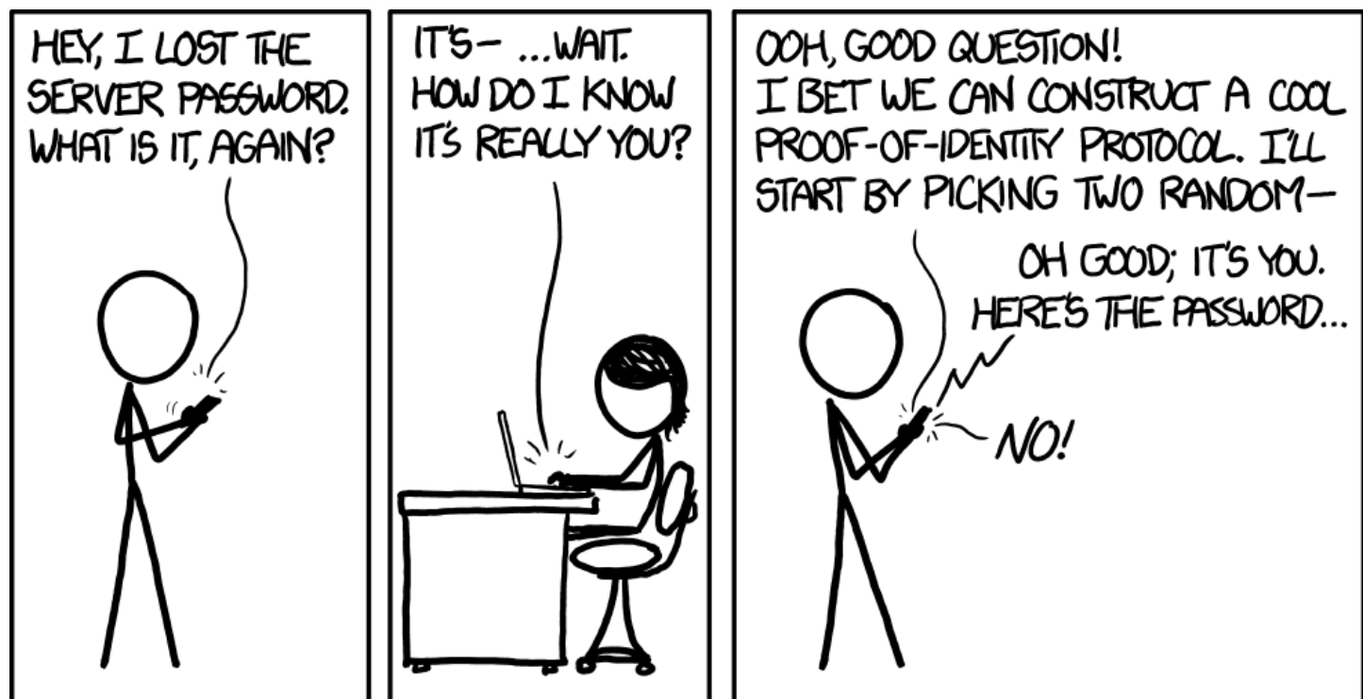
Applications of Mathematics in Computer Science (MACS)

LECTURE #2:

CHECKSUMS AND HASH CODES



The CS Problems:

- Redundancy in numbers
- Data integrity checking
- Authorization
- Hash tables

Redundancy in numbers

How can we quickly check that a number is correct?

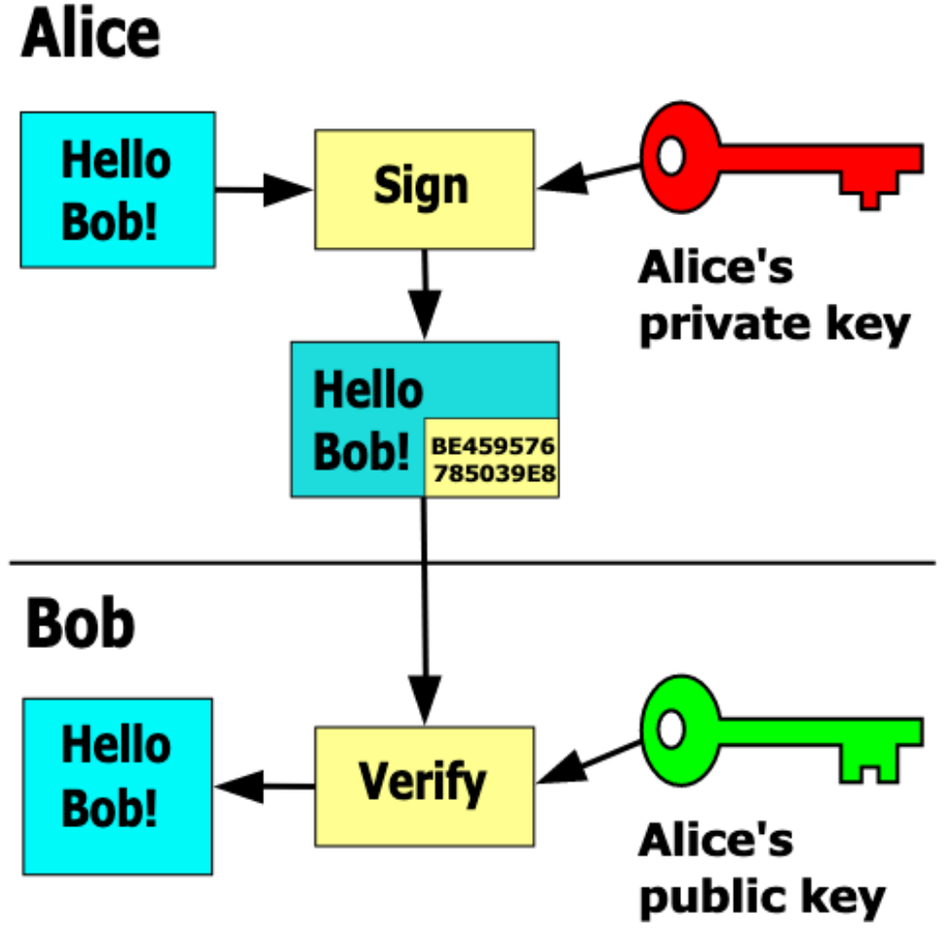
- ID number

- "card number is invalid"

How do they know?

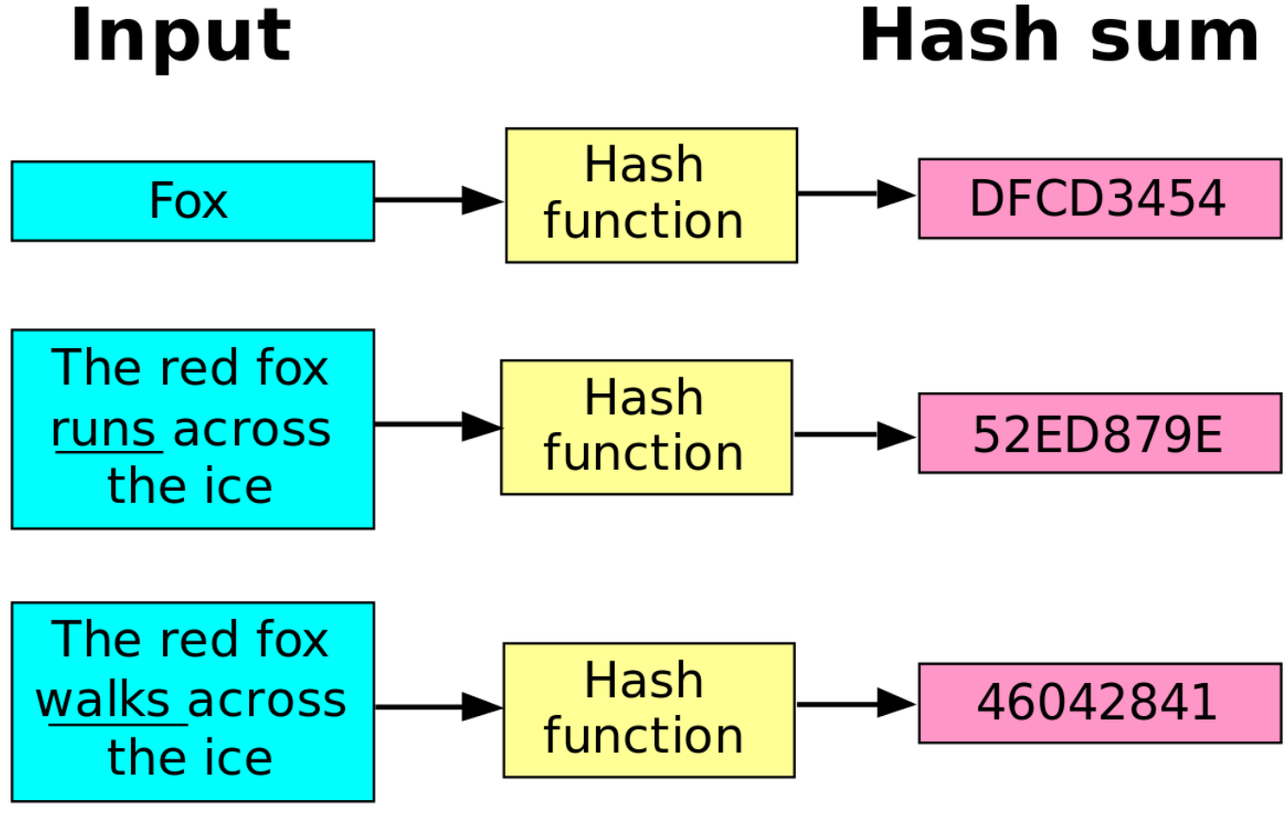
Data integrity checking

- Quickly check that a download is complete
- Verify a document:



Authorization

- Checking a person's identity
  - ID/password



Hash Tables (Dictionaries)

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'jack': 4098, 'sape': 4139, 'guido': 4127}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'jack': 4098, 'guido': 4127, 'irv': 4127}
>>> list(tel)
['jack', 'guido', 'irv']
>>> sorted(tel)
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
>>> 'jack' not in tel
False
```

How can we construct such a data type?

The math technique:

Modular Arithmetics

Modular Arithmetics

$$m = a \,\%\, b$$

$$0 \leq m < b$$

$$\exists n \,.\, a = n \cdot b + m$$

$$n \,\%\, 1 = 0$$

$$3 \,\%\, 2 = 1$$

$$10 \,\%\, 3 = 1$$

$$5 \,\%\, 3 = 2$$

Modular Equivalence

$$a \equiv b \pmod{c}$$

$$a \,\%\, c = b \,\%\, c$$

$9 \equiv 5 \pmod{4}$ $4 \equiv 2 \pmod{2}$ $3 \equiv 111 \pmod{2}$

- Reflexivity: $a \equiv a \pmod{n}$
- Symmetry: $a \equiv b \pmod{n} \Leftrightarrow b \equiv a \pmod{n}$
- Transitivity:

  $a \equiv b \pmod{n} \wedge b \equiv c \pmod{n} \Leftrightarrow a \equiv c \pmod{n}$

Properties

We can compute with remainders!

If $a \equiv b \pmod{n}$, then:

- $a + k \equiv b + k \pmod{n}$
- $ka \equiv kb \pmod{n}$ (for integer k)

If $a_1 \equiv b_1 \pmod{n}$ and $a_2 \equiv b_2 \pmod{n}$, then:

- $a_1 + a_2 \equiv b_1 + b_2 \pmod{n}$
- $a_1 - a_2 \equiv b_1 - b_2 \pmod{n}$
- $a_1 \cdot a_2 \equiv b_1 \cdot b_2 \pmod{n}$

Example

- S: 1000 integers sent
- R: 1000 integers received

S=R?

```
def checksum(A):
  cksum = 0
  for a in A:
    cksum += a % 101
  return cksum % 101
```

Will overflow if integers are big

Let's try...