**Why SALTSTACK?**

The following lists the advantages of saltstack:-

• enable commands to remote systems to be called in parallel rather than serially

• use a secure and encrypted protocol

• use the smallest and fastest network payloads possible

• provide a simple programming interface

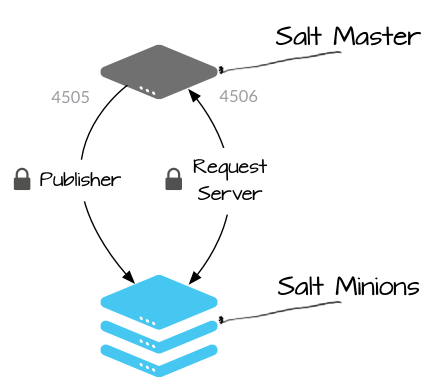
**AUTHENTICATION BETWEEN SALT-MASTER AND SALT-MINION: -**

Salt uses a server-agent communication model.

The Salt master is responsible for sending commands to Salt minions, and then aggregating and displaying the results of those commands.

.

**COMMUNICATION MODEL:**



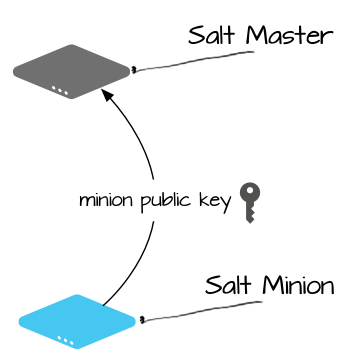
Salt communicates with managed systems using a publish-subscribe pattern. Connections are initiated by the Salt minion, which means that you do not need to open any incoming ports on those systems. The Salt master uses ports 4505 and 4506, which must be opened to accept incoming connections.

**PUBLISHER port**: (port 4505)- All Salt minions establish a persistent connection to the publisher port where they listen for messages. Commands are sent asynchronously to all connections over this port, which enables commands to be executed over large numbers of systems simultaneously.

**REQUEST SERVER**:(port 4506) Salt minions connect to the request server as needed to send results to the Salt master, and to securely request files and minion-specific data values (called Salt pillar). Connections to this port are 1:1 between the Salt master and Salt minion (not asynchronous).

**SALT MINION AUTHENTICATION:**

When the minion starts for the first time, it searches the network for a system named salt. When found, the minion initiates a handshake and then sends its public key to the Salt master.



After this initial connection, the Salt minion’s public key is stored on the server, and it must be accepted on the Salt master using the salt-key command (or through some automated mechanism). This can be a source of confusion for new users since Salt won’t provide the security key required to decode messages until the Salt minion’s public key is accepted (which means that the Salt minion won’t run any commands until its key is accepted).

After the minion key is accepted, the Salt master returns its public key along with a rotating AES key that is used to encrypt and decrypt messages that are sent by the Salt master. The returned AES key is encrypted using the public key initially sent by the Salt minion, and can therefore be decrypted only by that Salt minion.

**ROTATING SECURITY KEYS:**

A rotating AES key is used to encrypt jobs that are sent to the Salt minion by the Salt master, and to encrypt connections to the Salt master file server. A new key is generated and used each time the Salt master restarts and each time a Salt minion key is deleted using the salt-key command.

After the key is rotated, all Salt minions must re-authenticate to receive the updated key. This enables the AES key to rotate without interruptng the minion connection.

**USER ACCESS CONTROL:**

Before a command is sent to minions, Salt performs some internal checks against the ***Publisher ACL*** to make sure that the user executing the command has the correct privileges. If the user is authorized to run the specified command against the specified target, the command is sent. If not, an error is returned. Salt also returns a list of the minions that are expected to respond to the command line interface, so it knows how long to wait for returns.

Saltstack also provides a **peer-interface** whereby a

saltminion connected to the saltmaster can ask the master to run some specific commands on its behalf