Optimising the SHA256 Hashing Algorithm for Faster and More Efficient Bitcoin Mining

Presented by:
Rahul P. Naik (12026189)

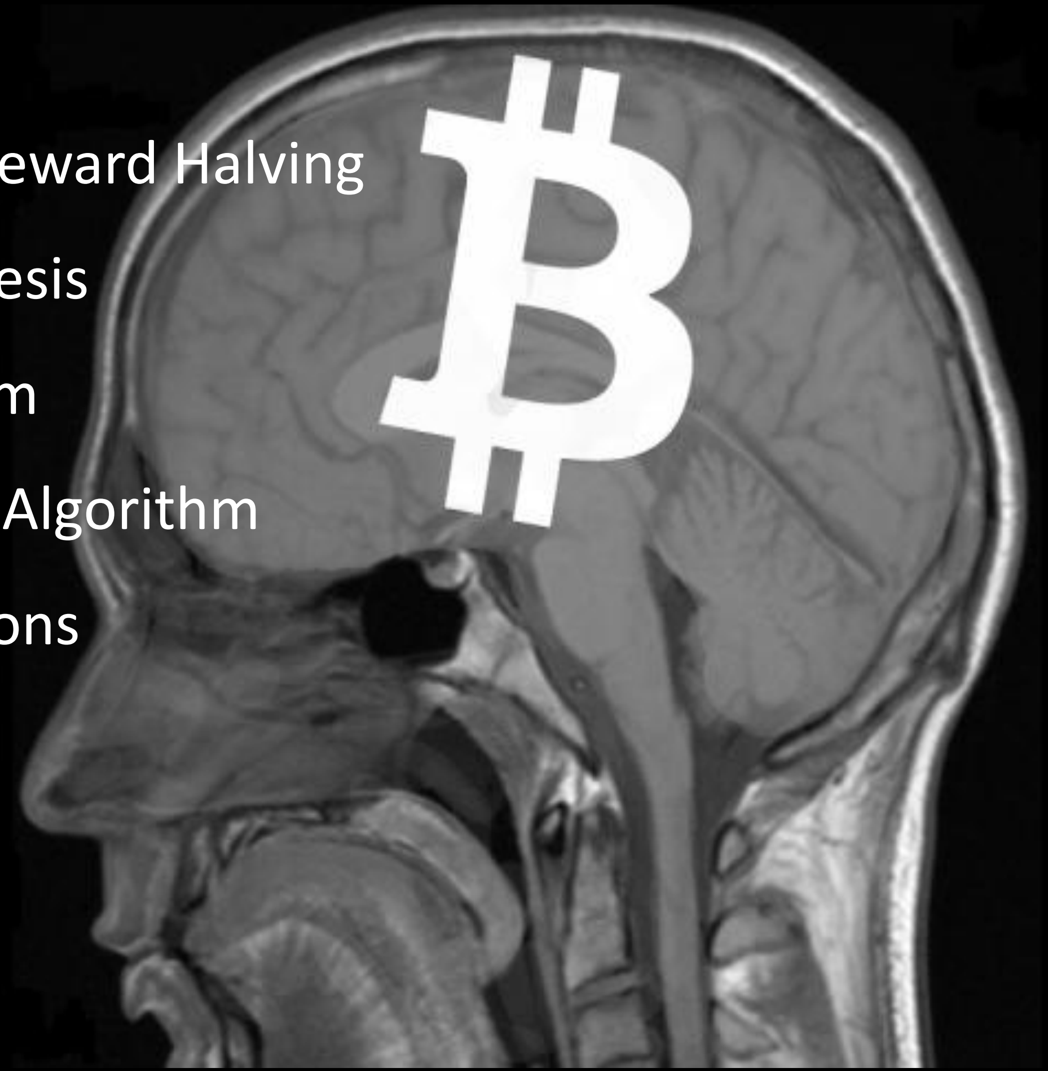
Supervisor:
Dr. Nicolas T. Courtois

MSc Information Security

DEPARTMENT OF COMPUTER SCIENCE
September 17, 2013

Agenda

- Bitcoin and Bitcoin mining
- Improvement to the Mining Reward Halving
- Motivation and Aim of the Thesis
- The SHA256 Hashing Algorithm
- Bitcoin Block Header Hashing Algorithm
- SHA256 Algorithm Optimisations
- Discussion
- Limitations and Future Work
- Conclusion



What is Bitcoin & Bitcoin Mining?

- A global, decentralised virtual currency scheme
- Not backed by any government or legal entity
- Invented in 2008 by Satoshi Nakamoto (A Pseudonym)
- Total number of Bitcoins are limited to about 21m and are divisible up to 8 decimal places
- Bitcoins are minted into existence by a process called Bitcoin mining i.e. calculating the double SHA256 hash
- Currently 25 Bitcoins are mined every 10 minutes
- Mining is essentially finding a new block accepted by the Bitcoin network
- Bitcoin Transactions are indirectly included into each block

Improvement Proposal for Mining Reward Halving

- Currently 25 BTCs awarded for every new block found
- Reward is halved every 210000 blocks (roughly every 4 years)
- Reward suddenly halves i.e. it suddenly becomes twice as costly to mine Bitcoins

12.5 BTC → 6.25 BTC and 420000 → 630000

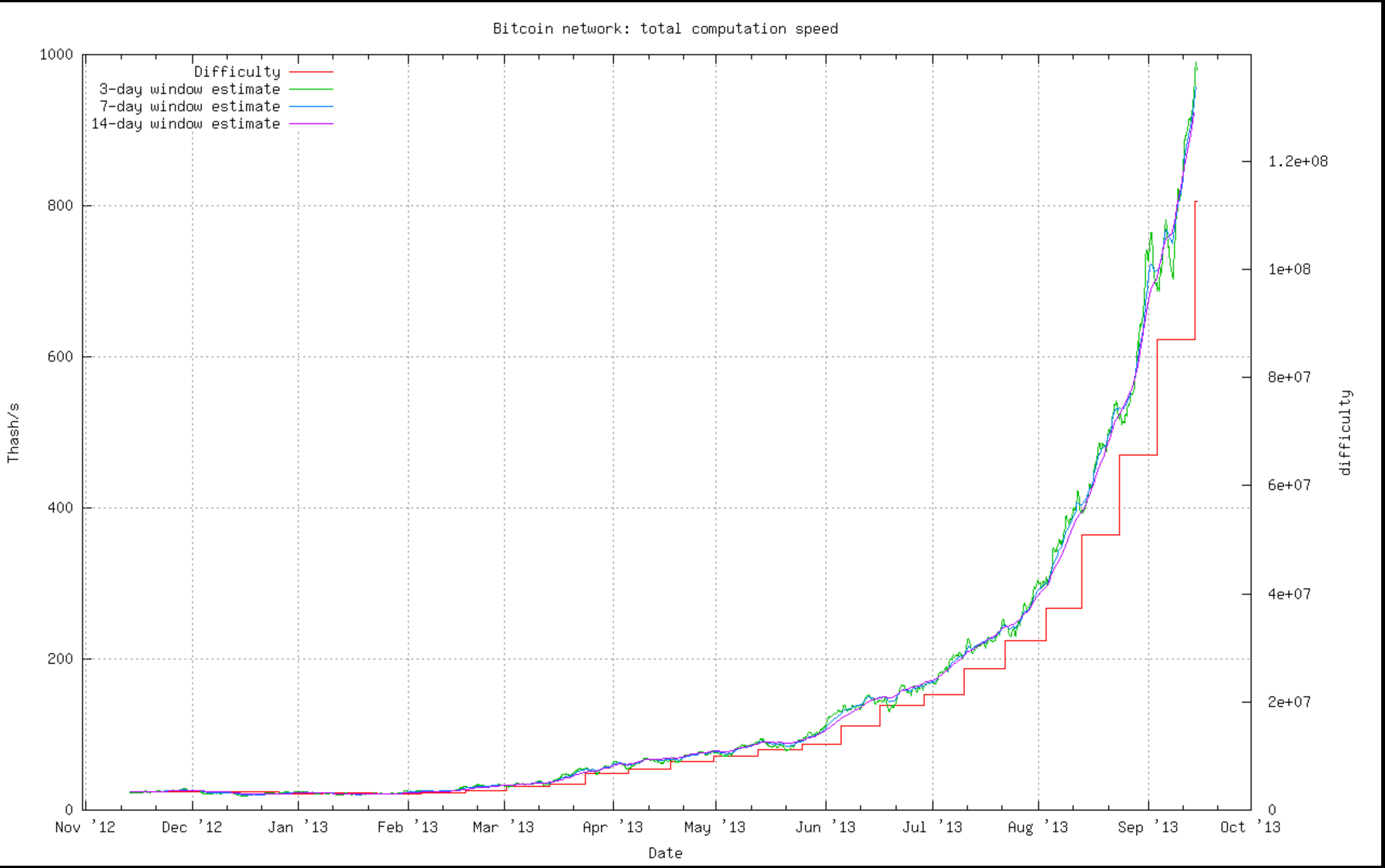
Therefore, Reward = Reward - 6.25/(630000-420000) = Reward - 0.00002976

Thus, the number of Bitcoins awarded when block 420001 is mined will be 12.5 - 0.00002976 = 12.49997024 and so on

Market Price ($) of Bitcoin over the Years

Bitcoin Network Hash Rate over the Years

Bitcoin Hash Rate & Difficulty



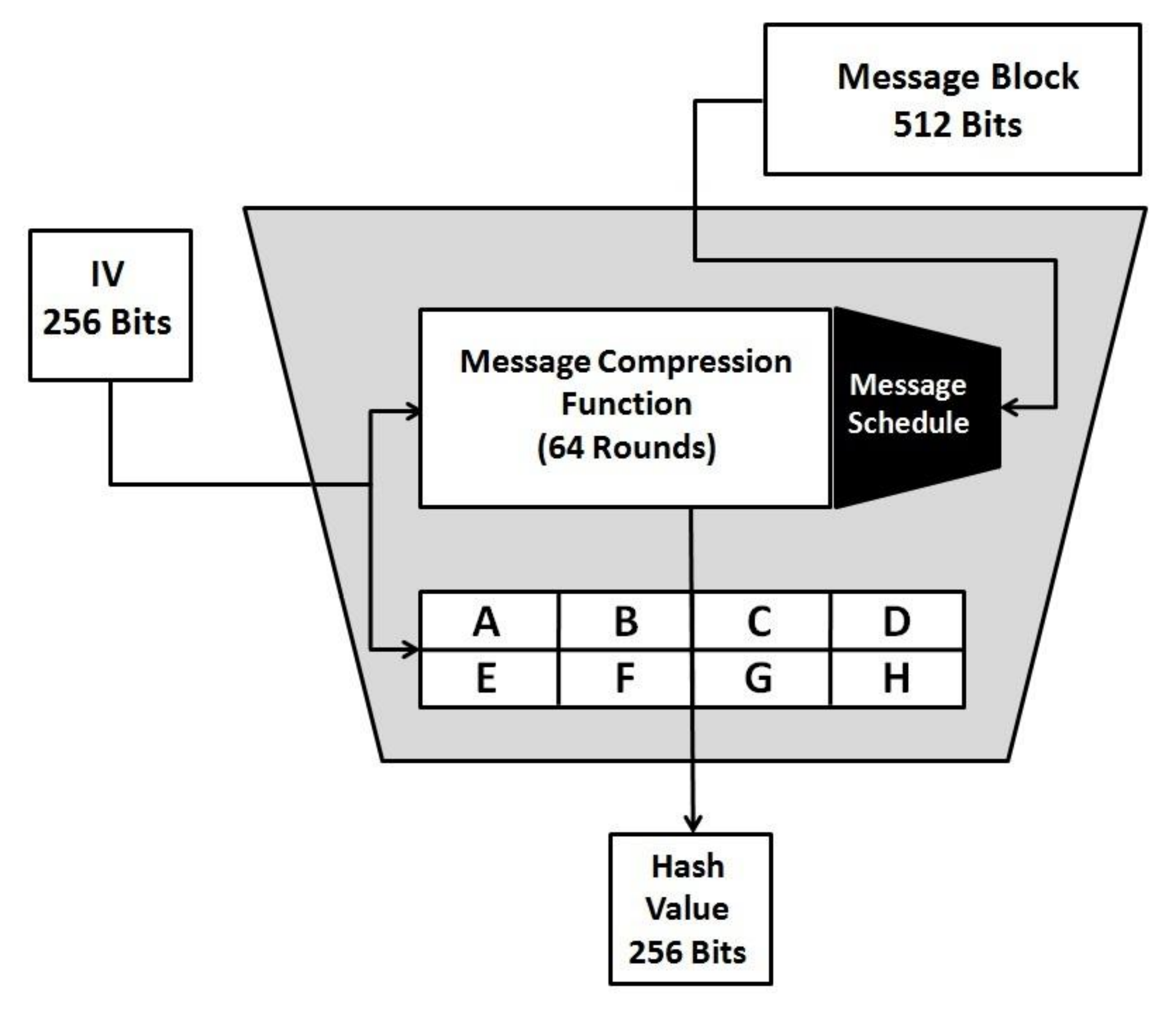
Bitcoin Energy Consumption Statistics

Source: blockchain.info

Hash Rate and Electricity Consumption	
Difficulty	108,730,911.96
Hash Rate	962,097.65 GH/s
Electricity Consumption	15,008.72 megawatt hours
Electricity Cost	$2,251,308.49

There must be a more efficient way to mine Bitcoins!

The SHA256 Hashing Algorithm



SHA256 Message Scheduler

For $0 \leq t \leq 15$,

$W_t = M_t$

For $16 \leq t \leq 63$,

$W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-5}) + W_{t-16}$

$\sigma_0(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$

$\sigma_1(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$

SHA256 Message Compression Function

$T_1 = H + \sum_1(E) + Ch(E, F, G) + K_t + W_t$

$T_2 = \sum_0(A) + Maj(A, B, C)$

$H = G; G = F; F = E$

$E = D + T_1 = D + H + \sum_1(E) + Ch(E, F, G) + K_t + W_t$

$D = C; C = B; B = A$

$A = T_1 + T_2 = H + \sum_1(E) + Ch(E, F, G) + \sum_0(A) + Maj(A, B, C) + K_t + W_t$

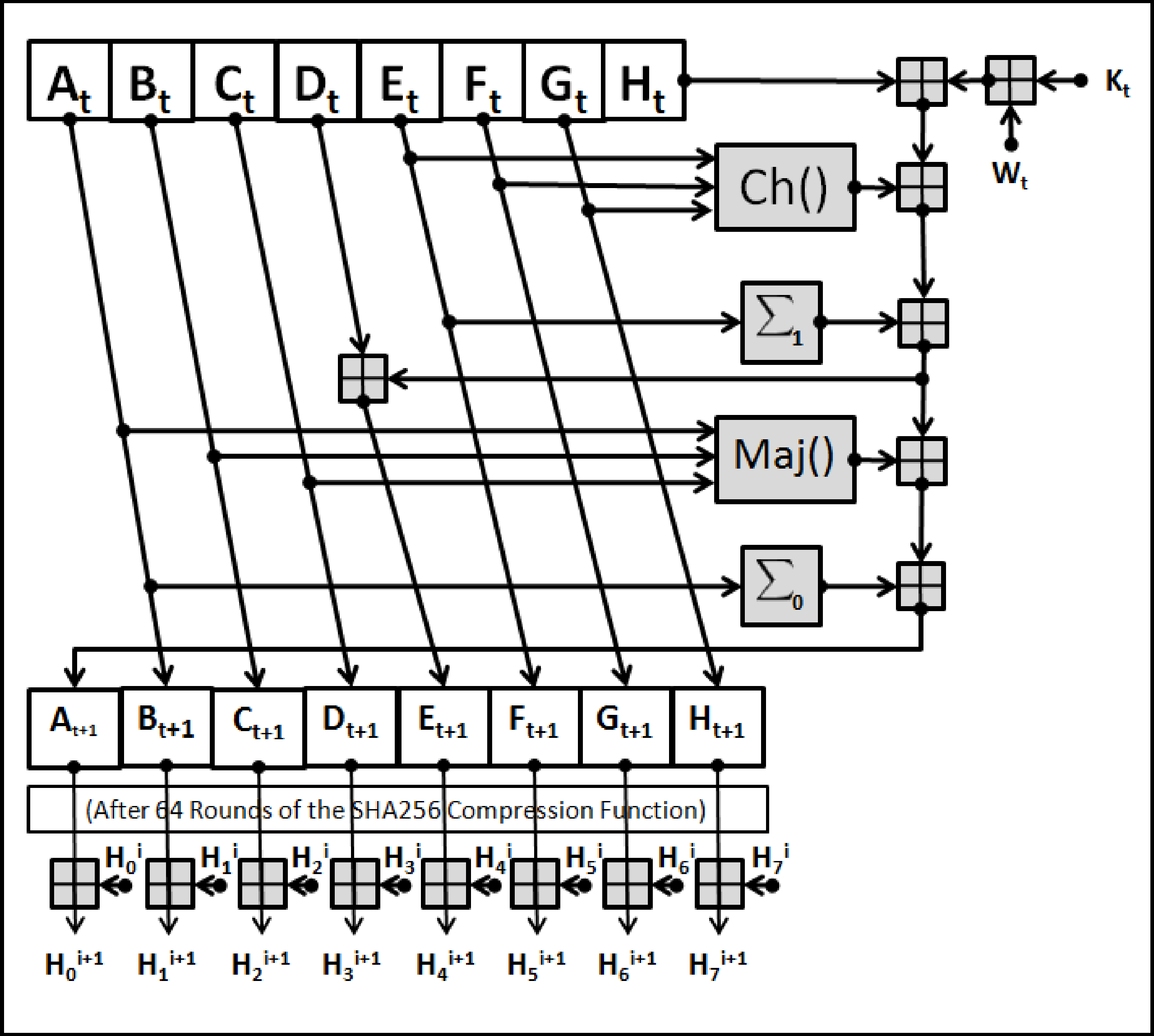
$Ch(X, Y, Z) = (X \wedge Y) \oplus (\neg X \wedge Z)$

$Maj(X, Y, Z) = (X \wedge Y) \oplus (X \wedge Z) \oplus (Y \wedge Z)$

$\sum_0(X) = ROTR^2(X) \oplus ROTR^{13}(X) \oplus ROTR^{22}(X)$

$\sum_1(X) = ROTR^6(X) \oplus ROTR^{11}(X) \oplus ROTR^{25}(X)$

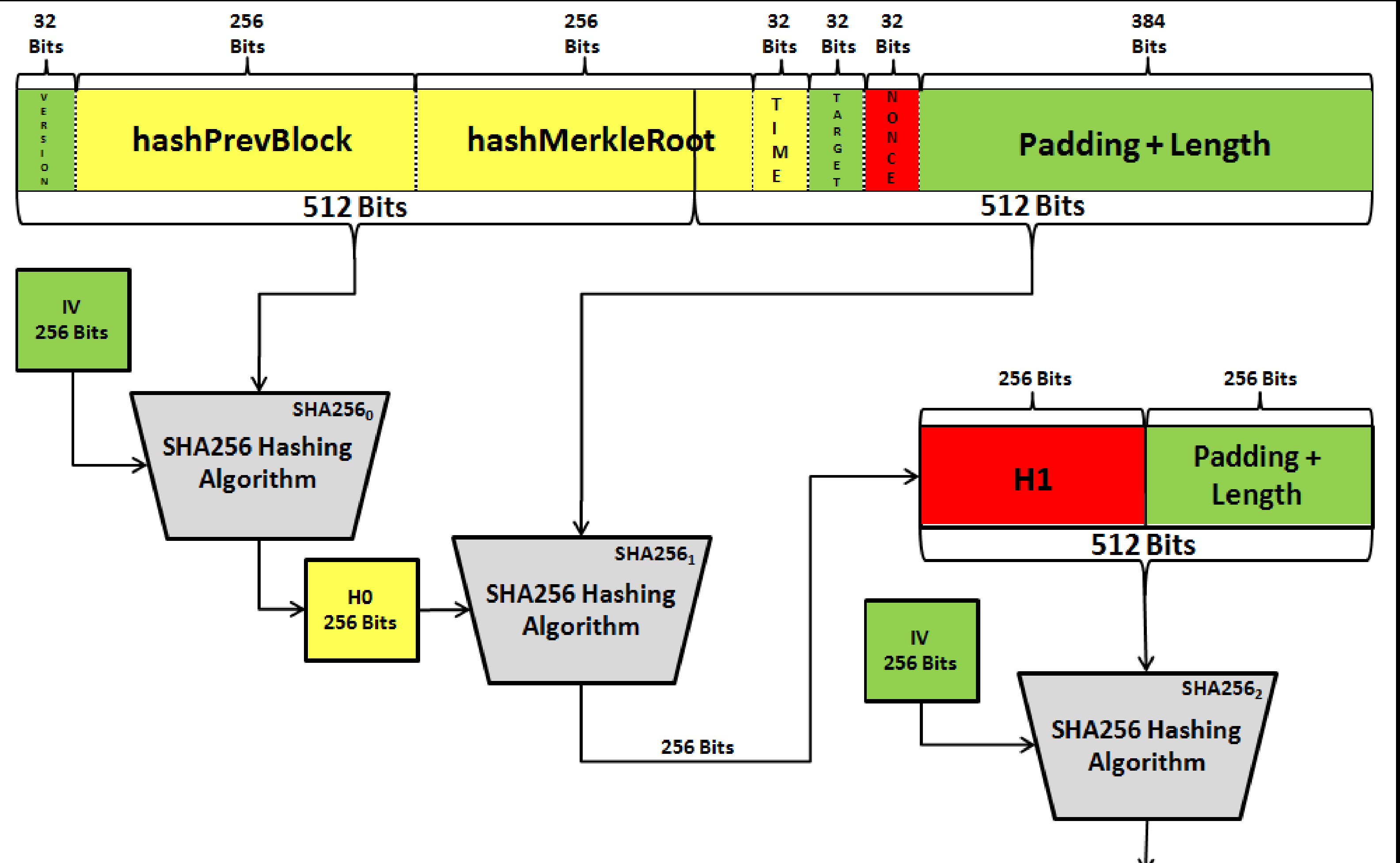
The SHA256 Hashing Algorithm Contd..



The Number of Operations in SHA256

Additions (Mod 2^{32})	= (7*64) + (3*48) + 8 = 448 + 144 + 8 = **600**	(message compression) + (message scheduler) + (intermediate/final hash computation)
Bitwise Rotations (ROTR)	= (6*64) + (4*48) = 384 + 192 = **576**	$(\Sigma_0, \Sigma_1) + (\sigma_0, \sigma_1)$
Bitwise Shifts (SHR)	= 2*48 = **96**	σ_0, σ_1
Bitwise AND ($\wedge$)	= 5*64 = **320**	Maj, Ch
Bitwise EX-OR ($\oplus$)	= (7*64) + (4*48) = 448 + 192 = **640**	(message compression) + (message scheduler)
Total Operations	= 600 + 576 + 96 + 320 + 640 = **2232**	

The Bitcoin Block Header Hashing Algorithm

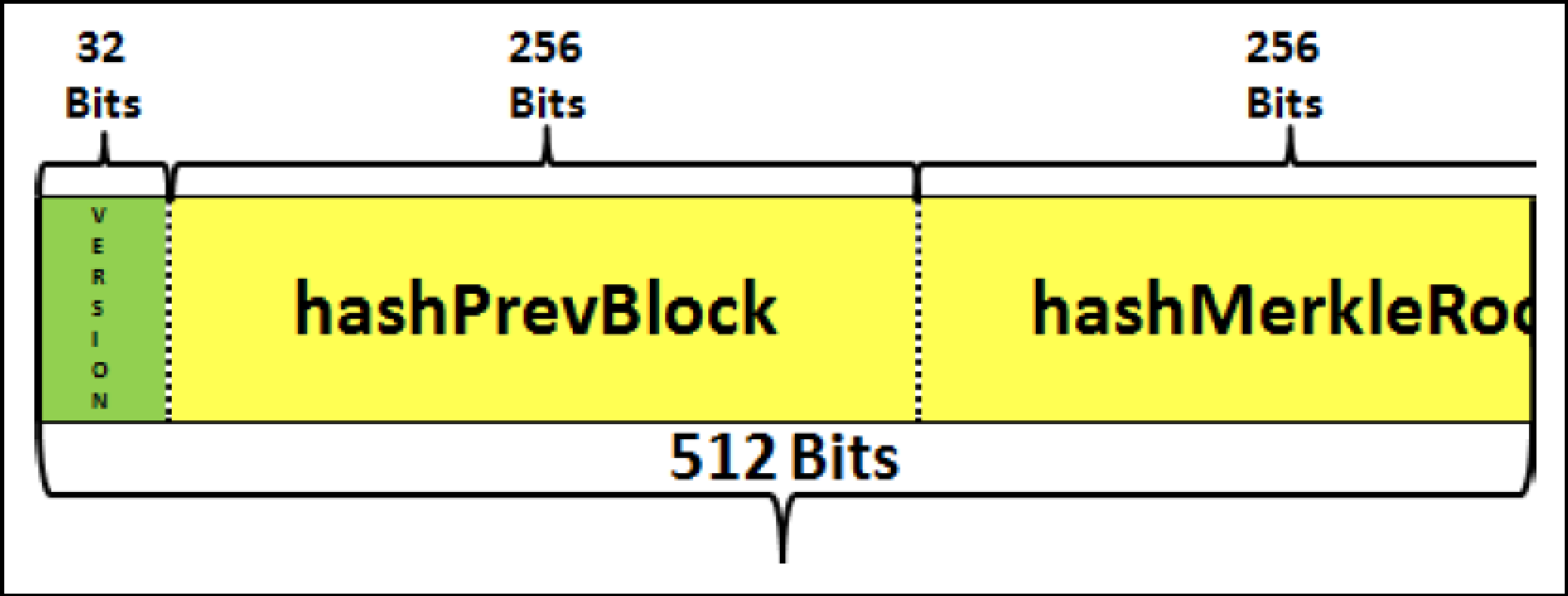


H2 = SHA256(SHA256(Block_Header))

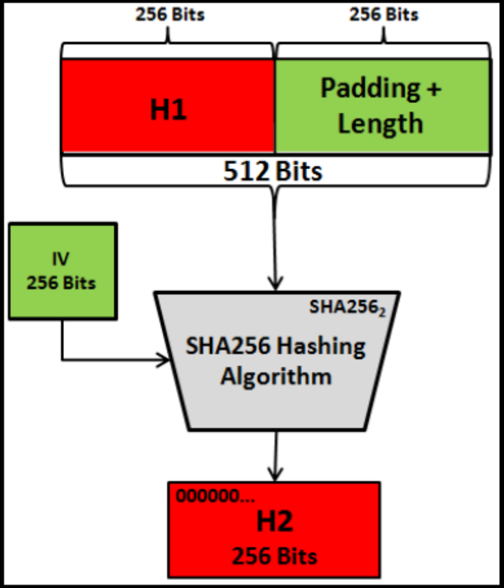
The Bitcoin Block Header Details

Field	Size	Description
Version	32 bits	Block version information that is based on the Bitcoin software version creating this block
hashPrevBlock	256 bits	The hash of the previous block accepted by the Bitcoin network
hashMerkleRoot	256 bits	Bitcoin transactions are hashed indirectly through the Merkle Root
Timestamp	32 bits	The current timestamp in seconds since 1970-01-01 T00:00 UTC
Target	32 bits	The current Target represented in a 32 bit compact format
Nonce	32 bits	Goes from 0x00000000 to 0xFFFFFFFF and is incremented after a hash has been tried
Padding + Length	384 bits	Standard SHA256 padding that is appended to the data above

#1 The Calculation of H0 for $SHA256_0$



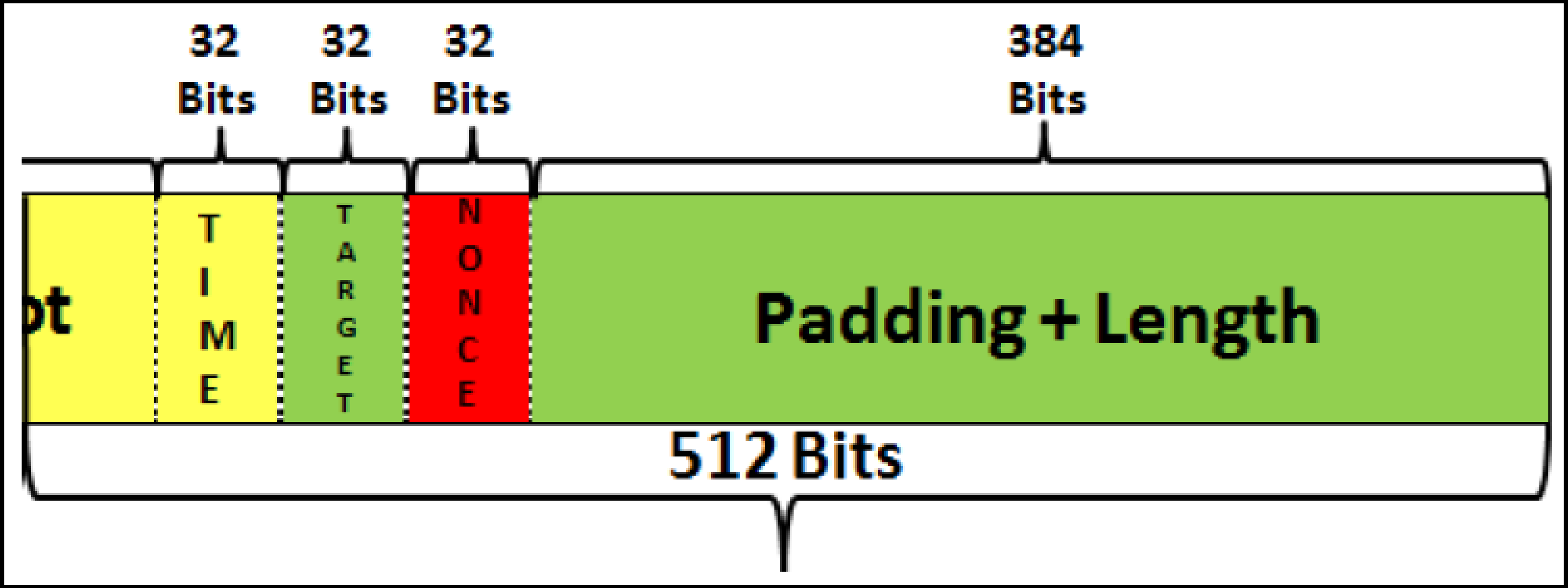
#2 Early Rejection at Rounds 61 and 62 for $SHA256_2$



	A	B	C	D	E	F	G	H
t=59:	B6AE8FFF	FFB70472	C062D46F	FCD1887B	B21BAD3D	6D83BFC6	7E44008E	9B5E906C
t=60:	B85E2CE9	B6AE8FFF	FFB70472	C062D46F	961F4894	B21BAD3D	6D83BFC6	7E44008E
t=61:	04D24D6C	B85E2CE9	B6AE8FFF	FFB70472	948D25B6	961F4894	B21BAD3D	6D83BFC6
t=62:	D39A2165	04D24D6C	B85E2CE9	B6AE8FFF	FB121210	948D25B6	961F4894	B21BAD3D
t=63:	506E3058	D39A2165	04D24D6C	B85E2CE9	5EF50F24	FB121210	948D25B6	961F4894

Source: http://csrc.nist.gov/groups/ST/toolkit/documents/Examples/SHA256.pdf

#3 First 3 Rounds of SHA256$_1$



#4 Round 4 Incremental Calculations for SHA256[1]

$$A = H + \sum_1 (E) + Ch(E, F, G) + \sum_0 (A) + Maj(A, B, C) + K_t + W_t$$

$$E = D + H + \sum_1 (E) + Ch(E, F, G) + K_t + W_t$$

Nonce	A	B	C	D	E	F	G	H
0x00000000	c14c28c6	fdd86aa7	1184d36	2703413e	346785c7	c1abdbc7	8f925db9	a4b56f21
0x00000001	c14c28c7	fdd86aa7	1184d36	2703413e	346785c8	c1abdbc7	8f925db9	a4b56f21
0x00000002	c14c28c8	fdd86aa7	1184d36	2703413e	346785c9	c1abdbc7	8f925db9	a4b56f21
0x00000003	c14c28c9	fdd86aa7	1184d36	2703413e	346785ca	c1abdbc7	8f925db9	a4b56f21
0x00000004	c14c28ca	fdd86aa7	1184d36	2703413e	346785cb	c1abdbc7	8f925db9	a4b56f21
0x00000005	c14c28cb	fdd86aa7	1184d36	2703413e	346785cc	c1abdbc7	8f925db9	a4b56f21

#5 Saving Additions Using the Long Trail of 0s for $SHA256_1$ and $SHA256_2$

$$l + 1 + k = 448 \mod 512$$

$$k = 1024 - (640 + 1 + 64) = 319$$

$$k = 512 - (256 + 1 + 64) = 191$$

#5 Saving Additions Using Long Trail of 0s

SHA256$_1$ (For H1)			SHA256$_2$ (For H2)		
Round (t)	32 bit W_t (In Hex)	Description	Round(t)	32 bit W_t (In Hex)	Description
0	XXXXXXXX	Last 32 Bits of hashMerkleRoot	0	XXXXXXXX	$H1_0$
1	XXXXXXXX	Timestamp	1	XXXXXXXX	$H1_1$
2	XXXXXXXX	Target	2	XXXXXXXX	$H1_2$
3	XXXXXXXX	Nonce (00000000 to FFFFFFFF)	3	XXXXXXXX	$H1_3$
4	0x80000000	Padding Starts	4	XXXXXXXX	$H1_4$
5	0x00000000	|	5	XXXXXXXX	$H1_5$
6	0x00000000	|	6	XXXXXXXX	$H1_6$
7	0x00000000	|	7	XXXXXXXX	$H1_7$
8	0x00000000	|	8	0x80000000	Padding Starts
9	0x00000000	|	9	0x00000000	|
10	0x00000000	|	10	0x00000000	|
11	0x00000000	|	11	0x00000000	|
12	0x00000000	|	12	0x00000000	|
13	0x00000000	Padding Ends	13	0x00000000	Padding Ends
14	0x00000000	Length 1	14	0x00000000	Length 1
15	0x00000280	Length 2	15	0x00000100	Length 2

#6 Saving Additions With Hard Coding

$SHA256_1$ (For H1)				$SHA256_2$ (For H2)		
Round(t)	32 bit W_t (In Hex)	Description		Round(t)	32 bit W_t (In Hex)	Description
4	0x80000000	Padding Starts		8	0x80000000	Padding Starts
15	0x00000280	Length 2		15	0x00000100	Length 2

- For $SHA256_1$, at round 16, $W_{15}+K_{15}$ can be hardcoded as 0x00000280+0xc19bf174=**0xc19bf3f4**. The same is true in Round 16 for $SHA256_2$ where $W_{15}+K_{15}$ can be hardcoded as 0x00000100+0xc19bf174=**0xc19bf274**.

- A similar technique can be applied to Round 5 for $SHA256_1$ and Round 9 for $SHA256_2$. Hardcode with 0x80000000+0x3956c25b=**0xb956c25b** for $SHA256_1$ and 0x80000000+0xd807aa98=**0x5807aa98** for $SHA256_2$.

#7 Message Scheduler Bypass

For $SHA256_1$	Rounds 5 to 16 (12 in total)
For $SHA256_2$	Rounds 9 to 16 (8 in total)

#8 Constant Message Schedule for SHA256

For $16 \le t \le 63$, we have,

$W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-5}) + W_{t-16}$

Therefore, $W_{16} = \sigma_1(W_{14}) + W_9 + \sigma_0(W_1) + W_0$

Hence, $W_{16} = 0 + 0 + \sigma_0(W_1) + W_0$

For $16 \le t \le 63$, we have,

$W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-5}) + W_{t-16}$

Therefore, $W_{17} = \sigma_1(W_{15}) + W_{10} + \sigma_0(W_2) + W_1$

Hence, $W_{17} = \sigma_1(0x00000280) + 0 + \sigma_0(W_2) + W_1$

#9 Incremental Message Schedule at Round 20 for SHA256[1]

$W_{19} = \sigma_1(W_{17}) + W_{12} + \sigma_0(W_4) + W_3$

Hence, $W_{19} = \sigma_1(W_{17}) + 0 + \sigma_0(0x80000000) + W_3$

W_0	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff
W_1	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff
W_2	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff
W_3	0x00000000	0x00000001	0x00000002	0x00000003	0x00000004
W_{19}	0x1108b75**9**	0x1108b75**a**	0x1108b75**b**	0x1108b75**c**	0x1108b75**d**

#10 Saving Additions by Dynamic Hard Coding for SHA256[1]

Dynamically hardcoded new values:

K_{16} = 0xXXXXXXXX + 0xe49b69c1

K_{17} = 0xXXXXXXXX + 0xefbe4786

K_{19} = 0xXXXXXXXX + 0x240ca1cc

Discussion – Summary of Savings

SHA256 Application	Optimisation	Calculations Saved
$SHA256_0$	#1 - The Calculation of H0 for $SHA256_0$	None
$SHA256_1$	#3 - First 3 Rounds of $SHA256_1$	SHA256 Rounds: 3
	#4 - Round 4 Incremental Calculations for $SHA256_1$	SHA256 Rounds: 1
	#5 - Saving Additions Using the Long Trail of 0s for $SHA256_1$	Mod 2^{32} additions: 10
	#6 - Saving Additions with Hard Coding	Mod 2^{32} additions: 2
	#8 - Constant Message Schedule for $SHA256_1$	2 calculations of Message Scheduler Mod 2^{32} additions: 3*2=6 Bitwise Rotations: 4*2=8 Bitwise Shifts: 2*2=4 Bitwise AND: 0 Bitwise EX-OR: 4*2=8

Discussion – Summary of Savings II

$SHA256_1$	#9 - Incremental Message Schedule Calculation at Round 20 for $SHA256_1$	1 calculation of Message Scheduler Mod 2^{32} additions: 3*1=3 Bitwise Rotations: 4*1=4 Bitwise Shifts: 2*1=2 Bitwise AND: 0 Bitwise EX-OR: 4*1=4
	#10 - Saving Additions by Dynamic Hard Coding for $SHA256_1$	Mod 2^{32} additions: 3
$SHA256_2$	#2 - Early Rejection at Rounds 61 and 62 for SHA2562	SHA256 Rounds: 3
	#5 - Saving Additions Using the Long Trail of 0s for $SHA256_2$	Mod 2^{32} additions: 6
	#6 - Saving Additions with Hard Coding	Mod 2^{32} additions: 2

Total Savings Introduced by the Algorithm Optimisations

$SHA256_1$	SHA256 Rounds: 4 Mod 2^{32} additions: 24 Bitwise Rotations: 12 Bitwise Shifts: 6 Bitwise AND: 0 Bitwise EX-OR: 12
$SHA256_2$	SHA256 Rounds: 3 Mod 2^{32} additions: 8 Bitwise Rotations: 0 Bitwise Shifts: 0 Bitwise AND: 0 Bitwise EX-OR: 0

Savings Factor Calculation

Savings Factor = 60/64 + 61/64 ≈ 0.9375 + 0.9535 ≈

1.891

Additions (Mod 2^{32})	7 + (3*48/64) = 7 + (3*0.75) = 9.25
Bitwise Rotations (ROTR)	6 + (4*48/64) = 6 + (4*0.75) = 9
Bitwise Shifts (SHR)	2*48/64 = 1.5
Bitwise AND (∧)	5
Bitwise EX-OR (⊕)	7 + (4*48/64) = 7 + (4*0.75) = 10

For $SHA256_1$: ((24/9.25)+(12/9)+(6/1.5)+0+(12/10))/5 = (2.5946+1.334+3+0+1.2)/5 ≈ 1.6257

For $SHA256_2$: ((8/9.25)+0+0+0+0)/5 ≈ 0.8649/5 ≈ 0.173

Savings Factor = (60-1.6257)/64 + (61-0.173)/64 ≈ 0.912 + 0.9504 ≈

1.8624

Limitations & Future Work

Limitations:

- Thesis had more of a theoretical approach
- After optimisation, generic SHA256 hashing cannot be performed
- Savings Factor of 1.8624 not entirely accurate but reasonably close
- Optimisations more concentrated towards $SHA256_1$ than $SHA256_2$
- Still more room for improvements and optimisations

Future Work:

- Need for implementation on a common platform
- Performance Comparison of OOTB and Optimised SHA256 for a more accurate rendering of the Savings Factor
- Compatibility Analysis of Algorithm Optimisations with Hardware Optimisations

Conclusion

- Managed to reduce the Bitcoin mining calculation of 2xSHA256 to approximately 1.8624xSHA256
- Entire Bitcoin network currently consuming about 15000 megawatt hour of electricity per day
- The optimisations will lead to an approximate savings of 1000 megawatt hours per day
- This is roughly equivalent to saving about $150000 each day on electricity!
- Optimisation ideas decided to be made public for the betterment of the Bitcoin community

Thank You

Questions?