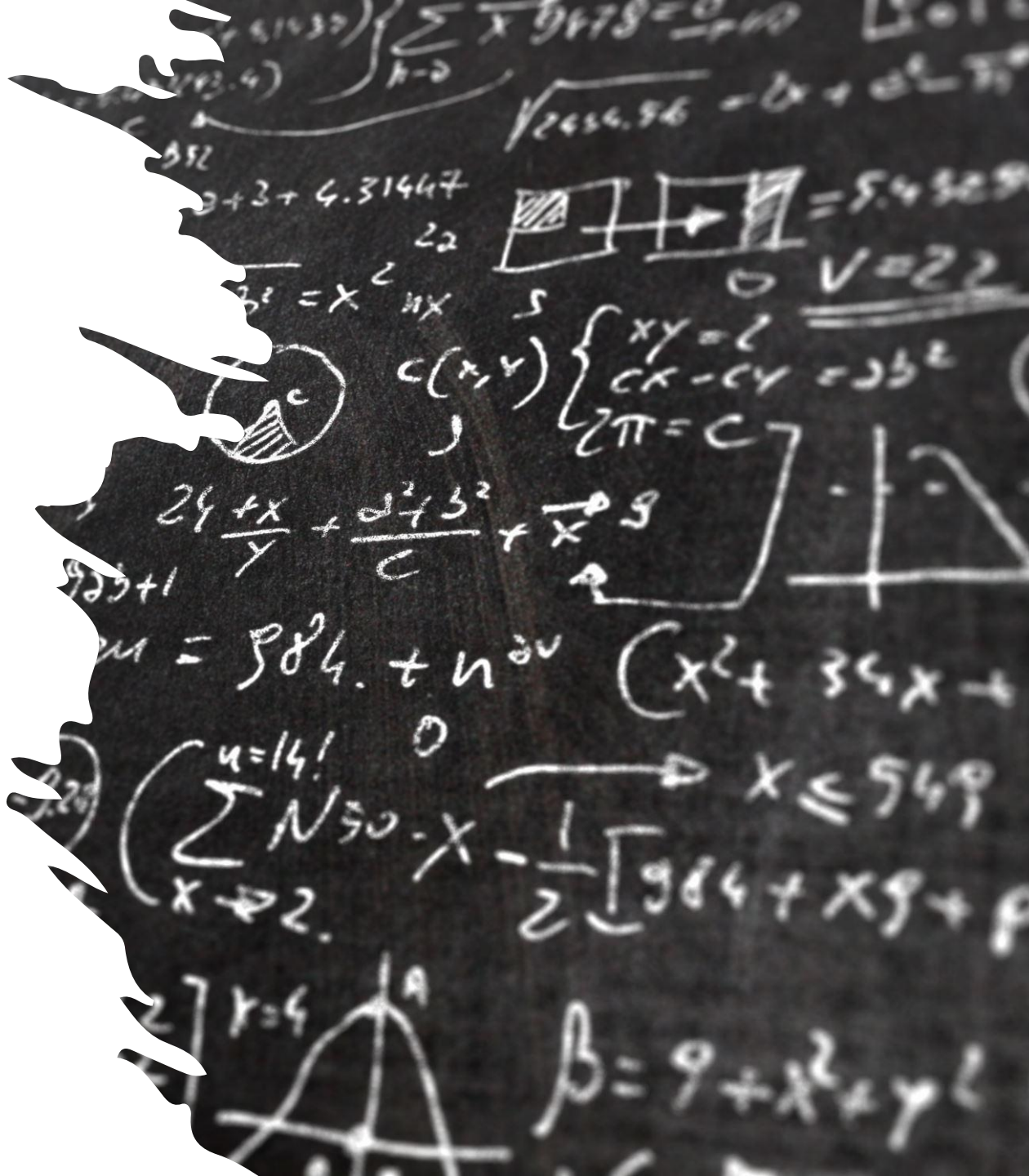
Hash Functions

Liron Shani

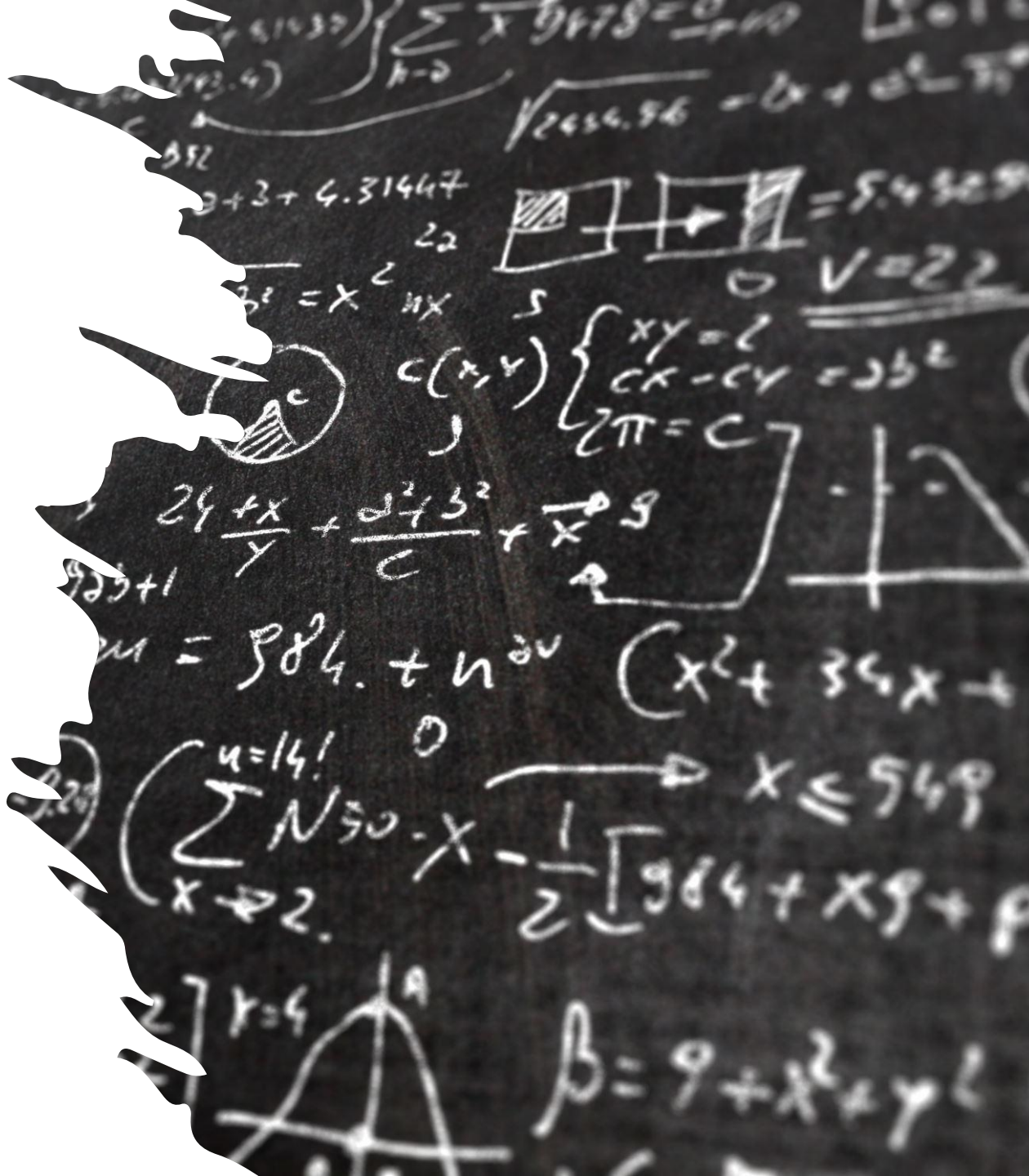
Background

- Hashing algorithms were first introduced in the 1950s by Hans Peter Luhn.
- Checksum algorithm
- The “check” number was used to easily verify if the result matched the final digit of the original number
- Would lead to several new developments of hash functions



Definitions

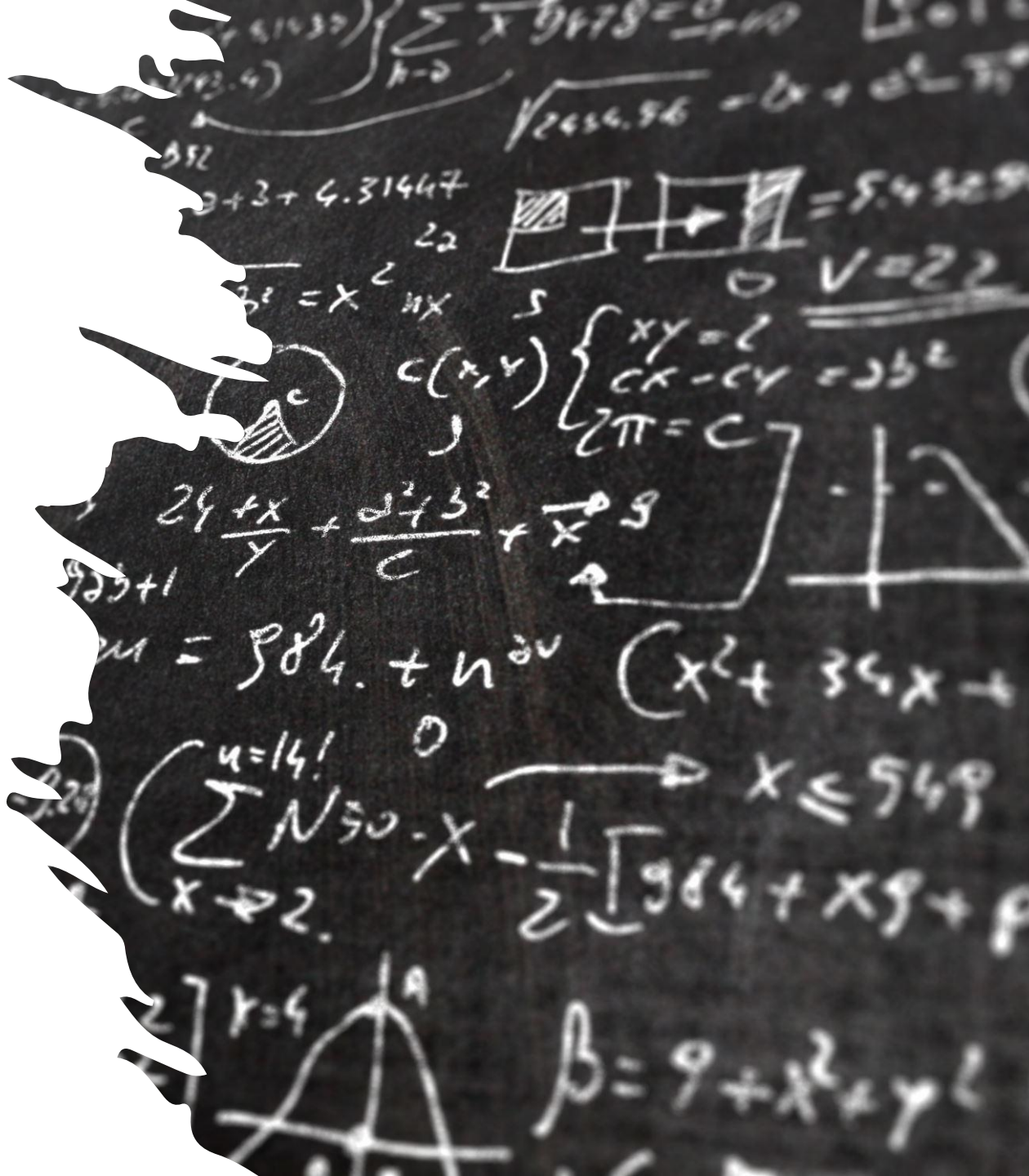
- Hash functions are cryptographic functions than can map data inputs of any size to a fixed-size length of bits.
- Typically run faster than symmetric encryptions
- Properties:
  - Irreversible
  - Collision-resistant
  - Avalanche effect



Quick example

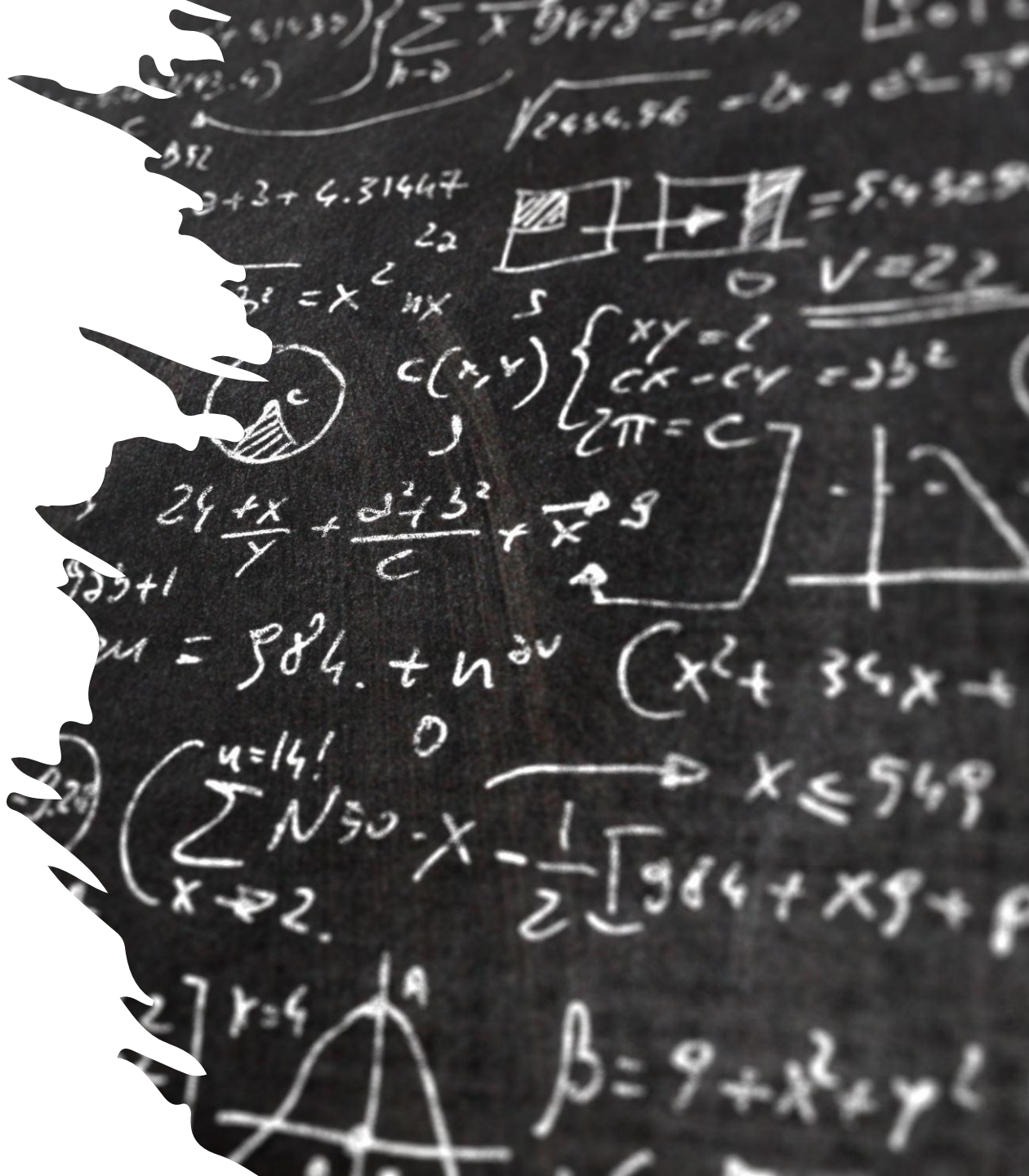
- Message: “test”
- Hashing algorithm: 20+5+19+20=64
- Hash value: 64

- New message: ”best”
- Hashing algorithm: 2+5+19+20=46
- New hash value: 46



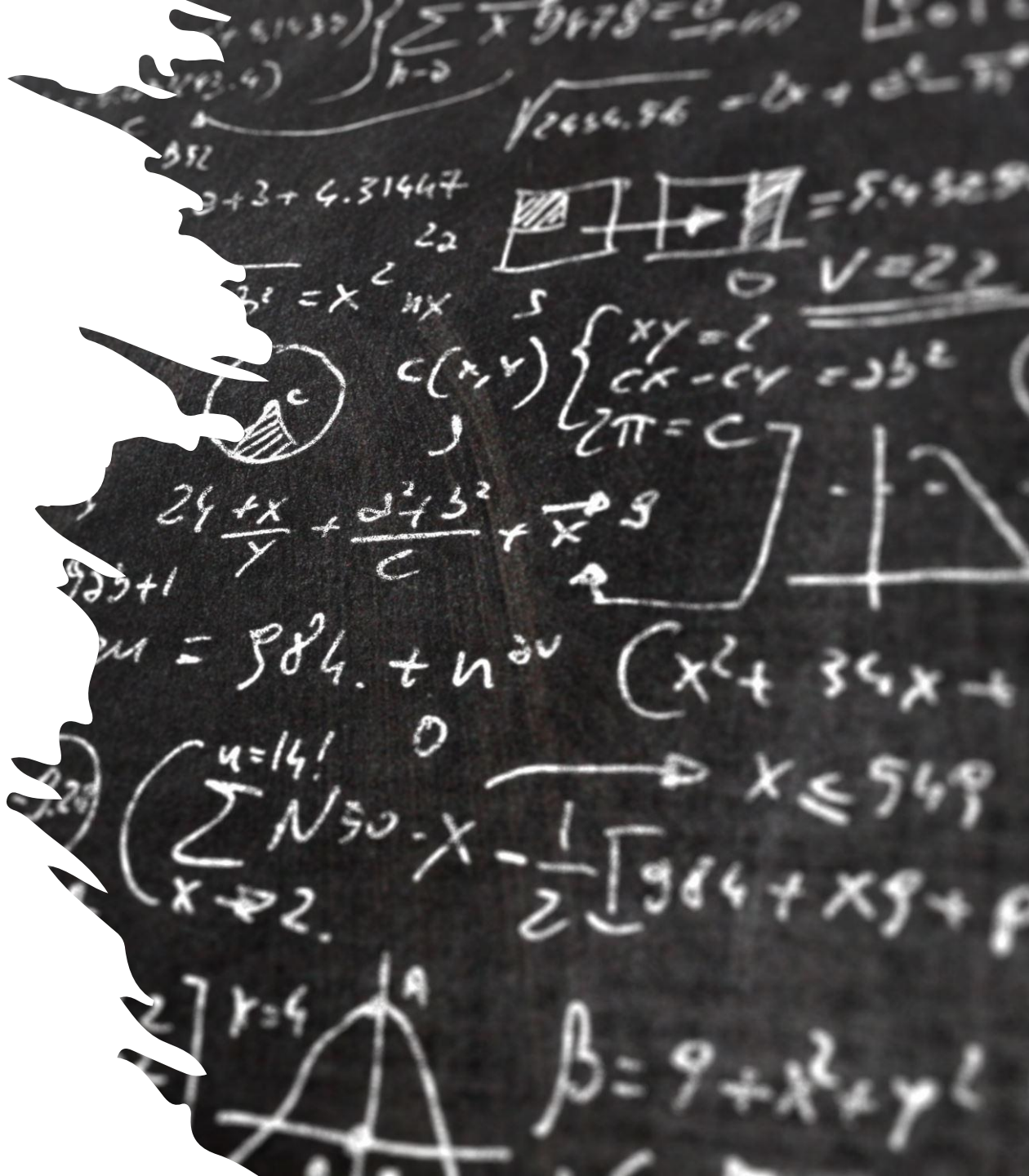
Implementations

- Originally used for organizing and classifying data
- Digital signatures
- Verify message and file integrity
- Used for cryptocurrencies
- Password verification



Example: SHA-1

- Developed and published by the NSA
- 40-digit hexadecimal hash value
- Wide use in the early 2000s
- No longer considered secure



Step 1 – Take an input text and split up each character. Convert to ASCII codes

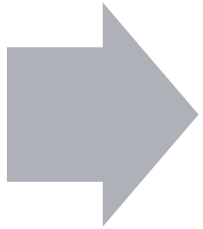
Message: “A Test”

[A, , T, e, s, t]

Result: [65, 32, 84, 101, 115, 116]

Step 2 – Convert ASCII codes to binary

[65, 32, 84, 101, 115, 116]



Result: [1000001, 100000, 1010100, 1100101, 1110011, 1110100]

Step 3 – Attach zeros to the front of each value until they are each 8 bits long

[1000001, 100000, 1010100, 1100101, 1110011, 1110100]

Result: [01000001, 00100000, 01010100, 01100101, 01110011, 01110100]

Step 4 – Join the values and attach a 1 to the end

[01000001, 00100000, 01010100, 01100101, 01110011, 01110100]

Result:
1000001100000101010011001011110011111010 01

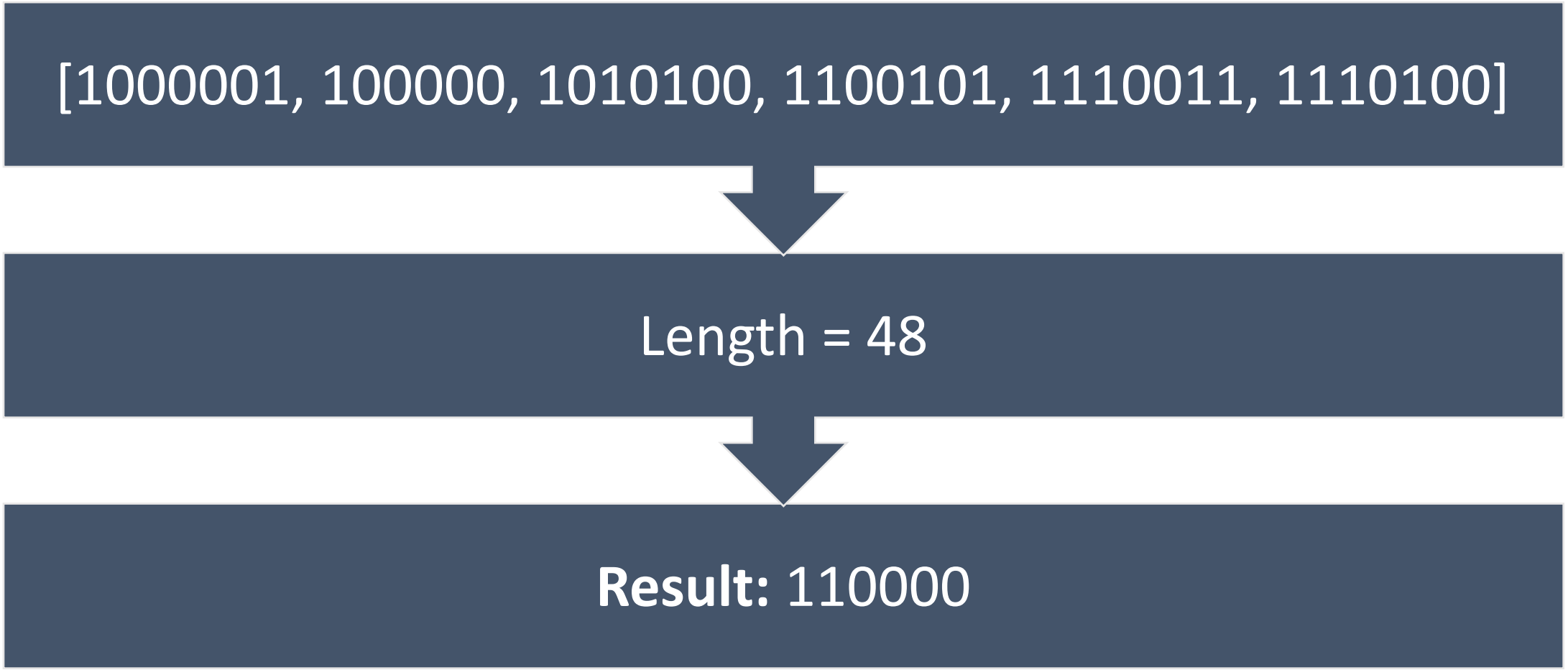
Step 5 – Add zeros to the end until the value's length is 448 characters (or 448 mod(512))

100000110000010101001100101111001111101001

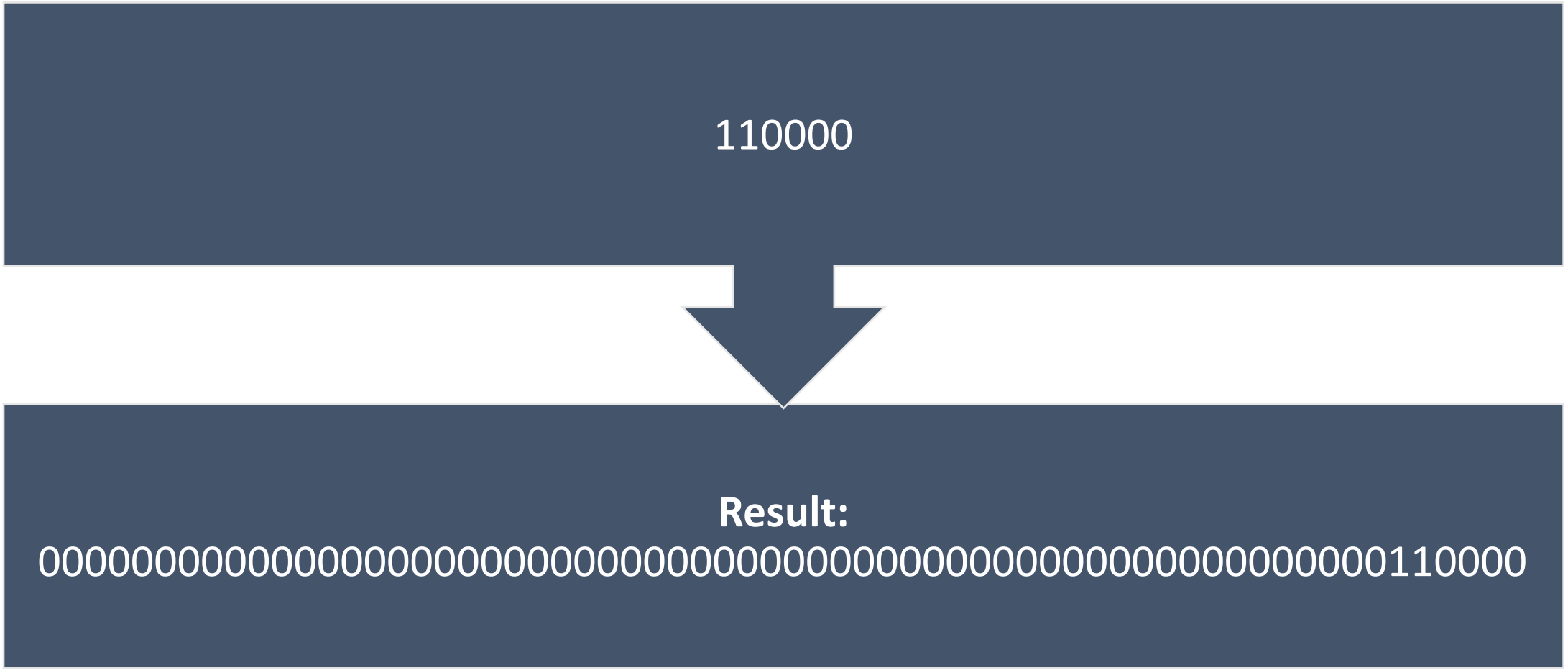
Result:

1000001100000101010011001011110011111010010000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000

Step 6 – Take the values from step 3 and get the total length. Convert that number to binary.



Step 7 – Attach zeros to the front until it has 64 characters



Step 8 – Attach the value from step 7 to the binary message from step 5

Result:
1000001100000101010011001011110011111010010000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000110000

Step 9 – Break up the value into chunks of 512 characters

Result:

[10000011000001010100110010111100111110100100000000
00000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000110000]

Step 10 – Break up the value into smaller chunks of sixteen 32-bit messages

Result:

[[10000011000001010100110010111100, 11111010010000000000000000000000,
00000000000000000000000000000000, 00000000000000000000000000000000,
00000000000000000000000000000000, 00000000000000000000000000000000,
00000000000000000000000000000000, 00000000000000000000000000000000,
00000000000000000000000000000000, 00000000000000000000000000000000,
00000000000000000000000000000000, 00000000000000000000000000000000,
00000000000000000000000000000000, 00000000000000000000000110000]]

Step 11 – Loop through each chunk of sixteen 32-bit messages and extend each to 80 messages using bitwise operations.

Part of the Result:

[[10000011000001010100110010111100,
11111010010000000000000000000000, 00000000000000000000000000000000,
00000000000000000000000000000000, 00000000000000000000000000000000,
00000000000000000000000000000000, 00000000000000000000000000000000,
00000000000000000000000000000000, 00000000000000000000000000000000,
00000000000000000000000000000000, 00000000000000000000000000000000,
00000000000000000000000000000000, 00000000000000000000000000000000,
00000000000000000000000110000,

10000010010000001010100011001010, 11100110111010010000000000000000,
00000000000000000000000001100000, 00000100100000010101000110010101,

…]]

Step 12 – Initialize these variables

Result:

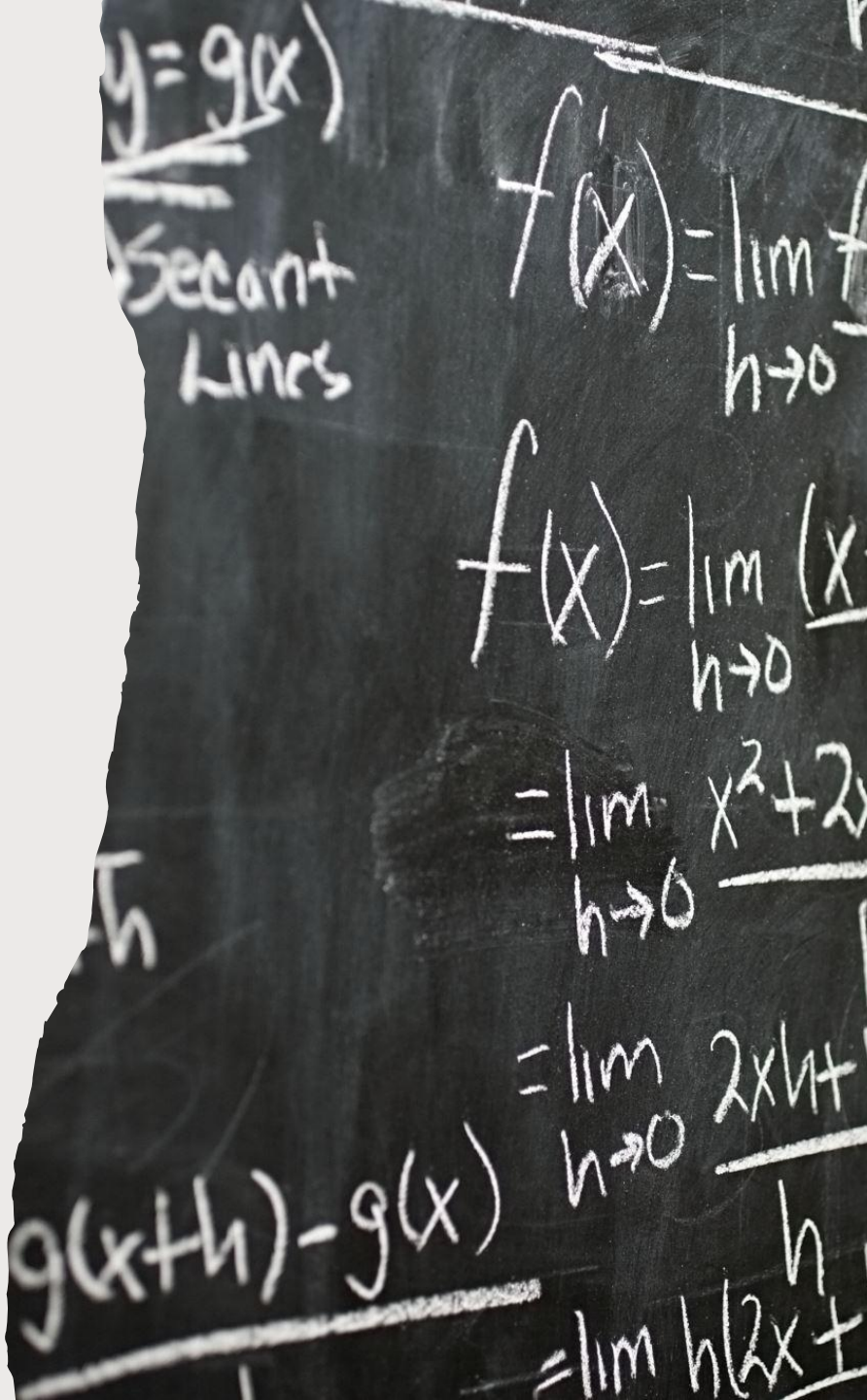
a = 01100111010001010010001100000001

b = 11101111110011011010101110001001

c = 10011000101110101101110011111110

d = 00010000001100100101010001110110

e = 11000011110100101110000111110000



Step 13 – loop through the chunks to perform bitwise operations and variable reassignment on the variables

Result:
a = 10001111000011000000100001010101
b = 10010001010101100011001111100100
c = 10100111110111100001100101000110
d = 10001011001110000111010011001000
e = 10010000000111011111000001000011

Step 14 – Convert each of the binary values to hexadecimal

a = 10001111000011000000100001010101

b = 10010001010101100011001111100100

c = 10100111110111100001100101000110

d = 10001011001110000111010011001000

e = 10010000000111011111000001000011

Result:

a = 8f0c0855

b = 915633e4

c = a7de1946

d = 8b3874c8

e = 901df043

Step 15 – Join the hexadecimals to get the final hash value

a = 8f0c0855

b = 915633e4

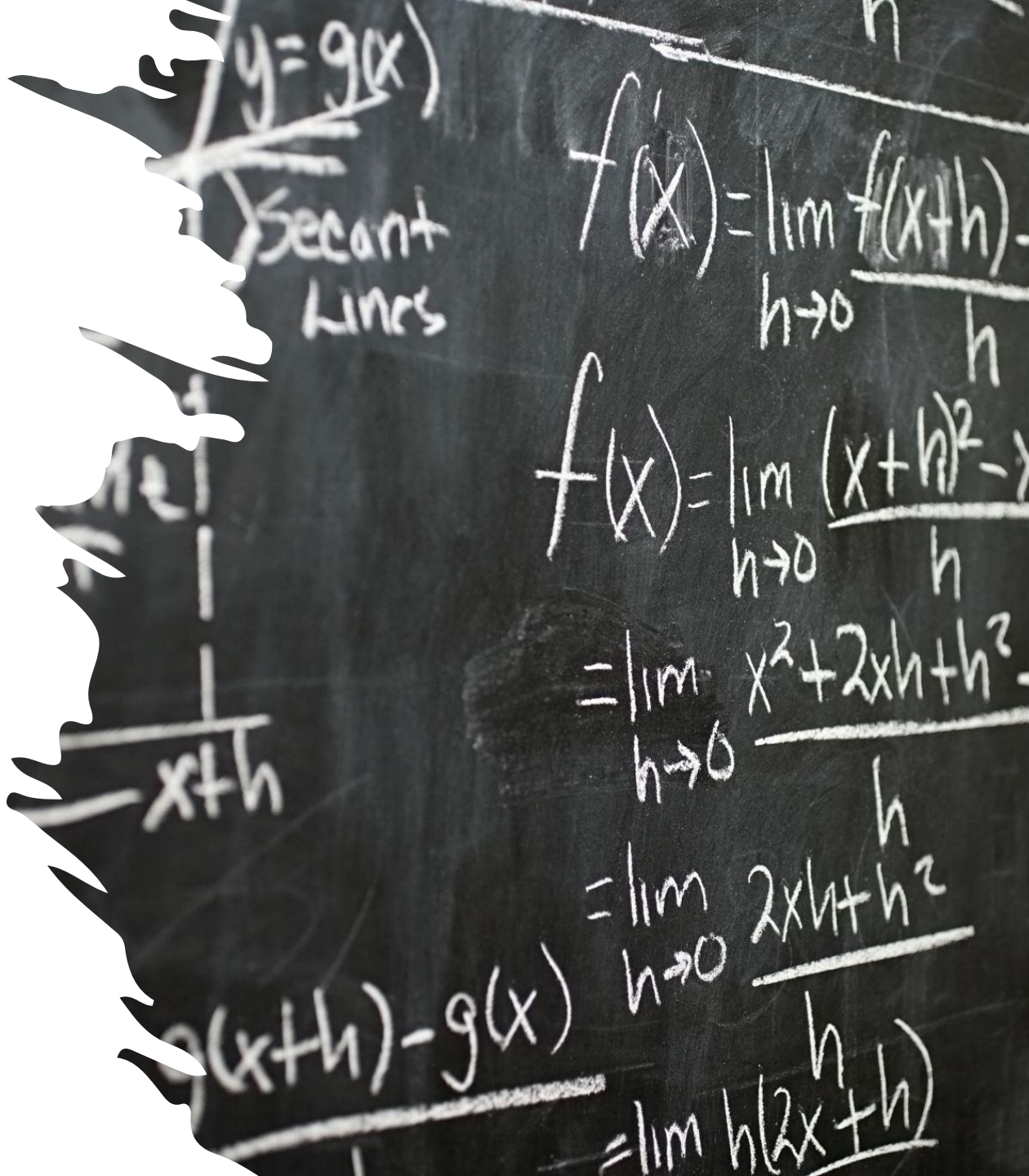
c = a7de1946

d = 8b3874c8

e = 901df043

Final Hash Value:

8f0c0855915633e4a7de19468b3874c8901df043



Citations

- Crane, C. (2023, May 25). *What is a hash function in cryptography? A beginner's guide*. Hashed Out. https://www.thesslstore.com/blog/what-is-a-hash-function-in-cryptography-a-beginners-guide/
- Gallo, K. (2022, August 23). *What is hashing? A guide with examples*. Built In. https://builtin.com/cybersecurity/what-is-hashing
- Okta. (2023, February 14). *Hashing algorithm overview: Types, methodologies & usage*. Okta. https://www.okta.com/identity-101/hashing-algorithms/#:~:text=What%20Are%20Hashing%20Algorithms%20Used,for%20classifying%20and%20organizing%20data.
- Rebecca. (2022, December 1). *What is hashing (hash function) and how does it work?*. History Computer. https://history-computer.com/hashing-guide/
- Fullstack Academy. (2017). *How Does SHA-1 Work - Intro to Cryptographic Hash Functions and SHA-1*. *YouTube*. Retrieved July 22, 2023, from https://www.youtube.com/watch?v=kmHojGMUn0Q.

Thank you for listening