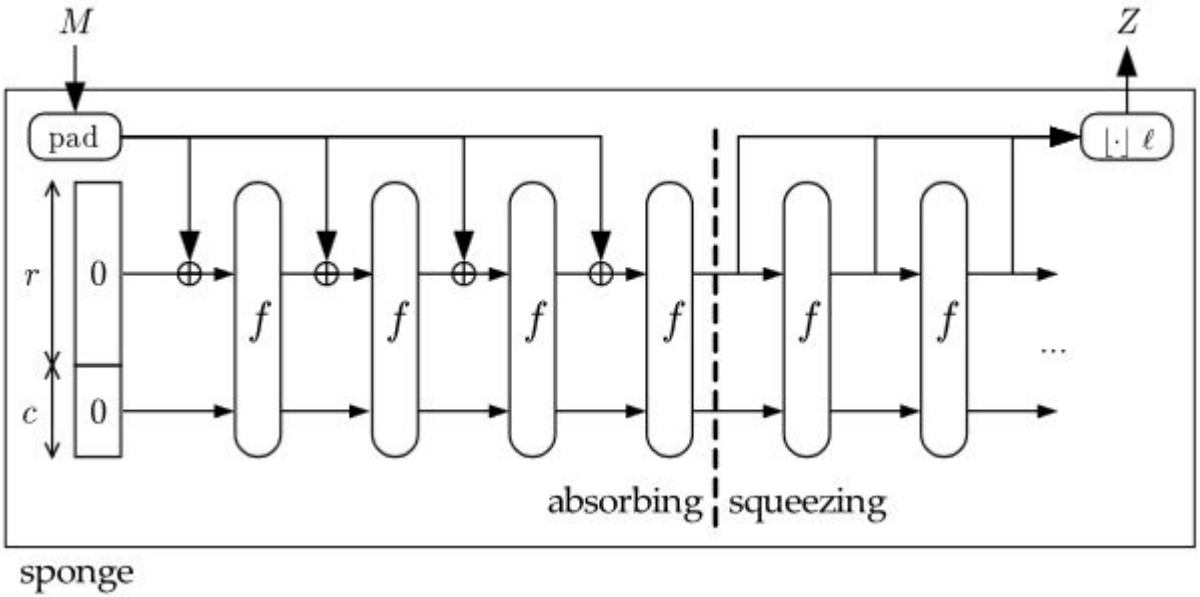
Poseidon Hash

- New hash function designed to be used in proof systems.[1]
- Arithmetization-oriented.
- Sacrifices execution times in favor of arithmetic circuit sizes.
- Based on a sponge construction.
- Poseidon permutation based on a Hades strategy.

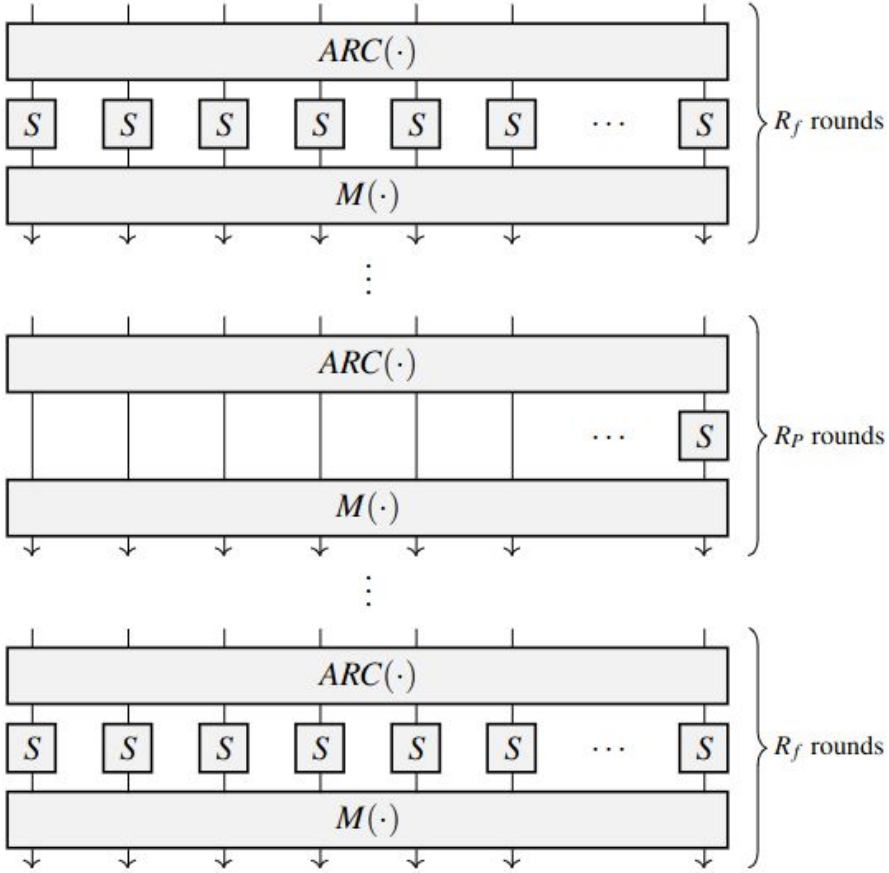
[1]Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., & Schofnegger, M. (2021). Poseidon: A new hash function for Zero-Knowledge proof systems. In *30th USENIX Security Symposium (USENIX Security 21)* (pp. 519-535).

Poseidon Hash (cont'd)

Sponge construction



Poseidon Hash (cont'd)



Poseidon one-way permutation

- **ARC:** add round constans.
- **S:** x^n-box, n>=3.
- **M**: multiply by *t*×*t* MDS matrix.

Parameters:

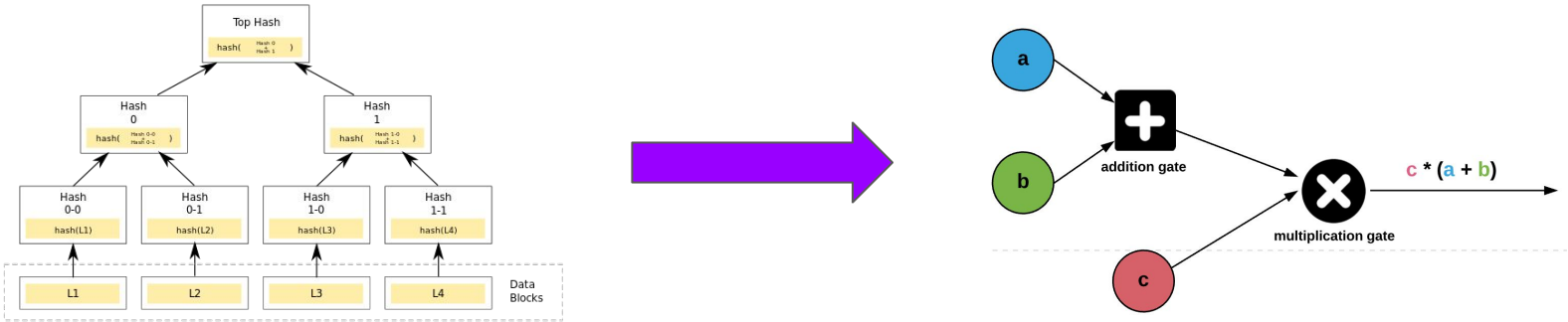
- Round constants.
- Number of rounds.
- S-box.
- MDS matrix.

Outline

1. Precompiled contracts.
2. Poseidon Hash
3. Motivation behind EIP-5988.
4. Project proposal.
    a. Goals
    b. Methodology.
    c. Expected results.
    d. Potential problems.

Motivation behind EIP-5988

- Reduce prover computation times and proof lengths in proof systems.
- Efficient zk-rollups.
- Imagine proving knowledge of a leaf in a Merkle tree.



- Number of constraints (R1CS) is linearly related to the size of the circuit.

Motivation behind EIP-5988 (cont'd)

Table 4: Number of R1CS constraints for a circuit proving a leaf knowledge in the Merkle tree of 2^{30} elements.

Arity	Width	R_F	R_P	Total constraints
POSEIDON-128				
2:1	3	8	57	7290
4:1	5	8	60	4500
8:1	9	8	63	4050
Rescue-x^5				
2:1	3	16	-	8640
4:1	5	10	-	4500
8:1	9	10	-	5400
Pedersen hash				
510	171	-	-	41400
SHA-256				
510	171	-	-	826020
Blake2s				
510	171	-	-	630180
MiMC-2p/p (Feistel)				
1:1	2	324	-	19440

Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., & Schofnegger, M. (2021). Poseidon: A new hash function for Zero-Knowledge proof systems. In *30th USENIX Security Symposium (USENIX Security 21)* (pp. 519-535).

Outline

1. Precompiled contracts.
2. Poseidon Hash.
3. Motivation behind EIP-5988.
4. Project proposal.
    a. Goals
    b. Methodology.
    c. Expected results.
    d. Potential problems.

Project Proposal

- Main goal: Propose execution benchmarks to support the acceptance or rejection of EIP-5988.

Justification:

Poseidon was designed to reduced the number of constraints in R1CS and not on improving execution times.

Similar debate regarding the adoption of the `blake2f` precompile:

- What are the gas cost for a “normal” smart contract?
- What are the execution times?
- Implementation in all languages currently used by clients.
- Estimated gas cost: 1 gas per 1 microsecond.

Project Proposal (cont’d)

Methodology:

- Code Poseidon in Solidity to estimate gas costs for a common smart contract.
- Code Poseidon in some compiled language (C, C++, Go, and/or similar) following the specs of EIP-5988 to estimate execution times.
- Try different combinations of parameters of Poseidon.
- Compare the execution times of Poseidon against other hash functions and hash precompiles.
- Proposed a gas cost for the precompile.
- Do the same for Poseidon2. (maybe not enough time)

Project Proposal (cont’d)

Expected results:

- Open source code of Poseidon in Solidity.
- Open source code of Poseidon in the selected language/s.
- Technical report with benchmark results.
- The same for Poseidon2 (maybe not enough time).

Project Proposal (cont'd)

Roadmap:

Week	# Week	Activities						
		Learn Rust	Poseidon (Rust)	Benchmarking	Poseidon (Solidity)	Tecnical Report	Devconnect prep.	Devconnect
14-Aug to 20-Aug	1	2 Weeks						
21-Aug to 27-Aug	2		3 weeks					
28-Aug to 3-Sep	3							
4-Sep to 10-Sep	4			3 weeks				
11-Sep to 17-Sep	5							
18-Sep to 24-Sep	7							
25-Sep to 01-Oct	8				3 weeks			
2-Oct to 8 Oct	9							
9-Oct to 15-Oct	10					4 weeks		
16-Oct to 22-Oct	11							
23-Oct to 29-Oct	12							
30-Oct to 5-Nov	13						2 weeks	
6-Nov to 12-Nov	14							
13-Nov to 19-Nov	15							1 week

Project Proposal (cont’d)

Potential problems:

- Not finishing on time by Devconnect 2023.

Questions?