

SM3 Implementation And Application

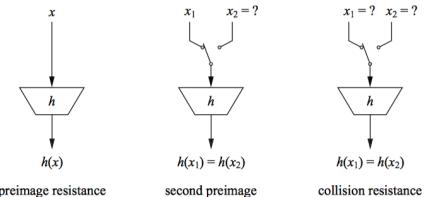
Long Wen
longwen6@gmail.com
20250713 @ Qingdao

Hash Function Basics

- Security of hash functions

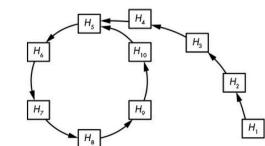
- Preimage resistance / first-preimage resistance
 - Second-preimage resistance
 - Collision resistance
 - Q&A: which one is weaker?
 - Q&A: MD5 and SHA1 is broken, meaning?
 - Q&A: collision could be found, so what ?
 - Q&A: how do you argue no one knows a secret ?

Address = RIPEMD160(SHA256(pubkey)) = 0000000000000000000000000000000071E76C



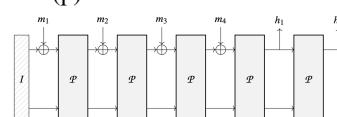
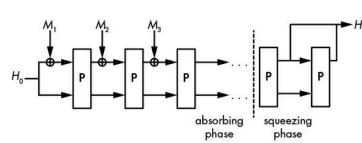
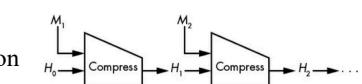
- Finding Collisions

- 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
 - Poseidon, rescue
 - Will cover this part later in zkSNARK parts



Merkel-Damgard Construction

- A way to turn a secure compression function taking small, fixed-length inputs into a secure hash function taking inputs of arbitrary lengths

- MD construction workflow
 - 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 \rightarrow compress function is secure
 - Weak block cipher led to bad hash function: Xbox game
 - Finding fixed point of DM construction
 - Q&A is the existence of fixed point a problem to hash fun

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

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

SM3 - Merkel-Damgård Construction

1.1.5 SM3 密码杂凑算法的消息填充

对长度为 $l < 2^{32}$ 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 $l+k+1 \equiv 448 \pmod{512}$ 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:

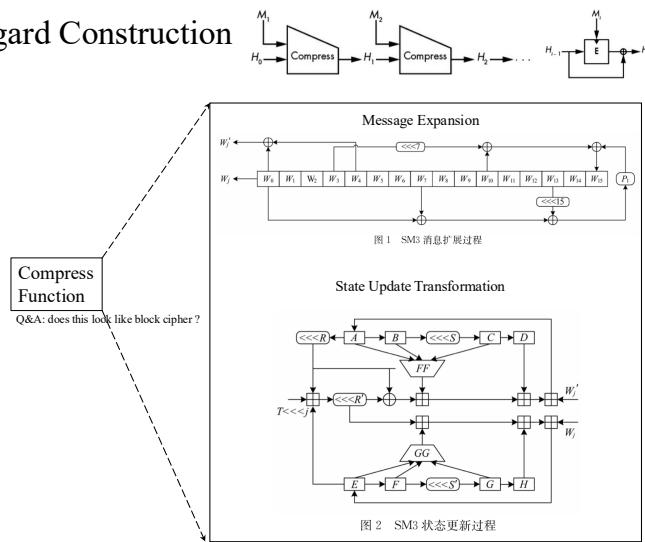
$\frac{12b}{01100001} 01100010 0110001110\cdots00\cdots011000$
(每 3 位制表示)

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: $m' = B^{(0)}B^{(1)}\dots B^{(n-1)}$, 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 $V^{(i)}$.



SM3 - Merkel-Damgård Construction

1.1.5 SM3 密码杂凑算法的消息填充

对长度为 $l < 2^{32}$ 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 $l+k+1 \equiv 448 \pmod{512}$ 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:

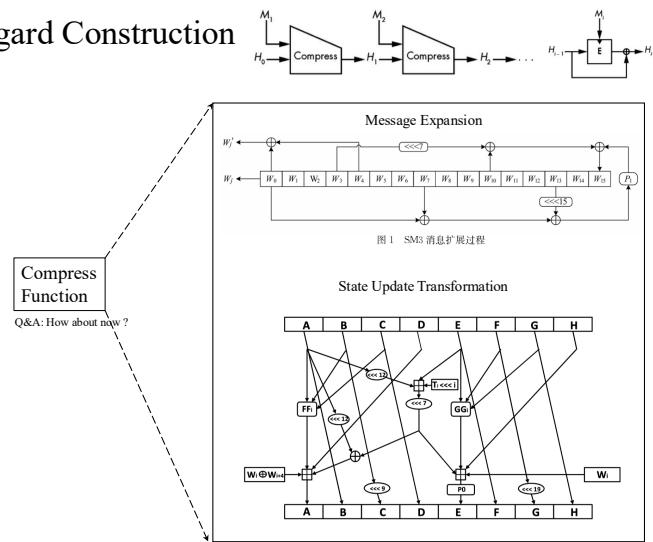
$\frac{12b}{01100001} 01100010 0110001110\cdots00\cdots011000$
(每 3 位制表示)

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: $m' = B^{(0)}B^{(1)}\dots B^{(n-1)}$, 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 $V^{(i)}$.



Length Extension Attack of The Merkel-Damgård Construction

- Length extension attack is the main threat to MD construction

Attack outline

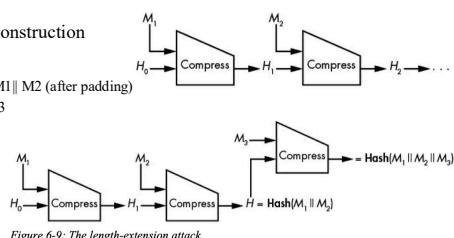
- If you know $\text{Hash}(M)$ for unknown message where $M = M_1 \parallel M_2$ (after padding)
- You can determine $\text{Hash}(M_1 \parallel M_2 \parallel M_3)$ for any block, M_3

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.



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 $\text{Hash}(M||C)$ as a response and sends the result to the client
- Client also computes $\text{Hash}(M||C)$ and checks that both hash values matches

Q&A: Is the protocol above secure? Why?

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: reinterpret 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.charCodeAt(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
> sm3
{
  _digest: '2f143c0fe7ca61f5f5071c0fd4d09f13056b1ca4997e70fc68',
  _block: '2f143c0fe7ca61f5f5071c0fd4d09f13056b1ca4997e70fc68'
}
> sm3()
'2f143c0fe7ca61f5f5071c0fd4d09f13056b1ca4997e70fc68'
> sm3(62)
'457a10a2e7c2f143c0fe7ca61f5f5071c0fd4d09f13056b1ca4997e70fc68'

```



问题概述

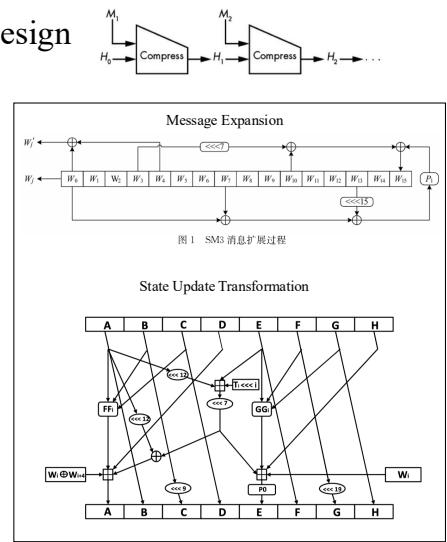
2021年2月，董语安全在对客户产品进行安全代码审计时，发现其依赖的微信小程序sm-crypto开源NodeJS国密算法库中的SM3算法实现（该算法库使用NodeJS对国密算法中的SM2、SM3和SM4三个算法进行了实现，并封装成接口提供给开发者使用。该国密算法库的具体实现实在GitHub上开源：<https://github.com/wechat-miniprogram/sm-crypto>，其中sm3的实现实在文件src/sm3/index.js中，微信小程序官方的参考文档链接为<https://developers.weixin.qq.com/miniprogram/dev/extended/utils/sm-crypto.html>）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

```

10 int sm3_hash_openssl(uint8_t *dgst, const void *msg, size_t len) {
11     int res = 0;
12     const EVP_MD *md = EVP_get_digestbyname("sm3");
13     EVP_MD_CTX *mdctx = EVP_MD_CTX_new();
14     if (!mdctx) goto done;
15
16     EVP_DigestInit_ex(mdctx, md, NULL);
17     EVP_DigestUpdate(mdctx, msg, len);
18     res = EVP_DigestFinal_ex(mdctx, dgst, NULL);
19
20 done:
21     EVP_MD_CTX_free(mdctx);
22     return res;
23 }
24
25 int sm3_hash_verify_openssl(const void *msg, size_t len, const void *dgst) {
26     uint8_t buf[32];
27     sm3_hash_openssl(buf, msg, len);
28     return memcmp(buf, dgst, 32);
29 }

```

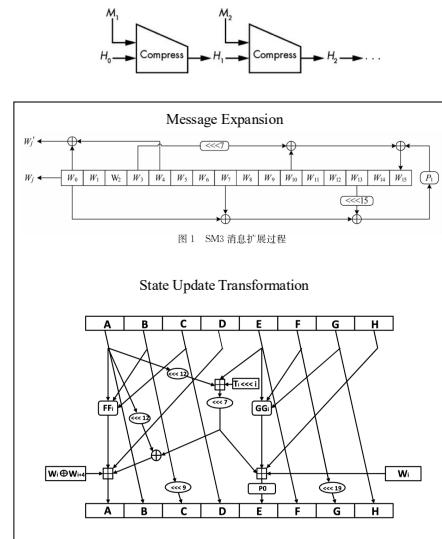


SM3 Implementation – Interface

```

5 #define sm3_digest_BYTIES 32
6 #define sm3_block_BYTIES 64
7 #define sm3_hmac_BYTIES sm3_digest_BYTIES
8
9 typedef struct sm3_ctx_t {
10     uint32_t digest[sm3_digest_BYTIES / sizeof(uint32_t)];
11     int nblocks; // number of blocks that have been processed
12     uint8_t block[sm3_block_BYTIES * 4];
13     int num;
14 } sm3_ctx;
15
16 void sm3_init(sm3_ctx *ctx);
17 void sm3_update(sm3_ctx *ctx, const uint8_t *data, size_t data_len);
18 void sm3_final(sm3_ctx *ctx, uint8_t *digest);
19
20 void sm3_hash(uint8_t *digest, const uint8_t *data, size_t dlen);
21 int sm3_hash_verify(const uint8_t *data, size_t dlen, const uint8_t *digest);
22
23 void sm3_init(sm3_ctx *ctx) {
24     ctx->digest[0] = 0x7300166F;
25     ctx->digest[1] = 0x4914B2B9;
26     ctx->digest[2] = 0x17244D7;
27     ctx->digest[3] = 0x0ABA0600;
28     ctx->digest[4] = 0xA96F30BC;
29     ctx->digest[5] = 0x163138AA;
30     ctx->digest[6] = 0x3B0EE4D;
31     ctx->digest[7] = 0x80F80E4E;
32
33     ctx->nblocks = 0;
34     ctx->num = 0;
35 }

```

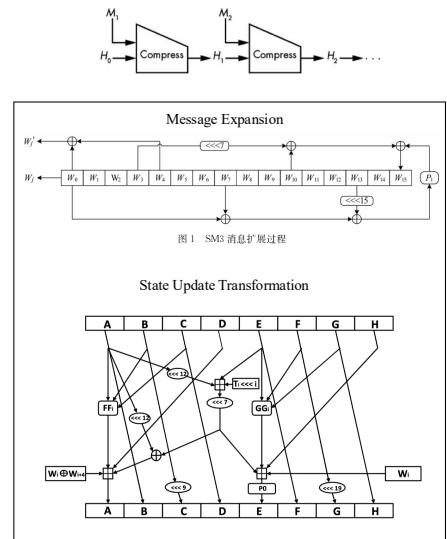


SM3 Implementation – Update

```

20 void sm3_update(sm3_ctx *ctx, const uint8_t *data, size_t dlen) {
21     if (ctx->num) {
22         unsigned int left = sm3_block_BYTIES - ctx->num;
23         if (dlen < left) {
24             memcpy(ctx->block + ctx->num, data, dlen);
25             ctx->num += dlen;
26             return;
27         } else {
28             memcpy(ctx->block + ctx->num, data, left);
29             sm3_compress(ctx->digest, ctx->block);
30             ctx->nblocks++;
31             data += left;
32             dlen -= left;
33         }
34     }
35     while (dlen >= sm3_block_BYTIES) {
36         sm3_compress(ctx->digest, data);
37         ctx->nblocks++;
38         data += sm3_block_BYTIES;
39         dlen -= sm3_block_BYTIES;
40     }
41     ctx->num = dlen;
42     if (dlen) {
43         memcpy(ctx->block, data, dlen);
44     }
45 }

```



SM3 Implementation – Final

```

47 void sm3_final(sm3_ctx *ctx, uint8_t *digest) {
48     size_t i;
49     uint32_t *pdigest = (uint32_t *) (digest);
50     uint64_t *count = (uint64_t *) (ctx->block + sm3_block_BYTES - 8);
51
52     ctx->block[ctx->num] = 0x80;
53
54     if (ctx->num + 9 <= sm3_block_BYTES) {
55         memset(ctx->block + ctx->num + 1, 0, sm3_block_BYTES - ctx->num - 9);
56     } else {
57         memset(ctx->block + ctx->num + 1, 0, sm3_block_BYTES - ctx->num - 1);
58         sm3_compress(ctx->digest, ctx->block);
59         memset(ctx->block, 0, sm3_block_BYTES - 8);
60     }
61
62     count[0] = (uint64_t) (ctx->nblocks) * 512 + (ctx->num << 3);
63     count[0] = byte_swap64(count[0]);
64
65     sm3_compress(ctx->digest, ctx->block);
66     for (i = 0; i < sizeof(ctx->digest) / sizeof(ctx->digest[0]); i++) {
67         pdigest[i] = byte_swap32(ctx->digest[i]);
68     }
69 }

```

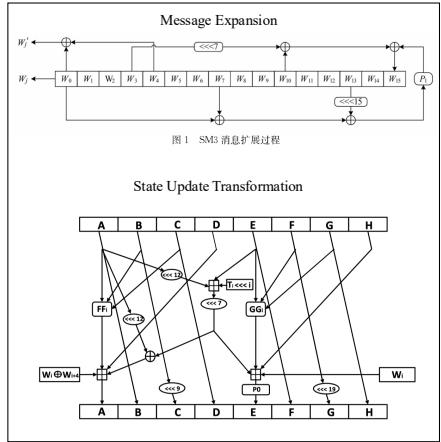
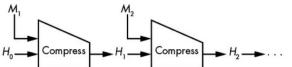


图 1 SM3 消息扩展过程

```

97 static void sm3_compress(uint32_t digest[sm3_digest_BYTES / sizeof(uint32_t)],
98                         const uint8_t block[sm3_block_BYTES]) {
99     int j;
100    uint32_t_ W[68], W1[64];
101    const uint32_t_ *pblock = (const uint32_t_ *) (block);
102
103    uint32_t_ A = digest[0], B = digest[1], C = digest[2], D = digest[3];
104    uint32_t_ E = digest[4], F = digest[5], G = digest[6], H = digest[7];
105
106    uint32_t_ SS1, SS2, TT1, TT2, T[64];
107
108    for (j = 0; j < 16; j++) W[j] = byte_swap32(pblock[j]);
109
110    for (j = 16; j < 68; j++) {
111        W[j] = P1(W[j - 16] ^ W[j - 9] ^ rol(W[j - 31], 15)) ^
112            rol(W[j - 13], 7) ^ W[j - 6];
113
114        for (i = 0; i < 64; i++) W1[i] = W[j] ^ W[i];
115
116        for (i = 0; i < 16; i++) {
117            T[i] = 0x9CC45495;
118            SS1 = rol((rol(A, 12) + E + rol(T[i], j)), 7);
119            SS2 = SS1 ^ rol(A, 12);
120            TT1 = FF(A, B, C) + D + SS2 + W1[i];
121            TT2 = GG(E, F, G) + H + SS1 + W1[i];
122            D = C, C = rol(B, 9), B = A, A = TT1;
123            H = G, G = rol(F, 19), F = E, E = P0(TT2);
124        }
125
126        for (j = 16; j < 64; j++) {
127            T[i] = 0x7AB790B4;
128            SS1 = rol((rol(A, 12) + E + rol(T[i], j)), 7);
129            SS2 = SS1 ^ rol(A, 12);
130            TT1 = FF(A, B, C) + D + SS2 + W1[j];
131            TT2 = GG(E, F, G) + H + SS1 + W1[j];
132            D = C, C = rol(B, 9), B = A, A = TT1;
133            H = G, G = rol(F, 19), F = E, E = P0(TT2);
134        }
135
136        digest[0] ^= A, digest[1] ^= B, digest[2] ^= C, digest[3] ^= D;
137        digest[4] ^= E, digest[5] ^= F, digest[6] ^= G, digest[7] ^= H;
138    }

```

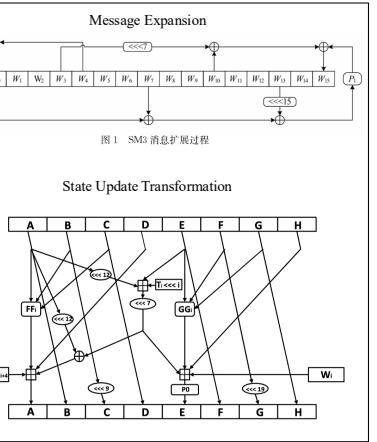
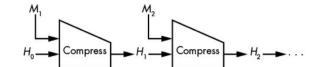


图 1 SM3 消息扩展过程

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

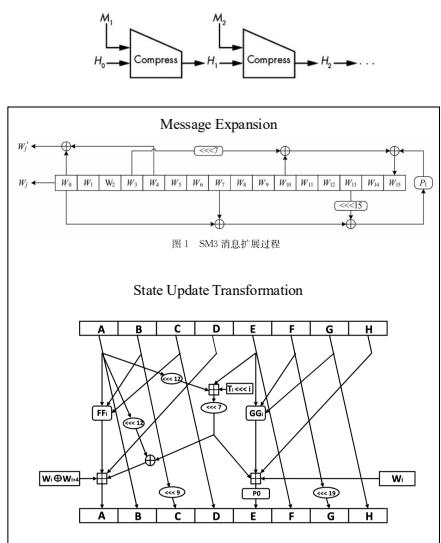
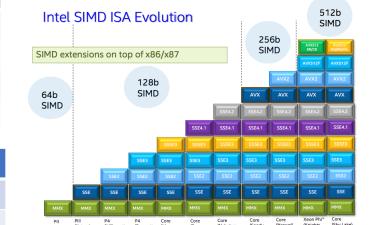


图 1 SM3 消息扩展过程

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	psrlq xmm, imm8	_m128i_mm_srli_epi32 (_m128i A, int n)

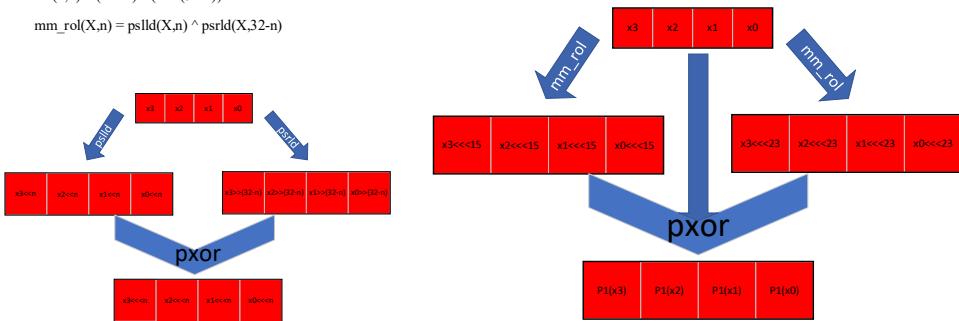
	instruction	C Language
shuffle bytes in XMM	pshufb xmm, xmm	_m128i_mm_shuffle_ep8 (_m128i A, _m128i mask)
shift whole XMM left by n bytes	pslldq xmm, imm8	_m128i_mm_slli_si128 (_m128i a, int n)
shift whole XMM right by n bytes	psrldq xmm, imm8	_m128i_mm_srli_si128 (_m128i a, int n)
shift Al B right by n bytes	palignr xmm, xmm, imm8	_m128i_mm_alignr_ep8 (_m128i a, _m128i b, int n)



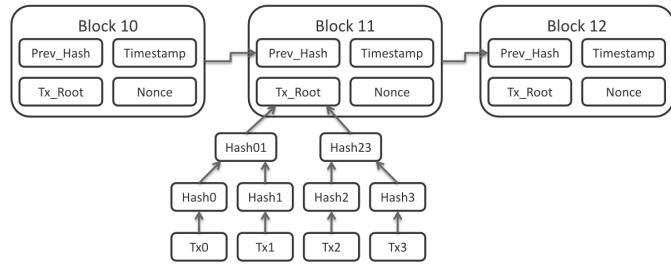
A Taste of SIMD, SM3's P1 Function

SM3 密码杂凑算法的置换函数定义如下：
 $P_0(X) = X \oplus (X \lll 9) \oplus (X \lll 17)$,
 $P_1(X) = X \oplus (X \lll 15) \oplus (X \lll 23)$.

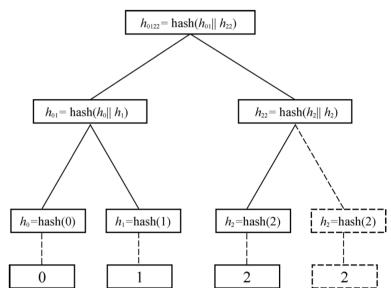
$\text{rol}(x,n) = (x << n) \wedge (x >> (32-n))$
 $\text{mm_rol}(X,n) = \text{pslld}(X,n) \wedge \text{psrld}(X,32-n)$



Blockchain – Chained By Hash Link

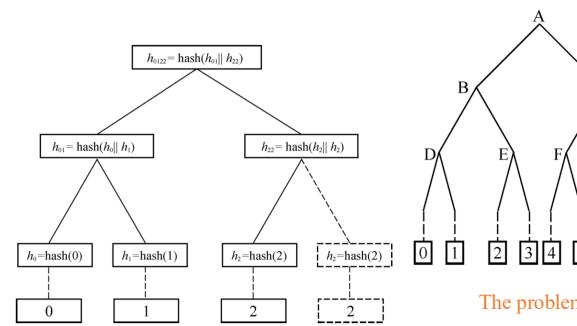


Merkle Tree In Bitcoin

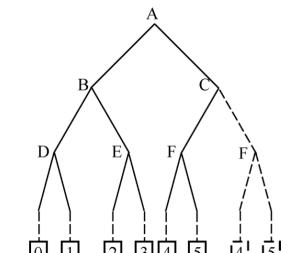


Q&A: Is the construction safe?

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

```

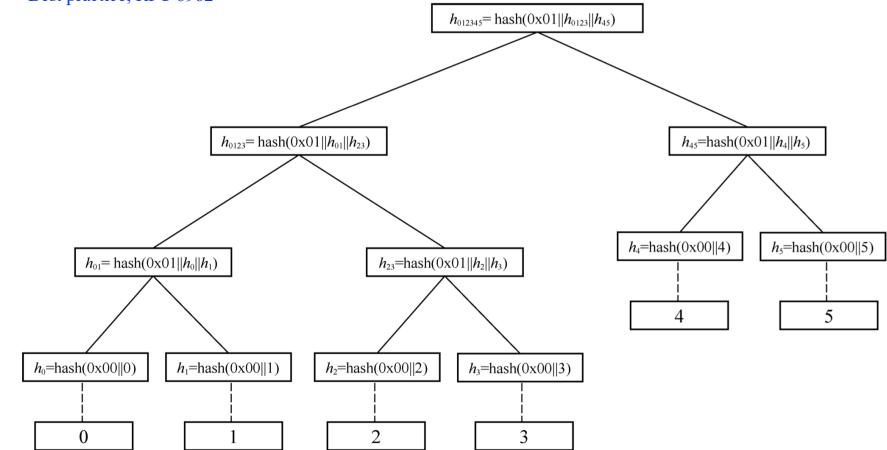
uint256 ComputeMerkleRoot(std::vector<uint256> hashes, bool* mutated) {
    bool mutation = false;
    while (hashes.size() > 1) {
        if (mutated) {
            for (size_t pos = 0; pos + 1 < hashes.size(); pos += 2) {
                if (hashes[pos] == hashes[pos + 1]) mutation = true;
            }
        }
        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 (mutation) *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);
}

```

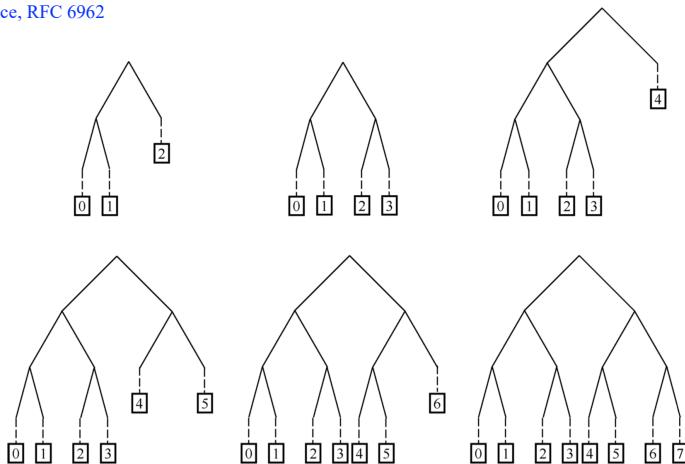
Engineering Merkle Tree

Best practice, RFC 6962



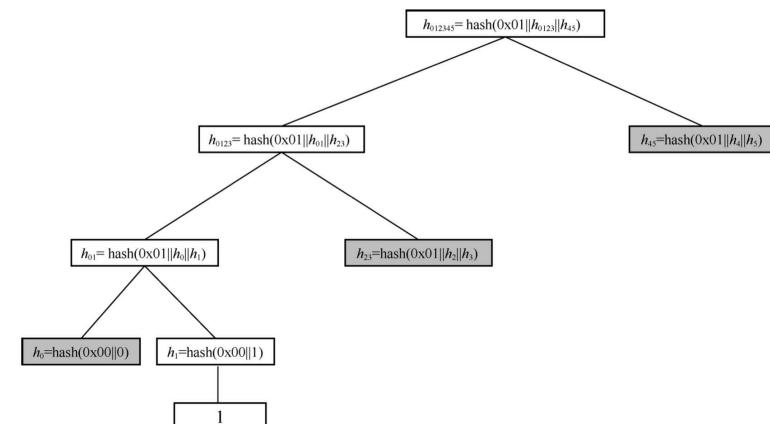
Engineering Merkle Tree

Best practice, RFC 6962



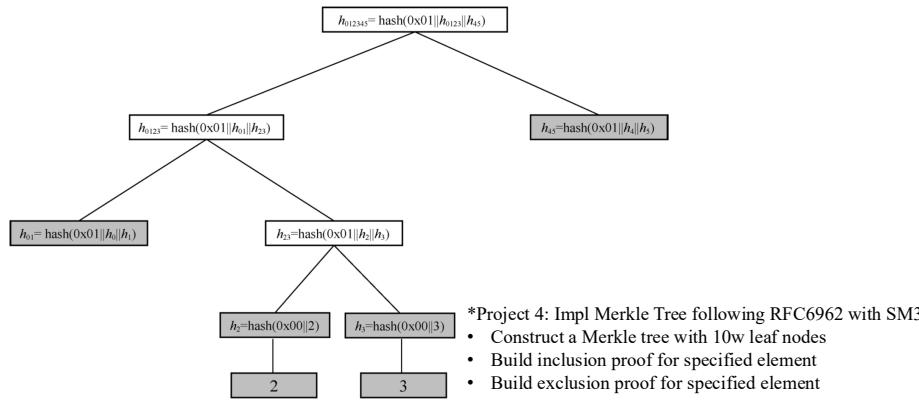
Merkle Tree – Inclusion Proof

Following RFC 6962



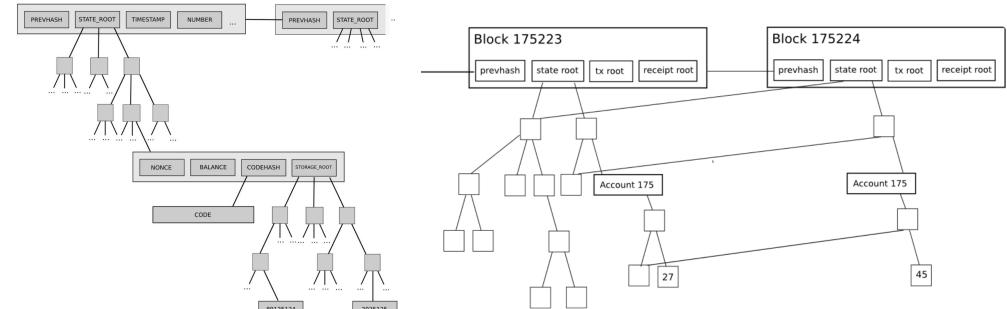
Merkle Tree – Exclusion Proof

Following RFC 6962, mind the extra requirement

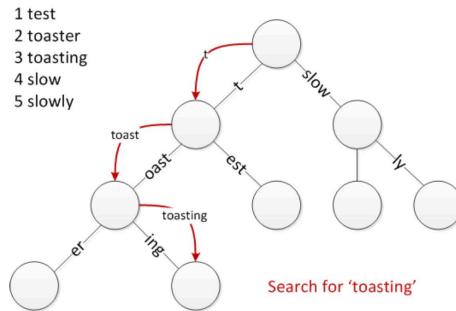


Global State in Ethereum - Updatable Merkle Trees

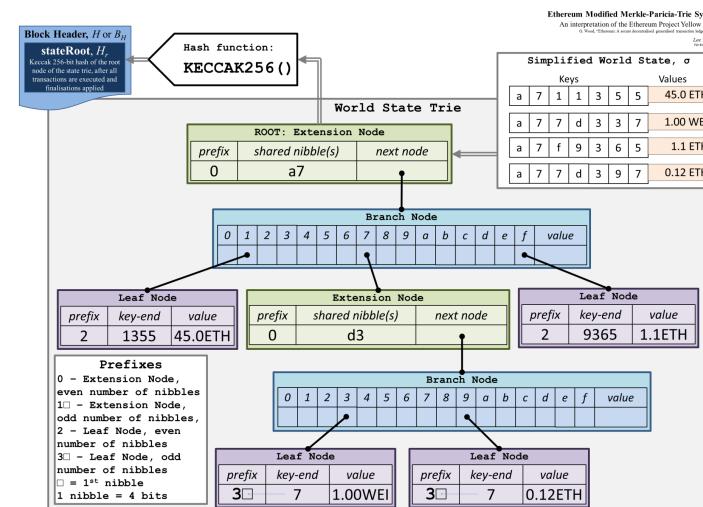
Q&A: how hard it is to add/remove/modify element in Merkle tree of RFC6962?



Global State in Ethereum - Merkle Patricia Tree



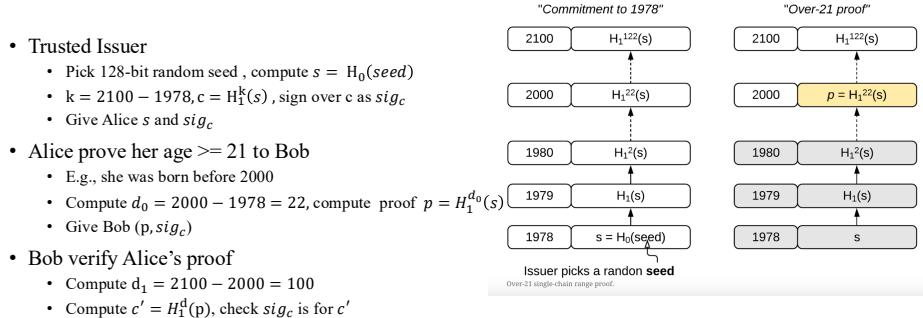
Global State in Ethereum - Merkle Patricia Tree



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

2021



Trusted Issuer

- Pick 128-bit random seed , compute $s = H_0(\text{seed})$
- $k = 2100 - 1978, c = H_1^k(s)$, sign over c as sig_c
- Give Alice s and sig_c

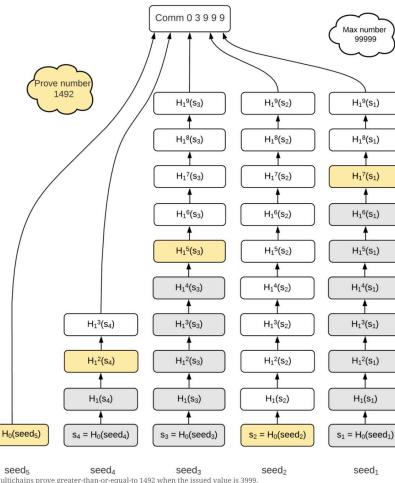
Alice prove her age ≥ 21 to Bob

- 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)

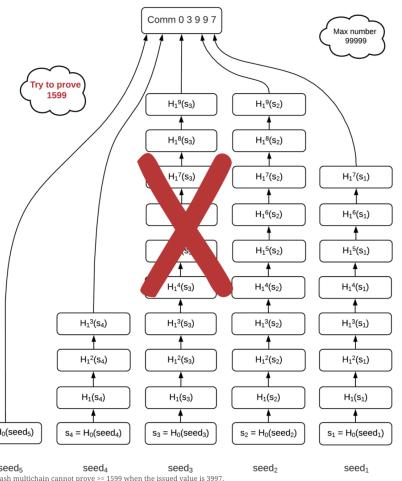
Bob verify Alice's proof

- Compute $d_1 = 2100 - 2000 = 100$
- Compute $c' = H_1^{d_1}(p)$, check sig_c is for c'

Generalizing Hash Chains



Cannot prove 1598 or 1599 if Comm. is for 03997

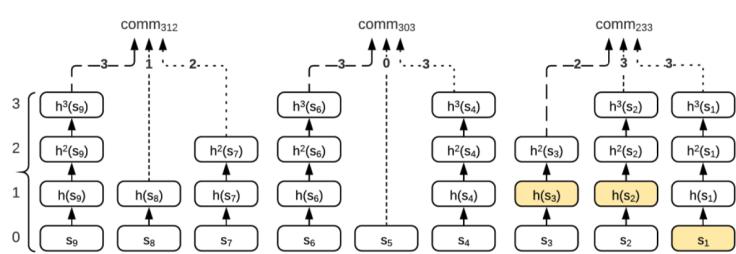


Generalizing Hash Chains: Fix Previous Problem

[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]

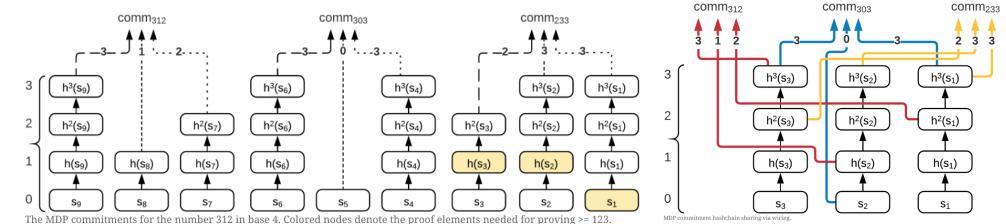
Call [312, 303, 233] MDP-list for base4 number 312



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:

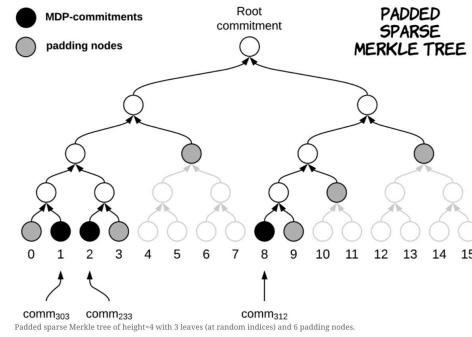
Reusing the chain

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

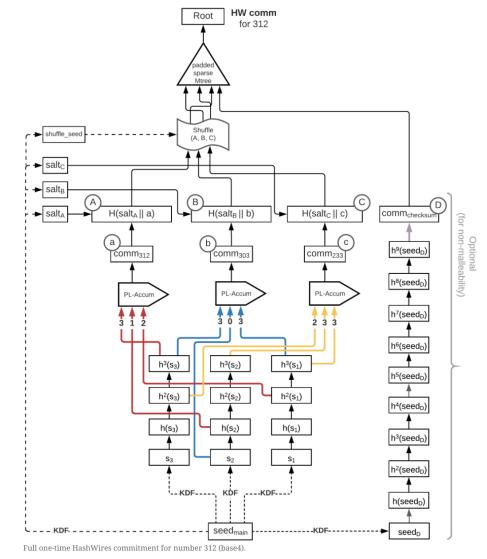


Generalizing Hash Chains: Putting All Things Together

HashWires: Hyperefficient Credential-Based Range Proofs

Both proof size & proof generation improved

*Project: Try to Implement this scheme



THANKS
Q&A