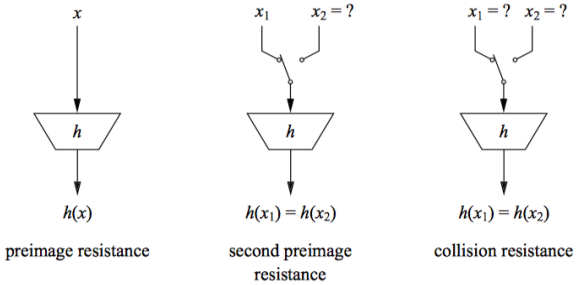
SM3 Implementation And Application

Long Wen
longwen6@gmail.com

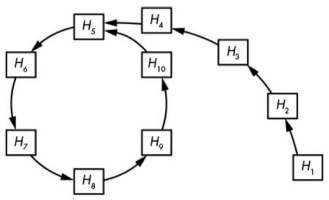
Hash Function Basics

- Security of hash functions
  - Preimage resistance / first-preimage resistance
  - Second-preimage resistance
  - Collision resistance
  - Q&A: which one is weaker?
  - Q&A: MD5 and SHA1 is broken, meaning?
  - Q&A: collision could be found, so what ?
  - Q&A: how do you argue no one knows a secret key



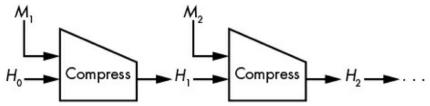
Address = RIPEMD160(SHA256(pubkey)) = 00000000000000000000000000000000071E76C

- Finding Collisions
  - The naïve birthday attack
  - The Rho method: low-memory collision search
  - Project: implement the naïve birthday attack of reduced SM3
  - Project: implement the Rho method of reduced SM3
  - *The longer collision you can find, the higher score

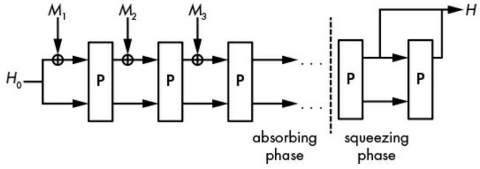


Hash Function Construction

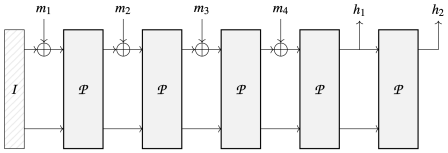
- Compression-Based: Merkel-Damgard construction
  - All hash functions from 1980s to 2010s are MD construction
  - MD4, MD5, SHA-1, SHA-2 (**SHA256**), **SM3**, **RIPEMD**, and Whirlpool



- Permutation-Based: Sponge Functions,
  - SHA3-family, BLAKE-family
  - Absorbing phasing
  - Squeezing phasing

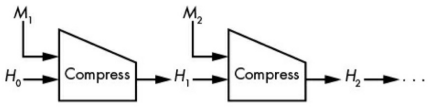


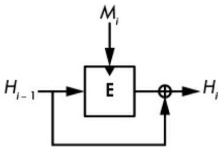
- New guy in town: zkSNARK/MPC friendly hash on GF(p):
  - Poseidon, rescue
  - Will cover this part later in zkSNARK reports



Merkel-Damgard Construction

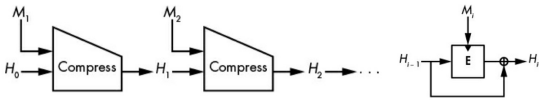
- MD construction work flow
  - Splits message into blocks of fixed size, usually 512 or 1024 bits
  - Mix blocks with internal state using Compression function
  - Final internal state is the message's hash value
- Dealing with any message length – padding blocks
  - For 512-bit block, forms the last block by
  - append bit 1, then bits 0, and finally the length of the original message
  - Guarantee: any two distinct messages will give a distinct sequence of blocks
- Building compress functions: the Davies-Meyer construction
  - Build compress function with block cipher E
  - Given message block M_i, and previous chain value H_{i-1}
  - $H_i = E(M_i, H_{i-1}) \oplus H_{i-1}$
  - M_i acts like block cipher key, H_{i-1} acts as plaintext block
  - Block cipher is secure → compress function is secure
  - Weak block cipher lead to bad hash function: Xbox game
  - Finding fixed point of DM construction
  - Q&A: is the existence of fixed point a problem to hash function





$$H_i = \mathbf{E}(M_i, H_{i-1}) \oplus H_{i-1} = \mathbf{E}(M_i, D(M_i, 0)) \oplus \mathbf{D}(M_i, 0)$$
$$= 0 \oplus \mathbf{D}(M_i, 0) = \mathbf{D}(M_i, 0) = H_{i-1}$$

SM3 - Merkel-Damgard Construction



1.1.5 SM3 密码杂凑算法的消息填充

对长度为 $l(l<2^{64})$ 比特的消息 m，SM3 密码杂凑算法首先将比特“1”添加到消息的末尾，再添加 k 个“0”，k 是满足 $l+k+1\equiv 448 \bmod 512$ 的最小非负整数. 然后再添加一个 64 位比特串，该比特串是长度 l 的二进制表示. 填充后的消息 m' 的比特长度为 512 的倍数.

例如：对消息 01100001 01100010 01100011，其长度 $l=24$，经填充得到的比特串如下：

$$\underbrace{01100001\ 01100010\ 01100011}_{24\text{ b}}\ \underbrace{1\ 0\cdots 0}_{423\text{ b}}\ \underbrace{0\cdots 011000}_{64\text{ b}}$$

（l 的二进制表示）

1.1.6 SM3 密码杂凑算法的迭代压缩过程

将填充后的消息 m' 按 512 b 进行分组：$m'=B^{(0)}B^{(1)}\cdots B^{(n-1)}$，其中 $n=(l+k+65)/512$. 对 m' 按如下方式迭代：

FOR $i=0$ TO $(n-1)$

$V^{(i+1)}=CF(V^{(i)},B^{(i)})$；

ENDFOR

其中，CF 是压缩函数，$V^{(0)}$ 为 256 b 初始值 IV，$B^{(i)}$ 为填充后的消息分组，迭代压缩的结果为 $V^{(n)}$.

Compress Function

Q&A: does this look like block cipher ?

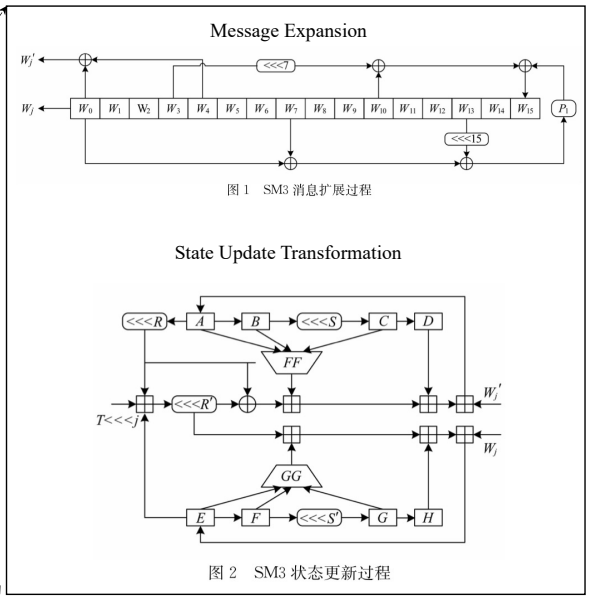
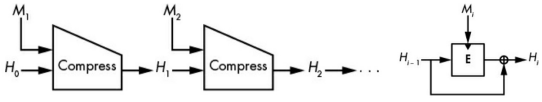


图 1　SM3 消息扩展过程

图 2　SM3 状态更新过程

SM3 - Merkel-Damgard Construction



1.1.5 SM3 密码杂凑算法的消息填充

对长度为 $l(l<2^{64})$ 比特的消息 m，SM3 密码杂凑算法首先将比特“1”添加到消息的末尾，再添加 k 个“0”，k 是满足 $l+k+1\equiv 448 \bmod 512$ 的最小非负整数. 然后再添加一个 64 位比特串，该比特串是长度 l 的二进制表示. 填充后的消息 m' 的比特长度为 512 的倍数.

例如：对消息 01100001 01100010 01100011，其长度 $l=24$，经填充得到的比特串如下：

$$\underbrace{01100001\ 01100010\ 01100011}_{24\text{ b}}\ \underbrace{1\ 0\cdots 0}_{423\text{ b}}\ \underbrace{0\cdots 011000}_{64\text{ b}}$$

（l 的二进制表示）

1.1.6 SM3 密码杂凑算法的迭代压缩过程

将填充后的消息 m' 按 512 b 进行分组：$m'=B^{(0)}B^{(1)}\cdots B^{(n-1)}$，其中 $n=(l+k+65)/512$. 对 m' 按如下方式迭代：

FOR $i=0$ TO $(n-1)$

$V^{(i+1)}=CF(V^{(i)},B^{(i)})$；

ENDFOR

其中，CF 是压缩函数，$V^{(0)}$ 为 256 b 初始值 IV，$B^{(i)}$ 为填充后的消息分组，迭代压缩的结果为 $V^{(n)}$.

Compress Function

Q&A: How about now ?

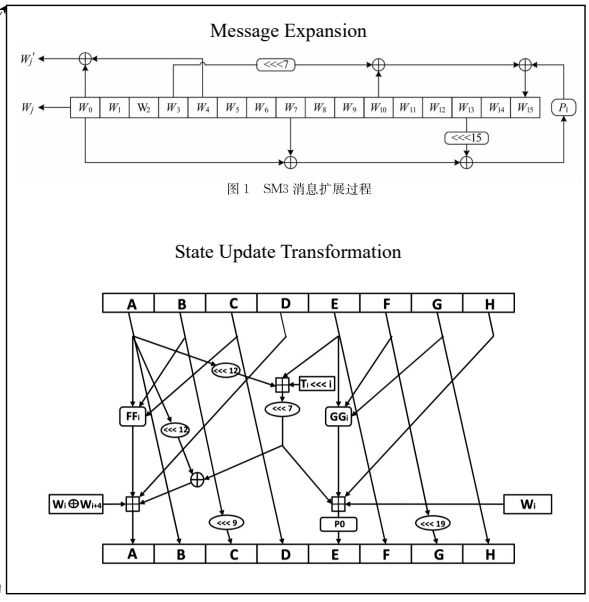
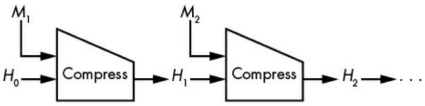


图 1　SM3 消息扩展过程

Length Extension Attack of The Merkel-Damgard Construction

- Length extension attack is the main thread to MD construction
- Attack outline
  - If you know Hash(M) for unknown message where
  - M = M1|| M2 (after padding)
  - You can determine Hash(M1 || M2 || M3) for any block, M3
- Can be generalized to any number of blocks
  - either in the unknown message part or the suffix part
- Affects & mitigation
  - Won't affect most application scenarios
  - Should bear this attack when design system
  - Q&A: does SM3 have the same security issue ?
  - Q&A: how to mitigate? - Just make the last compression function different from all others
  - New hash function design take this into consideration, refer to BLAKE2
  - Q&A: any other hash function example you can find?
  - *Project: implement length extension attack for SM3, SHA256, etc.



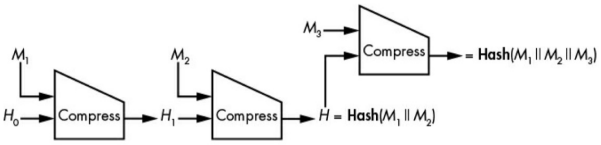
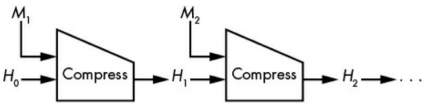


Figure 6-9: The length-extension attack

Length Extension Attack of The Merkel-Damgard Construction

- Length extension attack is the main thread to MD construction
- Attack outline
  - If you know Hash(M) for unknown message where
  - M = M1|| M2 (after padding)
  - You can determine Hash(M1 || M2 || M3) for any block, M3
- Can be generalized to any number of blocks
  - either in the unknown message part or the suffix part
- Affects & mitigation
  - Won't affect most application scenarios
  - Should bear this attack when design system
  - Q&A: does SM3 have the same security issue ?
  - Q&A: how to mitigate? - Just make the last compression function different from all others
  - New hash function design take this into consideration, refer to BLAKE2
  - Q&A: any other hash function example you can find?
  - *Project: implement length extension attack for SM3, SHA256, etc.



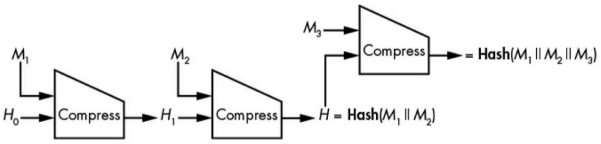


Figure 6-9: The length-extension attack

Hash Function in Action: Proof-of-Storage Protocol

- Proof-of-Storage: server (cloud provider) proves to a client that the server does in fact store a file
- design https://www.cs.utexas.edu/~lorenzo/papers/p129-kotla.pdf from The University of Texas at Austin
- Challenge and response protocol
  - Client picks a random value C as challenge
  - Server computes Hash(M||C) as a response and sends the result to the client
  - Client also computes Hash(M||C) and checks that both hash values matches
- Q&A: Is the procotol above secure? Why? If not, how to cheat ?
- Q&A: which kind of hash functions are affected?
- Q&A: how to fix ?
- More: will see more misuse of hash functions in Bitcoin later
- The security of hash function itself cannot protect you from misuse

SM3 Collision In WeChat

- Core issue: encode algorithm from string to binary allows one to construct same binary stream from different string
- Over 260 even more open-sourced projects has been affected
- Key: reinterprete the same binary as two printable characters

```
function str2binary(str) {
  let binary = ''
  for (let i = 0, len = str.length; i < len; i++) {
    const ch = str[i]
    binary += leftPad(ch.codePointAt(0).toString( radix: 2),  num: 8)
  }
  return binary
}
function leftPad(input, num) {
  // here! if input longer than num, return directly
  if (input.length >= num) return input

  return (new Array( arrayLength: num - input.length + 1)).join('0') + input
}
```

```
Welcome to Node.js v12.16.2.
Type ".help" for more information.
> const sm3 = require('miniprogram-sm-crypto').sm3
undefined
> s1 = "丧丙上蹿蠢不专汞"
'丧丙上蹿蠢不专汞'
> s2 = "鲆8fpT肩脚荧HNQ"
'鲆8fpT肩脚荧HNQ'
> sm3(s1)
'457a19a2e7c2f43c0bfe7ca61cf5f5071c0fd4da99f13856b1ca4997e70cfc69'
> sm3(s2)
'457a19a2e7c2f43c0bfe7ca61cf5f5071c0fd4da99f13856b1ca4997e70cfc69'
```

15:37 4G

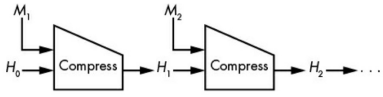
微信小程序国密算法SM3碰撞原理揭秘

原创 蜚语安全 蜚语安全 2021-03-10 14:24

问题概述

2021年2月，蜚语安全在对客户产品进行安全代码审计时，发现其依赖的**微信小程序sm-crypto 开源NodeJS国密算法库中的SM3算法实现**（该算法库使用NodeJS对国密算法中的SM2，SM3和SM4三个算法进行了实现，并封装成接口提供给开发者使用。该国密算法库的具体实现在GitHub上开源：https://github.com/wechat-miniprogram/sm-crypto，其中sm3的实现在文件src/sm3/index.js中，微信小程序官方的参考文档链接为 https://developers.weixin.qq.com/miniprogram/dev/extended/utils/sm-crypto.html ）存在重大的实现错误，**导致不同的输入可被编码为相同的bitstring，进而进行SM3计算后产生“伪”hash碰撞效果。**

SM3 Implementation – Interface Design




```c
int sm3_hash_openssl(uint8_t *dgst, const void *msg, size_t len) {
    int res = 0;
    const EVP_MD *md = EVP_get_digestbyname("sm3");
    EVP_MD_CTX *mdctx = EVP_MD_CTX_new();
    if (!mdctx) goto done;

    EVP_DigestInit_ex(mdctx, md, NULL);
    EVP_DigestUpdate(mdctx, msg, len);
    res = EVP_DigestFinal_ex(mdctx, dgst, NULL);

done:
    EVP_MD_CTX_free(mdctx);
    return res;
}

int sm3_hash_verify_openssl(const void *msg, size_t len, const void *dgst) {
    uint8_t buf[32];
    sm3_hash_openssl(buf, msg, len);
    return memcmp(buf, dgst, 32);
}
```

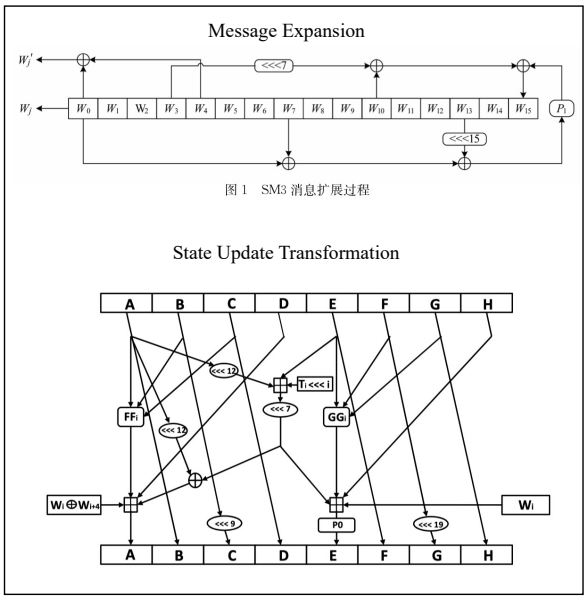
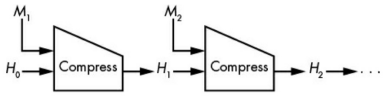


图 1　SM3 消息扩展过程

SM3 Implementation – Interface




```c
#define sm3_digest_BYTES 32
#define sm3_block_BYTES 64
#define sm3_hmac_BYTES sm3_digest_BYTES

typedef struct sm3_ctx_t {
    uint32_t digest[sm3_digest_BYTES / sizeof(uint32_t)];
    int nblocks;  // number of blocks that have been processed
    uint8_t block[sm3_block_BYTES * 4];
    int num;
} sm3_ctx;

void sm3_init(sm3_ctx *ctx);
void sm3_update(sm3_ctx *ctx, const uint8_t *data, size_t data_len);
void sm3_final(sm3_ctx *ctx, uint8_t *digest);

void sm3_hash(uint8_t *digest, const uint8_t *data, size_t dlen);
int sm3_hash_verify(const uint8_t *data, size_t dlen, const uint8_t *digest);
```

```c
void sm3_init(sm3_ctx *ctx) {
    ctx->digest[0] = 0x7380166F;
    ctx->digest[1] = 0x4914B2B9;
    ctx->digest[2] = 0x172442D7;
    ctx->digest[3] = 0xDA8A0600;
    ctx->digest[4] = 0xA96F30BC;
    ctx->digest[5] = 0x163138AA;
    ctx->digest[6] = 0xE38DEE4D;
    ctx->digest[7] = 0xB0FB0E4E;

    ctx->nblocks = 0;
    ctx->num = 0;
}
```

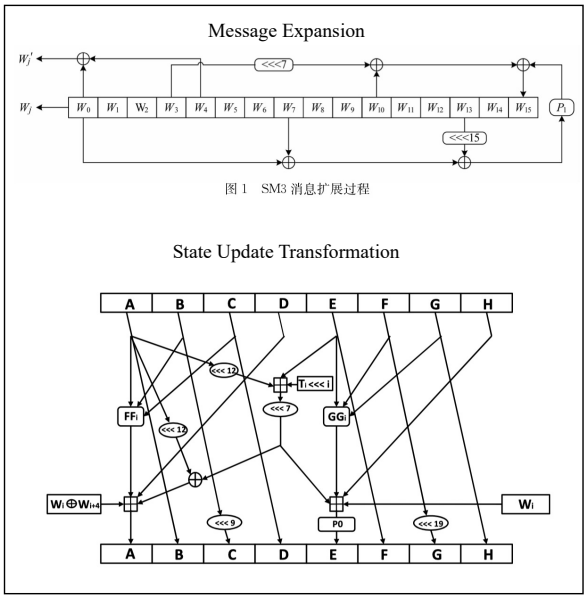
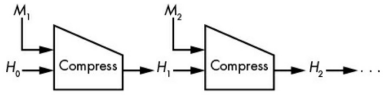


图 1　SM3 消息扩展过程

SM3 Implementation – Update




```c
void sm3_update(sm3_ctx *ctx, const uint8_t *data, size_t dlen) {
    if (ctx->num) {
        unsigned int left = sm3_block_BYTES - ctx->num;
        if (dlen < left) {
            memcpy(ctx->block + ctx->num, data, dlen);
            ctx->num += dlen;
            return;
        } else {
            memcpy(ctx->block + ctx->num, data, left);
            sm3_compress(ctx->digest, ctx->block);
            ctx->nblocks++;
            data += left;
            dlen -= left;
        }
    }
    while (dlen >= sm3_block_BYTES) {
        sm3_compress(ctx->digest, data);
        ctx->nblocks++;
        data += sm3_block_BYTES;
        dlen -= sm3_block_BYTES;
    }
    ctx->num = dlen;
    if (dlen) {
        memcpy(ctx->block, data, dlen);
    }
}
```

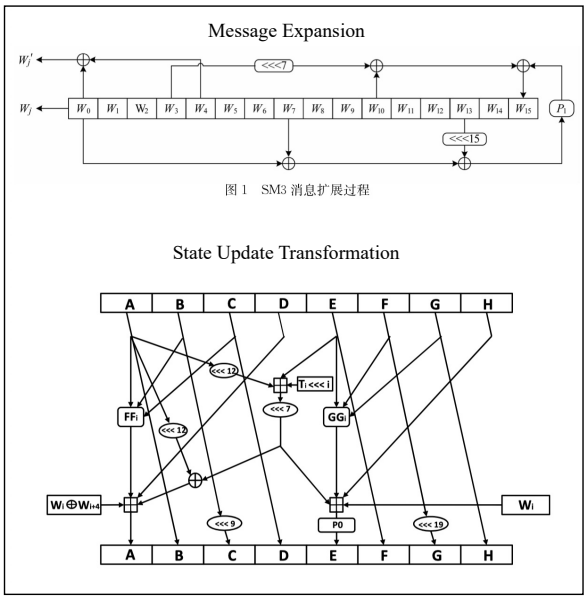
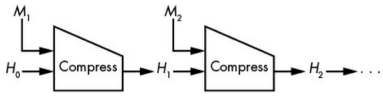


图 1　SM3 消息扩展过程

SM3 Implementation – Final




```c
void sm3_final(sm3_ctx *ctx, uint8_t *digest) {
    size_t i;
    uint32_t *pdigest = (uint32_t *)(digest);
    uint64_t *count = (uint64_t *)(ctx->block + sm3_block_BYTES - 8);

    ctx->block[ctx->num] = 0x80;

    if (ctx->num + 9 <= sm3_block_BYTES) {
        memset(ctx->block + ctx->num + 1, 0, sm3_block_BYTES - ctx->num - 9);
    } else {
        memset(ctx->block + ctx->num + 1, 0, sm3_block_BYTES - ctx->num - 1);
        sm3_compress(ctx->digest, ctx->block);
        memset(ctx->block, 0, sm3_block_BYTES - 8);
    }

    count[0] = (uint64_t)(ctx->nblocks) * 512 + (ctx->num << 3);
    count[0] = byte_swap64(count[0]);

    sm3_compress(ctx->digest, ctx->block);
    for (i = 0; i < sizeof(ctx->digest) / sizeof(ctx->digest[0]); i++) {
        pdigest[i] = byte_swap32(ctx->digest[i]);
    }
}
```

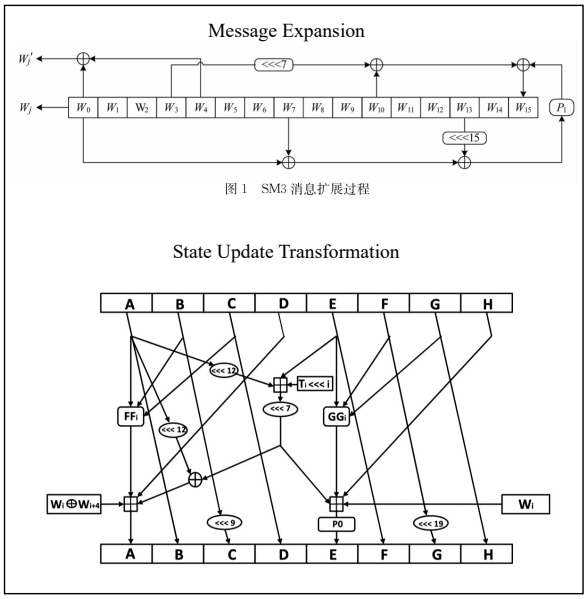
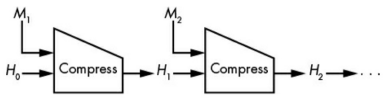


图 1　SM3 消息扩展过程

```c
static void sm3_compress(uint32_t digest[sm3_digest_BYTES / sizeof(uint32_t)],
                         const uint8_t block[sm3_block_BYTES]) {
    int j;
    uint32_t W[68], W1[64];
    const uint32_t *pblock = (const uint32_t *)(block);

    uint32_t A = digest[0], B = digest[1], C = digest[2], D = digest[3];
    uint32_t E = digest[4], F = digest[5], G = digest[6], H = digest[7];

    uint32_t SS1, SS2, TT1, TT2, T[64];

    for (j = 0; j < 16; j++) W[j] = byte_swap32(pblock[j]);

    for (j = 16; j < 68; j++)
        W[j] = P1(W[j - 16] ^ W[j - 9] ^ rol(W[j - 3], 15)) ^
               rol(W[j - 13], 7) ^ W[j - 6];

    for (j = 0; j < 64; j++) W1[j] = W[j] ^ W[j + 4];

    for (j = 0; j < 16; j++) {
        T[j] = 0x79CC4519;
        SS1 = rol((rol(A, 12) + E + rol(T[j], j)), 7);
        SS2 = SS1 ^ rol(A, 12);
        TT1 = FF0(A, B, C) + D + SS2 + W1[j];
        TT2 = GG0(E, F, G) + H + SS1 + W[j];
        D = C, C = rol(B, 9), B = A, A = TT1;
        H = G, G = rol(F, 19), F = E, E = P0(TT2);
    }

    for (j = 16; j < 64; j++) {
        T[j] = 0x7A879D8A;
        SS1 = rol((rol(A, 12) + E + rol(T[j], j)), 7);
        SS2 = SS1 ^ rol(A, 12);
        TT1 = FF1(A, B, C) + D + SS2 + W1[j];
        TT2 = GG1(E, F, G) + H + SS1 + W[j];
        D = C, C = rol(B, 9), B = A, A = TT1;
        H = G, G = rol(F, 19), F = E, E = P0(TT2);
    }

    digest[0] ^= A, digest[1] ^= B, digest[2] ^= C, digest[3] ^= D;
    digest[4] ^= E, digest[5] ^= F, digest[6] ^= G, digest[7] ^= H;
}
```



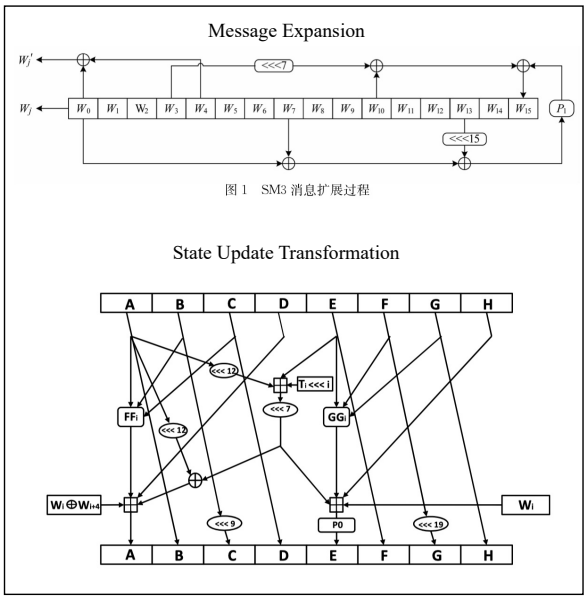
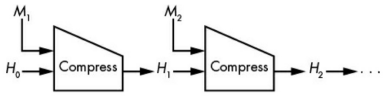


图 1　SM3 消息扩展过程

SM3 Implementation – Optimize



- The simple way - just refer to OpenSSL
  - Loop unwinding & employ the macro definition of C
- The hard way - Summon the SIMD instructions
  - Perform multiple arithmetic operation with one instruction
- 3 approaches to optimize with SIMD instructions
  - Hash data from multiple independent streams at the same time
  - Schedule multiple blocks (for the same data stream) in parallel
  - Schedule the data associated with a single block in parallel
- Reference
  - Fast SHA-256 Implementations on Intel® Architecture Processors
  - S. Gueron, V. Krasnov. Parallelizing message schedules to accelerate the computations of hash functions. http://eprint.iacr.org/2012/067.pdf
  - Processing Multiple Buffers in Parallel to Increase Performance on Intel® Architecture Processors
- *Project: do your best to optimize SM3 implementation (software)
- Make sure you understand each line written, (don't copy)
- Earn higher score with faster implementation

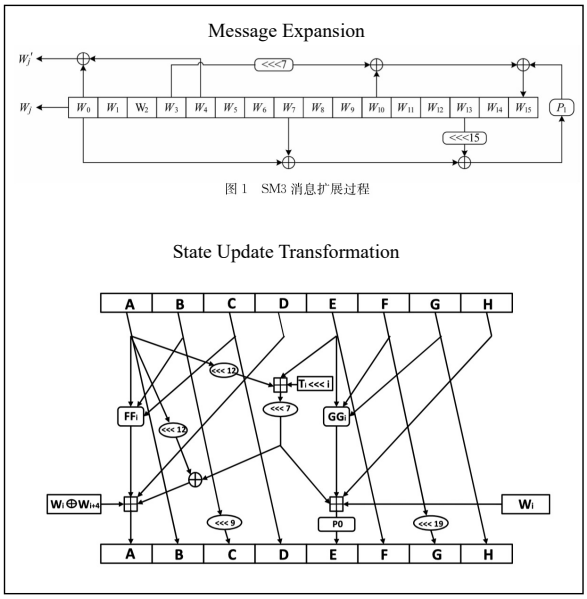
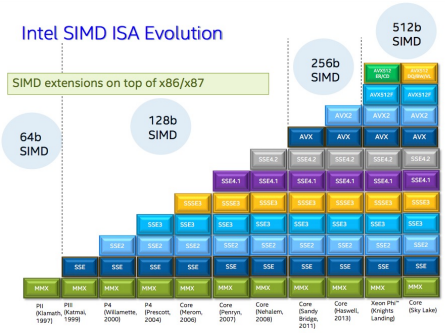


图 1　SM3 消息扩展过程

Brief Introduction to SIMD Instructions

	instruction	C Language
xor	pxor xmm, xmm	__m128i _mm_xor_si128 (__m128i A, __m128i B)
rotate scalars left	vprold xmm, xmm, imm8	__m128i _mm_rol_epi32 (__m128i A, int n)
shift scalars left	pslld xmm, imm8	__m128i _mm_slli_epi32 (__m128i A, int n)
shift scalars right	psrld xmm, imm8	__m128i _mm_srli_epi32 (__m128i A, int n)

	instruction	C Language
shuffle bytes in XMM	pshufb xmm, xmm	__m128i _mm_shuffle_epi8 (__m128i A, __m128i mask)
shift whole XMM left by n bytes	pslldq xmm, imm8	__m128i _mm_slli_si128 (__m128i a, int n)
shift whole XMM right by n bytes	psrldq xmm, imm8	__m128i _mm_srli_si128 (__m128i a, int n)
shift A||B right by n bytes	palignr xmm, xmm, imm8	__m128i _mm_alignr_epi8 (__m128i a, __m128i b, int n)



A Taste of SIMD, SM3's P1 Function

SM3 密码杂凑算法的置换函数定义如下：

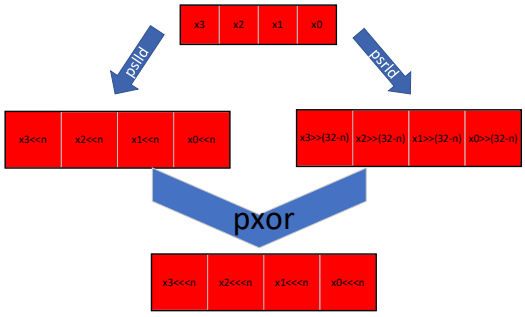
$$P_0(X) = X \oplus (X <<< 9) \oplus (X <<< 17),$$
$$P_1(X) = X \oplus (X <<< 15) \oplus (X <<< 23).$$

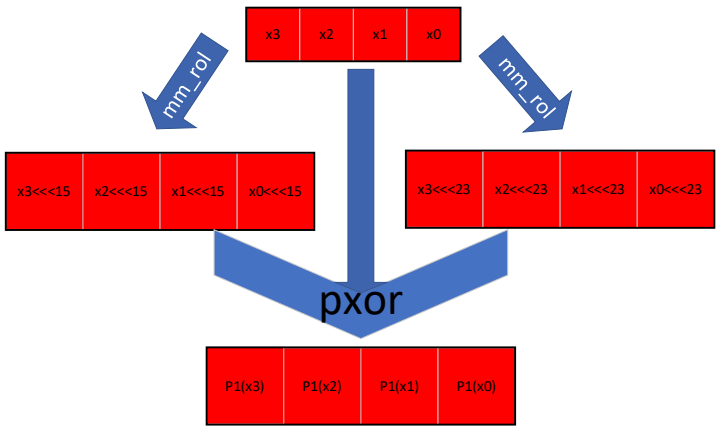
P1(x) = x ^ (x<<<15) ^ (x<<<23)

mm_P1(X) = X ^ mm_rol(X,15) ^ mm_rol(X,23)

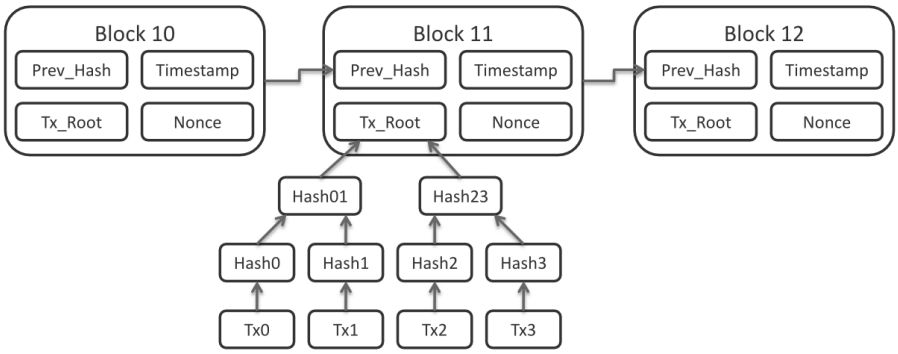
rol(x,n) = (x<<n) ^ (x>>(32-n))

mm_rol(X,n) = pslld(X,n) ^ psrld(X,32-n)

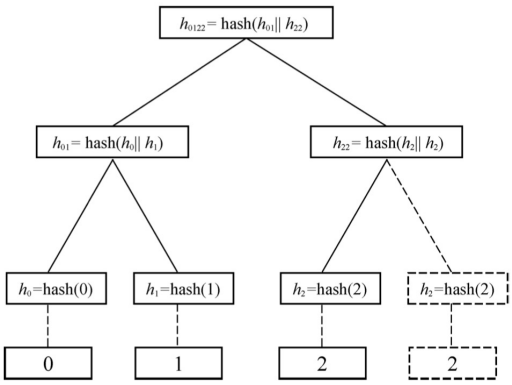




Blockchain – Chained By Hash Link

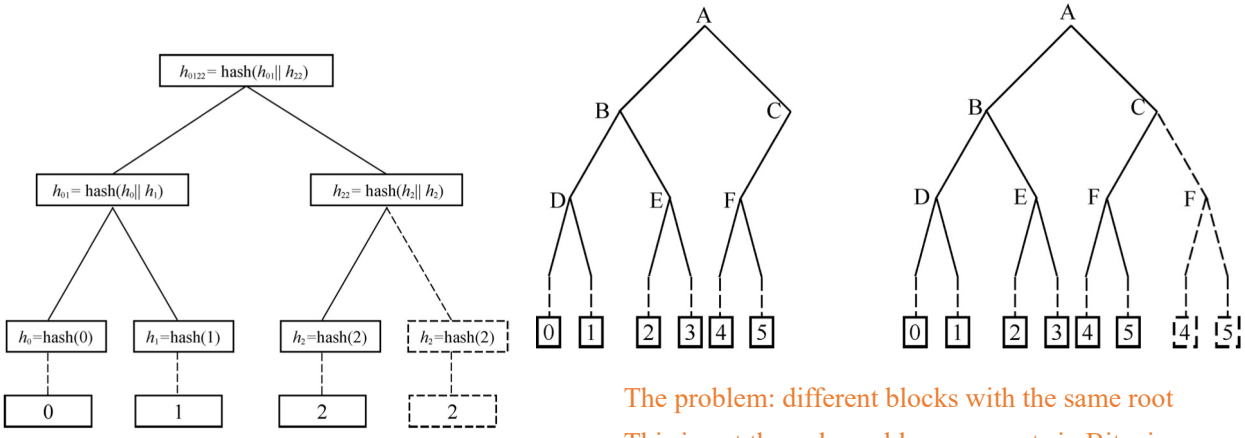


Merkle Tree In Bitcoin



Q&A: Is the construction safe?

The Problem with Bitcoin's Merkle Tree



The problem: different blocks with the same root

This is not the only problems presents in Bitcoin

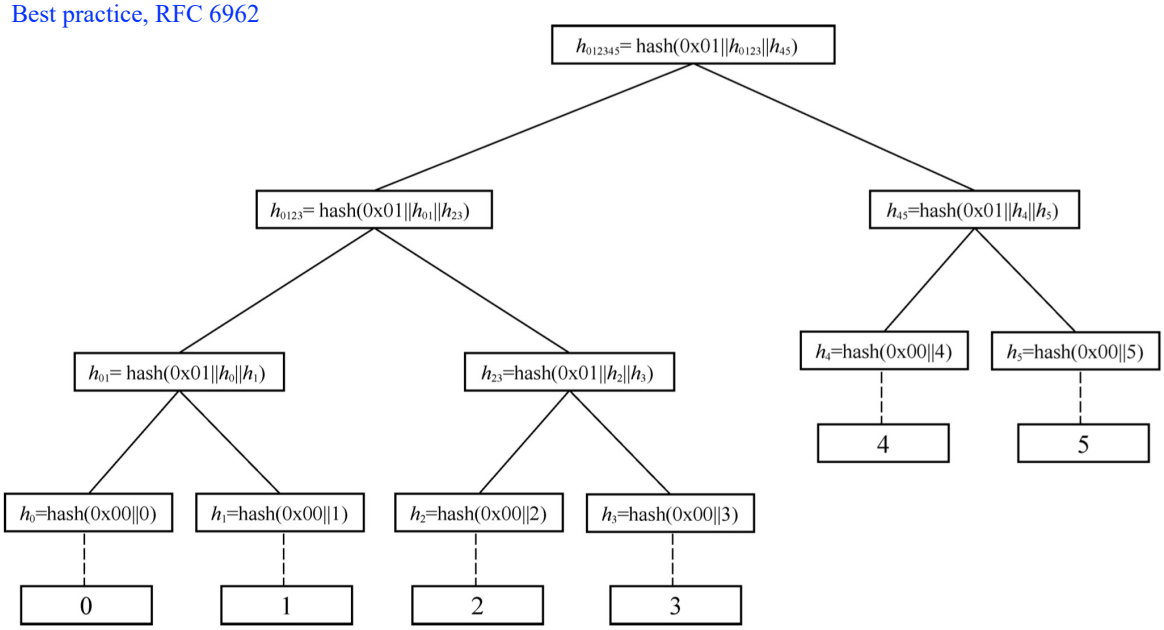
Bitcoin's Fix

```
uint256 ComputeMerkleRoot(std::vector<uint256> hashes, bool* mutated) {
    bool mutation = false;
    while (hashes.size() > 1) {
        if (mutated) {
            for (size_t pos = 0; pos + 1 < hashes.size(); pos += 2) {
                if (hashes[pos] == hashes[pos + 1]) mutation = true;
            }
        }
        if (hashes.size() & 1) {
            hashes.push_back(hashes.back());
        }
        SHA256D64(hashes[0].begin(), hashes[0].begin(), hashes.size() / 2);
        hashes.resize(hashes.size() / 2);
    }
    if (mutated) *mutated = mutation;
    if (hashes.size() == 0) return uint256();
    return hashes[0];
}
```

```
uint256 BlockMerkleRoot(const CBlock& block, bool* mutated)
{
    std::vector<uint256> leaves;
    leaves.resize(block.vtx.size());
    for (size_t s = 0; s < block.vtx.size(); s++) {
        leaves[s] = block.vtx[s]->GetHash();
    }
    return ComputeMerkleRoot(std::move(leaves), mutated);
}
```

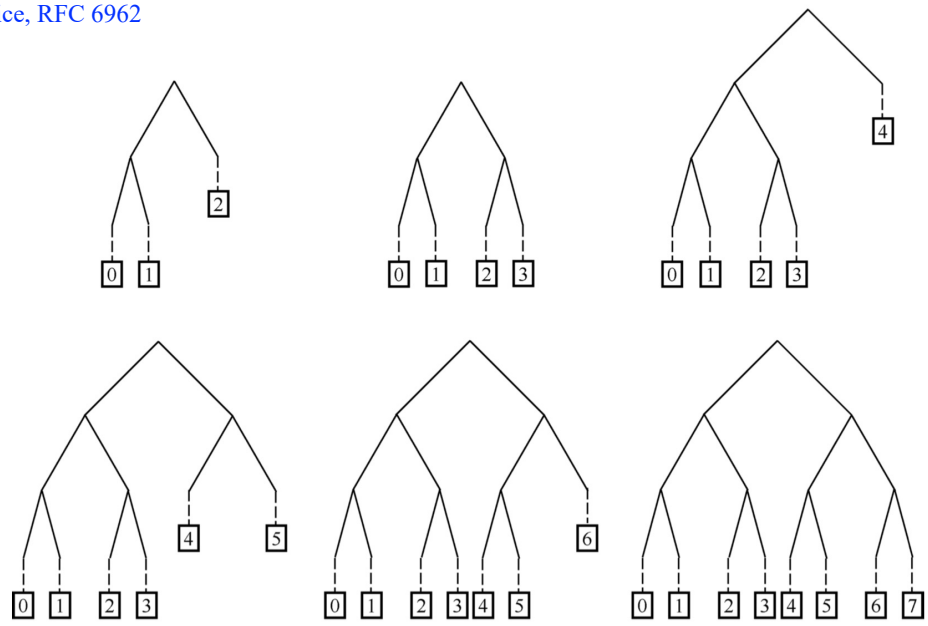
Engineering Merkle Tree

Best practice, RFC 6962



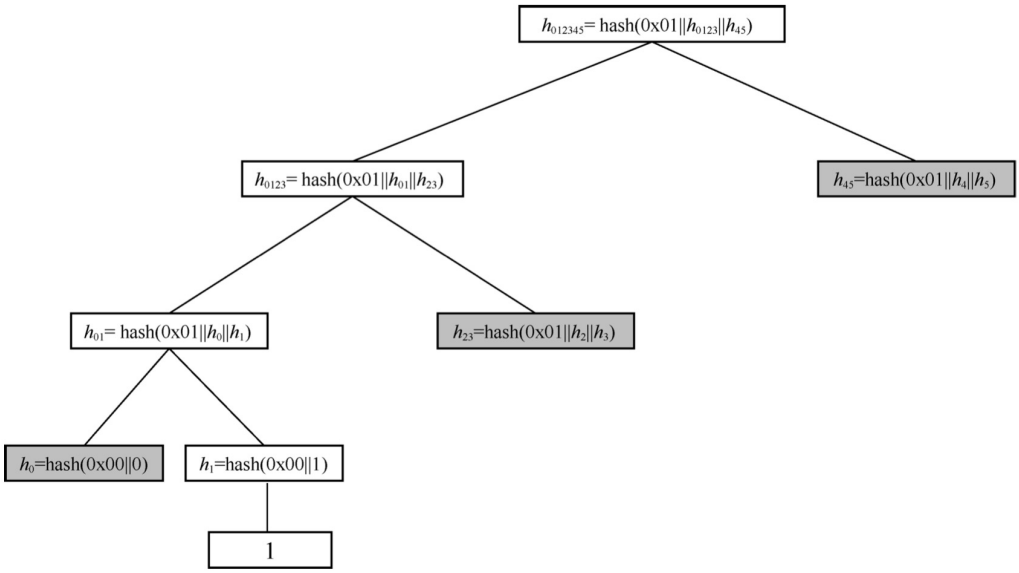
Engineering Merkle Tree

Best practice, RFC 6962



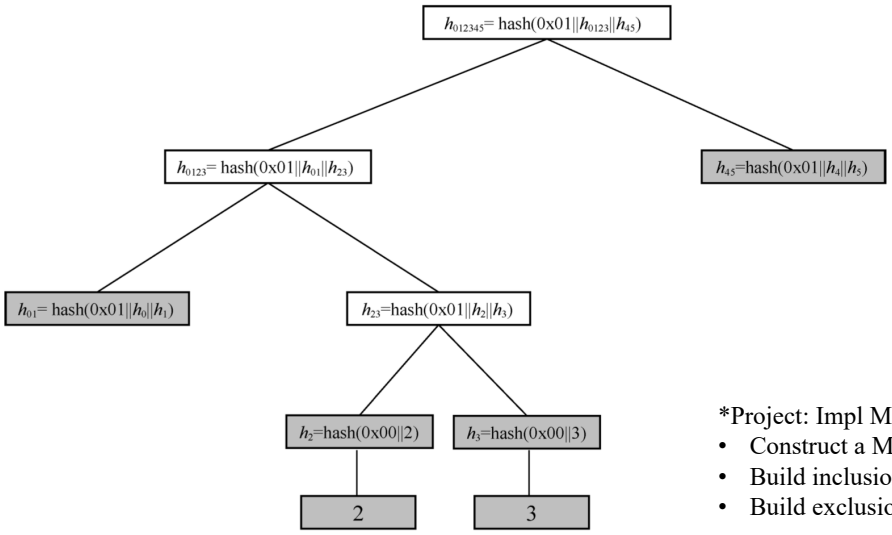
Merkle Tree – Inclusion Proof

Following RFC 6962



Merkle Tree – Exclusion Proof

Following RFC 6962, mind the extra requirement

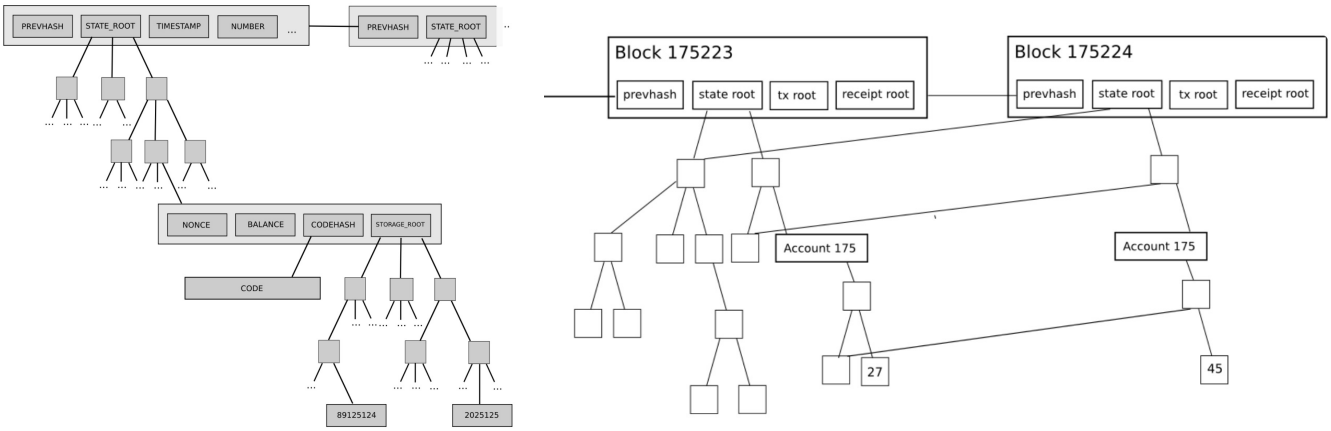


*Project: Impl Merkle Tree following RFC6962

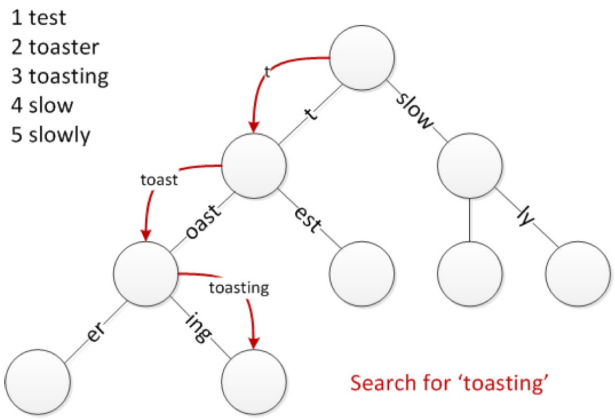
- Construct a Merkle tree with 10w leaf nodes
- Build inclusion proof for specified element
- Build exclusion proof for specified element

Global State in Ethereum - Updatable Merkle Trees

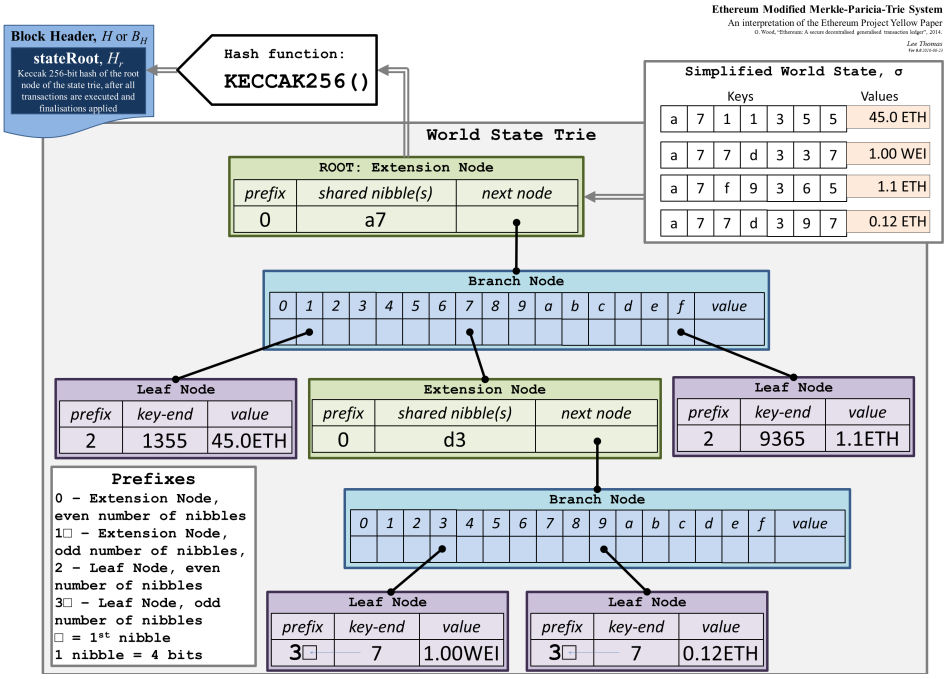
Q&A: how hard it is to add/remove/modify element in Merkle tree of RFC6962?



Global State in Ethereum - Merkle Patricia Tree

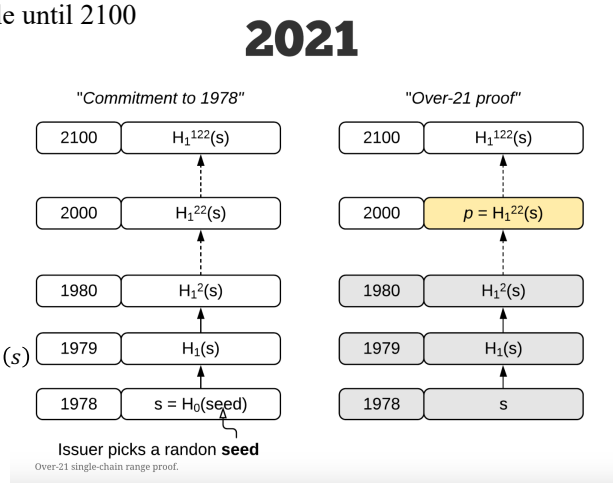


Global State in Ethereum - Merkle Patricia Tree



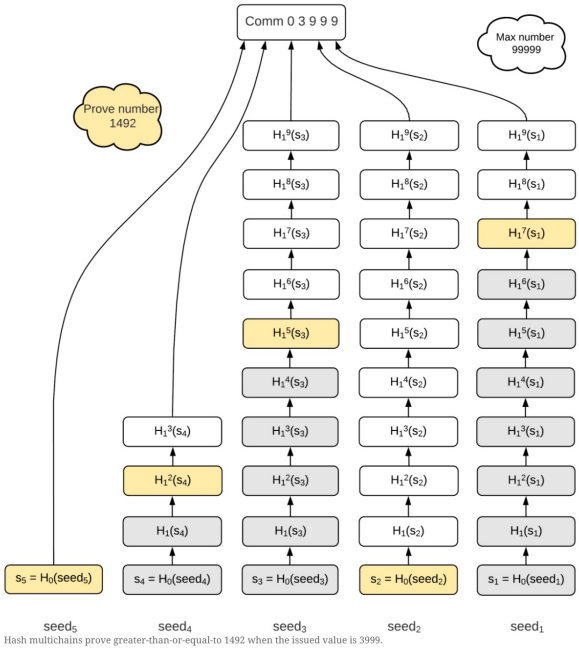
Range Proof With Hash Function

- Alice (born 1978) wants to prove to Bob that her age > 21
- Rely on trusted issuer, proof system need to be useable until 2100
- Suppose this year is 2021



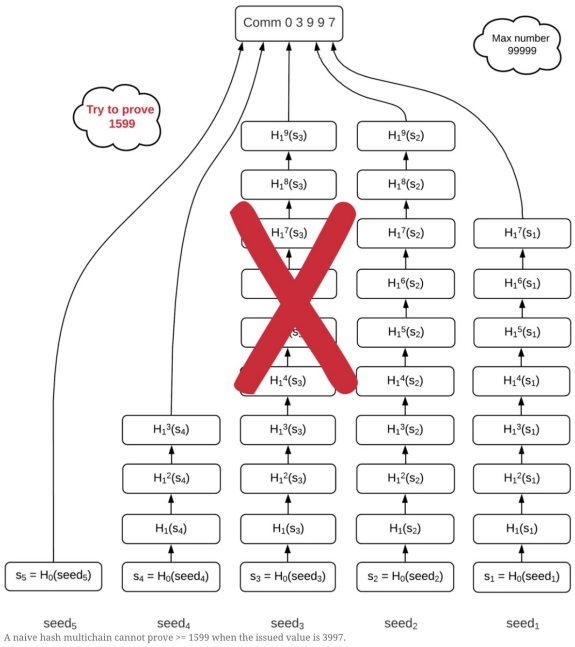
- Trusted Issuer
  - Pick 128-bit random seed , compute $s = \mathrm{H}_0(seed)$
  - $\mathrm{k} = 2100 - 1978, \mathrm{c} = \mathrm{H}_1^{\mathrm{k}}(s)$, sign over c as sig_c
  - Give Alice s and sig_c
- Alice prove her age >= 21 to Bob
  - E.g., she was born before 2000
  - Compute $d_0 = 2000 - 1978 = 22$, compute proof $p = H_1^{d_0}(s)$
  - Give Bob (p, sig_c)
- Bob verify Alice's proof
  - Compute $\mathrm{d}_1 = 2100 - 2000 = 100$
  - Compute $c' = H_1^{\mathrm{d}}(\mathrm{p})$, check sig_c is for c'

Generalizing Hash Chains



Hash multichains prove greater-than-or-equal-to 1492 when the issued value is 3999.

Cannot prove 1598 or 1599 if Comm. is for 03997



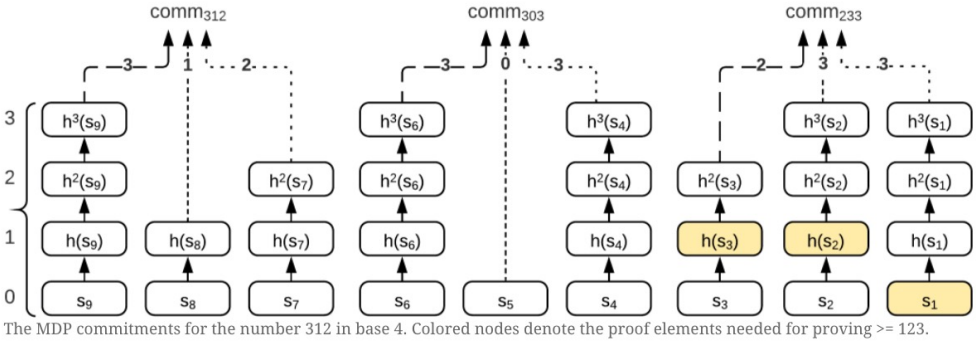
A naive hash multichain cannot prove >= 1599 when the issued value is 3997.

Generalizing Hash Chains: Fix Previous Problem

[312, 303, 233] is the minimum dominating partitions for number 312 in base4
With Comm312, Comm303, Comm233, Alice can prove number <312

- Use Comm233 if prove number [0, 233]
- Use Comm303 if prove number [300, 303]
- Use Comm312 if prove number [310, 312]

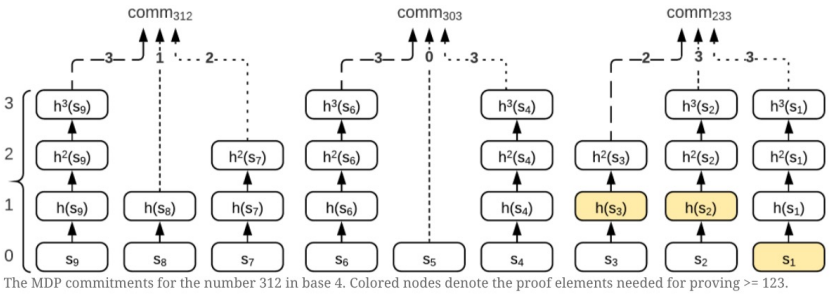
Call [312, 303, 233] MDP-list for base4 number 312



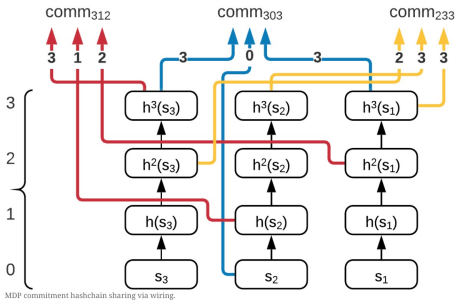
The MDP commitments for the number 312 in base 4. Colored nodes denote the proof elements needed for proving >= 123.

[3997, 3989, 3899, 2999] is the MDP-list for base10 number 3997

Generalizing Hash Chains: Reducing Commitment Number



[312, 303, 233] is the MDP list for number 312 in base4:



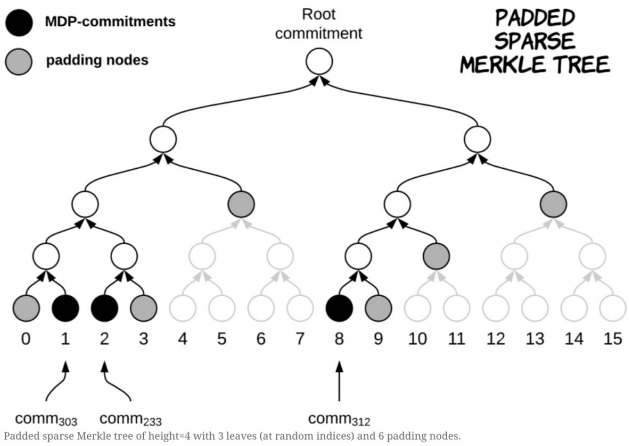
Reusing the chain

Generalizing Hash Chains: Hiding Commitment Number

Prove up to 2999 needs only one element in MDP list: [2999].

Verifier will learn from it the issued number can not be 2998 or any other integer

Use Merkle tree to padding random nodes with commitments nodes

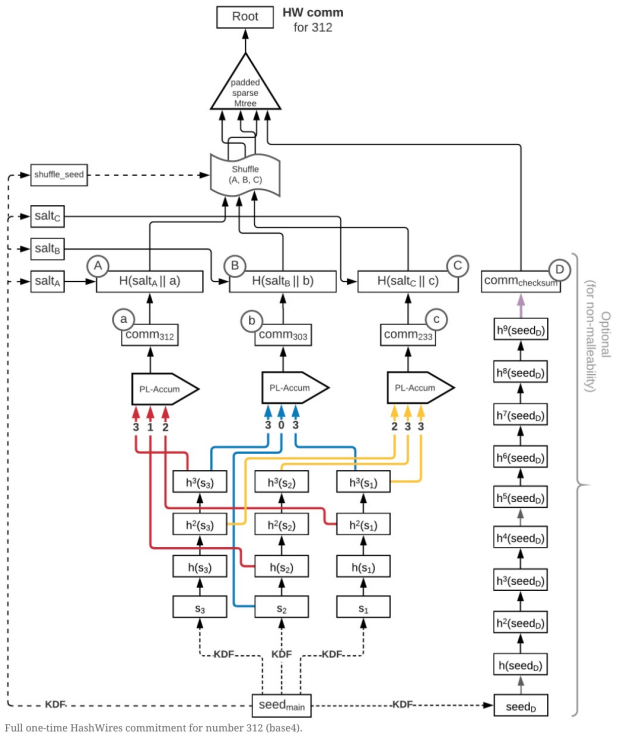


Generalizing Hash Chains: Putting All Things Together

HashWires: Hyperefficient Credential-Based Range Proofs

Both proof size & proof generation improved

*Project: Try to Implement this scheme



THANKS
Q&A