2CSDE93- Blockchain Technology

LAB 1-

To implement digital signature to sign and verify authenticated user. Also, show a message when tampering is detected.

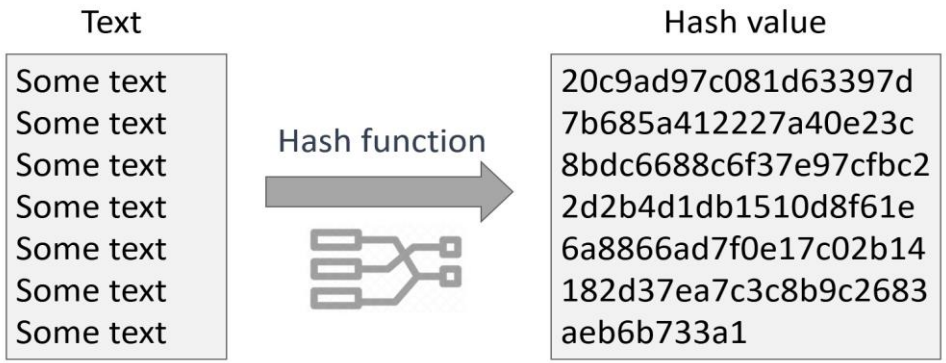
Outline

- Hash Functions
- Message Authentication Code (MAC)
- Digital Signatures
- Digital Signature Model
- Two Key Digital Signature Algorithms

HASH FUNCTIONS

What are Hash Functions?

- In cryptography, **hash functions** transform **input data** of arbitrary size (e.g. a text message) to a **result** of fixed size (e.g. 256 bits), which is called **hash value** (or hash code, message digest, or simply hash).
- Hash functions (hashing algorithms) used in computer cryptography are known as "**cryptographic hash functions**". Examples of such functions are **SHA-256** and **SHA3-256**, which transform arbitrary input to 256-bit output.



MESSAGE AUTHENTICATION CODE (MAC)

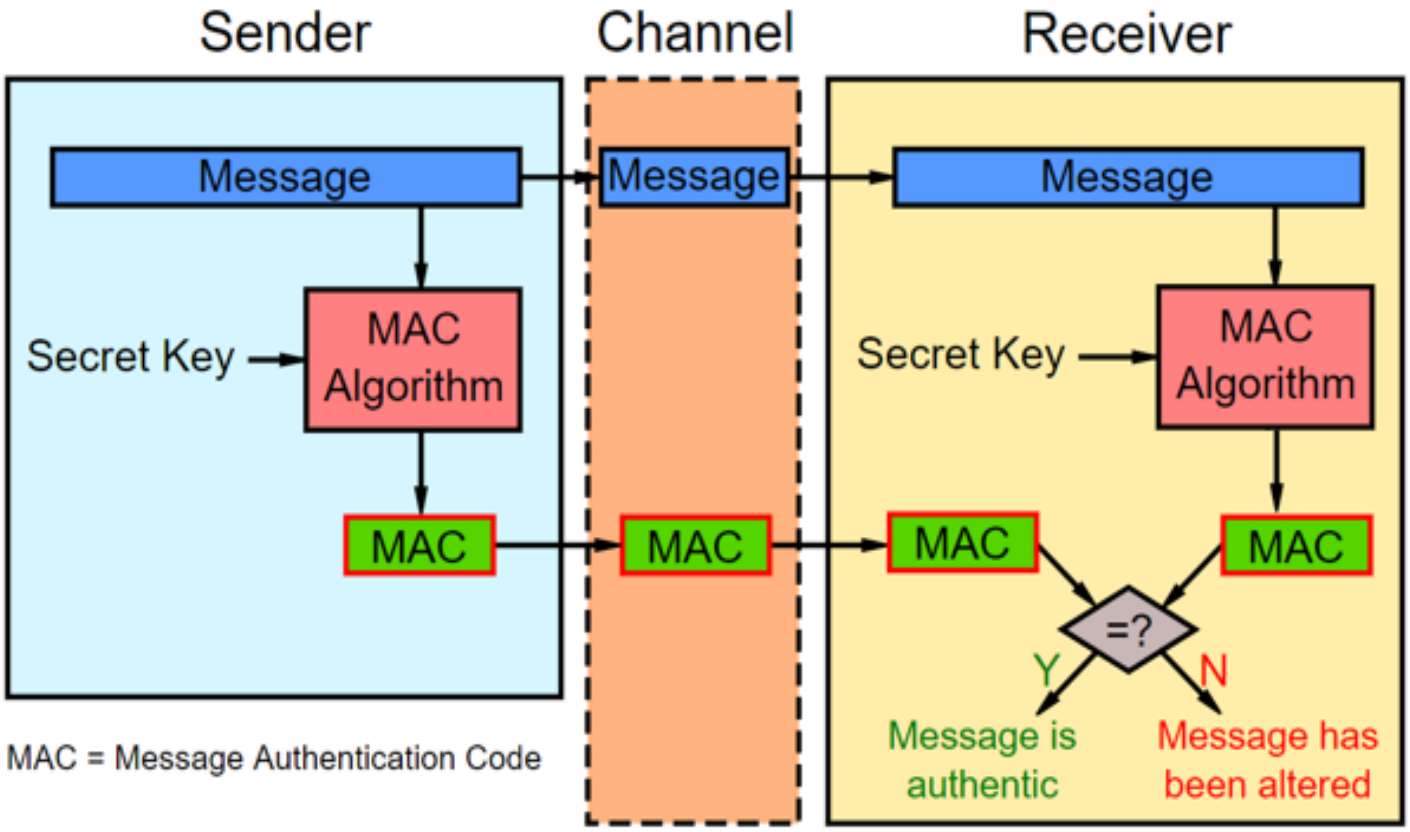
What is MAC?

- A message authentication code is an algorithm which takes as input a **message and a secret key** and produces a fixed-sized output which can be later on verified to match the message.
- The verification also requires the **same secret key**. Contrary to hash functions where everything is known and attackers are fighting against mathematics, MAC make sense in models where there are **entities with knowledge of a secret.**
- What we expect from a good MAC is **unforgeability**: it should be infeasible to compute a pair ***message+MAC*** value which successfully verifies with a given key *K* without knowing *K* exactly and in its entirety.

Hash vs MAC- What's the difference??

- The main difference is conceptual: while **hashes** are used to guarantee the **integrity of data**, a **MAC** guarantees **integrity AND authentication**.
- This means that a **hashcode** is blindly generated from the message without any kind of external input: what you obtain is something that can be used to check if the message got any alteration during its travel.
- A **MAC** instead uses a **private key as the seed** to the hash function it uses when generating the code: this should assure the receiver that, not only the message hasn't been modified, but also who sent it is what we were expecting: otherwise an attacker couldn't know the private key used to generate the code.

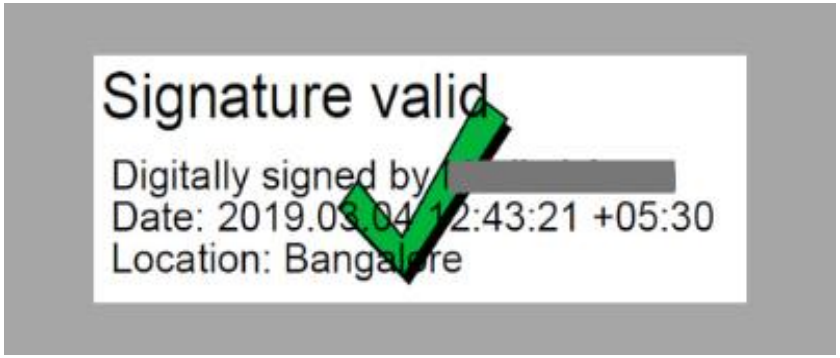
A flow scenario of MAC generation and verification



Digital Signature

What is Digital Signature?

- **Digital signature** allow us to verify the author, date and time of signatures, authenticate the message contents. It also includes authentication function for additional capabilities.



What is Digital Signature?

Technique for establishing the origin of a particular message in order to settle later disputes about what message (if any) was sent.

The purpose of a digital signature is thus for an entity to bind its identity to a message.

We use the term **signer** for an entity who creates a digital signature

We use the term **verifier** for an entity who receives a signed message and attempts to check whether the digital signature is “correct” or not.

Why Digital Signatures?

- To provide Authenticity, Integrity and Non - repudiation to electronic documents
- To use the Internet as the safe and secure medium for e-Governance,e-Commerce, and many more

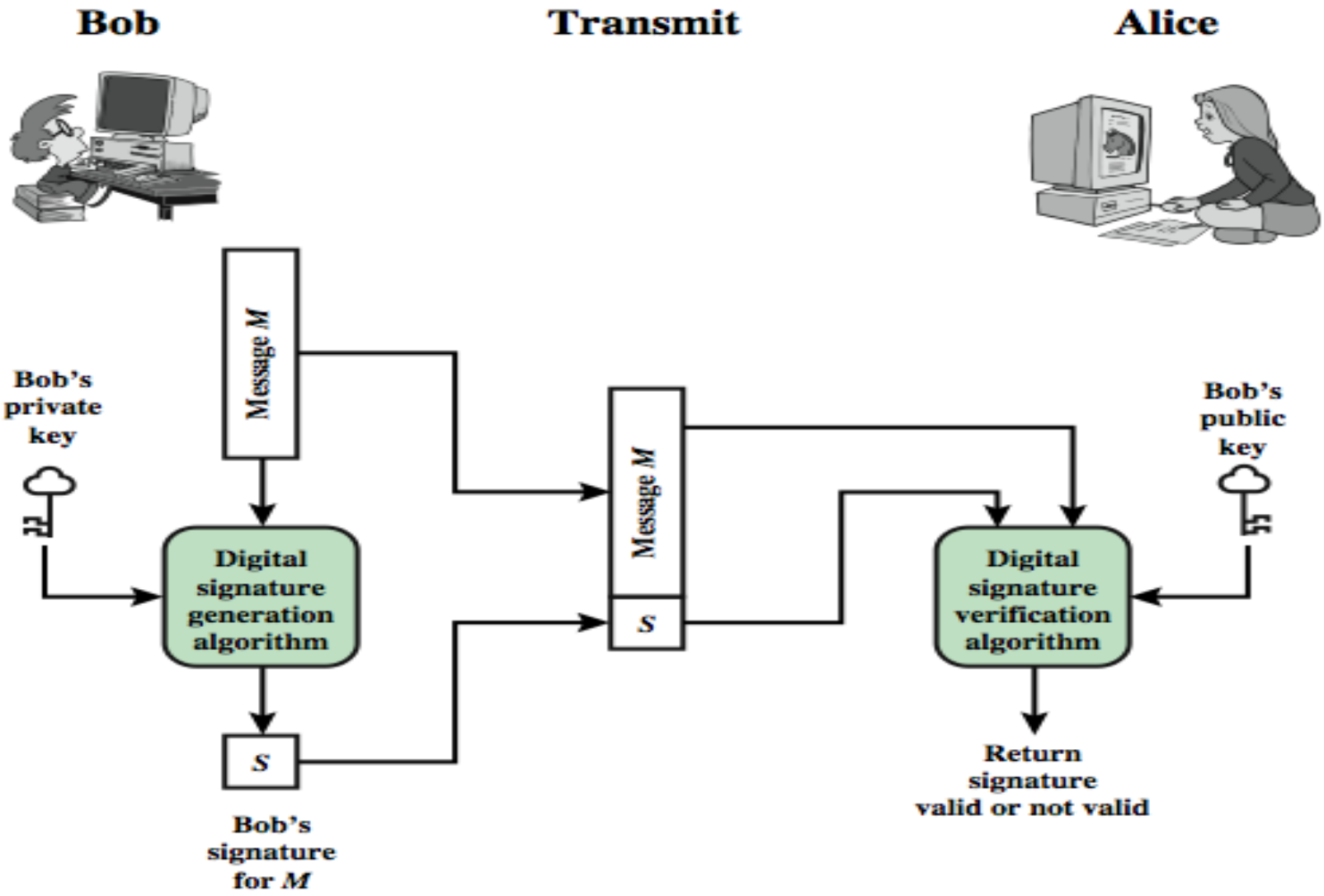
Paper Signatures V/s Digital Signatures

V/s

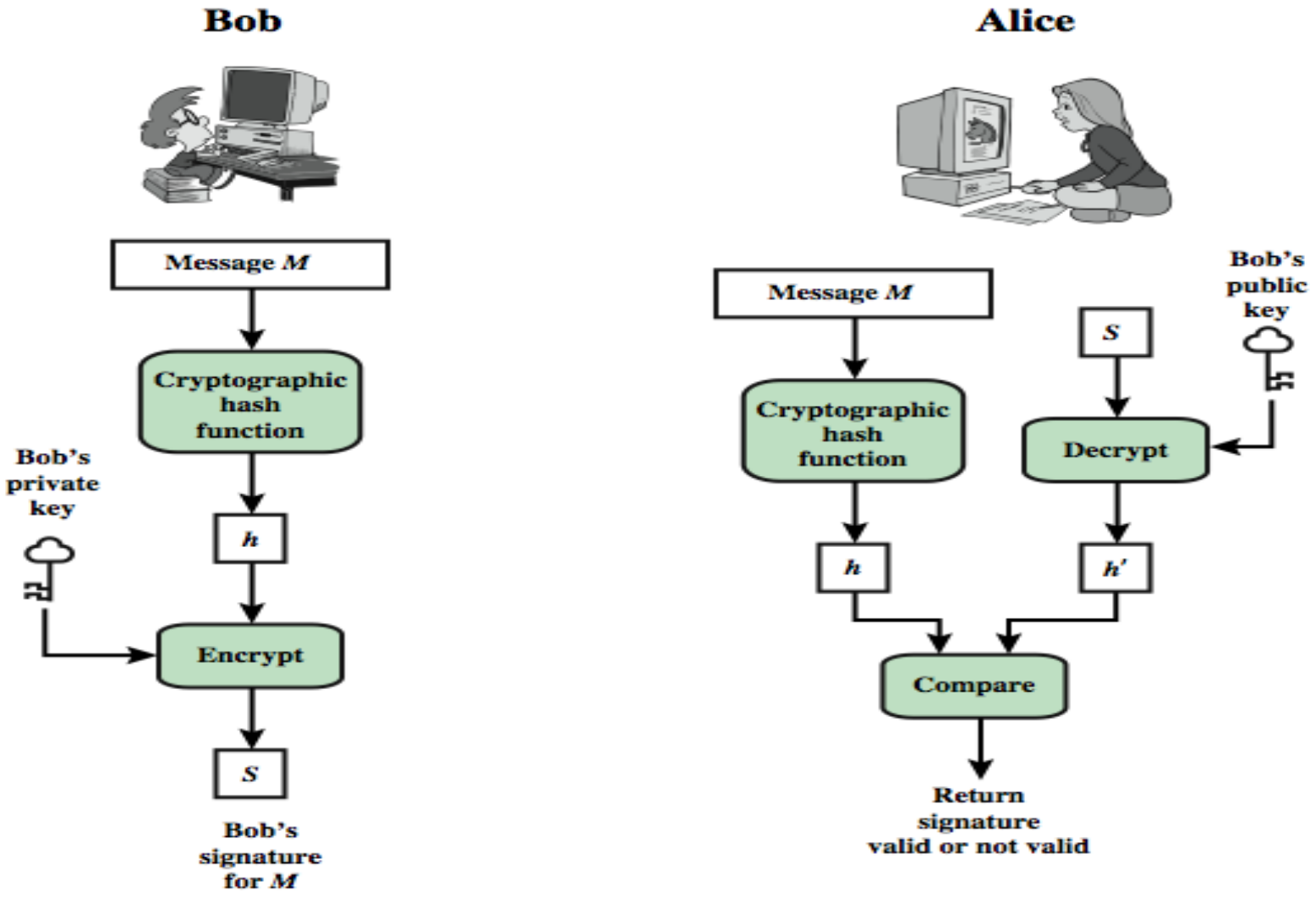
Parameter	Paper	Electronic
Authenticity	**May be forged**	**Can not be copied**
Integrity	**Signature independent of the document**	**Signature depends on the contents of the document**
Non-repudiation	a. Handwriting expert needed b. Error prone	a. Any computer user b. Error free

Digital Signature Model

Digital Signature Model (1)



Digital Signature Model (2)



Two Key Digital Signature Algorithms

Two Key Discussion algorithms

- RSA Digital Signature Algorithm
- ECC Digital Signature Algorithm

RSA Digital Signature Algorithm

- The **RSA** public-key cryptosystem provides a **digital signature scheme** (sign + verify), based on the math of the **modular exponentiations** and discrete logarithms and the computational difficulty of **the RSA problem** (and its related integer factorization problem).
- The RSA algorithm uses **keys** of size 1024, 2048, 4096, ..., 16384 bits. RSA supports also longer keys (e.g. 65536 bits), but the performance is too slow for practical use (some operations may take several minutes or even hours). For 128-bit security level, a 3072-bit key is required.

- The **RSA key-pair** consists of:

  - public key {***n***, ***e***}
  - private key {***n***, ***d***}
    - The numbers ***n*** and ***d*** are typically big integers (e.g. **3072** bits), while ***e*** is small, typically **65537**.

RSA-SIGN AND VERIFY

Signing a message ***msg*** with the private key exponent ***d***:

1. Calculate the message hash: ***h*** = hash(***msg***)
2. Encrypt ***h*** to calculate the signature: $s = h^d \pmod n$
3. The hash ***h*** should be in the range [0...***n***). The obtained **signature *s*** is an integer in the range [0...***n***).

Verifying a signature ***s*** for the message ***msg*** with the public key exponent ***e***:

1. Calculate the message hash: h = hash(***msg***)
2. Decrypt the signature: $h' = s^e \pmod n$
3. Compare **h** with **h'** to find whether the signature is valid or not

If the signature is correct, then the following will be true:

$$\mathbf{h' = s^e \pmod n = (h^d)^e \pmod n = h}$$

ECC-Digital Signature Algorithm

- The ECDSA (Elliptic Curve Digital Signature Algorithm) is a cryptographically secure **digital signature scheme**, based on the elliptic-curve cryptography (ECC).
- **ECDSA** relies on the math of the **cyclic groups of elliptic curves over finite fields** and on the difficulty of the ECDLP problem (***elliptic-curve discrete logarithm problem***).
- The ECDSA sign/verify algorithm relies on **EC point multiplication**
- ECDSA keys and signatures are **shorter than in RSA** for the same security level. A **256-bit ECDSA** signature has the same security strength like **3072-bit RSA** signature.

Elliptic Curve Families

ECDSA uses EC over finite fields in the classical Weierstrass form.

These curves are described by their **EC domain parameters**, specified by various cryptographic standards such as SECG and Brainpool (RFC 5639)

Elliptic curves, used in cryptography, define:

- **Generator point G**, used for scalar multiplication on the curve (multiply integer by EC point)
- **Order *n*** of the subgroup of EC points, generated by **G**, which defines the length of the private keys (e.g. 256 bits)

For example, the 256-bit elliptic curve **secp256k1** has:

- Order ***n*** = 115792089237316195423570985008687907852837564279074904382605163141518161494337 (prime number)
- Generator point **G** {***x***=55066263022277343669578718895168534326250603453777594175500187360389116729 2, ***y***= 32670510020758816978083085130507043184471273380659243275938904335757337482424}

ECDSA-Key generation

- The **ECDSA key-pair** consists of:
  - **private key** (integer): ***privKey***
  - **public key** (EC point): ***pubKey*** = ***privKey*** * **G**
- The **private key** is generated as a **random integer** in the range [0...***n***-1].
- The public key ***pubKey*** is a point on the elliptic curve, calculated by the EC point multiplication: ***pubKey*** = ***privKey*** * **G**
- (the private key, multiplied by the generator point **G**).
- The public key EC point {***x***, ***y***} can be **compressed** to just one of the coordinates + 1 bit (parity).
- For the secp256k1 curve, the **private key is 256-bit integer (32 bytes)** and the compressed **public key is 257-bit integer (~ 33 bytes).**

ECDSA- Signature Generation

- The ECDSA signing algorithm (RFC 6970) takes as input a message ***msg*** + a private key ***privKey*** and produces as output a **signature**, which consists of pair of integers {***r***, ***s***}.
- The **ECDSA signing** algorithm is based on the ElGamal signature scheme and works as follows (with minor simplifications):
  1. Calculate the message **hash**, using a cryptographic hash function like SHA-256: ***h*** = hash(***msg***)
  2. Generate securely a **random** number ***k*** in the range [1..***n***-1]
  3. In case of **deterministic-ECDSA**, the value ***k*** is HMAC-derived from ***h*** + ***privKey*.**
  4. Calculate the random point ***R*** = ***k*** * **G** and take its x-coordinate: ***r*** = ***R*.x**
  5. Calculate the signature proof: $\mathbf{s = k^{-1} * (h + r * privKey) (mod\ n)}$
  6. Return the **signature {*r*, *s*}**.

ECDSA- Signature Verification

- The algorithm to **verify a ECDSA signature** takes as input the signed message ***msg*** + the signature {***r***, ***s***} produced from the signing algorithm + the public key ***pubKey***, corresponding to the signer's private key.
- The output is boolean value: ***valid*** or ***invalid*** signature.
- The **ECDSA signature verify** algorithm works as follows (with minor simplifications):
  1. Calculate the message **hash**, with the same cryptographic hash function used during the signing: ***h*** = hash(***msg***)
  2. Calculate the modular inverse of the signature proof: ***s1*** = s^{-1}(mod n)
  3. Recover the random point used during the signing: ***R'*** = (***h*** * **s1**) * **G** + (***r*** * ***s1***) * ***pubKey***
  4. Take from ***R'*** its x-coordinate: ***r'*** = ***R'*.x**
  5. Calculate the signature validation **result** by comparing whether (***r'*** == ***r)***

Thank You