

CS553: Cryptography

Assignment 8: Solutions

Rohit Das (11910230)

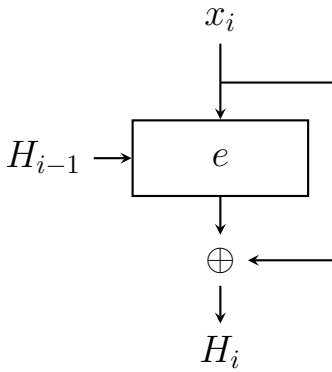
November 22, 2019

1. Authenticated Encryption Schemes

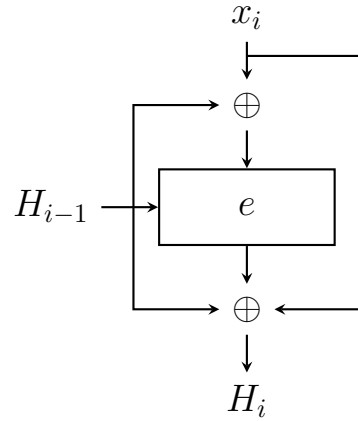
1. **Nyberg and Rueppel Scheme:** It is an application of their signature scheme, which was the first discrete logarithm based scheme with message recovery.
2. **Keyak:** It is a scheme based on Keccak-p. It takes as input a secret and unique value (SUV), then some associated data (or metadata) that are authenticated but not encrypted and finally some plaintext.
3. **Wu and Hsu's Convertible Scheme:** A new authenticated encryption scheme with (t,n) shared verification based on discrete logarithms was proposed.
4. **Hwang's AE Scheme with Message Linkage:** To reduce costs for redundancies in message blocks, the scheme is divided into two parts, generation of signature and encryption phase, and recovery message and verification phase.
5. **Galois/Counter Mode (GCM) for Block Ciphers:** It is an authenticated encryption algorithm designed to provide both data authenticity and integrity.
6. **Carter-Wegman Counter Mode (CWC):** It combines the use of CTR mode for encryption with an efficient polynomial Carter-Wegman MAC.
7. **Tiaoxin-346:** It is a nonce-based scheme, taking 4 inputs: Key K (128 bits), public nonce IV (128 bits), plaintext M (0 to $2^{128} - 1$ bits) and associated data AD (0 to $2^{128} - 1$ bits).
8. **Offset Codebook Mode (OCB):** It is essentially a scheme to integrate a MAC into the operation of a block cipher.
9. **EAX:** It is an AEAD scheme designed to simultaneously provide both authenticity and privacy with a two-pass scheme.
10. **Sophie Germain Counter Mode (SGCM):** Instead of the binary field $GF(2^{128})$, it used modular arithmetic in $GF(p)$, where p is a safe prime $2^{128} + 12451$ with corresponding Sophie Germain prime $\frac{p-1}{2} = 2^{127} + 6225$.

2. Hash: Mode-of-operation + collision

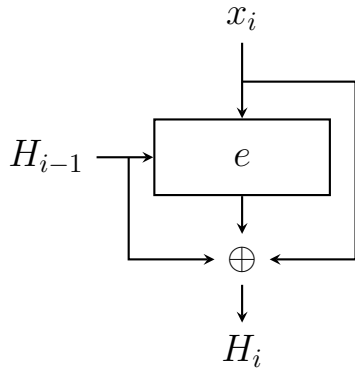
2.1 Pg. 315, Problem 11.3



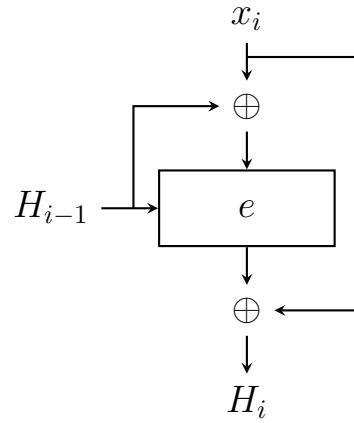
1. $e(H_{i-1}, x_i) \oplus x_i$



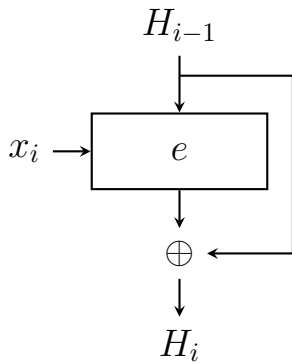
2. $e(H_{i-1}, x_i \oplus H_{i-1}) \oplus x_i \oplus H_{i-1}$



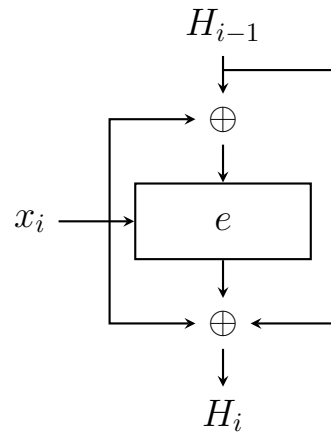
3. $e(H_{i-1}, x_i) \oplus x_i \oplus H_{i-1}$



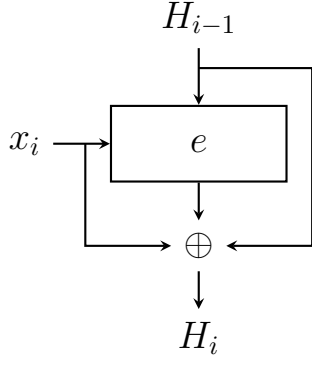
4. $e(H_{i-1}, x_i \oplus H_{i-1}) \oplus x_i$



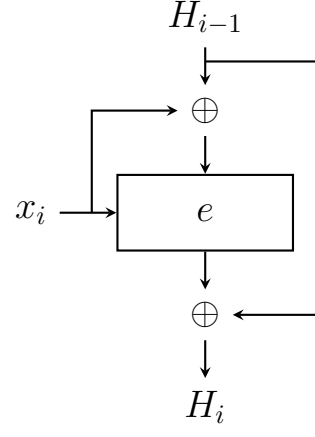
5. $e(x_i, H_{i-1}) \oplus H_{i-1}$



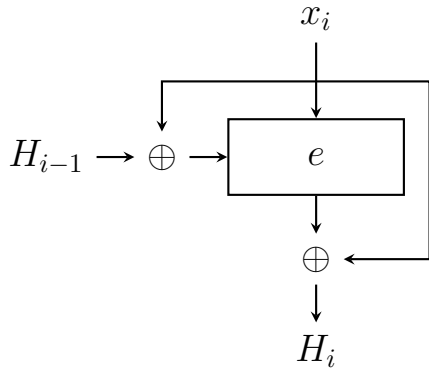
6. $e(x_i, x_i \oplus H_{i-1}) \oplus x_i \oplus H_{i-1}$



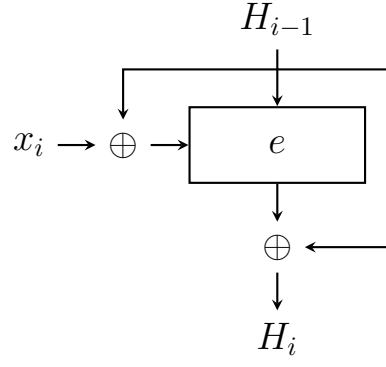
7. $e(x_i, H_{i-1}) \oplus x_i \oplus H_{i-1}$



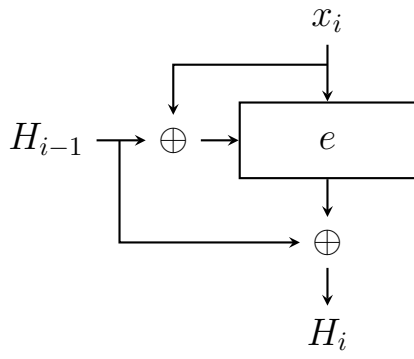
8. $e(x_i, x_i \oplus H_{i-1}) \oplus H_{i-1}$



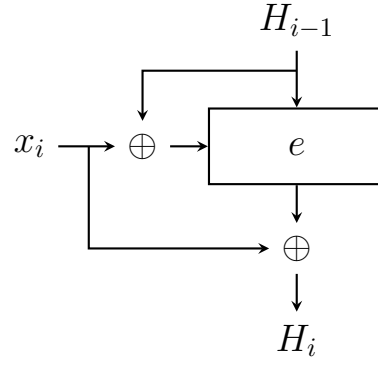
9. $e(x_i \oplus H_{i-1}, x_i) \oplus x_i$



10. $e(x_i \oplus H_{i-1}, H_{i-1}) \oplus H_{i-1}$



11. $e(x_i \oplus H_{i-1}, x_i) \oplus H_{i-1}$



12. $e(x_i \oplus H_{i-1}, H_{i-1}) \oplus x_i$

2..2 Pg. 316, Problem 11.8

Python Code (Python3): hashing.py

```
from functools import reduce

def hashing(msg):
    print("String:", msg)
    print("Binary:", ' '.join(['{:08b}'.format(ord(m), 'b') + ' '
        ↪ for m in msg]))
    hashed = ''
    for m in msg:
        hashed += str(reduce(lambda i, j: int(i) ^ int(j),
            ↪ format(ord(m), 'b')))
    print("Hash:", hashed)

hashing('CRYPTO') # 110011
hashing('FINALE')
```

Output: The hash function can be broken by choosing such characters whose bits will XOR out to be the XOR of bits of the target character. E.g., "CRYPTO" hashes to 110011, which is the same for "FINALE". The crucial property missing in this hash function is the second-preimage resistance, i.e., given a message M_1 , we are easily able to find another message M_2 hashing to the same value as M_1 .

```
String: CRYPTO
Binary: 01000011 01010010 01011001 01010000 01010100 01001111
Hash: 110011
String: FINALE
Binary: 01000110 01001001 01001110 01000001 01001100 01000101
Hash: 110011
```

3. Partial Collision: md5

The list of 2^{16} pairs of strings and their corresponding hashes was generated 4 times to find collision in the first and last 16 bits of the hash. Hence, the total number of trials stands at $2^{18} = 262144$. The strings have been generated randomly and hence may not form meaningful sentences.

Python Code (Python3): md5hash.py

```
# please run the code more than once to generate the collision

import string
import random
import hashlib
import os

def random_string(n): # to generate random string having
    ↪ letters and digits
    return ''.join(random.choices(string.ascii_letters +
    ↪ string.digits + ' ', k = n))

def hash_list(n,t): # to create a list of string and its hash
    ctr = 0
    str_hash = []
    for i in range(2 ** 16):
        str1 = random_string(n)
        hash1 = hashlib.md5(str1.encode()).hexdigest()
        list1 = []
        list1.append(str1)
        list1.append(hash1)
```

```

        str_hash.append(list1)
    return str_hash

def find_collision(n,t): # looking for collision in hash_list
    str_hash = hash_list(n,t)

    str_hash.sort(key = lambda x: x[1])

    file1 = open('string1.txt','w')
    file2 = open('string2.txt','w')

    for i in range(len(str_hash) - 1):
        if str_hash[i][1][:4] == str_hash[i + 1][1][:4] and
            ↪ str_hash[i][1][-4:] == str_hash[i + 1][1][-4:]:
            print(str_hash[i][0],str_hash[i][1])
            file1.write(str_hash[i][0])

            print(str_hash[i + 1][0],str_hash[i + 1][1])
            file2.write(str_hash[i + 1][0])

    file1.close()
    file2.close()

find_collision(100,4)

```

Output:

```

→ python_code git:(master) x python3 md5hash.py
→ python_code git:(master) x python3 md5hash.py
→ python_code git:(master) x python3 md5hash.py
→ python_code git:(master) x python3 md5hash.py
→ python_code git:(master) x python3 md5hash.py
tG8S4jr0cLP0aeGp0ln4gRYSXR5wAeBGBMYSW 6iQ5ComIdZk8Zf50KIeRz00rCavsf3fU2VFNzkHCwaNRsCaaFzB8hkFW1EyZvn f16deac24f10ada19a8d7f641f1defbc
8dKJ869vLFivQZcy7iTLtEQH1vDjDdY0duJk3gHEvfCh0sz7eCvQ m2A8peHiyZHqCwu kyt eBESINKR43D2hAHyMR3QjwB3bYLd f16dffcae8a5281a8f0aba1c7f7fefbc

```

4. SSL Certificates: X509 Format

4.1 About X.509

X.509 is a standard defining the format of public key certificates. An X.509 certificate contains a public key and an identity, and is either signed by a certificate authority, or self-signed.

In this system, an organization that wants a signed certificate requests one via a certificate signing request (CSR). To do this, it first generates a key pair, keeping the private key secret and using it to sign the CSR. This contains information identifying the applicant and the applicant's public key that is used to verify the signature of the CSR - and the Distinguished Name (DN) that the certificate is for. The CSR may be accompanied by other credentials or proofs of identity required by the certificate authority.

There are several commonly used filename extensions for X.509 certificates. Some of these extensions are also used for other data such as private keys.

- .pem - (Privacy-enhanced Electronic Mail) Base64 encoded DER certificate, enclosed between "—BEGIN CERTIFICATE—" and "—END CERTIFICATE—"
- .cer, .crt, .der - usually in binary DER form, but Base64-encoded certificates are common too (see .pem above)
- .p7b, .p7c - PKCS# 7 SignedData structure without data, just certificate(s) or CRL(s)
- .p12 - PKCS# 12, may contain certificate(s) (public) and private keys (password protected)
- .pfx - PFX, predecessor of PKCS# 12 (usually contains data in PKCS# 12 format, e.g., with PFX files generated in IIS)

4.2 SSL verification

Command to download SSL certificate from `www.facebook.com`:

```
openssl s_client -connect www.facebook.com:443 2>/dev/null </  
↪ dev/null | openssl x509 -outform PEM > fb_cert.pem
```

Command to verify SHA-1 fingerprint of above certificate:

```
openssl x509 -noout -in fb_cert.pem -fingerprint -sha1
```

Generated Fingerprint:

SHA1 Fingerprint=A4:27:D4:9C:21:BD:B6:E4:52:B3:F8:D6:DE:25:79:3A:8C:0E:45:2A

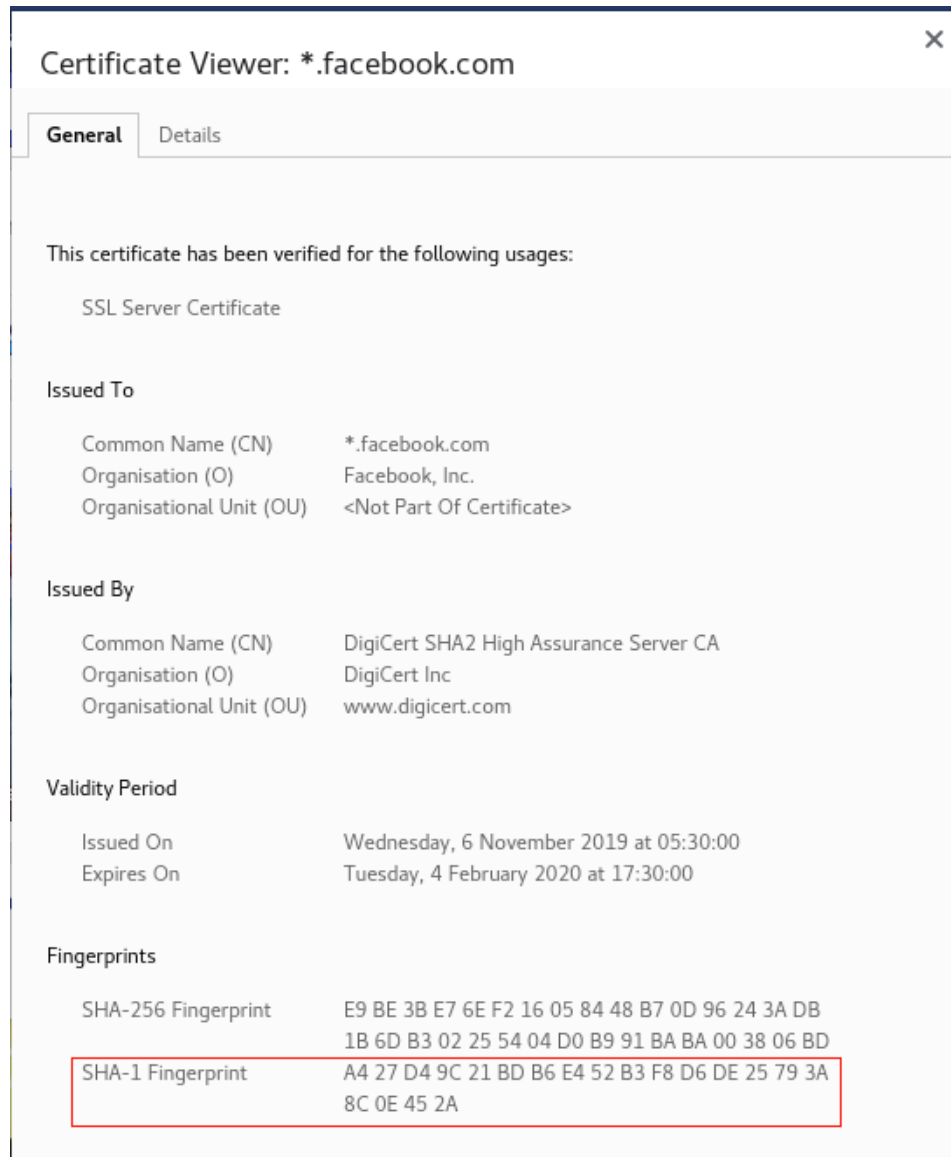


Figure 1: Screenshot from Google Chrome browser of actual fingerprint

5. MAC: Timing Attacks

Python Code (Python3): timing_attack.py

```
from time import time

def compare_mac(x, y, n):
    for i in range(n):
        if x[i] != y[i]:
            return False
    return True

def const_compare_mac(x,y,n):
    result = 0
    for i in range(n):
        result |= ord(x[i]) ^ ord(y[i])
    return result

def timing_measure(mac1,mac2, trials ,compare_mac):
    # each call checks the whole string
    print("Using ",compare_mac.__name__," : ")
    start = time()
    for i in range(trials):
        compare_mac(mac1,mac1,len(mac1))
    end = time()
    print('For same string: %0.5fs' % (end - start))

# each call checks only till mismatch appears
start = time()
```

```

for i in range(trials):
    compare_mac(mac1, mac2, len(mac1))
end = time()

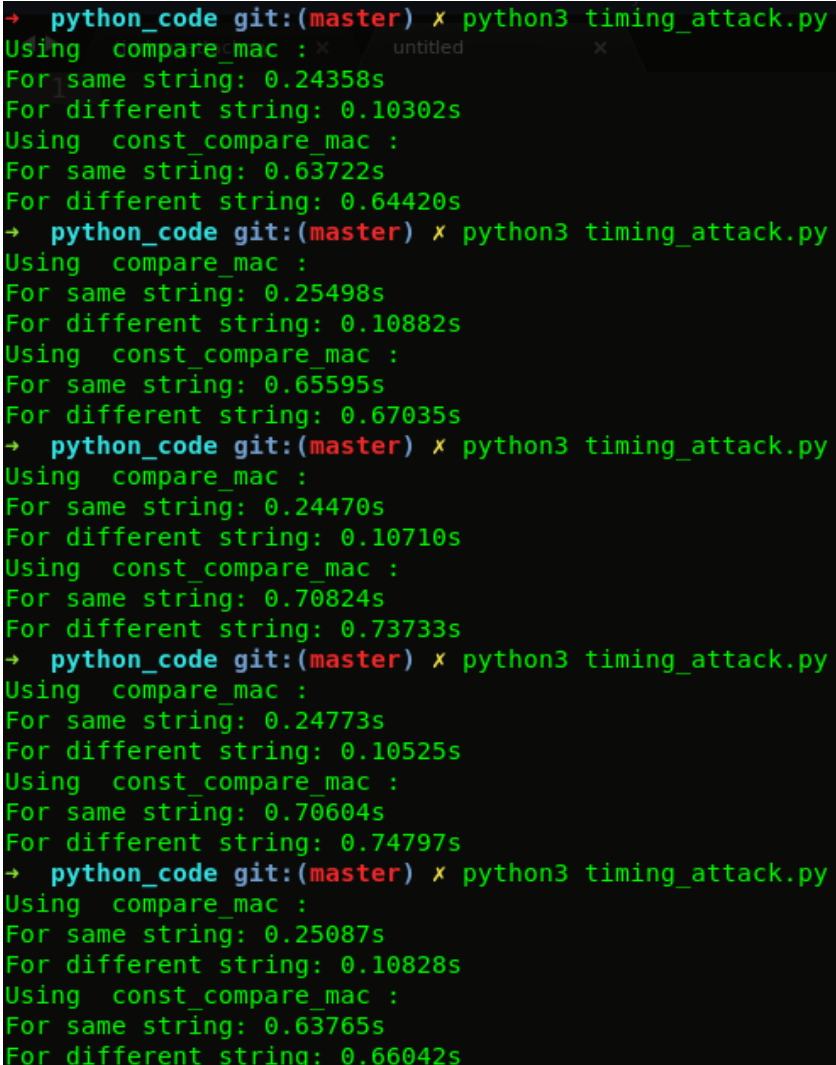
print('For different string: %0.5fs' % (end - start))

timing_measure('0123456789abcdef', '01X3456789abcdef', 100000,
    ↪ compare_mac)

timing_measure('0123456789abcdef', '01X3456789abcdef', 100000,
    ↪ const_compare_mac)

```

Output:



```

→ python_code git:(master) x python3 timing_attack.py
Using compare_mac :
For same string: 0.24358s
For different string: 0.10302s
Using const_compare_mac :
For same string: 0.63722s
For different string: 0.64420s
→ python_code git:(master) x python3 timing_attack.py
Using compare_mac :
For same string: 0.25498s
For different string: 0.10882s
Using const_compare_mac :
For same string: 0.65595s
For different string: 0.67035s
→ python_code git:(master) x python3 timing_attack.py
Using compare_mac :
For same string: 0.24470s
For different string: 0.10710s
Using const_compare_mac :
For same string: 0.70824s
For different string: 0.73733s
→ python_code git:(master) x python3 timing_attack.py
Using compare_mac :
For same string: 0.24773s
For different string: 0.10525s
Using const_compare_mac :
For same string: 0.70604s
For different string: 0.74797s
→ python_code git:(master) x python3 timing_attack.py
Using compare_mac :
For same string: 0.25087s
For different string: 0.10828s
Using const_compare_mac :
For same string: 0.63765s
For different string: 0.66042s

```

Figure 2: As evident from the screenshot, for `compare_mac()`, comparing identical strings takes more time than for different strings. However, it is not so distinguishable in `const_compare_mac()`.