Universally Unique Identifiers (UUIDs)

```
Module Code: ELEE1119

Module Name: Advanced Computer Engineering

Credits: 30

Module Leader: Seb Blair BEng(H) PGCAP MIET MIHEEM FHEA
```

Download as a PDF

UUIDs

- 32 base 16 characters ([0-9][A-F])

$$128 = 32 \cdot log_2(16)$$

- 128 bit numbers


center

UUID Version 1

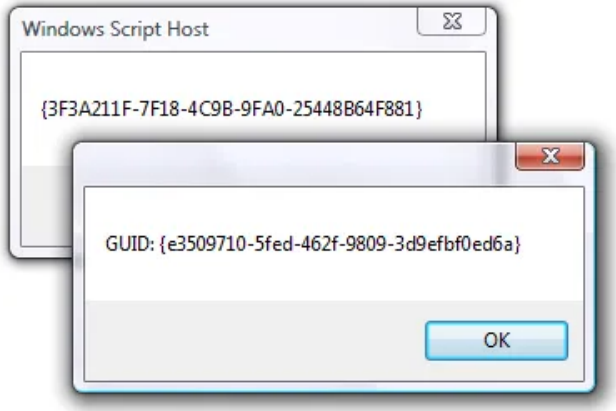
UUID Versions

There are 8 versions as of 23 June 2022

- UUID1 = Timebased + Unique or MAC [no repeats till 3603AD]
- UUID2 = Timebased(LSB) + userid
- UUID3 = Namespace+MD5 hash
- UUID4 = PRNG [1 trillion UUIDs for a chance of 2 repeats]
- UUID5 = Namespace + SHA-1 hash
- UUID6 = Timestamp and monotonic counter.
- UUID7 = UNIX Timestamp
- UUID8 - User defined Data

Who?

- Created by Microsoft but standardised by the **Internet Engineering Task Force** (IETF) and the **International Telecommunication Union** (ITU), so that each user or thing can be uniquely identifiable.
- **ITU-T X.667 | ISO/IEC 9834-8**



Where?

Combinations: UUID 4

- $0.0947mm^3$ grain of sand
- UUID4 has 122 random bits, $5.3e36$
- $5.0191e34mm^3 = 5.3e36 \cdot 0.0947mm^3$
- Volume of sand as UUID4 = $50,190,000,000,000,008km^3$

...and the volume of Jupiter

- $1,431,281,810,739,360km^3$

Uniqueness: UUID4

- In the version 4, 6 bits are fixed and the remaining 122 bits are randomly generated, for a total of 2^{122} possible UUIDs.
- $n = 2^{122}$
- So if the number of generated UUIDs exceeds $r > n$ then there must be duplicates
- If you assume perfect randomness you would expect to see collison after 2^{61}
- $2^{58} \approx 24,913,440,000,000,000 = 7.2e9 \cdot 365 \cdot 24 \cdot 60 \cdot 60$
- After a few years you would get the first collisions.

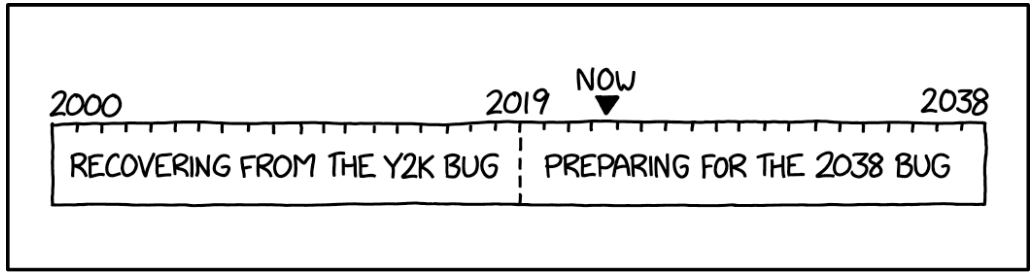
UUID 1

- combination of:
  - current time and date.
    - RFC 4122 60-bit count of 100ns since 00:00:00:00 15 October 1582 to 01/01/1970
    - $122192928000000000ns$
    - Current date time since 00:00:00:00 01 January 1970

  $$13928573027000000 0 = 17092802070000000 + 122192928000000000$$
- 48-bit MAC address of the host machine

Epochs/Time

- 01/01/1970
  - Unix engineers set the arbitrary datetime stamp because... it was convenient...
- 19/01/2038 03:14:07 the storage for 32-bit will become obsolete, as the value will be too large
  - Will need to migrate to 64-bit or just deal with the timestamp showing:
    - 19/01/1901 03:14:08



REMINDER: BY NOW YOU SHOULD HAVE FINISHED YOUR Y2K RECOVERY AND BE SEVERAL YEARS INTO 2038 PREPARATION.

UUID 1: Time format

Name	Bytes	Hex	Bits	Comments
time_low	4	8	32	integer giving the low 32-bits of time
time_mid	2	4	16	integer giving the middle 16-bits of time
time_hi+version	2	4	16	16 bits of time high with bits 6-7 multiplexed with version number
clock_seq_hi_and_res clock_seq_low	2	4	16	1 to 3-bit "variant" in the most significant bits, followed by the 13 to 15-bit clock sequence

Example

1. Current* timestamp in Unix Epoch Time (ns) = 1709280207346745902 .
2. Divide the timestamp by 100 to convert it to 100-nanosecond intervals:

$$1709280207346745902 = \frac{1709280207346745902}{100}$$

3. Add the number of 100-nanosecond intervals between the UUID epoch (1582-10-15) and the Unix epoch (1970-01-01):

$$13928573027346745902 = 1709280207346745902 + 12219292800000000$$

* at the time of making the slide

Example Continued...

Breaking down the UUID components as follows:

4. Time Low (32 bits): The first 32 bits of the timestamp in hexadecimal:

```
$ printf "0x%08X\n" 1392857302
> 0x530550D6
```

5. Time Mid (16 bits): The next 16 bits of the timestamp in hexadecimal:

```
$ printf "0x%04X\n" 7346
> 0x1CB2
```

6. Time High and Version (16 bits): The next 16 bits of the timestamp (7459) with the version (1) in hexadecimal: $0x1D24 = 0x1D23 + 1$

```
$ printf "%04X\n" <<< echo $((7459+1))
> 0x1D24
```

Example Continued....

7. Clock Sequence (14 bits), in truth this can be 14 random bits so:

```
$ printf "%04X\n" <<< echo $((RANDOM % 16384))
> 21FF
```

8. Node (48 bits): A randomly generated 48-bit value or MAC address if you must:

```
$ dd if=/dev/urandom bs=1 count=6 2>/dev/null | od -An -tx1 | tr -d ' \n'
> 2876c5202c7f
```

9. Put it all togehter:
    - timeLow - timeMid - timeHigh+version - clockSeq - Node
    - `530550D6-1CB2-1D24-21FF-2876C5202C7F`

UUID 2

- Distributed Computing Environment (DCE)
- combination of:
  - Current time and date.
  - The local identifier replaces the lower 32 bits of the timestamp.48-bit M1392857302AC address of the host machine
    - Domain Name or Hostname

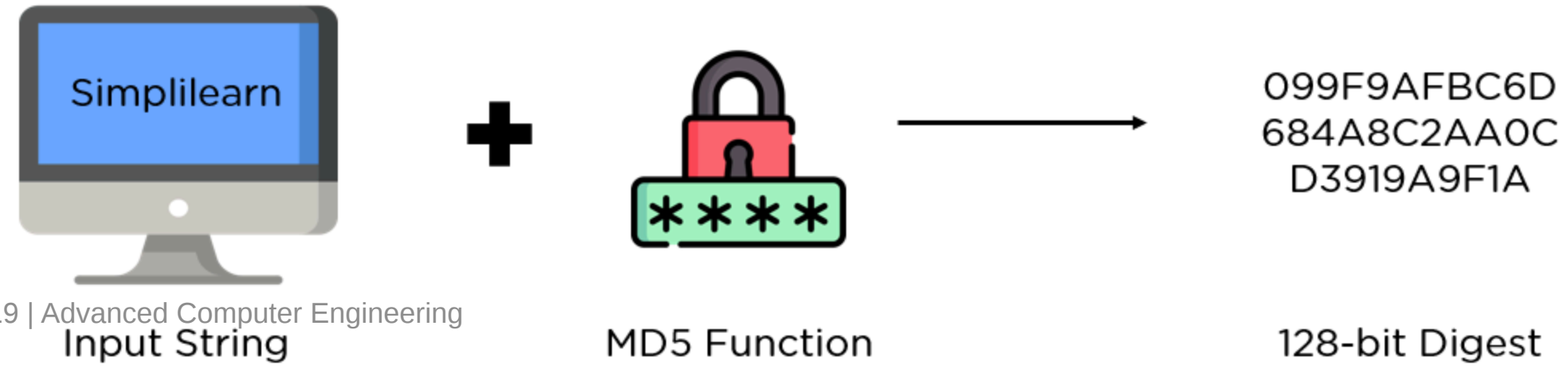
      ```
      $ id -u; id -g; whoami;
      ```

    - MacAddress or random generated Hex -> `sh1392857302`

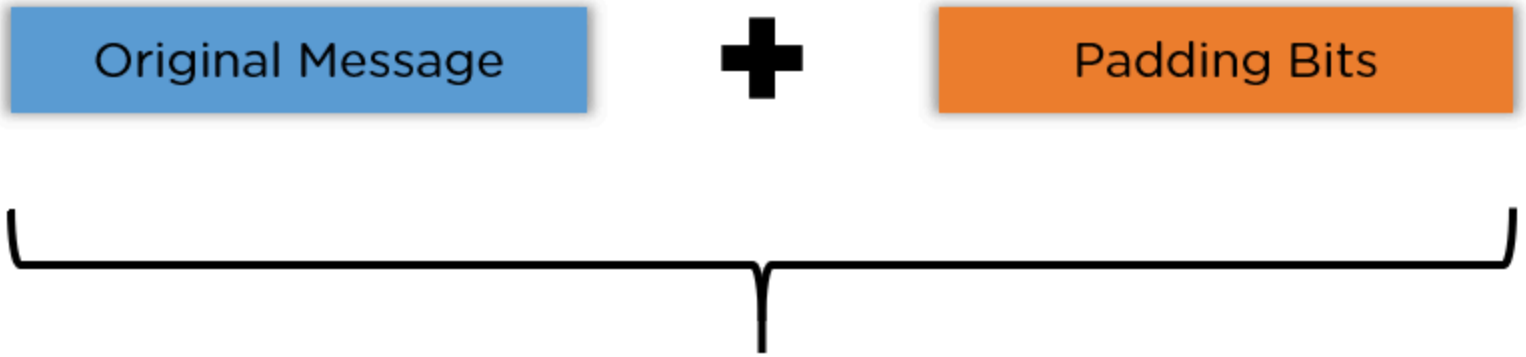
UUID 3

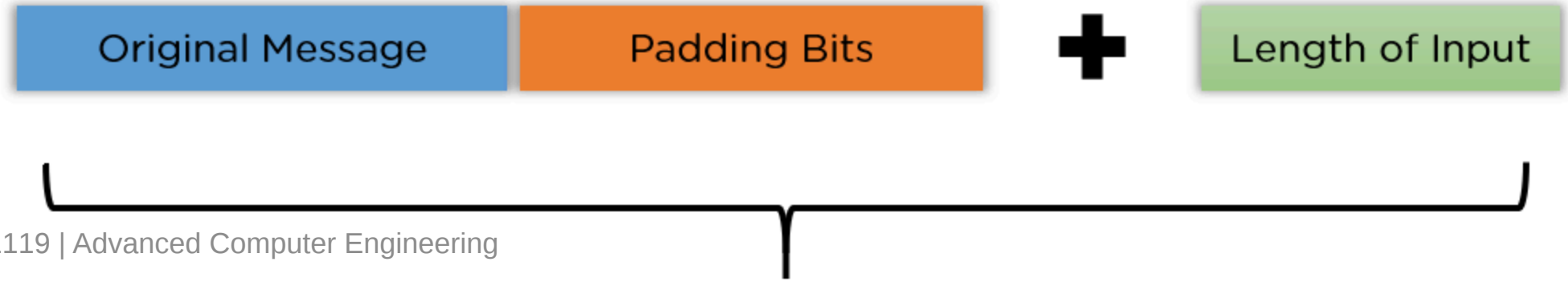
- namespace could be website, DNS information, plain text, etc
- the namespace value is hashed using the `md5hash` alogrithm
- GNU Coreutils implements this using `md5sum`

```
$ md5sum <<< "Test"
> 2205e48de5f93c784733ffcca841d2b5  -
```



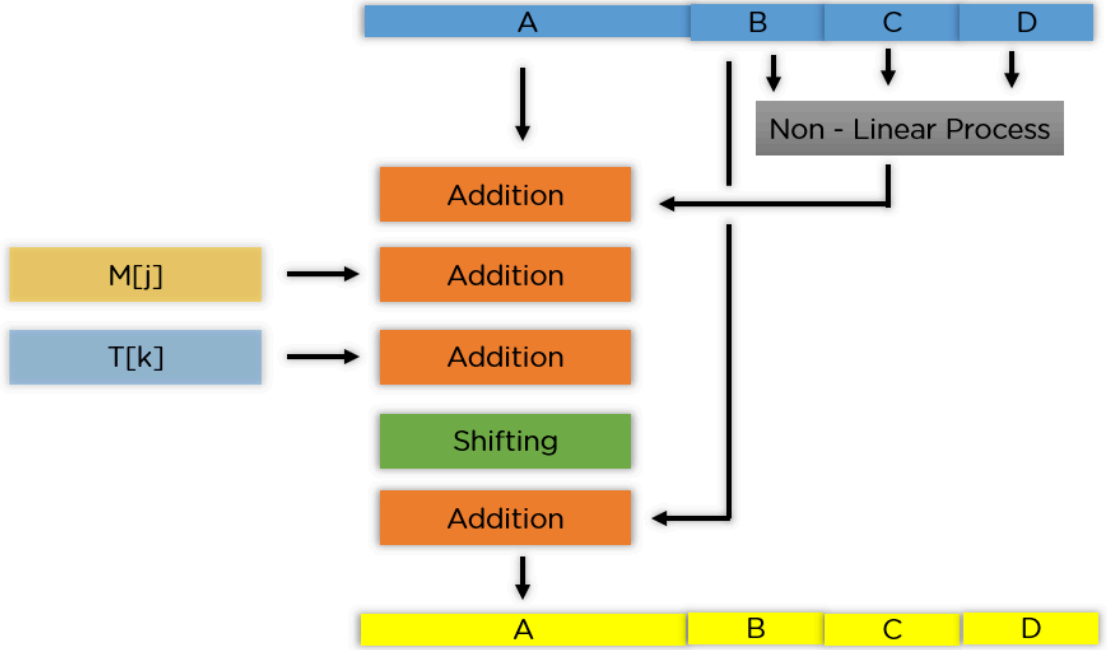
MD5 Algorithm





MD5 Alogrithm

- Round 1: (b AND c) OR ((NOT b) AND (d))
- Round 2: (b AND d) OR (c AND (NOT d))
- Round 3: b XOR c XOR d
- Round 4: c XOR (b OR (NOT d))



UUID4

1. Generate 128 random bits:

```
$ dd if=/dev/random count=16 bs=1 2> /dev/null | xxd -ps
> 7c1e598398eb691f3f4be4123c3ce9a7
```

[0]0111110 00001111 00101100 11000001 11001100 01110101 **10110100** 100011111

00111111 01001011 11100100 00010010 00111100 00111100 11101001 10100111

2. Take the 7th byte and perform an AND operation with `0x0F` to clear out the high nibble. Then, OR it with `0x40` to set the version number to 4.

00000100 = **10110100** & 00001111 (0x0f)

01000100 = **00000100** | 01000000 (0x40)

UUID4 Example

3. Next, take the 9th byte (**00111111**) and perform an AND operation with `0x3F` and then OR it with `0x80`.

$$00111111 = \mathbf{00111111} \ \& \ 00111111 \ (0x3f)$$

$$100111111 = \mathbf{00111111} \ | \ 10000000 \ (0x80)$$

1. Convert the 128 bits to hexadecimal representation and insert the hyphens to achieve the canonical text representation.

Before: 7C1E5983-98EB-691F-3f4B-E4123C3CE9A7

After: 7C1E5983-98EB-**44**1F-**CF**4B-E4123C3CE9A7

Your Turn

10101110 00100001 10110100 11111100 01101111 01110010 **10100011** 00110010 **10111010** 10000110 11010010 00001001 11010001 11100000 10101111 01101001

2. Take the 7th byte and perform an AND operation with `0x0F` to clear out the high nibble. Then, OR it with `0x40` to set the version number to 4.
3. Next, take the 9th byte and perform an AND operation with `0x3F` and then OR it with `0x80`.
4. Convert the 128 bits to hexadecimal representation and insert the hyphens to achieve the canonical text representation.

▶ Answer

UUID 5

- namespace could be a website, DNS information, plain text, etc
- the namespace value is hashed using the `sha1` alogrithm
- GNU Coreutils implements this using `sha1sum`

```
$ sha1sum "Test"
> 1c68ea370b40c06fcaf7f26c8b1dba9d9caf5dea  -
```