One-Way Hash Functions

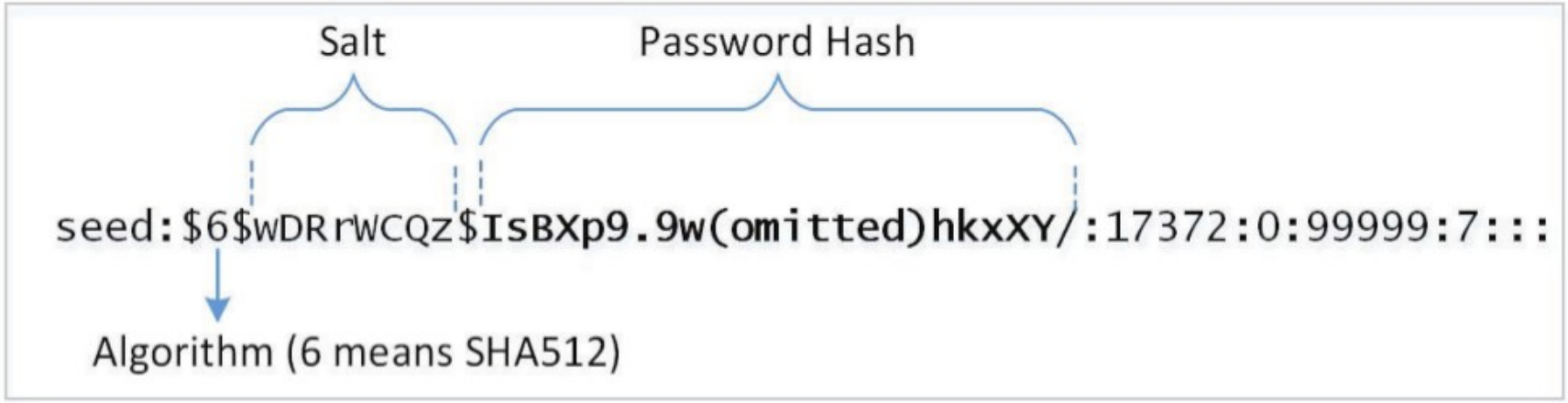
Password Verification

- To login into account, user needs to tell a secret (password)
- Cannot store the secrets in their plaintext
- Need for:
  - Password storage where nobody can know what the password is
  - If provided with a password, it verified against the stored password
- Solution: one-way hash function
- Example: Linux stores passwords in the /etc/shadow file

```
seed:$6$wDRrWCQz$IsBXp9.9wz9SG(omitted)sbCT7hkxXY/:17372:0:99999:7:::
test:$6$a6ftg3SI$apRiFL.jDCH7S(omitted)jAPXtcB9oC0:17543:0:99999:7:::
```

Case Study: Linux Shadow File

- Password field has 3 parts: algorithm used, salt, password hash
- Salt and password hash are encoded into printable characters
- Multiple rounds of hash function (slow down brute-force attack)



Purpose of Salt

- Using salt, same input can result in different hashes
- Password hash = one-way hash rounds (password || random string)
- Random string is the salt

```
The two password entries:
seed:$6$wDRrWCQz$IsBXp9.9wz9SG(omitted)sbCT7hkxXY/:17372:0:99999:7:::
test:$6$a6ftg3SI$apRiFL.jDCH7S(omitted)jAPXtcB9oC0:17543:0:99999:7:::
-------------------------------
$ python
>>> import crypt
>>> print crypt.crypt('dees', '$6$wDRrWCQz$')
$6$wDRrWCQz$IsBXp9.9wz9SG(omitted)sbCT7hkxXY/
>>> print crypt.crypt('dees', '$6$a6ftg3SI$')
$6$a6ftg3SI$apRiFL.jDCH7S(omitted)jAPXtcB9oC0
```

Attacks Prevented by Salt

- Dictionary Attack
  - Put candidate words in a dictionary
  - Try each against the targeted password hash to find a match
- Rainbow Table Attack
  - Precomputed table for reversing cryptographic hash functions
- Why Salt Prevents them ?
  - If target password is same as precomputed data, the hash will be the same
  - If this property does not hold, all the precomputed data are useless
  - Salt destroys that property