

原

SHA256算法原理详解

置顶

2018年07月03日 23:07:30

随煜而安

阅读数：13719

更多

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/u011583927/article/details/80905740

1. SHA256简介

SHA256是SHA-2下细分出的一种算法

SHA-2，名称来自于安全散列算法2（英语：Secure Hash Algorithm 2）的缩写，一种密码散列函数算法标准，由美国国家安全局研发，属于SHA算法之一，是SHA-1的后继者。

SHA-2下又可再分为六个不同的算法标准

包括了：SHA-224、SHA-256、SHA-384、SHA-512、SHA-512/224、SHA-512/256。

这些变体除了生成摘要的长度、循环运行的次数等一些微小差异外，算法的基本结构是一致的。

回到SHA256上，说白了，它就是一个哈希函数。

哈希函数，又称散列算法，是一种从任何一种数据中创建小的数字“指纹”的方法。散列函数把消息或数据压缩成摘要，使得数据量变小，将数据的格式固定下来。该函数将数据打乱混合，重新创建一个叫做散列值（或哈希值）的指纹。散列值通常用一个短的随机字母和数字组成的字符串来代表。

对于任意长度的消息，SHA256都会产生一个256bit长的哈希值，称作消息摘要。

这个摘要相当于是个长度为32个字节的数组，通常用一个长度为64的十六进制字符串来表示

来看一个例子：

干他100天成为区块链程序员，红军大叔带领着我们，fighting！

这句话，经过哈希函数SHA256后得到的哈希值为：

A7FCFC6B5269BDCCE571798D618EA219A68B96CB87A0E21080C2E758D23E4CE9

这里找到了一个SHA256在线验证工具，可以用来进行SHA256哈希结果的验证，后面也可以用来检验自己的SHA256代码是否正确。用起来很方便，不妨感受下。

2. SHA256原理详解

为了更好的理解SHA256的原理，这里首先将算法中可以单独抽出的模块，包括 常量的初始化、 信息预处理、 使用到的逻辑运算 分别进行介绍，甩开这些理解上的障碍后，一起来探索SHA256算法的主体部分，即消息摘要是如何计算的。

2.1 常量初始化

SHA256算法中用到了8个哈希初值以及64个哈希常量

其中，SHA256算法的8个哈希初值如下：

1

h0 := 0x6a09e667

2

h1 := 0xbb67ae85

3

h2 := 0x3c6ef372

4

h3 := 0xa54ff53a

5

h4 := 0x510e527f

6

h5 := 0x9b05688c

7

h6 := 0x1f83d9ab

8

h7 := 0x5be0cd19

这些初值是对自然数中前8个质数（2,3,5,7,11,13,17,19）的平方根的小数部分取前32bit而来

举个例子来说， $\sqrt{2}$ 小数部分约为0.414213562373095048，而

$$0.414213562373095048 \approx 6 * 16^{-1} + a * 16^{-2} + 0 * 16^{-3} + ...$$

```
1 | 428a2f98 71374491 b5c0fbcf e9b5dba5
2 | 3956c25b 59f111f1 923f82a4 ab1c5ed5
3 | d807aa98 12835b01 243185be 550c7dc3
4 | 72be5d74 80deb1fe 9bdc06a7 c19bf174
5 | e49b69c1 efbe4786 0fc19dc6 240ca1cc
6 | 2de92c6f 4a7484aa 5cb0a9dc 76f988da
7 | 983e5152 a831c66d b00327c8 bf597fc7
8 | c6e00bf3 d5a79147 06ca6351 14292967
9 | 27b70a85 2e1b2138 4d2c6dfc 53380d13
10 | 650a7354 766a0abb 81c2c92e 92722c85
11 | a2bfe8a1 a81a664b c24b8b70 c76c51a3
12 | d192e819 d6990624 f40e3585 106aa070
13 | 19a4c116 1e376c08 2748774c 34b0bcb5
14 | 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
15 | 748f82ee 78a5636f 84c87814 8cc70208
16 | 90bfeffa a4506ceb bef9a3f7 c67178f2
```

👍
4

💬
5

📖

🔖

📄

<

>

和8个哈希初值类似，这些常量是对自然数中前64个质数(2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97...)的立方根的小数部分取前32bit而来。

2.2 信息预处理(pre-processing)

SHA256算法中的预处理就是在想要Hash的消息后面补充需要的信息，使整个消息满足指定的结构。

信息的预处理分为两个步骤：[附加填充比特](#) 和 [附加长度](#)

STEP1：附加填充比特

在报文末尾进行填充，使报文长度在对512取模以后的余数是448

填充是这样进行的：先补第一个比特为1，然后都补0，直到长度满足对512取模后余数是448。

需要注意的是，信息必须进行填充，也就是说，即使长度已经满足对512取模后余数是448，补位也必须要进行，这时要填充512个比特。

因此，填充是至少补一位，最多补512位。

例：以信息“abc”为例显示补位的过程。

a,b,c对应的ASCII码分别是97,98,99

于是原始信息的二进制编码为：01100001 01100010 01100011

补位第一步，首先补一个“1”：0110000101100010 01100011 1

补位第二步,补423个“0”：01100001 01100010 01100011 10000000 00000000 ... 00000000

补位完成后的数据如下（为了简介用16进制表示）：

```
1 | 61626380 00000000 00000000 00000000
2 | 00000000 00000000 00000000 00000000
3 | 00000000 00000000 00000000 00000000
4 | 00000000 00000000
```

为什么是448？

因为在第一步的预处理后，第二步会再附加上一个64bit的数据，用来表示原始报文的长度信息。而448+64=512，正好拼成了一个完整的结构。

STEP2：附加长度值

附加长度值就是将原始数据（第一步填充前的消息）的长度信息补到已经进行了填充操作的消息后面。

wiki百科中给出的原文是：*append length of message (before pre-processing), in bits, as 64-bit big-endian integer*

SHA256用一个64位的数据来表示原始消息的长度。

因此，通过SHA256计算的消息长度必须要小于\$ 2^64 \$，当然绝大多数情况这足够了。

长度信息的编码方式为64-bit big-endian integer

关于Big endian的含义，文末给出了补充

开发者调查

Python学习路线!

会员任意学

中国大数据技术大会

登录 注册 ×

```
1 | 61626380 00000000 00000000 00000000
2 | 00000000 00000000 00000000 00000000
3 | 00000000 00000000 00000000 00000000
4 | 00000000 00000000 00000000 00000018
```

2.3 逻辑运算

SHA256散列函数中涉及的操作全部是逻辑的位运算

包括如下的逻辑函数：

$$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$$

$$Ma(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

$$\Sigma_0(x) = S^2(x) \oplus S^{13}(x) \oplus S^{22}(x)$$

$$\Sigma_1(x) = S^6(x) \oplus S^{11}(x) \oplus S^{25}(x)$$

$$\sigma_0(x) = S^7(x) \oplus S^{18}(x) \oplus R^3(x)$$

$$\sigma_1(x) = S^{17}(x) \oplus S^{19}(x) \oplus R^{10}(x)$$

其中：


| 逻辑运算 | 含义 |
|----------|-----------|
| \wedge | 按位“与” |
| \neg | 按位“补” |
| \oplus | 按位“异或” |
| S^n | 循环右移n个bit |
| R^n | 右移n个bit |

2.4 计算消息摘要

现在来介绍SHA256算法的主体部分，即消息摘要是如何计算的。


首先：将消息分解成512-bit大小的块

(break message into 512-bit chunks)

4

5

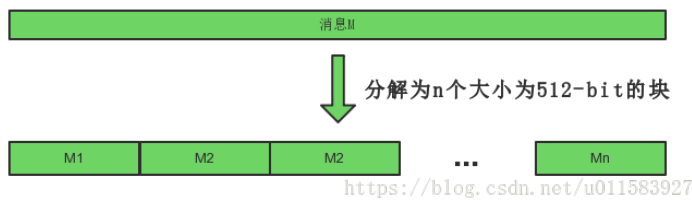










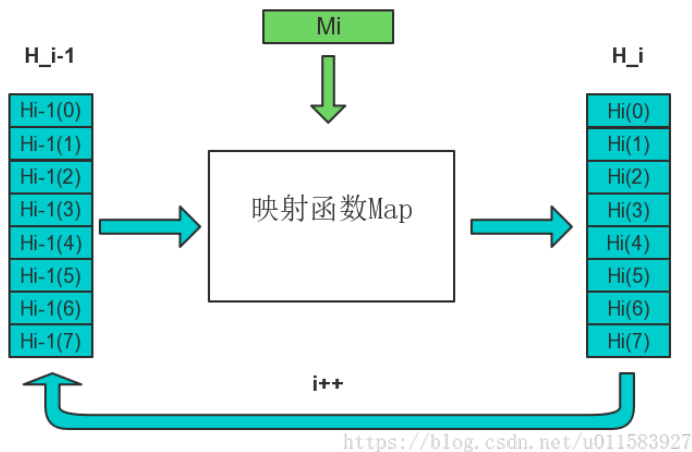


假设消息M可以被分解为n个块，于是整个算法需要做的就是完成n次迭代，n次迭代的结果就是最终的哈希值，即256bit的数字摘要。

一个256-bit的摘要的初始值H0，经过第一个数据块进行运算，得到H1，即完成了第一次迭代

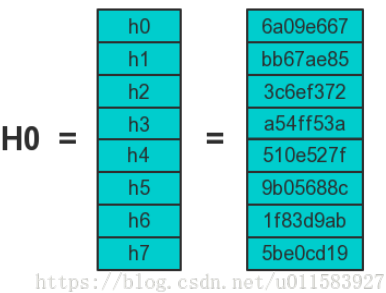
H1经过第二个数据块得到H2，.....，依次处理，最后得到Hn，Hn即为最终的256-bit消息摘要

将每次迭代进行的映射用 $Map(H_{i-1}) = H_i$ 表示，于是迭代可以更形象的展示为：



图中256-bit的 H_i 被描述8个小块，这是因为SHA256算法中的最小运算单元称为“字”（Word），一个字是32位。

此外，第一次迭代中，映射的初值设置为前面介绍的8个哈希初值，如下图所示：



下面开始介绍每一次迭代的内容，即映射 $Map(H_{i-1}) = H_i$ 的具体算法

STEP1: 构造64个字（word）

对于每一块，将块分解为16个32-bit的big-endian的字，记为w[0], ..., w[15]

也就是说，前16个字直接由消息的第i个块分解得到

其余的字由如下迭代公式得到：

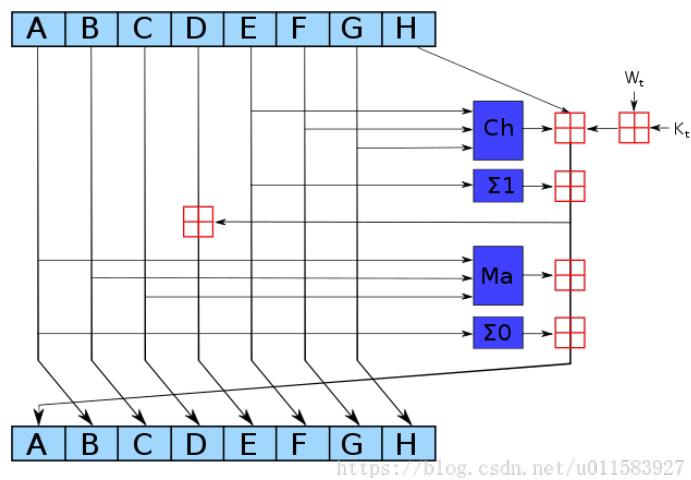
$$W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}$$

STEP2：进行64次循环

映射 $\text{Map}(H_{[i-1]}) = H_{[i]}$ 包含了64次加密循环

即进行64次加密循环即可完成一次迭代

每次加密循环可以由下图描述：



图中，ABCDEFGH这8个字（word）在按照一定的规则进行更新，其中

深蓝色方块是事先定义好的非线性逻辑函数，上文已经做过铺垫

红色田字方块代表 $\text{mod } 2^{32}$ addition，即将两个数字加在一起，如果结果大于 2^{32} ，你必须除以 2^{32} 并找到余数。

ABCDEFGH一开始的初始值分别为 $H_{[i-1]}(0), H_{[i-1]}(1), \dots, H_{[i-1]}(7)$

Kt是第t个密钥，对应我们上文提到的64个常量

Wt是本区块产生第t个word。原消息被切成固定长度512-bit的区块，对每一个区块，产生64个word，通过重复运行循环n次对ABCDEFGH这八个字循环加密。

最后一次循环所产生的八个字合起来即是第i个块对应的散列字符串 $H_{[i]}$

由此变完成了SHA256算法的所有介绍

3. SHA256算法伪代码

现在我们可以结合SHA256算法的伪代码,将上述的所有步骤进行梳理整合：

```
1 | Note: All variables are unsigned 32 bits and wrap modulo 232 when calculating
2 |
3 |
4 | Initialize variables
5 | (first 32 bits of the fractional parts of the square roots of the first 8 primes 2..19):
6 | h0 := 0x6a09e667
7 | h1 := 0xbb67ae85
8 | h2 := 0x3c6ef372
9 | h3 := 0xa54ff53a
10 | h4 := 0x510e527f
11 | h5 := 0x9b05688c
```

...

调查

Python学习路线!

会员任意学

中国大数据技术大会

```
14
15
16 Initialize table of round constants
17 (first 32 bits of the fractional parts of the cube roots of the first 64 primes 2..311):
18 k[0..63] :=
19     0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
20     0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
21     0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
22     0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
23     0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
24     0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
25     0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
26     0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
27
28
29 Pre-processing:
30 append the bit '1' to the message
31 append k bits '0', where k is the minimum number >= 0 such that the resulting message
32     length (in bits) is congruent to 448(mod 512)
33 append length of message (before pre-processing), in bits, as 64-bit big-endian integer
34
35
36 Process the message in successive 512-bit chunks:
37 break message into 512-bit chunks
38 for each chunk
39     break chunk into sixteen 32-bit big-endian words w[0..15]
40
41     Extend the sixteen 32-bit words into sixty-four 32-bit words:
42     for i from 16 to 63
43         s0 := (w[i-15] rightrotate 7) xor (w[i-15] rightrotate 18) xor(w[i-15] rightshift 3)
44         s1 := (w[i-2] rightrotate 17) xor (w[i-2] rightrotate 19) xor(w[i-2] rightshift 10)
45         w[i] := w[i-16] + s0 + w[i-7] + s1
46
47     Initialize hash value for this chunk:
48     a := h0
49     b := h1
50     c := h2
51     d := h3
52     e := h4
53     f := h5
54     g := h6
55     h := h7
56
57     Main loop:
58     for i from 0 to 63
59         s0 := (a rightrotate 2) xor (a rightrotate 13) xor(a rightrotate 22)
60         maj := (a and b) xor (a and c) xor(b and c)
61         t2 := s0 + maj
62         s1 := (e rightrotate 6) xor (e rightrotate 11) xor(e rightrotate 25)
63         ch := (e and f) xor ((not e) and g)
64         t1 := h + s1 + ch + k[i] + w[i]
65         h := g
66         g := f
67         f := e
68         e := d + t1
69         d := c
70         c := b
71         b := a
72         a := t1 + t2
73
74     Add this chunk's hash to result so far:
75     h0 := h0 + a
76     h1 := h1 + b
77     h2 := h2 + c
78     h3 := h3 + d
79     h4 := h4 + e
80     h5 := h5 + f
81     h6 := h6 + g
82     h7 := h7 + h
83
84 Produce the final hash value (big-endian):
```

45

<

>

[开发者调查](#)[Python学习路线!](#)[会员任意学](#)[中国大数据技术大会](#)[登录](#)[注册](#)

4. 参考文献

本篇笔记主要参考整合的资料如下：

- SHA-2 wiki
- 比特币算法——SHA256算法介绍
- SHA-256算法实现
- 操作指南：验证SHA256

4

5

知识填补

大端和小端（Big endian and Little endian）

对于整型、长整型等数据类型，都存在字节排列的高低位顺序问题。

Big endian 认为第一个字节是最高位字节（按照从低地址到高地址的顺序存放数据的高位字节到低位字节）

而 Little endian 则相反，它认为第一个字节是最低位字节（按照从低地址到高地址的顺序存放据的低位字节到高位字节）。

例如，假设从内存地址 0x0000 开始有以下数据：

| 地址 | 数据 |
|--------|------|
| ... | ... |
| 0x0000 | 0x12 |
| 0x0001 | 0x34 |
| 0x0002 | 0xab |
| 0x0003 | 0xcd |
| ... | ... |

假设我们去读取一个地址为 0x0000 的四个字节变量

若字节序为big-endian，则读出结果为0x1234abcd；

若字节序为little-endian，则读出结果为0xcdab3412。

如果我们将0x1234abcd 写入到以 0x0000 开始的内存中，则Little endian 和 Big endian 模式的存放结果如下：

| 地址 | 0x0000 | 0x0001 | 0x0002 | 0x0003 |
|----------------|--------|--------|--------|--------|
| big-Big_endian | 0x12 | 0x34 | 0xab | 0xcd |
| little-endian | 0xcd | 0xab | 0x34 | 0x12 |

参考文献

[点击返回正文](#)

想对作者说点什么

行走天涯的象： Sn是循环右移，Rn是直接右移，你上面写错了，我查了别的资料都是这样的 （5天前 #3楼） [查看回复\(1\)](#)

ybinvest： STEP1：附加填充比特在报文末尾进行填充，使报文长度在对512取模以后的余数是448 填充是这样进行的：先补第一个比特为1，然后都补0，直到长度满足对512取模后余数是448。下面这句话实验发现 是错误的，字符数是56个的 原始数据，正好是 448 bit位，不需要补位，后面的8个字节直接存储原始数据的长度信息。需要注意的是，信息必须进行填充，也就是说，即使长度已经满足对512取模后余数是448，补位也必须要进行，这时要填充512个比特。因此，填充是至少补一位，最多补512位。 （1个月前 #2楼）

ybinvest： STEP1：构造64个字（word） break chunk into sixteen 32-bit big-endian words w[0], ..., w[15] 对于每一块，将块分解为16个32-bit的big-endian的字，记为w[0]. w[15] 也就是说，前16个字直接由消息的第i块分解得到 ?????????? 一个块就16个字，怎么还前16个？ 还有后16个字？ 其余的字由如下迭代公式得到： ? ? ? ? ? ? 其余的意思是什么，一个块有32个字？ （1个月前 #1楼） [查看回复\(1\)](#)

SHA-256算法实现 - 随心散人专栏

转载来自于 《基于FPGA的SHA-256算法实现》和 http://www.cnblogs.com/tofixer/p/3498048.ht...

5.2万

来自: 随心散人专栏

加密算法比较：SHA1，SHA256，MD5 - 极客神殿

MD5输出128bit、SHA1输出160bit、SHA256输出256bit SHA-1是160位的哈希值，而SHA-2是组...

810

来自: 极客神殿

sha256 加密算法 - 我不是大牛

sha256 加密算法 go 调用 sha256 加密字符串哈希值 package main import("fmt" "crypto/sha256" "os") func main() { s := "hello world" h := sha256.Sum256([]byte(s)) fmt.Println(h.Hex()) }

1772

来自: 我不是大牛

关于SHA256解密的处理方法 - 一个人的博客

PS：今天发现SHA256加密的问题，还有不少人关注。声明这只是我个人的处理，如果暴力解密也是...

6435

来自: 一个人的博客

sha256实现代码（C++模板类） - ak47000gb的博客

目前在网上找到的比较高效稳定的一个生成sha256的代码，只包含头文件就可以了，简单易用，同时...

638

来自: ak47000gb的博客

RSA加密与SHA签名用法详解 - Levilly的博客

基础知识什么是RSA？答：RSA是一种非对称加密算法，常用来对传输数据进行加密，配合上数字摘...

6674

来自: Levilly的博客

Java实现SHA256算法 - 王凯琦的博客

import java.io.UnsupportedEncodingException; import java.security.MessageDigest; import java.s...

1574

来自: 王凯琦的博客

Android RSA加密与SHA256算法工具类 - wylong1991的博客

Android开发中我们经常会用到各种加密，一般针对一些密码加密，下面给说一下RSA加密与SHA256...

608

来自: wylong1991的博客

sha256 - 韩帅的博客

import java.io.File; import java.io.FileInputStream; import java.math.BigInteger; import java.securi...

397

来自: 韩帅的博客

文章热词 机器学习 机器学习课程 机器学习教程 深度学习视频教程 深度学习学习

相关热词

c# sha256 加密 c# sha256解密 android查看sha256 c# sha256模拟 go语言实现sha256 python初级教程:入门详解 python3教程详解

SHA1和SHA256算法C语言实现 - 明潮的BLOG

SHA家族的五个算法，分别是SHA-1、SHA-224、SHA-256、SHA-384，和SHA-512，由美国国家...

572

来自: 明潮的BLOG



低调的洋仔

关注 311篇文章



罗小辉

关注 101篇文章



寸草心

关注 88篇文章

换一批

手把手教你android studio terminal 命令获取SHA1和MD5、SHA256值 - 悟-静的博客

在开发项目时，我们会获取项目的SHA1值或者MD5来作为第三方集成需要的值。而我们集成的时候...

7824

来自: 悟-静的博客

比特币中的SHA256是何方神圣？ - u013107902的博客

比特币中的SHA256是何方神圣？ 在比特币中钱包地址的产生过程中用到了一种叫做SHA256的...

3859

来自: u013107902的博客

下载 BM算法原理图示详细讲解

03-14

BM算法原理图示详细讲解

弹窗式恶搞网页的设计与发布 - lusongno1的博客

这个idea是我在去年春节的时候想起来的。原因是之前收到一个qq红包，看起来像真的一样，打开一...

8135

来自: lusongno1的博客

SuperSearch(超级网搜) - BoomWorks

软件简介(Introduction)免费、轻量、快速的多引擎搜索工具，拥有详细的搜索分类。免费：无须注册...

2.5万

来自: BoomWorks

精准平特一肖公式/平码三中三/平码3中3/平特一肖/平特一码/平特1码 - qq_37873853的博客

精准平特一肖公式/平码三中三/平码3中3/平特一肖/平特一码/平特1码平码三中三网站精准/精准平码...

4万

来自: qq_37873853的博客

sha-256.js - weixin_38198276

导入sha-256.js后；通过var password = SHA256(\$("#password").val());进行加密

下载 07-14

开发者调查

Python学习路线!

会员任意学

中国大数据技术大会

登录

注册

×

| | | |
|--|-------|--|
| <div>下载</div> <div>SHA256算法的verilog 实现</div> | 06-01 | |
| SHA256算法的verilog实现 IPCore 自动生成的FIFO和ROM没有上传 都是使用高端block memory 同时这里的rd_wr_dram模块是我实验室自己开发板子上的读写存控逻辑 | | |
| <div>哈希算法SHA-256实现示例 - QQ604666459的博客</div> <div>哈希算法SHA-256实现示例参考： https://en.wikipedia.org/wiki/SHA-2 https://en.wikipedia.org/wiki/SHA-256 来自： QQ604666459的博客</div> | 181 | |
| <div>下载</div> <div>Canny边缘检测算法原理及其VC实现详解 - zhaolong12323</div> | 04-15 | |
| Canny边缘检测算法原理及其VC实现详解Canny边缘检测算法原理及其VC实现详解 | | |
| <div>（转）比特币算法——SHA256算法介绍 - Julia & Rust & Python</div> <div>比特币算法——SHA256算法介绍标签： 密码技术 sha256 消息摘要 SHA256是安全散列算法SHA（S... 来自： Julia & Rust & Python</div> | 4945 | |
| <div>c# 实现MD5，SHA1，SHA256，SHA512等常用加密算法 - mituan1234567的专栏</div> <div>http://www.cnblogs.com/dreign/archive/2007/05/18/751089.html using System; using System.I... 来自： mituan1234567的专栏</div> | 2198 | |
| <div>SHA256算法 - code_segment的博客</div> <div>SHA-256 算法输入报文的最大长度不超过2^64 bit，输入按512-bit 分组进行处理，产生 的输出是一... 来自： code_segment的博客</div> | 371 | |
| <div>Go sha256使用实例介绍 - 用心做事</div> <div>SHA-256安全散列算法SHA（Secure Hash Algorithm）是美国国家安全局（NSA）设计，美国国家... 来自： 用心做事</div> | 4611 | |
| <div>最新坦白说破解方法！！ - WYJ的博客</div> <div>前言qq坦白说的推出让许多人感到烦恼，或是被骚扰，或是被撩，完事儿被戏弄之后你还不能屏蔽。... 来自： WYJ的博客</div> | 15.9 | |
| <div>常见加密算法分,用途,原理以及比较 - 远方的专栏</div> <div>常见加密算法分,用途,原理以及比较 密码学简介据记载，公元前400年，古希腊人发明了置换密码。1... 来自： 远方的专栏</div> | 6385 | |
| <div>消息摘要算法-HMAC算法 - feiyangxiaomi的专栏</div> <div>一、简述 mac（Message Authentication Code，消息认证码算法）是含有密钥散列函数算法，兼容... 来自： feiyangxiaomi的专栏</div> | 2.3万 | |
| <div>深入理解SHA系列加密算法 - 孙启超</div> <div>介绍 SHA是一系列的加密算法，有SHA-1、SHA-2、SHA-3三大类，而SHA-1已经被破解，SHA-3应... 来自： 孙启超</div> | 141 | |
| <div>比特币钱包地址的概念以及SHA256和椭圆曲线乘法加密算法的详解 - zzy179982985的博客</div> <div>比特币钱包，密钥，地址钱包就是密钥所在之处，钱包是私钥的容器，通过有序文件或者简单的数据... 来自： zzy179982985的博客</div> | 599 | |
| <div>kmp算法原理详解 - tyut小郑</div> <div>字符串匹配是计算机的基本任务之一。举例来说，有一个字符串"BBC ABCDAB ABCDABCDABDE"... 来自： tyut小郑</div> | 1096 | |
| <div>C学习 - SHA256算法的实现 - jianhui_wang的专栏</div> <div>1. Sha2.h/** * \file sha2.h * * \brief SHA-224 and SHA-256 cryptographic hash function * *... 来自： jianhui_wang的专栏</div> | 382 | |
| <div>sha256算法 c语言实现 - maxzero的专栏</div> <div>sha256算法，网上有很多的介绍，摘抄一段如下： SHA-256 算法输入报文的最大长度不超过2^64 bi... 来自： maxzero的专栏</div> | 415 | |
| <div>下载</div> <div>MD5 SHA1 SHA256 的C语言源码</div> | 12-14 | |
| MD5/SHA1/SHA256 纯C语言源码,支持增量计算. 包括测试程序与官方文档. 该代码我已按纯C语言基于接口编程方式封装,可以直接调用.有问题请联系我. | | |
| <div>SHA1算法升级SHA256更新计划 - zzstack的专栏</div> <div>原文链接： http://www.ert7.com/service/question/4968.html 微软SHA1升级计划 2013年11月份的... 来自： zzstack的专栏</div> | 2731 | |
| <div>使用OpenSSL生成IIS可用的SHA-256自签名证书 - TMajier的博客</div> <div>使用OpenSSL生成IIS可用的SHA-256自签名证书好吧，2017年iOS就开始强制开启ATS了，那么所有... 来自： TMajier的博客</div> | 8436 | |
| <div>常用数字货币挖矿算法 - 金石软件</div> <div>算法 Scryptstratum+tcp://scrypt.LOCATION.nicehash.com:3333 SHA256stratum+tcp://sha256.L... 来自： 金石软件</div> | 2325 | |
| <div>表白密码:I Love you的42种密码表白方式 - qq_32047637的博客</div> <div>字母表白数字密码：9121522521 表白解密：从1开始到26，分别表示从A到Z，即：A（1） B（2） ... 来自： qq_32047637的博客</div> | 15万 | |

一个牛人给JAVA初学者的建议 - yaya_free的专栏

给初学者之一：浅谈java及应用学java不知不觉也已经三年了从不知java为何物到现在一个小小的j2ee...

来自： [yaya_free的专栏](#)

下载

Dijkstra算法原理详解 - rocyequ

01-20

Dijkstra算法原理详解，弄不懂的可以看一下

下载

SHA256加密

01-16

SHA256加密，SHA256加密，SHA256加密，SHA256加密，SHA256加密，SHA256加密，SHA256加密，SHA256加密，SHA256加密

MD5加密算法与SHA加密算法 - 王洋的专栏

2、MD5加密 2.1 概述 Message Digest Algorithm MD5（中文名为消息摘要算法第五版）为计算机安...

来自： [王洋的专栏](#)

MD5, SHA256, SHA512哈希算法 - 极客神殿

StringHasher.cs /// /// 实现各种字符串hash散列算法的类 /// public class StringHasher { ...

来自： [极客神殿](#)

生成文件的MD5、SHA、SHA256 - sunny05296的博客

生成文件的MD5、SHA、SHA256 Linux系统生成MD5、SHA、SHA256 md5sum file1.zip >> MD5...

来自： [sunny05296的博客](#)

如何应对SHA-1加密算法升级为SHA-256 - CTO_ZhangHui_的博客

经过权威机构证实，sha1加密算法的不安全性越来越高，sha指纹造假成本越来越低，随即微软、谷歌...

来自： [CTO_ZhangHui_的博客](#)

网页版快手视频去水印下载工具 - Spring的博客

分享一个快手视频在线解析下载的工具，网上的大多都是要下载软件，只有这个不需要安装任何软件...

来自： [Spring的博客](#)

深入理解计算机系统（原书第3版）

和第2版相比，本版内容上的变化是，从以IA32和x86-64为基础转变为完全以x86-64为...

随煜而安

博客专家

关注

原创

75

粉丝

264

喜欢

110

评论

99

等级：

博客 5

访问：

27万+

积分：

2782

排名：

1万+

勋章：

博主专栏

探索比特币源码

阅读量：13331 14 篇

- 个人分类
- 区块链开发16篇
 - 数字图像处理与计算机视觉19篇
 - Machine Learning7篇
 - C++12篇
 - C#11篇
 - Python11篇
 - 量化投资探索4篇
 - 初级蜘蛛侠-编号897571篇
 - 最优化算法5篇
 - Deep Learning1篇

开发者调查

Python学习路线！

会员任意学

中国大数据技术大会

4

5

<

>

登录

注册

×

风机桨叶识别与故障诊断项目

7篇

展开

最新评论

利用python实现短信和电话提醒...

Realazas: 好! 支持! 威武! 有希望了!

SHA256算法原理详解

u011583927: [reply]qq_27854685[/reply] 感谢您的指点, 已更正

SHA256算法原理详解

qq_27854685: Sn是循环右移, Rn是直接右移, 你上面写错了, 我查了别的资料都是这样的

SHA256算法原理详解

qq_26369907: [reply]ybinvest[/reply] 消息块M是512位, 即16*32=512, 故前16...

使用Atom快速打造好用的Mark...

wskz876: 感谢博主,markdown的工具链困扰我好几个月了,,哎,像个没头苍蝇一样乱撞,乱折腾,哎,网上博...

联系我们



微信客服



QQ客服

 QQ客服

 kefu@csdn.net

 客服论坛

 400-660-0108

工作时间 8:00-22:00

关于我们 招聘 广告服务 网站地图

 百度提供站内搜索 京ICP证09002463号

©1999-2018 江苏乐知网络技术有限公司

江苏知之为计算机有限公司 北京创新乐知信息技术有限公司版权所有

网络110报警服务 经营性网站备案信息

北京互联网违法和不良信息举报中心

中国互联网举报中心



4



5





