Blockchain: Notes and Links

Last updated: Apr 9 15:24:19 2019

Contents

1 The Blockchain Concept

A **blockchain** can be defined as a public distributed ledger of transactions, where the order of the transactions is an essential property.

- **Dynamic:** The blockchain grows with time.
- **Block Structure:** For efficiency reasons, transactions are grouped in blocks.
- **(Almost) Linear:** The blockchain tends to grow as a linear structure, that is, a linked list of blocks.
- **Consensuated:** The blockchain contents and structure is consensuated among the blockchain workers.

(Generated by lineprocx)

- **Incentivated:** The workers (also known as "miners") collaborate in the creation and maintenance of the blockchain because of an incentivation system.
- **Replicated:** The blockchain contents (or the essential part of it) is replicated in every worker's local strorage.
- **Open Design:** Anyone can act as a blockchain worker, and the whole blockchain is public and publicly verifiable. In addition, no special structure is required in the communication network.

2 Linear Structure

The linear structure of the blockchain is the result of the best strategy for miners:

- The task of the miners is to pack recent transactions into a new block and to link it to the blockchain.
- All the miners are in competition, and the one who comes up first with the new block is who is entitled to receive a reward. All the other miners' effort to build this block is lost.
- A successful miner will receive its reward only after its block is deep enough in the blockchain.
- When there is a "fork", that is, different (and perhaps dishonest) miners try to make the blockchain grow in different directions, only the one that costed most work will be finally accepted.
- The best strategy for a miner is to link any new block to the longest path (actually, to the path with most accumulated work), also called the main path. The blocks outside the main path will be discarded, and the corresponding rewards are destroyed.
- A honest majority of active miners (actually, a honest majority of computational power) guarantees that the blockchain grows as a linear structure, and occasional forks are soon discarded.
- A transaction is considered as irreversible if it is registered in a deep enough block.
- Only valid blocks are accepted as part of the blockchain. A valid block can only contain acceptable transactions not contained in any other block.

3 Proofs of Work

The previous considerations are only valid if mining a block involves a nontrivial amount of computational effort. Then, every block contains a "proof of work" related to its contents and to the block in the blockchain that will be its parent. This proof of work is part of the block validation.

Hash puzzles are often used in blockchains in their proof-of-work system. The first hash-based puzzles were proposed in 1993 as a countermeasure against e-mail spam.

The preimage resistance property of a hash function can be used to define families of computational problems parameterized by the computational cost they require. This cost ranges from the very easy problems to the infeasible ones. The simplest example is the following problem:

Given a hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$ and a difficulty parameter $t \in \{1, ..., \lambda\}$, find m such that $H(m)$ has (at least) t leading zeros.

Assuming that H behaves as a random function, the expected number of trials (hash evaluations) to get a valid solution is 2^t, which is nowadays considered as an infeasible task if, for instance, $t = 128$.

The problem can be modified in a way that it is required to include some particular information as a prefix of m, like a timestamp or some predefined data, in order to prevent reusing previously known solutions.

In addition, to have a finer control of the difficulty of the puzzle, one can use an arithmetic comparison of the result of the hash computation instead of forcing a number of zeros:

Given a hash function $H : \{0,1\}^* \to \{0,...,2^\lambda - 1\}$ and a difficulty parameter $\Delta \in \{1,...,2^\lambda\}$, find m such that $H(m) < \Delta$. Then, the success probability, assuming the random behavior of the hash function, is now $\Delta/2^\lambda$.

In this case, the expected number of trials to find a solution is $2^\lambda/\Delta$.

In the blockchain, the hash puzzle involves the computation of the hash of the new block, which contains a timestamp, the hash of its future parent block (already contained in the blockchain) and the transactions included in the new block.

`https://en.wikipedia.org/wiki/Hashcash` (Hashcash puzzles at Wikipedia)
`https://patents.google.com/patent/US7197639` (Client Puzzles RSA patent)

4 Mathematical Model

To give a mathematical framework to evaluate the proof of work based on hash functions, we first recall some concepts about the security of a hash function.

4.1 Hash Functions

A **hash function** is a deterministic and efficiently computable function $H : \{0,1\}^* \to \{0,1\}^m$, that maps binary strings of arbitrary length to binary strings of a fixed length m (or to elements in a finite set), behaving like a random function.

A deterministic function cannot behave exactly as a random function, but at least the following properties are required to any hash function:

Preimage resistance. Given a random element $y \in \{0,1\}^m$, it is infeasible to compute $x \in \{0,1\}^*$ such that $H(x) = y$.

Second preimage resistance. Given a random element $x \in \{0,1\}^*$, it is infeasible to compute $x' \in \{0,1\}^*$ such that $H(x') = H(x)$ and $x' \neq x$.

Collision resistance. It is infeasible to come up with a **collision** pair $x, x' \in \{0,1\}^*$ such that $H(x) = H(x')$ and $x' \neq x$.

In a random function, for any different $x_1, ..., x_n \in \{0,1\}^*$, the images $H(x_1), ..., H(x_n)$ are independent random variables, uniformly distributed on $\{0,1\}^m$. Therefore, one expects to compute 2^m hash values to hit a given $y \in \{0,1\}^m$, and there is no difference between the first and second preimage resistance properties. However, only $2^{m/2}$ hash computations would be needed on average to find a collision (birthday paradox).

`https://en.wikipedia.org/wiki/Birthday_problem` (Birthday Paradox at Wikipedia)

Therefore, any cryptographic hash function will use a value of m large enough so that computing $2^{m/2}$ hash values is considered an infeasible task.

4.2 Hash Work Aggregation

In the blockchain scenario, several entities are in competition trying to solve hash puzzles, and even one entity can consist of a set of collaborating computing devices working in parallel.

Assuming that a device is computing hash values at a constant rate (say ρ hashes per second), the necessary time to find a solution to a (λ, Δ)-hash puzzle is $T = N/\rho$, where N is the number of hash values computed. N behaves as a **geometric** random variable of parameter $\Delta/2^\lambda$, and it can also be seen as the ceiling of an exponential random variable with the same parameter (i.e., with expected value $2^\lambda/\Delta$). Therefore, the time to find a solution $T = N/\rho$ can be approximated by an **exponential random variable** with parameter

$\mu = \rho\Delta/2^\lambda$. Hence, the probability that a solution to the hash puzzle if found before a given time t can be approximated by

$\Pr[T \leq t] = 1 - e^{-\mu t}$.

If the device works in an endless loop computing new solutions, the time instants in which the solutions are found behave as (a discretized version of) a **Poisson stochastic process**.

If several devices are working in parallel, the success time instants are aggregated. By the properties of Poisson stochastic processes, the aggregation of two (or more) independent Poisson processes is also a Poisson process. As a consequence, a set of devices is equivalent to a single and more powerful device, with a hash rate equal to the sum of the hash rates of the individual devices.

In addition, if two computing clusters with hash rates ρ_1 and ρ_2 are in competition solving hash puzzles, the probability that the first cluster reaches a solution before the second cluster is

$\Pr[T_1 < T_2] = \mu_2/(\mu_1 + \mu_2) = \rho_2/(\rho_1 + \rho_2)$

where $\mu_1 = \rho_1\Delta/2^\lambda$ and $\mu_2 = \rho_2\Delta/2^\lambda$ are the solution finding rates of the two clusters.

Other probabilities can also be explicitly computed (e.g., that the cluster 1 finds k_1 solutions before the cluster 2 finds k_2, for arbitrary k_1 and k_2) but the mathematical expressions are not so simple.

For instance, the probability law of the advantage of cluster 1 with respect to cluster 2, that is the difference $S_1(t) - S_2(t)$ between the number of solutions found by the two clusters, is given by

$\Pr[S_1(t) - S_2(t) = k] = (\mu_1/\mu_2)^{k/2} e^{-(\mu_1+\mu_2)t} I_k(2t\mu)$,

where μ is the geometric mean of μ_1 and μ_2, and $I_k()$ is the modified first order Bessel function.

However, the expected value of the advantage is given by the simple expression

$\mathrm{E}[S_1(t) - S_2(t)] = (\mu_1 - \mu_2)t$.

4.3 Sequential Work

In the blockchain, the miners work in competition to build new blocks and link them to the main chain. Ideally, everytime a new block is appended to the blockchain, the miners have to restart their work in order to solve the new hash puzzle associated to this new block in the chain. The reason of it is because every new valid block must contain in its header the hash of its (future) parent block in the chain. Therefore, changing the parent implies changing the block header, which in turn changes the hash puzzle associated to the new block.

As a consequence, in the two competing mining clusters scenario, once one of them (say cluster 1) finds a solution of the current hash puzzle and successfully appends a new block to the chain, all the computation efforts done by cluster 2 are discarded.

The game between the two clusters can be seen as the indefinite repetition of a basic step consisting of both clusters trying to find a solution of the current hash puzzle. All the basic steps are independent identical random experiments. Therefore, the number of new blocks appended to the blockchain by cluster 1 among that last n blocks appended by both clusters follows the **binomial** probability distribution.

Thus, in the honest behavior the fraction of blocks appended by a cluster is proportional to its hash rate. But a dishonest cluster could try to force a fork in the blockchain in order to defeat the other clusters, making their blocks being finally discarded. This means building a secondary chain that at some time becomes longer than the main chain.

The success probability of this attack can be evaluated with the above expression for $\Pr[S_1(t) - S_2(t) = k]$, where k is the initial disadvantage of the malicious cluster (Cluster 1) with respect to the honest cluster (Cluster 2).

Using proofs of work as the way to control the structure of the blockchain has the drawback of wasting energy and computing resources in useless computations. Some alternatives have been proposed to make the blockchains more environmental-friendly.

`https://chia.net/faq/index.html` (Chia proof of space)

5 Block Structure

In the Bitcoin blockchain, a block basically contains the following information:

Block Header:
- Hash of the parent block
- Hash of the transaction set
- Timestamp
- Difficulty level
- Nonce

Set of Transactions:
- Number of transactions
- Coinbase transaction
- Transaction 1
- ...
- Transaction n

`https://en.bitcoin.it/wiki/Block` (Bitcoin block structure)

The hash of a block is computed as the hash function applies only to the block header. A correct block has a hash value that is less than its difficulty level (that is, the block header itself is a solution to a hash puzzle). In order to solve the hash puzzle, the **nonce** in the block header is incremented every time until a solution is reached. If necessary, between iterations, the timestamp can be updated and also new transactions can be added to the block (and in this case, the hash of the set of transactions is recomputed).

If the block nonce wraps around to zero, there is an "extra nonce" field embedded into the first block transaction (also known as the "coinbase") that is incremented. This also requires the hash of the transaction set to be recomputed.

For efficiency reasons, the transactions are organized as a hashed binary tree, called a Merkle Tree.

6 Merkle Trees

A Merkle Tree is a tree of hash values that allows to authenticate a collection of objects with a single hash value (the root of the tree), and has only logarithmic complexity in the verification that a given object belongs to the collection.

The objects in the collection are arranged as the leaves of a binary tree. Each node in the tree is associated to a binary string, indicating the path from the root to the node ("0" = left, "1" = right). The hash values of the nodes are computed as follows:

$h_\emptyset = H(1, h_{00}, h_{01})$, for the root node
$h_x = H(1, h_{x0}, h_{x1})$, for any intermediate node x,
$h_x = H(0, m_x)$, for any leaf,
where m_x is the object at leaf node x.

The verification that an object m_x is at node x of a tree with root hash $h_\emptyset$, only requires to check the hash equations along the path to x. Therefore, a proof of the above fact consists of the hashes h_y for all nodes y that are siblings of the nodes of the path to x.

For instance, a proof that m belongs to the collection with root hash $h_\emptyset$ and it is at node $x = 10010$, consists of the hashes $(h_0, h_{11}, h_{101}, h_{1000}, h_{10011})$. The proof verification consists of checking the system of equations:

$h_{10010} = H(0, m)$

$h_{1001} = H(1, h_{10010}, h_{10011})$
$h_{100} = H(1, h_{1000}, h_{1001})$
$h_{10} = H(1, h_{100}, h_{101})$
$h_1 = H(1, h_{10}, h_{11})$
$h_\emptyset = H(1, h_0, h_1)$

The length of the proof and the number of hash evaluations performed in the verification depend logarithmically on the size of the collection.

New objects can be added to a Merkle tree, which requires to recompute some of the hash values. If we assume that all the leaves are at the same depth, then some intermediate nodes can have an empty siebling (that is, only the left siebling exists). In this case, the corresponding hash is computed as follows:

$h_x = H(1, h_{x0}, h_{x0})$.

If the tree is full, the insertion of a new object is done by creating a new root, then the old tree is put as its left sub-tree, and its right sub-tree consists only of a linear path to the new leave containing the inserted object.

Every insertion of an object at depth d implies the computation of d hashes, which depends logarithmically on the total number of stored objects.

A more efficient hashed tree is used in Ethereum block chain:

`https://github.com/ethereum/wiki/wiki/Patricia-Tree` (Ethereum Merkle Patricia Trie)

7 Transactions

A transaction is the atomic information stored into a blockchain. Transactions are linked, conforming a directed acyclic graph. The graph nodes are the transactions and the graph links or arcs are the linls between blockchain transactions.

Every transaction contains a list of inputs and outputs, and every transaction input comes from other transaction's output, conforming a transaction link.

In a cryptocurrency blockchain, nodes (transactions) and links have positive weights (or values) that follow some rules:

- The weight of a node is the sum of the weights of its inputs.
- The sum of the weights of the outputs of a node cannot exceed the node's weight.

Some node outputs can be free (i.e., they are still not used as inputs of other nodes), and they are known as **unspent transaction outputs**. Some other nodes are sources (that is, they have no inputs) and they correspond to the creation of cryptocurrency units (like the coinbase transaction contained in every Bitcoin block).

Thus, a transaction represents a transitory good ownership, and the links between transactions are ownership transfers. The multiple inputs and outputs reflect the aggregatability and the divisibility of the goods (e.g., the cryptocurrency).

As a consequence, there would be a secured (cryptographical) way to connect a new transaction to an existing one, fulfilling some requirements:

- Only the legitimate user can spend a transaction output (i.e., to identify a new transaction's input to the owned transaction output).
- The validity of the link and the transaction itself must be publicly verifiable, using only the information stored in the blockchain.

To that end, every transaction output has an **address**, and every transaction input has a reference to a transaction output and a proof that it matches the corresponding address. In bitcoin transactions, the transaction inputs and outputs essentially consist of:

Transaction Output:
Transferred value (the weight of the graph link)
Public Key Script (the definition of the required owner's credentials)

Transaction Input:
Hash of the source transaction
Index of the output in the source transaction (both fields are a reference to the linked transaction output)
Signature Script (the credentials showed by the referenced output's owner)

`https://en.bitcoin.it/wiki/Transaction` (Bitcoin transactions structure)

The Public Key Script (PKScript) contains the output's intended address, and the Signature Script (SigScript) contains the "proof of ownership", that is, the necessary credentials or evidence that match an address given in other transaction's output.

In a simple Bitcoin transaction input/output matching (which is the actual cash transfer mechanism), PKScript contains the hash value of the recipient's public key, while SigScript consists of the recipient's public key and a signature of the transaction body (verifiable with that public key). In a more complex transaction input/output matching, PKScript and SigScript contain data and programs that, once concatenated and executed, result in a bit, indicating whether or not the two pieces of information match.

8 Elliptic Curve Based Signatures

The previous transaction input/output matching verification mechanism is based (in the simplest case) on a digital signature scheme. A digital signature provides at once message integrity, authentication and non-repudiation by securely combining a secret key with the message, and making the signature verification publicly available.

8.1 Digital Signatures

A digital signature scheme consists of three algorithms:

- A **Key Generation** algorithm, KeyGen, that produces a random key pair (pk, sk) of a suitable size.
- An **Signature** algorithm, Sig, that given a secret key sk and a message m, it produces a signature s, possibly using some randomness.
- A **Verification** algorithm, Ver, that given a public key pk, a message m and a signature s, it outputs 1 if s is a valid signature for pk and m, or 0 otherwise.

When a user A wants to be able to sign messages for other users, she produces a key pair (pk, sk) with KeyGen. Then she publishes pk and keeps sk secret as her long term key.

Now A can compute a digital signature for a message $m \in M_{pk}$ by computing $s = \mathrm{Sig}(sk, m)$.

Any other user B can verify the signature s on m with A's public key by checking whether $\mathrm{Ver}(pk, m, s) = 1$.

Security of a signature scheme refers to the hardness of generating a valid signature without knowing the secret key in any realistic attack scenario, like knowing some valid pairs message/signature from the target signer.

A signature can authenticate some extra information like place, time and purpose of the signature, as in conventional handwritten signatures, by appending in an unambiguous way the extra information to the original message.

Dealing with arbitrary length documents typically imply the use of a cryptographically secure hash function. Indeed, the chosen hash function is part of the specification of the signature scheme, and what is essentially signed is not the document itself but its hash value. This technique is referred to as the "hash-and-sign" paradigm.

Elliptic curve based signature schemes are efficient and compact. Therefore, they are preferred in applications where saving storage space is important.

8.2 Elliptic Curves

The set of points (x, y) fulfilling the (non-singular) cubic equation $y^2 = x^3 + ax + b$ along with the so-called "point at infinity" O form a group with the "tangent and chord" geometric operation. It can be shown that any straight line L intersects the elliptic curve at exactly tree points (counting multiplicity and the point at the infinity). In particular, a vertical line intersects the curve at two symmetric points $P = (x, y)$ and $-P = (x, -y)$. The third intersection point is the point at infinity O. In addition, if the line is tangent at a point P, then it counts as a double intersection, and there exists a second intersection point in the curve. There could be a unique intersection point with multiplicity 3.

Then, given two points P, Q in the curve such that Q is not P or $-P$, the straight line passing through P and Q intersects the curve at a third point R (technically, it can be $R = P$ or $R = Q$ if the line in tangent at P or at Q, or even $P = Q = R$ is the intersection point has multiplicity 3). The point $P + Q$ is defined to be the symmetric of R. That is, $P + Q = -R$. The same construction is used to define $2P = P + P$, but in this case the tangent line at P is used instead. The point at infinity is taken as the neutral element, and then $P + O = P$, $P + (-P) = O$ and $O + O = O$.

The equations of the sum of two points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ for $x_P \neq x_Q$ are:

$m = (y_P - y_Q)/(x_P - x_Q)$; $\quad x_{P+Q} = m^2 - x_P - x_Q$; $\quad y_{P+Q} = -m(x_{P+Q} - x_P) - y_P$.

The equations of $2P = P + P$ for a point $P = (x_P, y_P)$ and $y_P \neq 0$ are:

$m = (x_P^2 + a)/(2y_P)$; $\quad x_{2P} = m^2 - 2x_P$; $\quad y_{2P} = -m(x_{2P} - x_P) - y_P$.

If $y_P = 0$ then $-P = P$ and $2P = P - P = O$.

A scalar multiple of a point can be efficiently computed via doubles and additions, based on the binary representation of the scalar, like in the following example:

$269P = (256 + 8 + 4 + 1)P = 2(2(2(2(2(2(2P)))) + P) + P)) + P$.

In general, the previous computation would require a logarithmic number of additions and doublings.

In cryptographic applications, the elliptic curve is defined over a large finite field, and the group operation is defined by the same equations given above, but with the addition, multiplication and inversion defined in the finite field.

For instance, if the finite field is a prime field of order p, then its elements are $\{0, 1, 2, ..., p - 1\}$ and addition and multiplication are performed modulo p, that is, the result of any addition or multiplication is replaced by the reminder of the division of the result by p. Inverses modulo p are computed with a special algorithm (typically, the generalized GCD Euclid algorithm).

On the other hand, if the finite field has characteristic 2, then its elements are represented as polynomials of certain limited degree with binary coefficients, and addition and multiplication are performed modulo a special (irreducible) polynomial. However, the theory of elliptic curves in characteristic 2 is slightly different. Indeed, the elliptic curve equation has one of the reduced forms $y^2 + xy = x^3 + ax^2 + b$ or $y^2 + ay = x^3 + bx + c$, and the formulae for $-P$, $P + Q$ and $2P$ differ form the above.

8.3 ECDSA

The basic signature scheme on elliptic curves is ECDSA.

KeyGen(λ):

Choose a prime number p of λ_1 bits (typically λ_1 goes from λ to 2λ).
Choose an elliptic curve $E_p(a,b)$ and a base point B of prime order q, where q has λ bits.
Choose a secure hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$.
Choose a random $\alpha \in Z_q$ and compute the multiple $Y = \alpha B$.
Output $(pk, sk) = ((param, Y), \alpha)$, where param $= (p, E_p(a,b), q, B, H)$.

Sig(pk, sk, m):
Parse $pk = (\text{param}, Y)$.
Parse $sk = (\alpha)$.
Choose a random $k \in Z_q^\times$ and compute $r = (kB)_x \bmod q$.
If $r = 0$, then repeat the procedure by choosing a new random k.
Compute $t = (H(m) + \alpha r)k^{-1} \bmod q$.
If $t = 0$, then repeat the procedure by choosing a new random k.
Output $s = (r, t)$.

Ver(pk, m, s):
Parse $pk = (\text{param}, Y)$.
Parse $s = (r, t)$.
If $r \notin \{1, ..., q-1\}$ or $t \notin \{1, ..., q-1\}$, then output 0.
Compute $U = (t^{-1} \bmod q)(H(m)B + rY)$.
Output 1 if $r \equiv U_x (\bmod\ q)$, and 0 otherwise.

`https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm` (ECDSA signature at Wikipedia)

The correctness of the ECDSA scheme can be easily shown:

$t^{-1} \bmod q = k(H(m) + \alpha r)^{-1} \bmod q$.
$H(m)B + rY = (H(m) + \alpha r)B$.
$U = (t^{-1} \bmod q)(H(m)B + rY) = kB$.
$U_x \bmod q = (kB)_x \bmod q = r$.

8.4 The Certicom secp256k1 Elliptic Curve

The selected signature algorithm in the Bitcoin blockchain is ECDSA over a standarized elliptic curve called Certicom secp256k1.

`http://www.secg.org/sec2-v2.pdf` (Certicom recommended elliptic curves (pdf file))

The prime field has p elements, where the prime p has a binary length of 256 and is given by

$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^4 - 2^4 - 1$.

The elliptic curve is defined by the equation $y^2 = x^3 + b$, where $b = 7$. The group of (rational) points of this elliptic curve has a prime order q, where

$q = 2^{256} - 432420386565659656852420866394968145599$.

The standarized base point is $B = (x_B, y_B)$, where

$x_B = 55066263022277343669578718895168534326250603453777594175500187360389116729240$,
$y_B = 32670510020758816978083085130507043184471273380659243275938904335757337482424$.

For this elliptic curve, the equations of the sum of two points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ are:

- If $P = O$, then $P + Q = Q$.
- If $Q = O$, then $P + Q = P$.
- If $x_Q \neq x_P$, then:

$m = (y_P - y_Q)(x_P - x_Q)^{-1} \bmod q$;
$x_{P+Q} = m^2 - x_P - x_Q \bmod q$;
$y_{P+Q} = -m(x_{P+Q} - x_P) - y_P \bmod q$.

- If $x_Q = x_P$ and $y_Q = -y_P$, then $P + Q = O$.
- Otherwise, $x_Q = x_P$ and $y_Q = y_P \neq 0$, and then:
  $m = x_P^2(2y_P)^{-1} \bmod q$;
  $x_{2P} = m^2 - 2x_P \bmod q$;
  $y_{2P} = -m(x_{2P} - x_P) - y_P \bmod q$.

9 Hashing the Public Keys

Bitcoin blockchain achieves certain level of anonymity, by avoiding any reference to the users' identities. Indeed, only signature public/secret keypairs are involved in the generation and verification of the transactions stored in the blockchain. However, if the same public key is used for many transactions, they can be trivially linked to the same (anonymous) individual.

A way to increase the unlinkability of different transactions is the use on one-time signature keypairs: Every time a transaction is created, the intended recipient of every output creates a fresh keypair and she communicates to the sender the hash of its public key. This hash is the so-called recipient's address, and is part of the PKScript of the output.

A simplified version of the transaction is signed by the sender for every transaction input. Indeed, every input corresponds to a different address owned by the sender, and each address corresponds to a signature keypair. The public key and the signature of the simplified transaction are included in the transaction input's SigScript.

After a transaction is inserted into the blockchain, everybody learns that all addressed referred to by the transaction inputs correspond to the same owner, but nothing is known about the owners of the unspent transaction outputs, because each address is used only once.

Hashing the public key has two advantages:

- It saves space, because the hash value is shorter than the public key. In a more complex transaction input/output link the address is not the hash of a public key, but it is the hash of a script defining the way the ownership of the transaction output is going to be verified. Therefore, the hash is necessary to uniformize scripts of different lengths.
- Including the explicit public key in the still unspent transaction output would enable an attacker to forge a valid ECDSA signature (e.g., if she has access to a quantum computer). With only the hash of the public key, the previous attack is prevented, unless the hash function can be inverted (than is, the hash function is no longer preimage resistant).

10 Links and Documents

`https://en.wikipedia.org/wiki/Blockchain` (Blockchain at Wikipedia)
`https://en.wikipedia.org/wiki/Hashcash` (Hashcash puzzles at Wikipedia)
`https://patents.google.com/patent/US7197639` (Client Puzzles RSA patent)
`https://www.blockchain.com/` (Blockchain.com)
`http://www.goldmansachs.com/our-thinking/pages/blockchain/index.html?cid=sch-pd-google-blockchain-searc`
`mkwid=SvgCqExm` (GoldmanSachs presentation)
`https://blockgeeks.com/guides/what-is-blockchain-technology/` (Article at BlockGeeks)
`https://www.youtube.com/watch?v=r43LhSUUGTQ` (Introductory video at Youtube)
`https://blockchain.info/` (Bitcoin Block Explorer)

https://blockchain.info/api/blockchain_api (Blockchain Data API)
https://blockchain.info/rawblock/00000000000000000000960ed49f8b5b14943b67377a15bcaca76ce5675a0873 (Raw block example)
https://en.bitcoin.it/wiki/Main_Page (Bitcoin Wiki)
https://en.bitcoin.it/wiki/Block (Bitcoin block structure)
https://en.bitcoin.it/wiki/Transaction (Bitcoin transactions)
https://en.bitcoin.it/wiki/Confirmation (Transaction confirmation and double-spending)
https://en.bitcoin.it/wiki/Irreversible_Transactions#Attack_vectors (Double-spending attacks)
https://en.bitcoin.it/wiki/Address (Bitcoin addresses)
https://en.bitcoin.it/wiki/Protocol_documentation (Bitcoin protocol documentation)
https://en.bitcoin.it/wiki/Protocol_documentation#Merkle_Trees (Merkle trees)
https://en.bitcoin.it/wiki/Protocol_rules (Bitcoin protocol rules)
https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm (ECDSA signatures)
http://www.secg.org/sec2-v2.pdf (Certicom recommended elliptic curves: secp256k1 (pdf file))
https://bitcoin.org/bitcoin.pdf (Satoshi's bitcoin paper (pdf file))
https://en.wikipedia.org/wiki/Ethereum_Classic (Ethereum Classic at Wikipedia)
https://en.wikipedia.org/wiki/Ethereum (Ethereum at Wikipedia)
https://www.ethereum.org/ (Ethereum.org)
https://blockgeeks.com/guides/ethereum/ (Article at BlockGeeks)
https://github.com/ethereum/wiki/wiki (Ethereum Wiki)
https://github.com/ethereum/wiki/wiki/White-Paper (Ethereum white paper)
https://ethereum.github.io/yellowpaper/paper.pdf (Ethereum yellow paper (pdf file))
https://blog.ethereum.org/2016/12/05/zksnarks-in-a-nutshell/ (zkSNARKS in a nutshell)
https://www.etherchain.org/ (Ethereum Block Explorer)
https://www.etherchain.org/block/39656fcba26af27a6548ec5cbd06b15bb48c7f04a955c44fcd66dc395f88e5cd (Block example)
https://chia.net/faq/index.html (Chia FAQ)
https://eprint.iacr.org/2017/893.pdf (Asiacrypt 2017 paper (eprint version))
https://link.springer.com/chapter/10.1007/978-3-319-70697-9_13 (Asiacrypt 2017 paper (LNCS version))
https://www.youtube.com/watch?v=iqxkO7C-cyk (Asiacrypt 2017 presentation)
https://eprint.iacr.org/2018/183.pdf (Eurocrypt 2018 paper (eprint version))
