**1. Hash Functions**

* A hash function maps a message of arbitrary length to an m-bit output.
* The output is known as the fingerprint or message digest.
* Cryptographic hash functions are hash functions with additional security requirements.

**2. Examples of Hash Functions**

* **Fox**
  + **Input: Fox**
  + **Output: DFCD3454 BBEA788A 751A696C 24D97009 CA992D17**
* **The red fox runs across the ice**
  + **Input: The red fox runs across the ice**
  + **Output: 52ED879E 70F71D92 6EB69570 08E03CE4 CA6945D3**
* **The red fox walks across the ice**
  + **Input: The red fox walks across the ice**
  + **Output: 46042841 935C7FB0 9158585A B94AE214 26EB3CEA**

**3. Using Hash Functions for Message Integrity**

* Hash functions are many-to-one functions, so collisions can happen.
* **MD5**
  + **Output: 128 bits**
  + Collision resistance completely broken by researchers in China in 2004.
* **SHA1**
  + **Output: 160 bits**
  + No collision found yet, but methods exist to find collisions in less than 2^80.
  + Considered insecure for collision resistance.
  + One-wayness still holds.
* **SHA2 (SHA-224, SHA-256, SHA-384, SHA-512)**
  + **Outputs: 224, 256, 384, and 512 bits respectively.**
  + No real security concerns yet.

**4. Class Activity**

* Suggest two real-world scenarios where hash functions might be applied.
* **Scenario 1: Verifying File Integrity**
  + **Context**:

Ensuring that files have not been altered or corrupted during transfer or storage.

* + **Use**:

When downloading a file from the internet, a hash function can be used to generate a unique hash value for the original file.

* + **Verification**:

After downloading, users can re-calculate the hash value of the downloaded file and compare it with the original hash value provided by the source.

* + **Outcome**:

If the hash values match, the file is verified to be intact and unaltered; if they don't match, the file may have been tampered with or corrupted.

* **Scenario 2: Password Storage**
  + **Context**:

Protecting user passwords in a database.

* + **Use**:

Instead of storing plain text passwords, a hash function is applied to each password to generate a unique hash value.

* + **Authentication**:

When a user logs in, the hash value of the entered password is computed and compared with the stored hash value.

* + **Outcome**:

If the hash values match, the user is authenticated; if they don't match, the password is incorrect. This method ensures that even if the database is compromised, the actual passwords remain protected.

Imagine you have a super cool magic box. You can put any toy or object inside this box, and when you press a button, the box gives you a secret code. This secret code is like a special nickname for your toy, and it's always the same length no matter how big or small your toy is.

1. **Unique Nicknames:**
   * When you put the **same toy** in the box, you always get the **same secret code.**
   * If you put **different toys** in the box, you usually get **different secret codes**. Sometimes, two different toys might accidentally get the same code, but it's rare.
2. **One-Way Magic:**
   * You can easily get the secret code from your toy, but you can't turn the code back into the toy. It's a one-way process.

Let's see an example:

* You put a toy car in the box and press the button. The box gives you the code "ABC123".
* You put a toy dinosaur in the box and press the button. The box gives you the code "XYZ789".
* Every time you put the toy car in, you get "ABC123". And every time you put the toy dinosaur in, you get "XYZ789".

Hashing is like having this magic box that gives special nicknames (codes) to your toys (data).

These codes (nicknames) help you keep track of your data (toys), make sure they haven't changed.