Study & Design of Hashing Algorithms

SHA-256 | MurmurHash | BLAKE3 | Custom Algorithm (MODHASH)

Author: Sahil Madaan

What is Hashing?

- A function that transforms input data into a fixed-size value (hash).
- Widely used in:
  - Cryptography
  - Data Integrity
  - Hash Tables / Indexing
  - Digital Signatures
- Properties of good hash functions:
  - Deterministic
  - Fast to compute
  - Uniform output
  - Collision-resistant
  - Non-reversible (for crypto)

Algorithm #1 – SHA-256

- **Type:** Cryptographic Hash
  Standard: Part of SHA-2 (NIST)
  Output: 256-bit (32-byte) hash
  Structure: Merkle–Damgård construction

Applications:

- Bitcoin Blockchain
- TLS Certificates
- Digital Signatures

Algorithm #1 – SHA-256

Pros:

- Highly secure
- Collision-resistant

Cons:

- Slow for large datasets

C++ Pseudocode for SHA-256

```
// Use OpenSSL or Crypto++ libraries for real use.
// Simplified view:
string sha256(string input) {
    // Padding + Message Schedule + Compression Function
    // Complex, not practical to write manually
    return SHA256_HASH; // placeholder
}
```

Algorithm #2 – MurmurHash

- **Type:** Non-cryptographic
  Author: Austin Appleby
  Output: 32, 64, 128-bit variants
  Speed: Extremely fast

Use Cases:

- Hash tables
- In-memory databases (Cassandra, Redis)
- Compilers

Algorithm #2 – MurmurHash

Pros:

- Lightweight
- Good distribution
- Hardware friendly

Cons:

- Not secure (vulnerable to hash flooding)

MurmurHash C++ Sample (v3, 32-bit)

```
uint32_t murmurhash3_32(const char* key, size_t len, uint32_t seed) {
    uint32_t h = seed;
    const uint32_t c1 = 0xcc9e2d51;
    const uint32_t c2 = 0x1b873593;

    const int nblocks = len / 4;
    const uint32_t* blocks = (const uint32_t*)(key);
    for (int i = 0; i < nblocks; i++) {
        uint32_t k = blocks[i];
        k *= c1; k = (k << 15) | (k >> 17); k *= c2;
        h ^= k;
        h = (h << 13) | (h >> 19); h = h * 5 + 0xe6546b64;
    }

    // Tail and finalization skipped for brevity
    return h;
}
```

Algorithm #3 – BLAKE3

- **Type:** Cryptographic
  Output: Variable-length (default 256-bit)
  Based On: BLAKE2 + Merkle tree + SIMD + ARX

Strengths:

- Faster than SHA-256
- Safe for cryptographic use
- Parallelizable (multi-threaded hashing)

Use Cases:

- File integrity
- High-speed checksums
- Cryptography

BLAKE3 Pseudocode (Simplified View)

```c
// Use official BLAKE3 C library
blake3_hasher hasher;
blake3_hasher_init(&hasher);
blake3_hasher_update(&hasher, input, input_len);
blake3_hasher_finalize(&hasher, output, output_len);
```

Proposed Custom Hash – MODHASH

- **Goal:**
  Simple, fast, efficient for data indexing

Design:

- Modular polynomial hashing using large prime
- Positional entropy via powers of a base

Features:

- Deterministic
- Fast (non-cryptographic)
- Uniform for real-world text/data
- Suitable for embedded systems

C++ Code – MODHASH

- ```
  #include <iostream>
  #include <string>
  using namespace std;
  const uint64_t PRIME = 4294967311;
  const uint64_t BASE = 31;
  uint64_t modhash(const string& data) {
      uint64_t hash = 0;
      for (size_t i = 0; i < data.size(); ++i) {
          hash = (hash + (data[i] * pow(BASE, i))) % PRIME;
      }
      return hash;
  }
  ```
  ```

Comparison Table

Algorithm	Type	Speed	Output	Secure	Use Cases
SHA-256	Cryptographic	Slow	256bit	✅	Crypto, Blockchain
MurmurHash	Fast Hash	Very Fast	32-128	❌	DBs, Caches
BLAKE3	Cryptographic	Very Fast	Var.	✅	Fast Crypto Hashing
MODHASH	Custom/Fast	Very Fast	64bit	❌	Hash Tables, Indexes

Conclusion

- SHA-256 is secure but slower.
- MurmurHash is lightweight, non-secure, ideal for performance.
- BLAKE3 combines speed & security.
- **MODHASH** balances simplicity and performance — ideal for systems where speed

References

- https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
- https://github.com/aappleby/smhasher
- https://github.com/BLAKE3-team/BLAKE3
- https://en.wikipedia.org/wiki/Rabin%E2%80%93Karp_algorithm