Zero-knowledge Proofs

Merkle Tree Based Range Proofs (MTRP)

Maximilian Powers
International School of Nice

(Internal Assessment: Mathematics Higher Level)

(Wordcount:)

(Dated: October 13, 2018)

Contents

I. Zero-Knowledge Proofs

A Zero-knowledge Proofs, or ZKP's are a recent development in mathematics, often used in cryptography, to prove something without giving any specific information. To illustrate this, imagine Alice needs to prove to Bob she's not blind, however the constraint is that it can only be done with a red and a green ball. Now Bob will ask Alice which hand the red ball is in. Alice will then give a correct answer, but she could of guessed, in this case $p = 1/2$. But by repeating this several times, and Alice guessing correctly every time, p gets very small, in fact by repeating 7 times Alice would have less 1 in 100 chance. This is called a Zero-knowledge Proof, as Alice never had to present any prior information to prove herself and she has made a statistically sound claim, as in p is too low to be acknowledge. A Zero-knowledge Argument, ZKA, is computationally sound, and while a Zero-Knowledge Proof may be a Zero-Knowledge Argument, the same cannot be said vice versa. In this Internal Assessment, a new proof, by B. B́unz et a.l. (2018), and an old proof, by G.Maxwell (2015) will be presented for a range proof, a way of showing a number is in between two others without revealing it, and then a modification will be added to both, as well as a new approach that could be potentially more efficient for large scale permanent distributed ledgers as found in Blockchain networks.

Informally, a ZKA involves two parties, the prover P (Alice) and the verifier V (Bob). The prover must prove to the verifier that a given statement is true without revealing the statement itself. Statements will be written as $\mathbf{x} \in L_R$, where L is a language in NP, the type of complexity problem ZKA's address, over R, a statement that can be verified in polynomial time, but cannot be solved in polynomial time such as Soduko. The set of hidden values is given as $w \in W(\mathbf{x})$. Now the statement can be written as $(\mathbf{x}, w) \in R$ where R is a "polynomial time decidable binary relation associated with L_R" Jonathan Bottle et a.l. (2016). The ZKA must then satisfy the following,

Completeness. If $(\mathbf{x}, w) \in R$, the prover P who knows witness w for $\mathbf{x}$ will prove convincingly to the verifier of his knowledge. Bob can decide how convinced he is with Alice's proof by repeating the process, therefore decreasing p until he is convinced.

Soundness. A prover can never prove a statement $\mathbf{x}$ when $\mathbf{x} \notin L$. It would be very hard for Alice to prove she's not blind if she were and it can be probabilistically ignored for large numbers of trials.

Zero-Knowledge. The interaction can only reveal $\mathbf{x} \in L$ and should not reveal any $w \in W(\mathbf{x})$. Alice does not need to present any medical information, eye examination results... in order to prove she's not blind.

II. Preliminaries

The output, y of an algorithm A will be written as $y = A(\mathbf{x}; r)$ where r is some randomness. $y \hookleftarrow S$ will also be used to sample y uniformly at random from the set S and this selection can be done from sets such as $\mathbb{Z}_p$. These algorithms will receive a security parameter λ, which for a scheme such as RSA denotes the maximum bit length. A function $f(\lambda)$ that is close to 0 will be taken as secure.

Throughout this paper, $\mathbb{G}$ will be a group of prime order p and $\mathbf{g}$ be a generator of that group such that $\mathbf{g} = (g_1, g_2, ..., g_n) \in \mathbb{G}^n$. A second group $\mathbb{Z}_p^n$ will also be used such that $\mathbf{f} = (f_1, f_2, ..., f_n) \in \mathbb{Z}_p^n$. Therefore $\mathbf{g}^{\mathbf{f}} = \Pi_{i=1}^{n} g_i^{f_i}$ which will allow for the use of multi-exponentiation.

i. The Discrete Logarithm.

The proof presented in this paper will revolve around the Discrete Logarithm NP-Problem, hence that is our language L. Given a prime p, a generator g, which must be a primitive root, of the finite cyclic multiplicative group $(G_p, *)$, hence $g \in G_p^n$, and an element $a \in G_p$. Find the integer x wherein $1 \leq x \leq p-1$ such that,

$$g^x \mod p \equiv a \tag{1}$$

The solution will distribute randomly for example given the group $(G_{17}, *)$, and 3 is a primitive root hence

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$3^x \mod 17$	3	9	10	13	5	15	11	16	14	8	7	4	12	2	6	1

Therefore for our ZKA,

Definition 1. Discrete Logarithm Assumption. The discrete logarithm assumption holds as a Zero-knowledge proof wherein $\mathbf{x} = \{\mathbf{g}, p, G\}$ and $w = x$.

Discrete Logarithm Protocol. A Sound, Complete, and Zero-Knowledge Argument.

```
1: $P$'s Input: $(\mathbf{g} \in G_p^n, p, G, x)$
2: $P$ computes:
       $r \xleftarrow{\$} \mathbb{Z}_p^n$
       $h = g^{r} \mod p$
3: $V$ computes:
       $\mathcal{A}(b) \xleftarrow{\$} \mathbb{Z}_p^n$
4: $P \rightarrow V : h$
5: $V \rightarrow P : b$
6: $P$ computes:
       $s = (r + bx) \mod (p-1)$
7: $P \rightarrow V : s$
8: $V$ computes:
       $g^{s} \mod p$
       $ha^{b} \mod p$
9: Output: $\{V$ accepts, $V$ rejects $\}$
```

Figure 1. In step 1 the prover retrieves the information for the group, where G is the group, $\mathbf{g}$ is the generator, p is a large prime, and x is the witness value that Alice, the prover, is trying to prove she knows, and then in step 2 the prover computes for some randomness r. In Step 3 Bob generates a random challenging integer $\mathcal{A}(b)$. He is then sent $h = g^{r} \mod p$ which is an element of the group G, and he sends back the challenging integer. The prover returns another element of the group s which is equal to another element in group G. Then the verifier computes two values, g^{s} which is $g^{a+bx \mod p}$ and $ha^{b} \mod p$ wherein h a random element in the group, a is the initial known public value, and b is Bob's randomly generated value. If these two are equal, then Bob knows Alice has the private key x while not knowing what it is, therefore the argument is sound, complete, and zero-knowledge and could only be broken by getting lucky in finding what x and that becomes very unlikely as p increases.

If the computations in Operation 8 match then the verifier is sure that the prover has the discrete logarithm key, but he does not know what that key is.

ii. Hash Functions

A Hash function takes in a string and outputs a random string of fixed length. There are some useful properties of hash functions,

Deterministic. The same input, will always give the same output no matter how many times the hash function is applied. Therefore a hash function has a unique input. However if there is even a small change in the input, even one characters, the entire hash function will change. Using the input 'This is a test' the result of an SHA-256 hash is 'C7BE1ED902FB8DD4D48997C6452F5D7E509FBCDBE2808B16BCF4EDCE4C07D14E' but of the input 'this is a test' the output is 'E99758548972A8E8822AD47FA1017FF72F06F3FF6A016851F45C398732BC50C'.

Quick Computation. The process of hashing a string is very efficient, in most cases the computation is negligible.

Pre-Image Resistance. With knowledge of the hash function, it is very improbable to find the input, this is called probabilistic soundness wherein if the output is long enough, it is very difficult to guess the initial input. For an example a 2^{128} hash output, would require, in the worst cast, $2^{128}-1$ computations which is around $3.4 \cdot 10^{38}$ different possibilities.

Collision Resistant. Given two inputs m_1 and m_2, the probability of $H(m_1)$ and $H(m_2)$ is given by the birthday paradox. For $p = 0.5$, there would have to be $\sqrt{|H(m_1)|}$. For a 2^{128} bit hash, it would 'only' require 2^{64} iterations to get $p = 0.5$ which is much lower than previously described, this problem is called the Birthday Problem. However for large enough scheme like SHA256, it is safe enough to assume that no collisions will occur. So by just changing T to t, the entire output is different.

Puzzle Friendly. This states that for every output Y, and if k is chosen from a distribution of high entropy (meaning there is a lot of choice, for example pick a number between 0 and 2^{128}) then it is computationally inefficient, and hard, to find a number x such that.

$$H(k \mid x) = Y \tag{2}$$

The notation | refers to concatenation, for example if $k =$ WATER and $x =$ MELON, then $k \mid x =$ WATERMELON. This is the most important property of a hash function, and is the foundation of Bitcoin mining (which won't be explained in this internal assessment).

Formally a hash function can be written as a mapping from a large message space to a tag space $H : M \to T$ wherein $|M| \gg |T|$ and a collision is for H is defined as a pair $m_1, m_2 \in M$ such that,

$$H(m_1) = H(m_2) \qquad m_1 \neq m_2 \tag{3}$$

The hash functions that are of this nature, collision resistant, will not be discussed here however for the remainder of the investigation SHA256 will be the standard algorithm used and will be assumed to be safe to reveal publicly.

iii. Commitments

In financial blockchains, when a transaction is being negotiated between two companies, let's say company A & B, they need to ensure each party have the required amounts in their accounts, and that would require going through what is known as a transaction tree. Which is a tree of all transactions a company has gone through, wherein outputs and inputs are stated. When Company A will go to check that Company B's accounts are in order, they could potentially end up looking back at transactions that are ongoing, as well as being able to determine what Company B owns, assets, liabilities, liquidities etc. Furthermore if Company B had 10$ million in their account, and Company A initially wanted 1$ for the transaction, they could go back on what they said and increase that amount knowing Company B has more in their account. This goes against the expected privacy typical financial institutions require in order to operate properly. As such, a method in which one knows the inputs and outputs are equal without knowing they are must be developed, and this is where a commitment comes in.
A commitment is a hash of both a blinding value, and your data which can be shown as

$$\text{Commitment} = H\{\text{Blinding Value}||\text{Data}\} \tag{4}$$

Where blinding factor is a randomly generated value, and H is a hash function such as SHA256 which is a point on an elliptic curve. As you can see, once someone commits to a value, it cannot be changed due to the randomly generated value so when that someone wants to reveal that data, they send the blinding value, hash function, and data to the verifier and if that hash function matches then the data is good. Now we can expand upon this idea for proving inputs are equal to outputs by allowing commitments to add to each other.

iv. Pedersen Commitments

A Pedersen Commitment is a special commitment wherein different commitments can add and subtract in an associative manner. This can be illustrated by:

$$H\{\mathrm{BV}_1 \mid\mid \mathrm{Data}_1\} + H\{\mathrm{BV}_2 \mid\mid \mathrm{Data}_2\} = H\{\mathrm{BV}_1 + \mathrm{VF}_2 \mid\mid \mathrm{Data}_1 + \mathrm{Data}_2\} \tag{5}$$

And

$$H\{\mathrm{BF}_1 \mid\mid \mathrm{Data}_1\} - H\{\mathrm{BF}_2 \mid\mid \mathrm{Data}_2\} = 0 \tag{6}$$

So long as $\mathrm{Data}_1 = \mathrm{Data}_2$, which can be intrepreted as inputs and outputs, the inputs and outputs are equal, therefore Company A can be sure that company B's inputs and outputs are sound. This will be checked for the entire transaction tree, which over many years can become very large hence the need for a system that hashes the total values, not the individual values, therefore the computational speed goes from $O((4n)^2) \rightarrow O(4n+1)$ where n is the number of inputs and outputs. For a tree with 1000 inputs, which is common over a month for a small bank, individual commitments requires 16,000,000 computations, whereas the Pedersen Commitments requires 4001.

$$\mathrm{Pub} = xG \tag{7}$$

Then using the homomorphic property, $f(x_1 \# x_2) = f(x_1) \# f(x_2)$.

$$\mathrm{Pub}_1 + \mathrm{Pub}_2 = (x_1 + x_2 \mod n)G \tag{8}$$

Hence if $(x_1 \vee x_2) < 0$, $\bigvee$ $x_1 + x_2 > n(G)$ then we will be creating currency from thin air. To clarify as $x_1 = -3$, $x_2 = 6$, $x_3 = 7$, and $x_4 = 1$, and we want to make sure $(x_3 + x_4) = (x_2 + x_1)$, hence

$$(7 + 1) - ((-3) + 6) \equiv 0 \tag{9}$$

However since it uses ECC, then the following equation can be interpreted as I add 7 and 1, and then subtract $2^{64} - 3$ (the bit size used to encrypt) and then subtract 6. Which will overflow the Pedersen commitment and inevitably cause it to fail as you have just made from thin air 2^{64} dollars. Therefore a system needs to be constructed wherein the inputs need to be between a certain range, $[0, 2^{64}]$ without knowing what the value is, as this would defeat the purpose. In this Internal Assessment, the Borromean Ring Signature will be used as a range proof, and then the later more efficient Bullet Proof will be explained.

III. Hash Structures

As previously described, hash functions allow for secure release of information and act as a random trapdoor function. This can be expanded upon by using the properties of concatenation to break down larger data structures into their smaller parts.

i. Merkle-Damgard Paradigm

The paradigm works on the basis that given a collision resistant hash function for a short message, a longer collision resistant hash function can be created. First take a large message M and break it down into smaller message $m[i]$, and the hash of the message M, $H(M)$ is equal to all the smaller hashes of $m[i]$, $h(m[i])$, called the chaining variables. The first hash takes in the first value of the message, and adds a constant initialisation vector (IV) to it, a constant for N dimensional computation, $h(m[0]+IV)$. For each subsequent smaller hash, the corresponding message $m[i]$ is added such that for $m[3]$, the hash is $h(m[0]+m[1]+m[2]+m[3]+IV)$. For the last message, a padding block (PB) is added to the message, which is just a 1 followed by 0s so that its size is equal to the message size (often 2^{64}). The output of the chaining variables, h, is the hash of $H(M)$. More formally,

$$h_0 : m[0] \times \mathrm{IV} \rightarrow h(m[0] + \mathrm{IV}) \tag{10}$$

$$h : h(m[i-1]) \times m[i] \rightarrow h(m[i]) \tag{11}$$

For the last block n, this is given as,

$$h : h(m[n-1]) \times \mathrm{PB} \rightarrow h(m[n]) \tag{12}$$

Now to prove its collision resistant,

Theorem 1. If the small hash function, h, is collision resistant than so is the large hash function, H.

Proof. This is a proof by contradiction, firstly suppose $H(M) = H(M')$, hence not collision resistant, and now to build a smaller hash function, h, from that. Firstly, the chaining values for each large hash are given as,

$$(\mathrm{IV} = H_0), H_1, \ldots, H_t, H_{t+1} = H(M) \tag{13}$$

$$(\mathrm{IV} = H'_0), H'_1, \ldots, H'_r, H'_{r+1} = H(M') \tag{14}$$

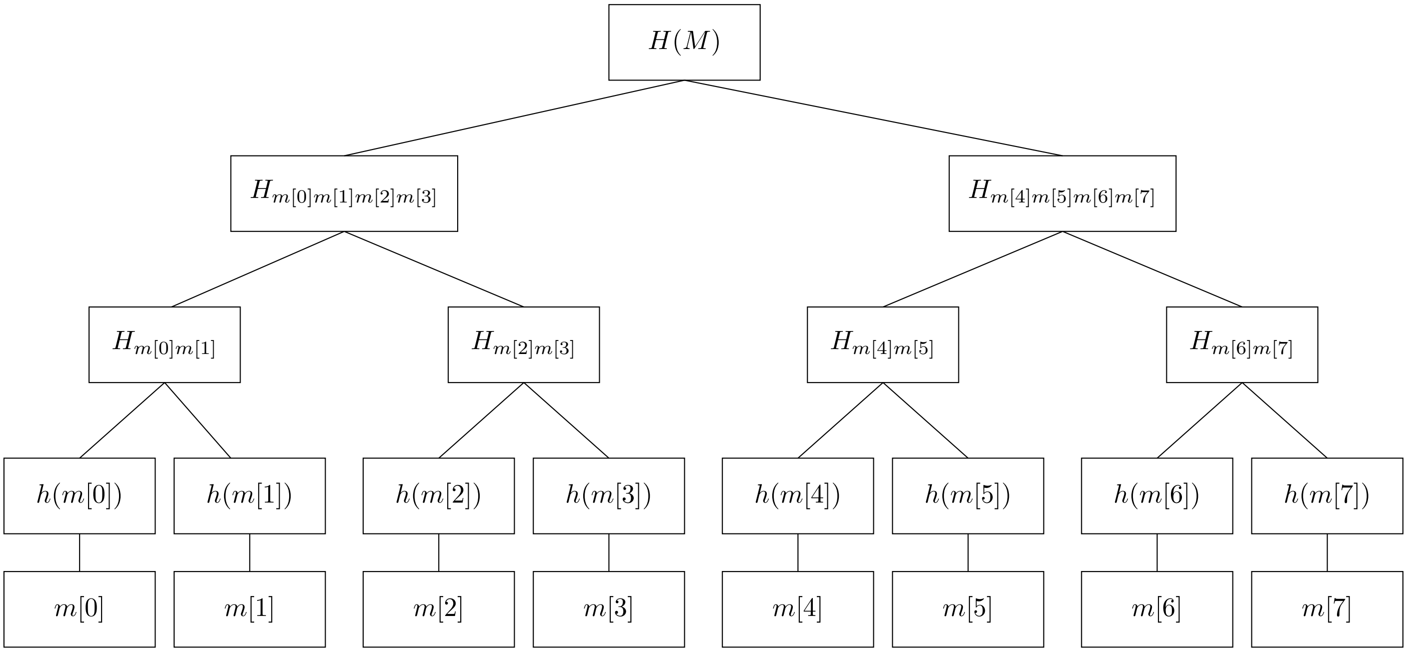
Now the last chaining variables need to be equal to each other, hence

$$h(H_t, M_t \mid\mid \mathrm{PB}) = H_{t+1} = H'_{r+1} = h(H_r, M_r \mid\mid \mathrm{PB}') \tag{15}$$

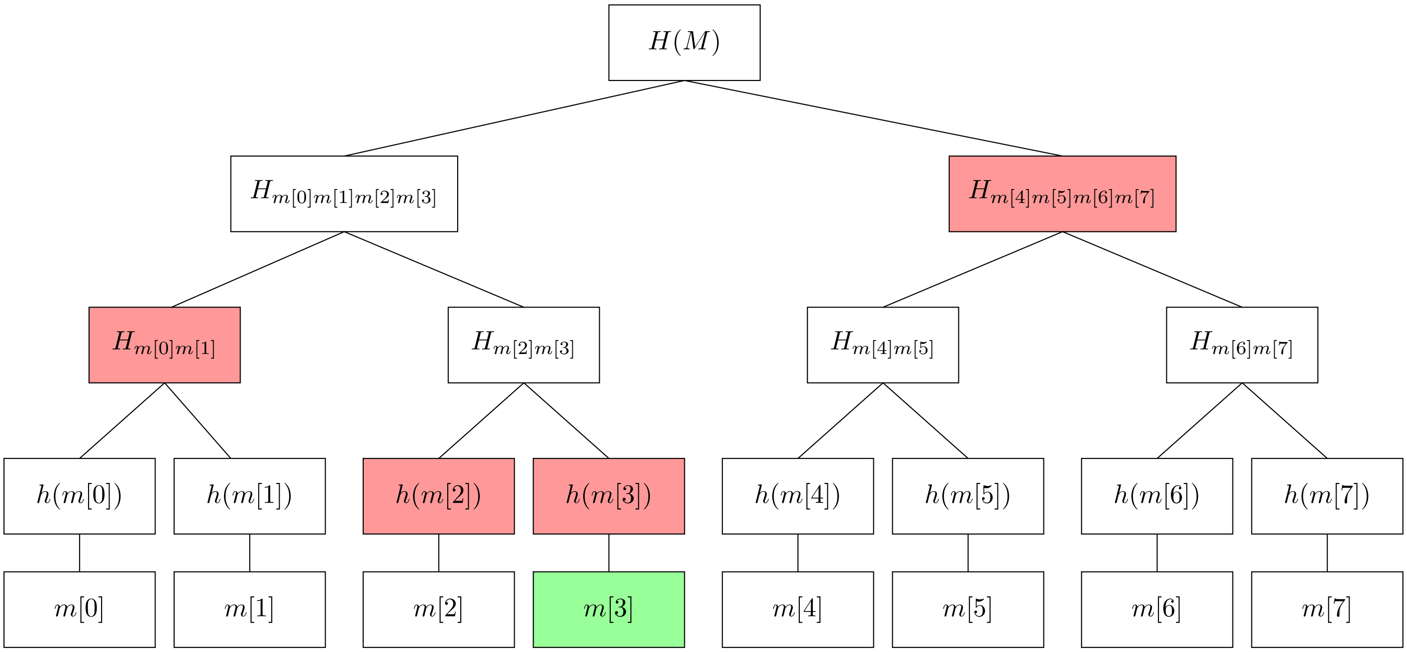
To summarise, the last small hash, is equal to the previous hash and the padding block, and that must be equal to the message M', therefore the same can be said for $H(M')$. From this it can be said so long $H_t \neq H_r$, $M_t \neq M'_r$ or PB $\neq$ PB' then these will be different, and hence they are collision resistant as the message M and M' are different, therefore give different small hashes, as well as different lengths so different padding block. However in the case all these three are the same, and hence equivalent, than the previous small hashes and messages would need to be checked. This would repeat until the values were not equal, in the case $m[0] = m'[0]$ then both the messages are equivalent hence it still preserves collision resistance. □

ii. Merkle Trees

As previously discussed, a large message M can be broken down into many sub parts, each of which are still collision resistant. To illustrate the next point, Bob is the verifier and has knowledge of the public variable $H(M)$, and Alice has the knowledge of all the smaller messages $m[i]$. In order to construct a zero knowledge argument, Alice would have to send every single $h(m[i])$, to create the Merkle-Damgard Paradigm, sending each hash to Bob. When dealing with large amounts of datas, this can be very inefficient, and if it weren't for Merkle Trees the blockchain could not exist. Firstly, a Merkle Tree can be visualised as follows,



To analyse this diagram, the fist layer is the message broken down into smaller parts, that are then all hashed using h, creating indistinguishable values that are safe to make public. Then for each layer, the hashes of the small messages are concatenated such that $H_{m[0]m[1]} = h(h(m[0]) \mid h(m[1]))$. This is repeated until the entire message has been concatenated until the hash of the entire message is taken. Now since Alice wants to prove knowledge of a $m[i]$ instead of revealing every other small hashes she just needs to reveal all the hashes that are directly connected to the path from her secret message to the public $H(M)$. It is easier to explain through an example, if Alice wants to prove she knows $m[3]$, in green, she would only need to reveal the following hashes shown in red,



This can be shown since by concatenating the hashes, there values are still conserved therefore $H(M)$ can be written as a combination of the chaining values, as well as concatenated values. For $m[3]$ it can be done such that,

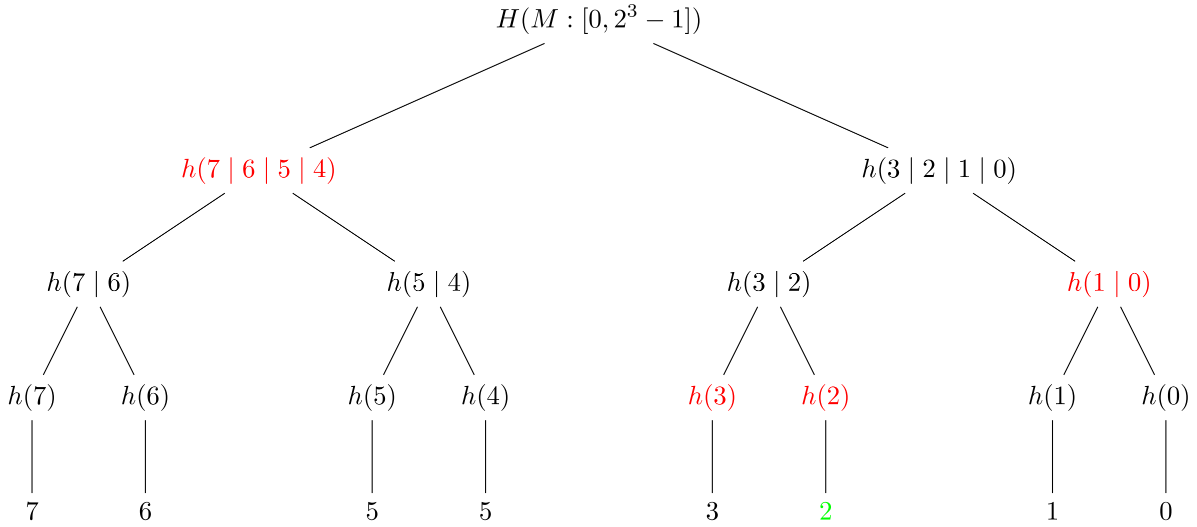
$$H(M) \equiv H\big(h(m[3]) \mid h(m[2]) \mid H_{m[0]m[1]} \mid H_{m[4]m[5]m[6]m[7]}\big) \tag{16}$$

So instead of revealing all 8 small message, Alice only needed to reveal 4 which has cut the number of exchanges in half. For larger data structures this cuts the information exchange exponentially. Now if Bob wants to make sure

Alice's message is valid, he can match the hash in Equation 16 to $H(M)$ and if they are the same, he can confidently say Alice's message is correct. This process is also done by computers on Blockchain networks to make sure a party's transactions are valid. Now this idea can be expanded to a Zero-knowledge Argument for a new range proof.

IV. Merkle Tree Range Proofs (MTRP)

The construct of a Merkle Tree will be broken into two, one for the inputs, and another for the outputs. Then each values will have its own branch associated to it, this means that the structure is broken into leaves of more than 2, this range proof works between $[0, 2^n - 1]$ where n is a natural number and greater than 0 (so $2^2, 2^3$) etc. To understand the concept it is much easier to visualise first, for example a range proof between $[0, 2^3 - 1]$ wherein Alice's transaction of 2$,



Now Bob can verify Alice's transaction value is between $[0, 2^3 - 1]$ by checking,

$$H(M) \equiv H\big(h(7 \mid 6 \mid 5 \mid 4) \mid h(3) \mid h(2) \mid h(1 \mid 0)\big) \tag{17}$$

This will reveal no information about what value Alice was hiding (although Bob could just check the hashes for each value between 1 and 8) however when the upper bound is large, it is infeasible for Bob to work out what Alice's secret value is. While the mathematics behind the range proof is fairly simply, the increase in computing in unparalleled. With a Merkle tree, the computing efficiency is given as $O(\log_2(n))$ for each value added to the Pedersen Commitment. Here are the current most used range proofs and their efficiency, wherein N is the number of verifications, and n is the signature size,

Name	**Verification**
AOS RS	$N + n$
Borromean RS	$N + 1$
Bulletproof	$2\big(\log_2(n) + \log_2(N)\big) + 9$
Merkle Tree	$\log_2(n) + 1$

Table I. Signature size for N verifications.

Which can be graphed, with a increasing verification sizes and at $n = 128$,

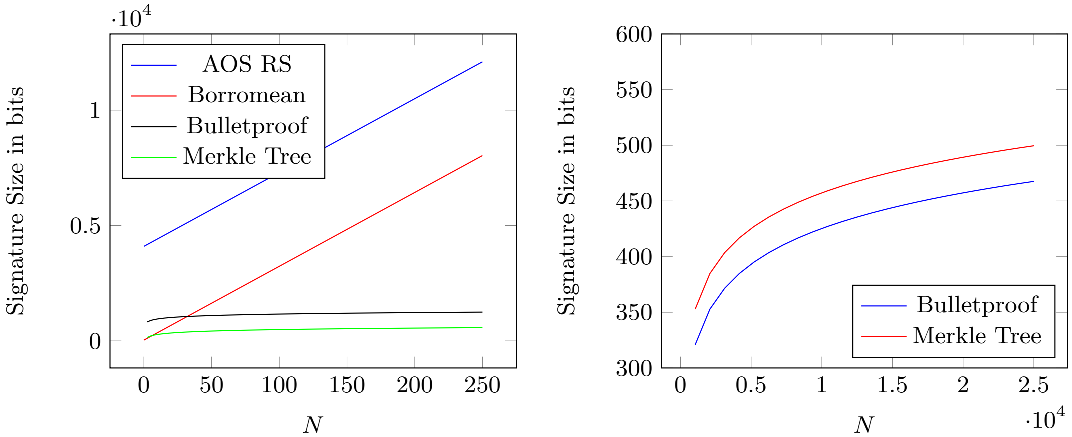


Figure 2. The first figure illustrates the signature size as the number of verifications, N, increases with a range proof between 0 and $2^n - 1$. The second figure shows how over time bullet proofs are more efficient to use as they are easier to aggregate. However this different is still relatively small (32 bits).

Merkle Trees are the fastest way for Bob to verify Alice's transaction is between 0 and $2^n - 1$. They are also far simpler to understand, with good visual intuition as opposed to the currently most efficient range proof, namely the Bullet Proof, which requires very good knowledge of the inner product argument as well as a many more complex interactions. The main drawback of using the Merkle Tree Range Proof is the initial cost of generating the Merkle tree for Alice, which would require to generate $2^{n+1} - 1$ hashes, however the entire Hash tree should be available publicly, and would work most efficiently in centralised systems as opposed to Blockchain networks, such as the new PAYE (Pay as you earn) tax scheme recently implemented wherein employers will be able to see your wage. Using one of these Merkle trees on a government server, and have each employer instead see the tax bracket range, instead of the specific amount, will ensure privacy while still having the benefits of the scheme such as reduced tax evasion and fraud. It can also be used by supermarkets for efficient age checking, and by advertisers in targeting age groups, while not revealing your age.

i. Open Problems

The main issue is the size of key generation for Alice, this could be addressed by creating more efficient range proofs working in the exponent size n of the input rather than the number itself. There have also been recent improvements to Merkle Trees namely Efficient Sparse Merkle Trees and the Bitcoin Improvement Proposal Merkle Tree, both of which could reduce the tree generation size significantly. There is also the question of renewing hashes, as the chance of collision increases significantly when multiple users are using the same hashes, and therefore an efficient way of refreshing Merkle Trees without creating a new one needs to be improved.

[1] Gregory Maxwell. “Confidential Transaction, the Initial Investigation” *ElementsProject*, 28 April 2018. https://elementsproject.org/elements/confidential-transactions/investigation.html

[2] Gregory Maxwell, and Polestra, Andrew. “Borromean Ring Signatures” *Blockstream*, 2 June 2015. https://pdfs.semanticscholar.org/4160/470c7f6cf05ffc81a98e8fd67fb0c84836ea.pdf

[3] Bootle, Jonathan et al . “Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting” *Cryptology ePrint Archive, Report 2016/263*, 2016. https://eprint.iacr.org/2016/263

[4] Bünz, Benedikt et al . “Bulletproofs: Short Proofs for Confidential Transactions and More” *Stanford University, University College London, and Blockstream*, 2018. https://eprint.iacr.org/2017/1066.pdf

[5] Curran, Brian. “What are Bulletproofs? Guide to Confidential Cryptocurrency Transactions.” *Blockonomi*, 2018. https://blockonomi.com/bullet-proofs/

[6] Mitra, Rajarshi. “The In’s and Out’s of Cryptographic Hash Functions” *Blockgeeks*, 2017. https://blockgeeks.com/guides/cryptographic-hash-functions/