24.05.14 Halo2-lib cost and Mithril chain

Xun Zhang Bingsheng Zhang
Zhejiang University, CHN
22221024@zju.edu.cn bingsheng@zju.edu.cn

May 14 2024

1 Halo2-lib Group Operation Benchmark

Following are the benchmark results of group operation in halo2-lib crate.

degree	advice	lookup	lookup_bits	limb_bits	proof_time	proof_size	verify_time
15	10	2	14	88	2.2243s	4128	8.055ms
16	5	1	15	90	2.5095s	2272	7.540ms
17	3	1	16	88	3.8706s	1696	9.775ms
18	2	1	17	88	6.7053s	1344	14.639ms
19	1	0	18	90	11.0751s	960	23.502ms

Table 1: Bn254 G2 Addition Cost(Batch size = 100)

degree	advice	lookup	lookup_bits	limb_bits	proof_time	proof_size	verify_time
16	170	23	15	88	51.5202s	59584	163.06ms
17	84	11	16	88	47.4618s	29120	133.59ms
18	42	6	17	88	49.9059s	15008	135.64ms
19	20	3	18	90	51.1550s	7360	131.00ms
20	11	2	19	90	67.8328s	4128	170.56ms

Table 2: Bn254 G2 MSM Cost(Batch size = 100)

degree	advice	lookup	lookup_bits	limb_bits	proof_time	proof_size	verify_time
14	211	27	13	91	19.5710s	72416	68.38ms
15	105	14	14	90	16.4763s	36416	47.59ms
16	50	6	15	90	14.8747s	17312	48.95ms
17	25	3	16	88	15.2069s	8864	31.46ms
18	13	2	17	88	17.9298s	4928	37.19ms
19	6	1	18	90	20.3571s	2496	38.85ms
20	3	1	19	88	30.7017s	1696	55.82ms
21	2	1	20	88	51.7027s	1344	105.05ms
22	1	1	21	88	91.7061s	960	194.89ms

Table 3: Bn254 Pairing Cost

2 Signature Verification Proving Time

Following are the benchmark results of signature verification in halo2-lib and halo2-lib-eddsa crates.

degree	advice	lookup	lookup_bits	limb_bits	proof_time	proof_size	verify_time
14	211	27	13	91	25.3671s	95808	119.31ms
15	105	14	14	90	22.8869s	48000	63.41ms
16	50	6	15	90	20.6673s	22752	54.23ms
17	25	3	16	88	20.4768s	11520	50.50ms
18	13	2	17	88	22.5167s	6080	52.44ms
19	6	1	18	90	24.9818s	3072	56.68ms
20	3	1	19	88	36.0663s	1920	85.64ms
21	2	1	20	88	56.5497s	1344	125.96ms

Table 4: Bn254 BLS Signature Verification Cost

degree	advice	lookup	lookup_bits	limb_bits	proof_time	proof_size	verify_time
19	1	1	18	88	11.7409s	960	20.68ms
18	2	1	17	88	7.1555s	1344	16.03ms
17	4	1	16	88	4.4740s	1920	10.86ms
16	8	2	15	90	3.9142s	3776	11.29ms
15	17	3	14	90	3.5269s	6784	12.91ms
14	34	6	13	91	3.8446s	13984	17.18ms
13	68	12	12	88	4.4652s	27072	30.02ms

Table 5: Secp256k1 ECDSA Signature Verification Cost

degree	advice	lookup	lookup_bits	limb_bits	proof_time	proof_size	verify_time
19	1	1	18	88	17.4019s	1920	28.54ms
18	2	1	17	88	13.1093s	3200	24.94ms
17	4	1	16	88	11.4594s	5632	22.69ms
16	8	2	15	90	10.1348s	10976	25.03ms
15	17	3	14	90	10.7261s	22688	37.71ms
14	34	6	13	91	12.2806s	46240	49.46ms
13	68	12	12	88	15.4048s	94048	82.81ms

Table 6: ED25519 EDDSA Signature Verification Cost

3 Mithril Certificate Chain

3.1 Certificate Chain Introduction

The **certificate chain** is a Mithril component that certifies the **stake distribution** used to create the multi-signature. Its primary purpose is to prevent adversaries from executing an **eclipse attack** on the blockchain.

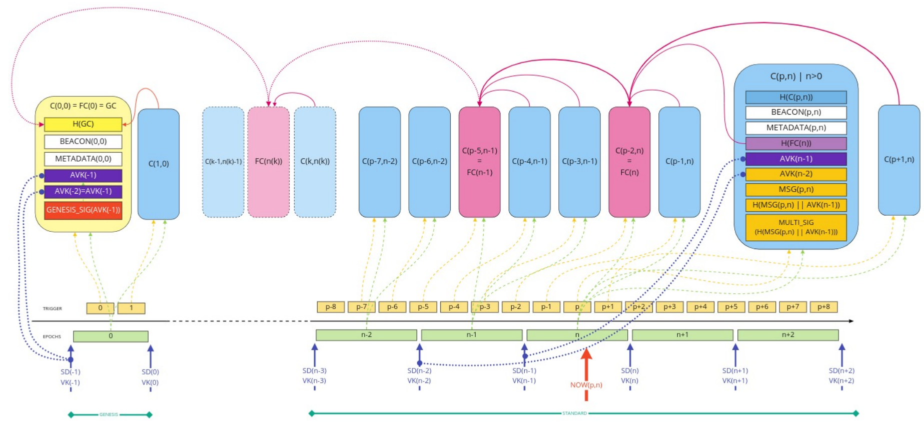
Without the certificate, the stake distribution can't be trusted. A malicious actor could relatively easily create a fake stake distribution and use it to produce a valid multi-signature, which would be embedded in a valid but non-genuine certificate. This certificate could be served by a dishonest Mithril aggregator node, leading an honest Mithril client to restore a non-genuine snapshot.

The way to certify the stake distribution used to create a multi-signature is by verifying that it has been previously signed in an earlier certificate. Then, one can recursively verify that the earlier certificate is valid in the same manner. This process can be structured as a chain of certificates, known as the Mithril certificate chain. The first certificate in the chain is discussed below.

Since multiple certificates can be created during the same epoch using the same stake distribution, it is not necessary to link to all of them for verification. Instead, it is sufficient to link to only one certificate from the previous epoch. By doing so, the verification process becomes faster and helps avoid network congestion.

The first certificate in the certificate chain is known as the genesis certificate. Validating the stake distribution embedded in the genesis certificate is only possible by signing it with a private key linked to a widely accessible public key called the genesis key. The use of these specific keys ensures the integrity and security of the initial stake distribution and subsequent transitions within the blockchain network.

The diagram below presents the certificate chain design:



Where the following notations have been used:

- C(p,n): Certificate at trigger p and epoch n
- FC(n): First certificate of epoch n
- GC: Genesis certificate
- H(): Hash
- SD(n): Stake distribution of epoch n
- VK(n): Verification key at epoch n
- AVK(n): Aggregrate verification key at epoch n such as $AVK(n) = MKT\_ROOT(SD(n)||VK(n))$
- MKT_ROOT(): Merkle-tree root
- BEACON(p,n): Beacon at trigger p and epoch n
- METADATA(p,n): Metadata of the certificate at trigger p and epoch n
- MSG(p,n): Message of the certificate at trigger p and epoch n
- MULTI_SIG(p,n): Multi-signature created to the message $H(MSG(p,n)||AVK(n-1))$

3.2 How to Validate a Certificate Chain

The **aggregate verification key (AVK)** is the root of the Merkle tree where each leaf is filled with $H(STAKE(signer)||VK(signer))$. It represents the corresponding stake distribution in a condensed way.

- To validate a **certificate chain**: a least a valid certificate per epoch.

- To validate a **non-genesis certificate**: iff the AVK used to verify the multi-signature is also part of the signed message used to create a valid multi-signature in a previously sealed certificate.
- To validate a **genesis certificate**: iff its genesis signature is verified with the advertised public genesis key.

An implementation of the algorithm would work as follows for a certificate:

- **Step 1**: Use this certificate as current_certificate.
- **Step 2**: Verify (or fail) that the current_hash of the current_certificate is valid by computing it and comparing it with the hash field of the certificate.
- **Step 3**: Get the previous_hash of the previous_certificate by reading its value in the current_certificate.
- **Step 4**: Verify (or fail) that the multi_signature of the current_certificate is valid.
- **Step 5**: Retrieve the previous_certificate that has the hash previous_hash.
  - **Step 5.1**: If it is not a genesis_certificate:
    - **Step 5.1.1**: Verify (or fail) that the previous_hash of the previous_certificate is valid by computing it and comparing it with the hash field of the certificate.
    - **Step 5.1.2**: Verify the current_avk:
      - **Step 5.1.2.1**: If the current_certificate is the first_certificate of the epoch, verify (or fail) that the current_avk of the current_certificate is part of the message signed by the multi-signature of the previous_certificate.
      - **Step 5.1.2.2**: Else verify (or fail) that the current_avk of the current_certificate is the same as the current_avk of the previous_certificate.
    - **Step 5.1.3**: Verify (or fail) that the multi_signature of the previous_certificate is valid .
    - **Step 5.1.4**: Use the previous_certificate as current_certificate and start again at **Step 2**.
  - **Step 5.2**: If it is a genesis_certificate:
    - **Step 5.2.1**: Verify (or fail) that the previous_hash of the previous_certificate is valid by computing it and comparing it with the hash field of the certificate.
    - **Step 5.2.2**: Verify (or fail) that the current_avk of the current_certificate is part of the message signed by the genesis signature of the previous_certificate.
    - **Step 5.2.3**: The certificate is valid (success).

3.3 The coexistence of multiple certificate chains

What would happen if some Mithril aggregator claims that not enough signatures were received? This doesn't really matter, as there will be a different Mithril aggregator that would collect sufficient signatures and aggregate them into a valid certificate.

Similarly, different Mithril aggregators might have different views of the individual signatures submitted (one aggregator might receive 10 signatures, and a different one could receive 11), which would result in different certificates signing the same message.

This would result in different certificate chains that would all link back to the genesis certificate. Indeed they would be represented by a tree of certificates where each traversal path from the root to a leaf represents a valid certificate chain.