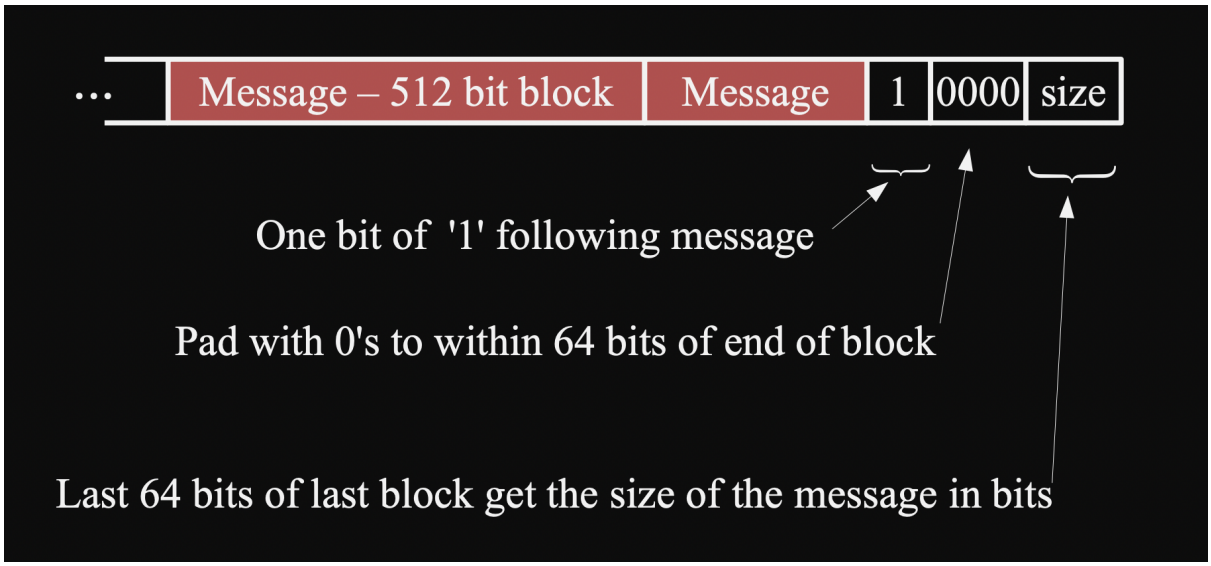
SHA-1 Implementation

SHA-1 (Secure Hash Algorithm) Implementation

Pre-processing of message:

1. End 64 bits represent the 512 bit size of the message.
2. Parity bits are added using 1 and 0
3. Add '1' after the message followed by '0's until the last 64 bits.



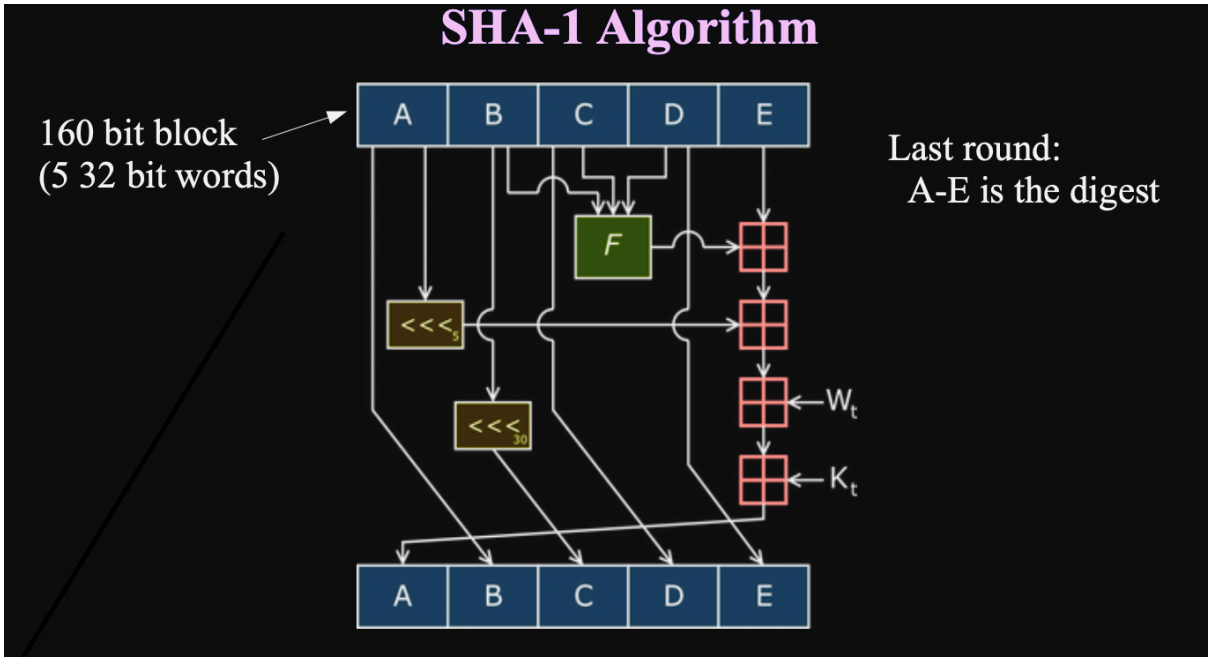
Noob Concepts:

1. Takes an input and generates 160-bit hash value.
2. Input of 512 bits is given to the system -> input block broken down into 16 32 bit words.
3. 16 32 bit words are then extended to 80 words using.

```
for t from 16 to 79
  Wt
 = (Wt-3 ⊕ Wt-8 ⊕ Wt-14 ⊕ Wt-16) <<< 1
```

4 different Rounds in the encryption process:

#define variables A,B,C,D,E,F



The stage block logic of SHA-1 Algorithm

A=0x67452301, B =0xEFCDAB89, C =0x98BADCFE, D =0x10325476, E =0xC3D2E1F0

5. We define F as the following for the different rounds

```
Rounds 1 to 20: F = (B ∧ C) ∨ (¬B ∧ D)
 Kt = 0x5A827999
Rounds 21 to 40: F = (B ⊕ C ⊕ D)
 Kt = 0x6ED9EBA1
Rounds 41 to 60: F = (B ∧ C) ∨ (B ∧ D) ∨ (C ∧ D)
 Kt = 0x8F1BBCDC
Rounds 61 to 80: F = (B ⊕ C ⊕ D)
 Kt = 0xCA62C1D6
```

6. **The next level variables are updated as follows:**

```
N = 2³²
temp = ((A<<5) + F + E + Kt + Wt) mod N
E = D
D = C
C = B<<<30
A = temp
B = A
```

7. **After 80 rounds output the hash.**

Description:

Pre-processing:

1. The input is in the form of string (message digest)
2. 8-bit binary ASCII value

Padding if the input string

- input string can be converted into an array of unsigned long integers

```
vector<unsigned long> convert_to_binary(const string message){
  vector<unsigned long> block;
  for(int i=0;i<message.length();i++){
```

```
    //8-bit ASCII for each one
    bitset<8> message_char(message.substr(i,1));
    block.push_back(message_char.to_ulong());
  }
  return block;
}
```

- Padding relation:

```
int l = block.size()*8; //8-bit for every ASCII character
k -> number of 0 used for padding.
given by the relation
(l+1+k) = 448 mod 512;
for the base case:
k = 447-l;
unsigned long padd_1 = 0x80; //to make sure no zeroes are added to the front
block.push_back(padd_1);
k = k-7;
for(int i=0;i<k/8;i++) block.push_back(0x00000000);

The last 64 bits represent the size of the message in bits (l)
bitset<64> 64_bit_size(l);
string 64_bss = 64_bit_size.to_String();
bitset<8> te(64_bss.substr(0,8));
block.push_back(te.to_ulong());
for(int i=8;i<64;i+=8)
  bitset<8> te2(64_bss.substr(i,8));
  block.push_back(te2.to_ulong());
//padding done
```

- **Resizing the block**

```
vector<unsigned long> resize_block(vector<unsigned long> input)
  vector<unsigned long> output(16);
  //16 32bit words
  for(int i=0;i<64;i+=4)
    bitset<32> temp(0);
    temp |= (unsigned long)input[i]<<24;
    temp |= (unsigned long)input[i+1]<<16;
    temp |= (unsigned long)input[i+2]<<8;
    temp |= (unsigned long)input[i+3];
    output[i/4] = temp.to_ulong();
```

Computing Hash of the resized message

- **Storing the keys**

```
vector<unsigned long> k[4] = {0x5A827999, 0x6ED9EBA1, 0x8F1BBCDC, 0xCA62C1D6}
//(Wt-3 ⊕ Wt-8 ⊕ Wt-14 ⊕ Wt-16) <<< 1
//RIGHTROTATE
#define LFTROT(word,bit) ((word<<bit) | (word>>(32-bit)))
vector<unsigned long> words_for_rounds(vector<unsigned long>& block)
  for(int i=16;i<80;i++)
    bitset<32> temp(0),t1(to_string(block[i-3])),t2(to_string(block[i-8])),t3(to_string(block[i-14])),t4(to_string(block[i-16]));
    temp |= (t1^t2^t3^t4);
    temp = LFTROT(temp,1);
    block.push_back(temp.to_ulong());


//FINALLY THE BLOCK ARRAY CONTAINS ALL THE MESSAGE WORDS.
```

Computing hashes of the message digest

- Divided into 4 different rounds with four different keys
- Assume that all the hash variables have been initialised.
- Append the hash to the variable

```
unsigned long k, f;
  unsigned long A,B,C,D,E;
  unsigned long N = 4294967296;
  bitset<32> a(A),b(B),c(C),d(D),e(E),temp(0);
  for(int i=0;i<20;i++){
    k = 0x5A827999
    f = (b & c) | (~b & d);
    temp = (LFTROT(a,5).to_ulong()+f.to_ulong()+e.to_ulong()+k+block[i])%N;
    e = d;
    d = c;
    c = LFTROT(b,30);
    b = a;
    a = temp;
  }
//update the hashes
```