
Cryptography behind Blockchain - Part 2

(Collision Resistant Hash Functions)

Ragesh Jaiswal, Department of Computer Science and Engineering, IIT Delhi

Blockchain is a decentralized consensus-based method for maintaining a public ledger that is **immutable** and contains authenticated records. In these notes, we discuss the core cryptographic primitives at the heart of blockchain technology. Part 1 of this series discussed public key cryptography (PKC), the main idea behind authenticated records. In this part, we will discuss **Collision Resistant Hash Functions** that help to achieve immutability.

Suppose you had a stroke of genius one fine morning, and you have a million-dollar idea $\mathcal{I}$ ¹ that you would like to announce to the world and simultaneously maintain ownership over the idea. You are now in a dilemma since the requirements are conflicting. By not telling anyone, you can maintain full ownership over $\mathcal{I}$. On the other hand, if you announce it to the world, you can lose ownership since someone else can claim to have come up with the idea $\mathcal{I}$. This is an extremely practical problem, and we need an effective solution to encourage innovation in society. The typical solution employed for this problem is a centralized one. A trusted party (in most cases, this is the government agency) makes sure that both requirements of the innovator are met. So, what you would do is reveal to the trusted party (through a *patent* application) your million-dollar idea $\mathcal{I}$. The trusted party checks if your idea does not conflict with an earlier idea and then promises to ensure that your ownership over the idea will not be violated. On obtaining this promise, you can now announce $\mathcal{I}$ to the world and earn from using your idea through *royalties*. Now imagine a situation where there is no central trusted party. How do we achieve the same goals without a central trusted authority (the

blockchain story's main premise)?

Let us try a few simple approaches for a decentralized solution. We shall incrementally build towards a solution by pointing out issues with the previous ones. Suppose every new idea is published in one of the newspapers that anyone can refer to. So, the proposed decentralized solution is for you to publish your million-dollar idea $\mathcal{I}$ in one of the newspapers. Of course, your idea $\mathcal{I}$ is published along with your identity information so that ownership gets recorded along with the idea. Any conflicts can be resolved by just comparing who published the idea first. Since one can publish in any one of the hundreds of newspapers with decent enough circulation², this seems like a decentralized solution that eliminates the central authority. So, are we done? A bit of thinking reveals that even though the proposed solution does not require a central authority, it requires a “trusted” authority. This is the editor of the newspaper that you send your idea $\mathcal{I}$ to. You are trusting the editor not to act maliciously in publishing your idea. The editor can act maliciously in so many different ways. He can choose not to publish your idea or publish your idea with a different name or publish your idea in a different newspaper with his name and then publish your idea in his paper the next day (which kills your ownership). Getting rid of the trusted authority will require some more ideas, and we will defer the discussion for some time. For now, let us assume that the newspaper's editor is trusted. However, publishing involves people other than the editor, and we cannot assume everyone at the newspaper publishing company is trusted. So, when you send the communication to the editor at his email address containing idea $\mathcal{I}$, most likely, the secretary will see the email body, and your idea might be stolen.

Clearly, sending $\mathcal{I}$ in the clear³ to the newspaper

¹ $\mathcal{I}$ can be interpreted as a string in some language

²Large circulation translates to a large number of witnesses.

³In Cryptography literature, we often use “in the clear” to

is a bad idea in the previous proposal. So, let the new proposal be to send an encrypted version of $\mathcal{I}$. We can do this using PKC that we discussed in part 1. So, you would send $C \leftarrow \text{Encrypt}(\mathcal{I}, pk)$ in the newspaper which is the encryption of your idea $\mathcal{I}$ using your public key pk . Once C gets published, you announce $\mathcal{I}$ to the world. Anyone can verify your ownership by checking that $C = \text{Encrypt}(\mathcal{I}, pk)$. So, are we done now? Not yet. One issue is that since only your public key is used in the communication, anyone can publish any arbitrary idea in your name and tarnish your reputation. This is because encryption is only meant for secret communication and does not provide authenticated communication. Fixing this is simple and is a standard idea in secure communication. What you will do is send C and signature $s \leftarrow \text{Sign}(C, sk)$ to the newspaper. It is infeasible for a malicious person to forge your signature. Once (C, s) gets published, at an appropriate later time, you can announce $\mathcal{I}$ to the world. So, are we done now? Unfortunately, there are still a few issues remaining. One issue is that $\mathcal{I}$ might be a very long string (such as a long computer program), and hence C would also be very long, and publishing C will be extremely costly⁴. Another issue is a bit subtle but very important. The basic idea behind publishing in the newspaper is known in cryptography as *commitment*. There are two important aspects of cryptographic commitment:

- *Hiding*: The commitment does not reveal the information to which the commitment has been made until revealed.
- *Binding*: The information to which the commitment has been made cannot be changed once the commitment has been made.

The above discussion only focussed on the hiding property. The binding property is important because of the following reason. A malicious person may publish (C, s) , where $C \leftarrow \text{Encrypt}(\mathcal{I}, pk)$ and $s \leftarrow \text{Sign}(C, sk)$ and then while revealing may declare an idea $\mathcal{I}'$ such that $\text{Encrypt}(\mathcal{I}', pk) = C$. So, this person can change the idea after the commitment has been published, which is very dangerous. Note that the encryption/decryption scheme within PKC does not provide any guarantees against this kind of attack.

The above discussion suggests that the published commitment should be such that the idea cannot be changed after the commitment has been made. How

mean not encrypted.

⁴Imagine seeing page after page of encrypted text in your subscribed newspaper

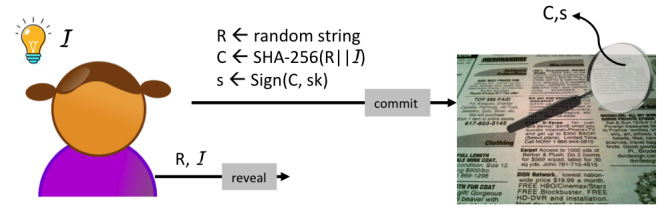


Figure 2: Protocol for publishing idea. ($||$ denotes concatenation)

can this be achieved? Suppose we have an easy-to-compute function h that maps arbitrary large strings to strings of fixed size (say 256 bits) and suppose this is a function that is hard to invert. That is, if you are given $h(s)$, for a randomly chosen string s , then it is computationally hard to figure out an s' such that $h(s') = h(s)$. In other words, h acts like a *one-way function*. We observe one-way behavior in so many things in everyday life, such as the breaking of an egg. So, we might not find it hard to believe that one-way functions exist. Now, given such a one-way function h , what we can do is commit to $h(\mathcal{I})$ instead of $\mathcal{I}$. Note that this is hiding and binding at the same time (since it is hard to find an $\mathcal{I}'$ such that $h(\mathcal{I}') = h(\mathcal{I})$).⁵ Moreover, since $h(\mathcal{I})$ is a small length string (256 bits), it will be easier to print in the newspaper. Figure 2 gives the protocol for secure publishing of idea $\mathcal{I}$. So, the final piece of the puzzle is to figure out whether such hard-to-invert functions exist. This leads us naturally to Collision Resistant Hash Functions that are discussed in the next section. You might have read about SHA-256 hash function in the context of blockchains. This is simply one such hard-to-invert function.

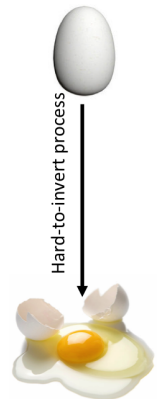


Figure 1: One-way process

Collision Resistant Hash Functions (CRHF)

In fact, SHA-256 satisfies a property stronger than the hard-to-invert property. This property is called *Collision Resistance* (CR). The basic idea is that it is hard to find two strings s and s' such that $\text{SHA-256}(s) = \text{SHA-256}(s')$. At first, it seems hard to believe.

⁵The final protocol will involve a few more details that deal with the fact that $\mathcal{I}$ may not be a random string. However, the basic idea is to use a function that is hard to invert. These are shown in Figure 2.

Since the function maps arbitrary long strings to 256-bit strings, there naturally exist *collisions* (that is, two strings that map to the same 256-bit string). However, one should realize that number of possible 256-bit strings is 2^{256} , which is unimaginably large. It is an open challenge to find two strings that map to the same 256-bit string, and no one has succeeded in this challenge. So, there is a reason to believe in the collision resistance property of **SHA-256**. Note that collision resistance is stronger than hard-to-invert property since if a function can be inverted, then a collision can be found.

It has been observed that collision-resistant hash functions such as **SHA-256** mimic the behavior of random functions. One property of a random function f that maps the set of strings to a set of 256-bit strings is that for every new input string, the random function maps it to a random 256-bit string. This means that the least significant bit (LSB) of $f(s)$ for every second string is 1 (or 0). Extending this notion, we get that one needs to try 2^{10} strings on the average (and hence spend some minimum amount of time) to be able to find a string s such that the ten least significant bits of $f(s)$ is a fixed 10-bit string. This is an extremely useful property that can be used as *Proof of Work* (POW). This is very useful in many scenarios and, in particular, in blockchains. We discuss proof of work in the next section.

Proof of Work (POW)

The idea of attaching proof of computational work in an electronic communication is not new and has been there since the early 90s [Dwork and Naor, 1992]. Here is a scenario that will motivate the concept. Spam emails are a nuisance. Even the most advanced filters do not resolve the issue completely. How do we solve this issue if we do not improve the filtering technology? Let us attack the problem at the very source. Who is generating the spam emails, and what are the costs involved in generating spam? The spam generation business is extremely cheap. As a spammer, all you need to do is to rent a few machines (which are available cheaply over the web) and run code on these machines that send billions of spam emails. What if every email needs to contain some proof of computational work? The email client rejects emails if such proof is not attached. This ensures that a minimum number of CPU cycles are employed to process valid emails. This does not cause a problem for regular users who generate only a small number

of daily emails. On the other hand, this disrupts the spammers' business because they now have to spend a lot more since generating a vast number of emails requires a considerable amount of computational power.

So, let us now work out the details of proof of work in the context of spam emails. Suppose Alice wants to send Bob an email with the text M . What could be a valid proof of work? Using the ideas in the previous section, the protocol can require the sender Alice to attach a string s (of some fixed size) such that the 10 least significant bits of **SHA-256**($s||M$) is a fixed 10-bit string t .⁶ Alice would have to try around 2^{10} strings before finding an appropriate one that she can attach. So, s indeed captures proof of work. Does this protocol solve the spamming problem, though? Not in its current form. The spammer usually sends the same email M to multiple email users, and once the spammer has found an appropriate string s for M , it just attaches the same s in all spam emails with message M . This is the typical *replay* attack. This attack can be thwarted by a three-round protocol where the sender, Alice, first sends interest of sending an email to Bob, who replies with a *nonce* (a fresh randomly chosen string) N . Alice then needs to generate a proof of work with respect to the string N and attach this with her email.⁷ See Figure 3 for an illustration of the email protocol.

Conclusion (part 2)

In this section, we considered collision-resistant hash functions (CRHF), an extremely important cryptographic primitive. We saw how the collision resistance property could be exploited to build commitment protocols, and its random function-like property can be used for proof of work. Commitment and proof of work are core ideas in blockchain technology. We have a good handle on these concepts using CRHFs such as **SHA-256**. We can now try to use the core ingredients developed in the first two parts, PKC and CRHF, to design a public ledger that is immutable and contains authenticated records. The public ledger we will design will be immutable in two ways, and CRHF will play a crucial role in both. First, proof of work will be utilized to ensure that the distributed consensus is reached on a block of entries in the ledger. Once the consensus has been reached, this block becomes immutable and

⁶Here $||$ denotes a simple concatenation or append operation.

⁷The nonce can be saved for future communication to reduce the rounds of communication

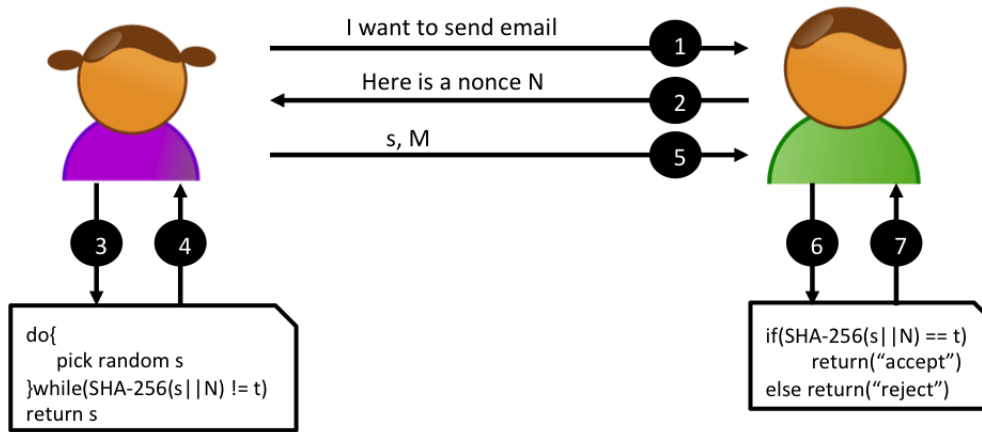


Figure 3: Solving the spam email problem using proof of work.

cannot be changed. Second, many entries in the ledger may not contain data but only commitments to the data (much smaller than the data). From the discussion in this document, we know that once a commitment has been added to the public ledger, the corresponding data cannot be changed and hence becomes immutable. In the next part of the series, we will start building blockchains using the ingredients discussed so far.

This write-up gives a very concise coverage of concepts. The readers are advised to consult books on cryptography to get a detailed picture. References to two such books are given below.

References

- [Katz and Lindell, 2014] Jon Katz and Yehuda Lindell. Introduction to Modern Cryptography *Chapman and Hall/CRC*, Second edition, 2014.
- [Dwork and Naor, 1992] Dwork Cynthia, Naor Moni. Pricing via Processing or Combatting Junk Mail. In *Brickell E.F. (eds) Advances in Cryptology – CRYPTO’ 92. Lecture Notes in Computer Science, vol 740. Springer, Berlin, Heidelberg*, 1992.
- [Schneier, 1996] Bruce Schneier. Applied Cryptography. *John Wiley & Sons*, Second edition, 1996.