MTAT.07.017
Applied Cryptography

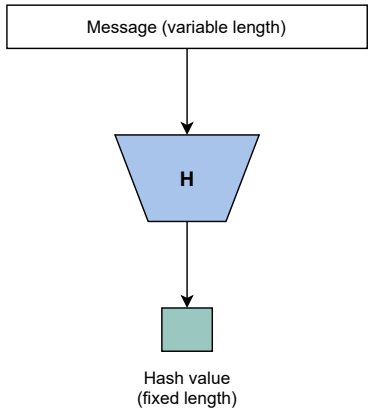
Hash functions

University of Tartu

Spring 2023

Hash function

A hash function is a function that takes an arbitrary block of data and returns a fixed-size unique bit string representation.



Related terms: hash, message digest, fingerprint, checksum

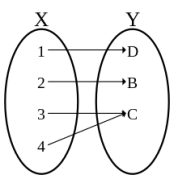
Cryptographic hash function

Properties:

- Easy to compute hash value (fast)
- Hard to restore message from hash (one-way)
- Hard to find messages with the same hash (collision resistant)
- Similar messages have very different hashes (avalanche effect)

Attacks:

- Collision attack:
  $hash(x) = hash(y)$ | find any x and y such that $x \neq y$
- (First) Preimage attack:
  $hash(x) = h$ | given h, find any x
- Second preimage attack:
  $hash(x) = hash(y)$ | given x and hash(x), find $y \neq x$



Security Level

"In cryptography, a **brute-force attack**, or exhaustive key search [..] might be used when it is not possible to take advantage of other weaknesses in an encryption system. It consists of systematically checking all possible keys or passwords until the correct one is found."

`https://en.wikipedia.org/wiki/Brute-force_attack`

E.g.: If a cryptosystem which has a 56-bit key can be brute-forced using 2^{56} operations[1] then the cryptosystem has a security level of 56 bits.

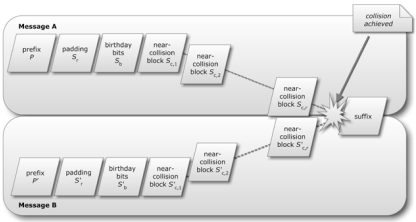
- 2^{128} operations infeasible
- 2^{80} become feasible

Note, 2^{81} operations take twice the time of 2^{80} (2^{128} vs 2^{256})

[1] The term "operation" is not defined.

Cryptographic hash functions

- MD5 – 128-bit output
  - collision attack in $2^{24.1}$ (brute-force in 2^{64})
    - chosen-prefix collision attack in 2^{39}



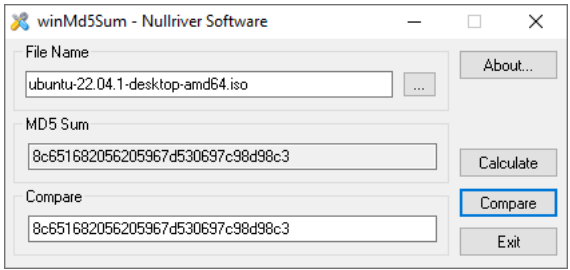

```
C:\TEMP> md5sum hello.exe
cdc47d670159eef60916ca03a9d4a007
C:\TEMP> .\hello.exe
Hello, world!
C:\TEMP> md5sum erase.exe
cdc47d670159eef60916ca03a9d4a007
C:\TEMP> .\erase.exe
This program is evil!!!
Erasing hard drive...1Gb...2Gb... just kidding!
Nothing was erased.

(press enter to quit)
C:\TEMP>
```

    - preimage and second preimage attack in $2^{123.4}$ (brute-force in 2^{128})
- SHA-1 – 160-bit output
  - collision attack in $2^{61.2}$ (brute-force in 2^{80})
    - chosen-prefix collision attack in $2^{63.4}$
- SHA-256 – 256-bit output
- SHA-512 – 512-bit output
- SHA-3 – 224/256/384/512-bit output

Data identification and integrity verification

- Integrity and authenticity of distributed files



- Distribution of MS Windows updates
- Disk imaging in digital forensics
- Remote file comparison (rsync)

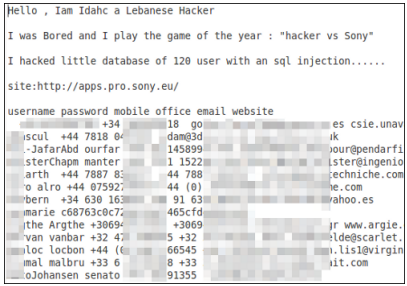
Server-side password storage

LAW & DISORDER / CIVILIZATION & DISCONTENTS

Sony hacked yet again, plaintext passwords, e-mails, DOB posted

The hackers of Lulz Security have broken into yet more Sony websites, this …

by **Peter Bright** - June 3 2011, 4:06am EEST



- Solution – store password hashes in database
  - Compare received plaintext password with hash from db

Server-side password storage

```
sql> SELECT username, password FROM users;
+------------+------------------------------------------+
| username   | password                                 |
+------------+------------------------------------------+
| jeff       | b1b3773a05c0ed0176787a4f1574ff0075f7521e |
| katrin     | 2730f2c29354932611d328cfff0c9f01e10328ec |
| mike       | e72e941812b920c908bba17798d5e27ebf627912 |
+------------+------------------------------------------+
```

Free Password Hash Cracker

Enter up to 10 hashes:

```
b1b3773a05c0ed0176787a4f1574ff0075f7521e
2730f2c29354932611d328cfff0c9f01e10328ec
e72e941812b920c908bba17798d5e27ebf627912
```

Supports: LM, NTLM, md2, md4, md5, md5(md5), md5-half, sha1, sha1(sha1_bin()), sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+

Hash	Type	Result
b1b3773a05c0ed0176787a4f1574ff0075f7521e	sha1	qwerty
2730f2c29354932611d328cfff0c9f01e10328ec	sha1	hillary999
e72e941812b920c908bba17798d5e27ebf627912	Unknown	Not Found

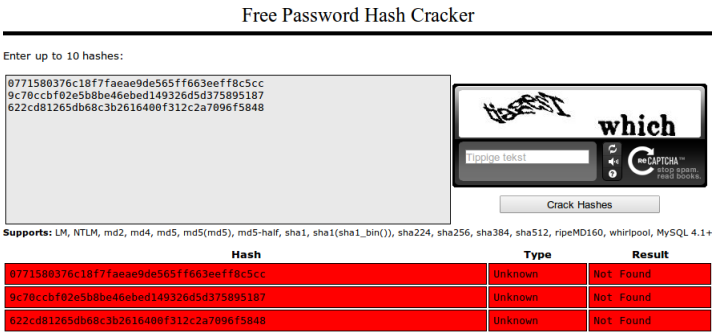
hilllary99

- Solution – add user-specific salt to the password

```
db_salt = os.urandom(8).hex()
db_password = hashlib.sha1((password + db_salt).encode()).hexdigest()
```

Server-side password storage

```
sql> SELECT username, password, salt FROM users;
+------------+------------------------------------+------------------+
| username   | password                           | salt             |
+------------+------------------------------------+------------------+
| jeff       | 0771580376c18f7faeae9de565ff663eeff8c5cc | 7d3a5ccd7fc28aa9 |
| katrin     | 9c70ccbf02e5b8be46ebed149326d5d375895187 | df9372246bfcd8d0 |
| mike       | 622cd81265db68c3b2616400f312c2a7096f5848 | 9a73764e2bf40db8 |
+------------+------------------------------------+------------------+
```



- Not feasible to build a lookup table for every possible salt
- Even the same passwords will have a different hash

Server-side password storage

- Brute-force cracking still possible:

```
>>> hashlib.sha1(('qwerty'+'7d3a5ccd7fc28aa9').encode()).hexdigest()
'0771580376c18f7faeae9de565ff663eeff8c5cc'
```

MD5	23,070.7 M/s
SHA-1	7,973.8 M/s
SHA-256	3,110.2 M/s
SHA-512	267.1 M/s
NTLM	44,035.3 M/s
DES	185.1 M/s
WPA/WPA2	348.0 k/s

Table: GPU speed

- Slow down brute-force by arbitrary factor using iterated hash: $h(h(h(h(x))))$
  - Trade-off: performance vs cost of brute-forcing
  - Goal is to increase asymmetry
- Recommended to use: bcrypt, scrypt (RAM-intensive)

Server-side password storage

Exercise: Calculate the security level of password hashes stored in a company's database. The company's password security policy requires the password to be exactly 7 lowercase letters.

- Number of letters in the English alphabet: 26
- Hash operations required:
  - to brute-force 1-letter password: 26
  - to brute-force 2-letter password: 26×26
  - to brute-force 3-letter password: $26 \times 26 \times 26$
  - to brute-force 7-letter password: 26^7

$$26^7 = 8031810176 \approx 2^{32}$$

If using 2000 iterations:

$$2^{32} \times 2000 = 2^{32} \times 2^{11} \approx 2^{43}$$

Password storage has a 43-bit security level.[2]

[2] Assuming users choose letters randomly.

Commitment scheme

Here is the proof of my clairvoyant powers – the next U.S. president will be SHA256(x)=d99d0b129d5864e1813438a885034452...

- Binding – due to collision resistance of SHA256
- Hiding – due to one-wayness of SHA256

```
>>> for candidate in ['Joe Biden', 'Bernie Sanders', 'Donald Trump']:
...     print(hashlib.sha256(candidate.encode()).hexdigest(), candidate)
...
d99d0b129d5864e1813438a885034452... Joe Biden
d181f00b6eb6ae128e950e8a2bc39a51... Bernie Sanders
e4f2e1f0e2ae4d3ce7018cf3b4f3577c... Donald Trump
```

- Improve hiding property by adding randomness

```
>>> prediction = b'Joe Biden|' + os.urandom(16)
>>> hashlib.sha256(prediction).hexdigest()
'9b8ca016ea530921e2b7bd9046424441...'
>>> prediction
b'Joe Biden|)\xdeg\x14\xb0\xb1h\x88\x83<%u\xa6x\xfe\xc5'
```

How can the committment scheme be implemented without using cryptography?

Coin flipping over phone

How to generate a random number from 0 to 99 over the phone:

1. Alice: "my commitment value (SHA256(x)) is 108c995b953c8a35561103e2014cf828..."
2. Bob: "my value is 84"
3. Alice: "my value was 65"
4. Bob checks if SHA256("65")="108c995b953c8a35561103e2014cf828..."

Random number generated:

$$65 + 84 = 149 \bmod 100 = 49$$

Hash-based PRNG

~~sha1(seed)~~
~~sha1(sha1(seed))~~
~~sha1(sha1(sha1(seed)))~~
~~sha1(sha1(sha1(sha1(seed))))~~
...

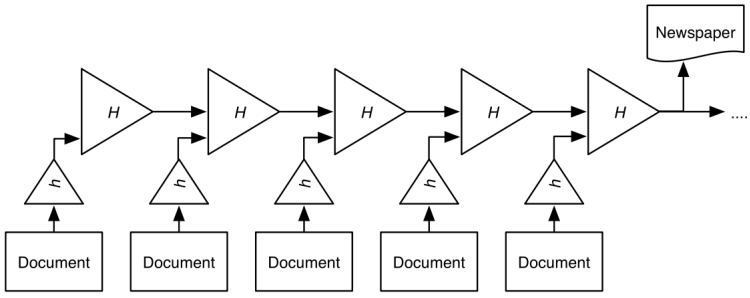
sha1(seed + "0")
sha1(seed + "1")
sha1(seed + "2")
sha1(seed + "3")
...

```
import hashlib
def hash_prng(seed):
    i = 0
    while True:
        print(hashlib.sha1(seed + str(i).encode()).hexdigest())
        i += 1

hash_prng(b'fookey')
```

- Standardized construction – Hash-DRBG (NIST SP 800-90)
- If we have a PRNG we can build a stream cipher

Hash chain



- Linking-based time-stamping
- Also known as a blockchain

Hash chain

Used in cash registers:

VIADA

VIADA Baltija, AS
Jur. adrese Alīses 3, Rīga, LV-1046
DUS/GUS 17 "Straupe"
Aleta, Straupe, Straupes pag.
Pārgaujas nov., LV-4152
tālr. 67301517
NMR kods: LV40103867145
ČEKS
ID 370/1748

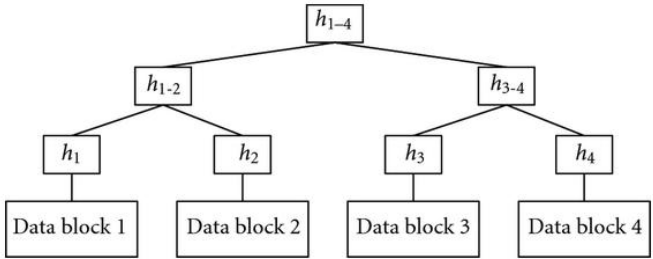
1.HOT DOG ITĀĻU+COLA/FANTA/S 1x3,29 3,29 D
Atlaide -1,64 D
2.pikantā uzkoda PRINGLES NA 1x2,75 2,75 D
Atlaide -1,37 D

KOPĀ 3,03 EUR
Kredīts 3,03 EUR
FAM NR: 05DCFF3639314E3457048114
FAM INICIALIZĒTS: 2019-05-03 19:27

2020-02-29 20:54 3
SN: 10171915 01 DOK.NR: 0187161

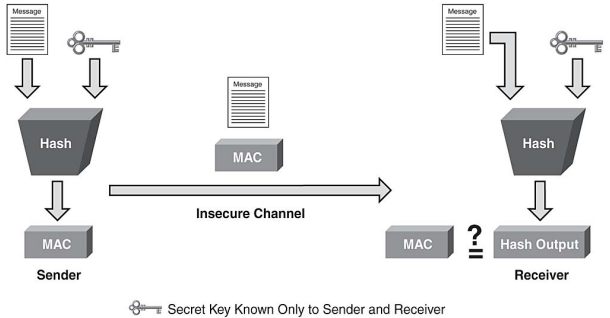
DOKUMENTA SHA-1:
1D48B325353A8473BD65E985532E2837385890C6
IEPR.DOK. SHA-1:
2FAD40E5CEB9E2833CE26AF5E3A1DADA366B188C

Hash tree (Merkle tree)



- Easy to prove that a node belongs to the tree
- To prove that "Data block 3" is part of the tree h_{1-4}:
  - "Data block 3"
  - "h_4"
  - "h_{1-2}"
- Used in BitTorrent to identify downloads

HMAC: Hash-based Message Authentication Code



- Without knowing the key, a valid MAC cannot be produced
- Naive implementation `hash(key + message)` is vulnerable!
  - Safe to use HMAC construction (RFC 2104)
- MAC does not guarantee freshness of the message
- Can we use MAC as a digital signature?

Questions

- What are the properties of a cryptographic hash function?
- What attacks must a cryptographic hash function resist?
- What does the size of the output from a hash function influence?
- What is a “security level” in cryptography?
- What is the commitment scheme useful for?
- Why is it better to store hashed passwords in a db?
- How can we increase the security level of password hashing?
- How can we create an encryption scheme from a hash function?
- What is HMAC useful for?
- Why is using MD5/SHA1 for HMAC not insecure?

Task: HMAC

Implement a tool that calculates and verifies the integrity of a file using HMAC.

```
$ ./hmac.py
Usage:
-mac <filename>
-verify <filename>

$ ./hmac.py -mac somefile
[?] Enter key: secretkey
[+] Calculated HMAC-SHA256: f5e94378cae5a3d0836e145f28807bb7076d28cd22b2481d45f92a904be9d2e8
[+] Writing HMAC DigestInfo to somefile.hmac

$ ./hmac.py -verify somefile
[+] Reading HMAC DigestInfo from somefile.hmac
[+] HMAC-SHA256 digest: f5e94378cae5a3d0836e145f28807bb7076d28cd22b2481d45f92a904be9d2e8
[?] Enter key: secretkey
[+] Calculated HMAC-SHA256: f5e94378cae5a3d0836e145f28807bb7076d28cd22b2481d45f92a904be9d2e8
[+] HMAC verification successful!

$ dumpasn1 somefile.hmac
  0  49: SEQUENCE {
  2  13:   SEQUENCE {
  4   9:     OBJECT IDENTIFIER sha-256 (2 16 840 1 101 3 4 2 1)
 15   0:     NULL
      :     }
 17  32:   OCTET STRING
      :     F5 E9 43 78 CA E5 A3 D0 83 6E 14 5F 28 80 7B B7
      :     07 6D 28 CD 22 B2 48 1D 45 F9 2A 90 4B E9 D2 E8
      :   }
```

DigestInfo

```
DigestInfo ::= SEQUENCE {
    digestAlgorithm    AlgorithmIdentifier,
    digest    OCTET STRING
}

AlgorithmIdentifier ::= SEQUENCE {
    algorithm    OBJECT IDENTIFIER,
    parameters    ANY DEFINED BY algorithm OPTIONAL
}
```

```
$ dumpasn1 hashobject
  0  33: SEQUENCE {
  2   9:   SEQUENCE {
  4   5:     OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
 11   0:     NULL
        :     }
 13  20:   OCTET STRING DA 39 A3 EE 5E 6B 4B 0D 32 55 BF ...
       :   }
```

- Standard structure to store the algorithm and calculated digest
- Defined in PKCS#1 v1.5 signature creation (RFC 2313)
- Our hash functions have no parameters (ASN.1 NULL)

Task: HMAC

- Use python's `hashlib` and `hmac` libraries
  e.g., `hmac.new(b'somekey', None, hashlib.md5)`
- Must support hashing of huge files
  - Read file in chunks of max 512 bytes
  - Feed chunks sequentially to `hmac_object.update()`
  - Finally call `hmac_object.digest()`
- HMAC digest must be writen to ".hmac" file using DigestInfo ASN.1 structure
  - Use your own ASN.1 encoder
    - Please embed your encoder in your solution
  - For decoding use `pyasn1`
- MAC'er must use HMAC-SHA256
- Verifier must support HMAC-MD5, HMAC-SHA1 and HMAC-SHA256 (algorithm must be read from DigestInfo)
  - OID for MD5: 1.2.840.113549.2.5
  - OID for SHA1: 1.3.14.3.2.26
  - OID for SHA256: 2.16.840.1.101.3.4.2.1

Task: Test Cases

```
$ echo -e -n "\x01" > file
$ python hmac.py -mac file
[?] Enter key: testkey
[+] Calculated HMAC-SHA256: a8be648dd48738b964391a00d4522fe988d10e3d5b2dbf8629a3dcbc0ce93ffd
[+] Writing HMAC DigestInfo to file.hmac

$ python hmac.py -verify file
[+] Reading HMAC DigestInfo from file.hmac
[+] HMAC-SHA256 digest: a8be648dd48738b964391a00d4522fe988d10e3d5b2dbf8629a3dcbc0ce93ffd
[?] Enter key: testkey
[+] Calculated HMAC-SHA256: a8be648dd48738b964391a00d4522fe988d10e3d5b2dbf8629a3dcbc0ce93ffd
[+] HMAC verification successful!
```

```
$ tar -zxvf hmac_testcases.tgz

$ python hmac.py -verify file_md5
[+] Reading HMAC DigestInfo from file_md5.hmac
[+] HMAC-MD5 digest: 9e8031ab9d85a5fa0753344bc8c31a2f
[?] Enter key: secretkey
[+] Calculated HMAC-MD5: 9e8031ab9d85a5fa0753344bc8c31a2f
[+] HMAC verification successful!

$ python hmac.py -verify file_sha1
[+] Reading HMAC DigestInfo from file_sha1.hmac
[+] HMAC-SHA1 digest: ebfb4fc1a84d5f9fcbd1b7c8d5d625ac9f5b4c81
[?] Enter key: secretkey
[+] Calculated HMAC-SHA1: ebfb4fc1a84d5f9fcbd1b7c8d5d625ac9f5b4c81
[+] HMAC verification successful!

$ python hmac.py -verify file_sha256
[+] Reading HMAC DigestInfo from file_sha256.hmac
[+] HMAC-SHA256 digest: c40932474350a3f29a9f800e68b6429c64b7526800f8701ae9b4e73db8a3b700
[?] Enter key: secrettkey
[+] Calculated HMAC-SHA256: 737f438db779461e6163aa236797099f08b154de6f5741843a549866ae57a5fd
[-] Wrong key or message has been manipulated!
```

pyasn1 library for decoding DER

```
$ sudo apt install python3-pyasn1
$ python3
>>> from pyasn1.codec.der import decoder
>>> der = open('asn1.der', 'rb').read()
>>> decoder.decode(der)
(<Sequence value object at 0x7f2a0cc3cf10 componentType=<NamedTypes object at 0x7f2a0cc128d0
     types > tagSet=<TagSet object at 0x7f2a0cc3cfd0 tags 0:32:16-128:32:0>
     subtypeSpec=<ConstraintsIntersection object at 0x7f2a0cc12850>
     sizeSpec=<ConstraintsIntersection object at 0x7f2a0cc12890> payload [<Set value object
     at 0x7f2a0cc3c890 componentType=[...] <UTCTime value object at 0x7f2a0cc3ca50 tagSet
     <TagSet object at 0x7f2a0cc3c590 tags 0:0:23> encoding us-ascii payload
     [150223010900Z]>]>, '')
>>> decoder.decode(der)[0][0][2]
<Integer value object at 0x7f2a0cc3ced0 tagSet <TagSet object at 0x7f2a0cc3ca90 tags
     0:0:2-128:32:11> payload [65407]>
>>> int(decoder.decode(der)[0][0][2])
65407
>>> decoder.decode(der)[0][0][2].__class__.__name__
'Integer'
```

- Can't handle large (>3MB) DER encoded structures
- Can't handle DER structures with implicit tagging