谁能想到，我们之前用作校验文件，核验数据的一种算法，现在却奠定了区块链数字货币帝国大厦，这个算法有很多名字，比如一般叫做哈希算法（HASH的英文而来），有的也称在杂凑算法（很形象~~）或者散列算法等（以下我们统称哈希）。那么现在我们就来解开这种算法的层层面纱，在本文中，我们将解析加密货币中最常用的四种加密哈希函数的特性和差异。但在此之前，还是需要了解哈希的含义。

什么是HASH（哈希）

简单来说，哈希意味着输入任意长度的字符串通过密码运算来实现固定长度的输出。在像比特币这样的加密货币的情况下，交易被视为输入并通过哈希算法（比特币使用SHA-256）运行（该算法提供固定长度的输出）。

先了解下哈希算法是如何工作的。我们以SHA-256（256位安全哈希算法）为例。

在SHA-256的情况下，无论你的输入有多大或多小，输出总是有一个固定的256位长度。这对于处理大量的数据和事务校验时非常关键，这里先看看哈希函数的各种特性以及它们在区块链中的实现方式。

加密哈希算法特性

哈希算法其特俗的特性使得非常适合密码应用，加密哈希算法具有以下几个特性：

特性 1：确定性

这意味着无论 你通过哈希函数解析特定输入多少次， 你都会得到相同的结果。这是至关重要的，因为如果每次都得到不同的哈希值，就不可能跟踪输入。

特性2：快速计算

哈希函数应该能够快速返回输入的哈希。

性质3：单向性

单向性指的是：假设A是输入数据并且H(A)是输出哈希结果，假设给定H(A)来倒推出A是“不可行”的（注意使用“不可行”一词而不是“不可能”）。

因为我们已经知道从它的哈希值中确定原始输入并不是“不可能”的。举个例子吧。

> 假设你掷骰子，输出是骰子出现的数字的哈希。我们将如何确定原始号码是什么？很简单，所要做的就是从1-6中找出所有数字的哈希值并进行比较。由于哈希函数是确定性的，所以特定输入的哈希总是相同的，因此我们可以简单地比较哈希并找出原始输入。
>
> 但是，只有在给定的数据量非常少的情况下，这才有效。当你有大量的数据时会发生什么？假设你正在处理128位哈希。唯一必须找到原始输入的方法是使用“暴力群举”。暴力群举基本上意味着你必须选取一个随机输入，对其进行哈希，然后将输出与目标哈希进行比较并重复，直到找到匹配。

如果你使用这种暴力方法后：

最好的结果： 你可能在第一次尝试时获得答案，不过这只是理论上的可能，因为他的概率是2的128次方分之1（自己去算算）。

最坏的情况：你在2 ^ 128 - 1次之后得到答案。基本上，这意味着你会在所有数据的末尾找到你的答案。

平均情景：在2 ^ 128/2 = 2 ^ 127次之后，你会在中间的某处找到它。从这个角度来看，2 ^ 127 = 1.7 *10 ^ 38。换句话说，这是一个巨大的数字。

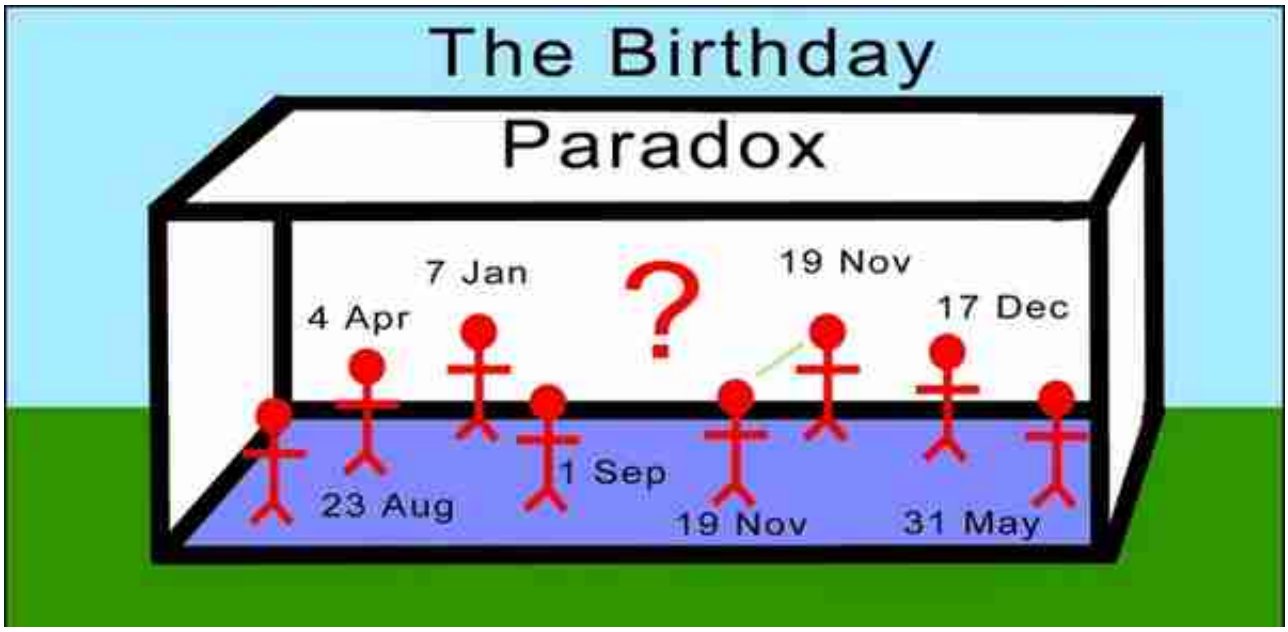
所以，虽然可以通过暴力方法来打破单向性，但同时也要有巨大的算力投入。

特性4：雪崩效应。

即使 你对原始输入数据进行了小改动，哈希的结果差异也非常大。让我们使用SHA-256对其进行测试：

即使你只是改变了输入的第一个字母表的大小写，看看它对输出哈希值有多大影响。

这个特性也被称为雪崩效应。



生日悖论

这是基于一个简单的概率原理：

假设你有一个事件发生N种不同的可能性，那么你需要N个随机项的平方根，以使它们有50%的碰撞几率。

因此，将这个理论应用于生日时， 你有365种不同的生日可能性，所以
你只需要Sqrt（365），即约23%，随机选择两个人分享生日的可能性为50%。

假设你有一个128位哈希，有2 ^
128种不同的可能性。通过使用生日悖论，你有50%的几率在sqrt（2 ^ 128）= 2 ^ 64的情况下打破防碰撞机制。

正如你所看到的，打破防碰撞要比打破单向性容易得多。所以，如果你使用的是SHA-256这样的函数，假设如果H（A）= H（B），那么A = B是可以认为成立。

那么，你如何创建一个抗碰撞的哈希函数？为此，我们使用一种称为Merkle-Damgard的结构。

什么是Merkle-Damgard结构？

该结构非常简单，并且遵循以下原则：给定短消息的防碰撞哈希函数，我们可以为长消息构造防碰撞哈希函数。

Comparison of SHA functions

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Rounds	Operations	Security bits (bits)	Capacity against length extension attacks	Performance on Skylake (median cpb)[1]		First Published
										long messages	8 bytes	
MD5 (as reference)		128	128 (4 × 32)	512	Unlimited[2]	64	And, Xor, Rot, Add (mod 2^{32}), Or	<64 (collisions found)	0	4.99	99.00	1992
SHA-0		160	160 (5 × 32)	512	$2^{64}-1$	80	And, Xor, Rot, Add (mod 2^{32}), Or	<34 (collisions found)	0	≈ SHA-1	≈ SHA-1	1993
SHA-1								<63 (collisions found[3])		3.47	52.00	1995
SHA-2	SHA-224	224	256 (8 × 32)	512	$2^{64}-1$	64	And, Xor, Rot, Add (mod 2^{32}), Or, Shr	112	32	7.62	84.50	2004
	SHA-256	256						128	0	7.63	85.25	2001
	SHA-384	384	512 (8 × 64)	1024	$2^{128}-1$	80	And, Xor, Rot, Add (mod 2^{64}), Or, Shr	192	128 (≈384)	5.12	130.75	
	SHA-512	512						256	0	5.09	135.50	
	SHA-512/224	224						112	288	≈ SHA-384	≈ SHA-384	
	SHA-512/256	256						128	256			
SHA-3	SHA3-224	224	1600 (5 × 5 × 64)	1152	Unlimited[4]	24[5]	And, Xor, Rot, Not	112	448	8.12	154.25	2015
	SHA3-256	256		1088				128	512	8.59	155.50	

各类哈希比较

让我们仔细看看SHA-256和SHA-3。

SHA-256

SHA-256是一个SHA-2函数，它使用32个单词，而不是使用64位单词的SHA-512。比特币在以下两种情况下使用SHA-256：

- 采矿。
- 创建地址。

采矿：

比特币采矿涉及矿工解决复杂的计算难题，以找到一个块，然后将其附加到比特币区块链。就是我们经常称的工作量证明（proof-of-work），它涉及SHA-256哈希函数的计算。

创建比特币地址、；

SHA-256哈希函数用于哈希比特币公钥以生成公共地址。哈希密钥为身份认证增加了一层额外的保护。此外，哈希地址的长度仅比存储更好的比特币公钥更短。

SHA-256运行实例：

输入：Hi

输出：98EA6E4F216F2FB4B69FFF9B3A44842C38686CA685F3F55DC48C5D3FB1107BE4

SHA-3

如上所述，这种算法以前被称为keccak，并被以太坊使用。它是在non-NSA设计师的公开竞赛之后创建的。SHA-3使用“海绵（sponge）机制”。

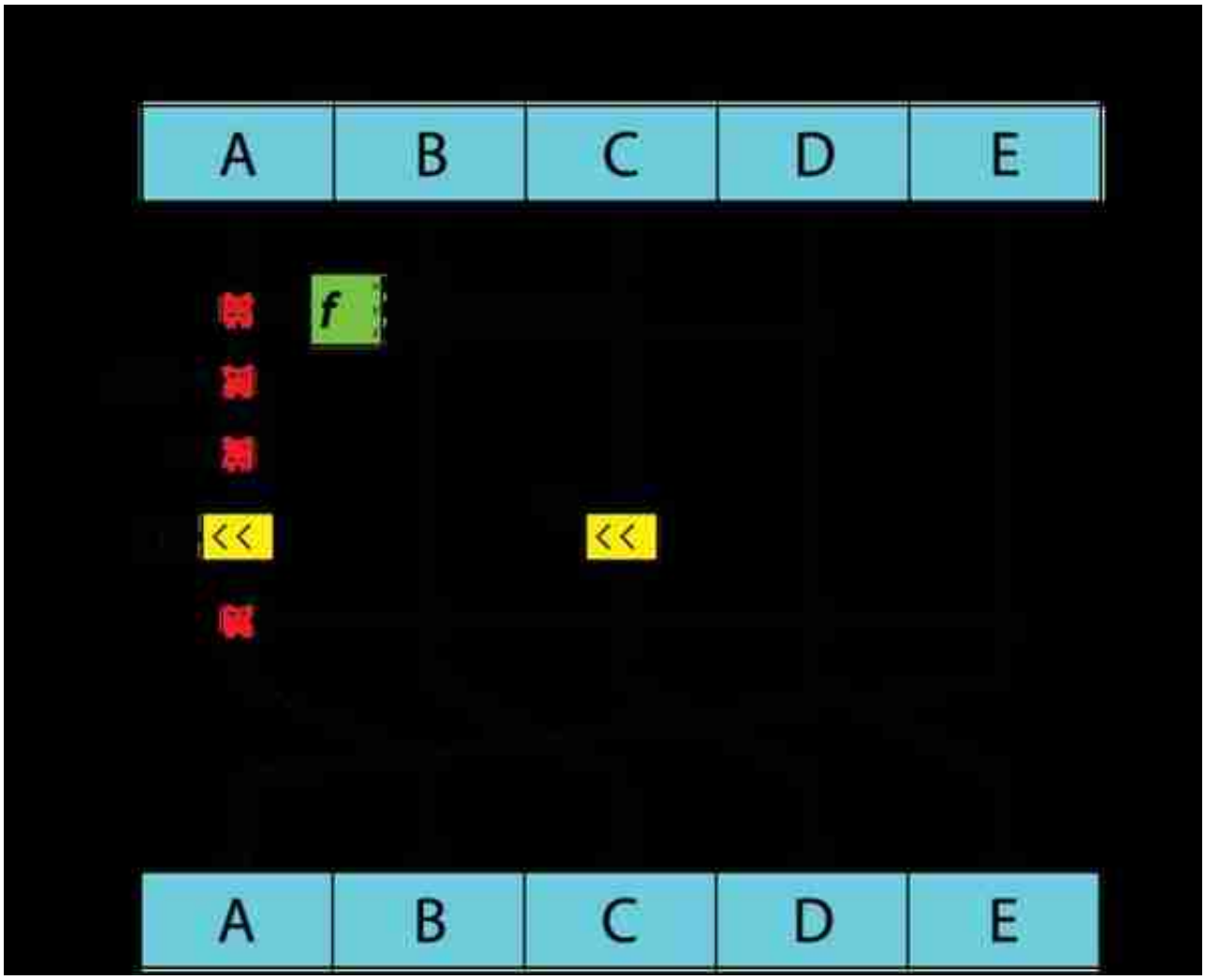
什么是海绵机制？

海绵功能是具有有限内部状态的一类算法，其获取任意长度的输入比特流并产生预定长度的输出比特流。

在继续之前，我们需要定义一些术语：

我们知道，在海绵功能中，数据被“吸收”到海绵中，然后将结果“挤出”。

所以有一个“吸收”阶段和一个“挤压”阶段。



RIPEMD-160

上面的图像显示了RIPEMD-160哈希算法压缩函数的子块快照。

RIPEMD -160实例：

输入：Hi

输出：242485ab6bfd3502bcb3442ea2e211687b8e4d89

CryptoNight哈希函数

现在我们拥有由Monero使用的CryptoNight哈希函数。与比特币不同，Monero希望他们的采矿尽可能地不利于GPU。他们可以做到这一点的唯一方法是让他们的哈希算法难以记忆。

输入CryptoNight

CryptoNight是一种内存硬件哈希函数。它被设计为在GPU，FPGA和ASIC体系结构上无法有效计算。CryptoNight的工作原理如下：

- 该算法首先用伪随机数据初始化一个大暂存器。
- 之后，大量的读/写操作发生在伪随机地址上
- 包含在便签簿中。
- 最后，整个暂存器被哈希以产生最终值。