Bitloop

**Introduction**

Thousands of encryption methods have been used in human history. Many of these were used for reasons of prohibition, clandestine or warfare. However, it is easily deciphered in today's technology due to the clarity in the encryption methods that have survived until today. We know the a lot of encrypted system but they easily cracked by the Intelligence Service or any crack systems. The best way to prevent this is to solve it with the encryption method in the network environment and over and over. So we can provide ourself by Bitloop. Bitloop is two time encrypted user any data and also give this encrypted data sha256 key id so that users using this id for decrypted the encrypted data. Also if users want remove the message on server they need just enter the key id and remove that. So we decide that Bitloop is secure and flexible.

**How It’s Work**

General Encrypted System’s Formula:

**Y^2 = (X^3 - 3x + b)**

System must be working to on server. Bitloop system work:

Modules:

**from cryptography.fernet import Fernet**

**from flask import Flask, jsonify, request**

Initialize 2 The Fernet Key and Create a List for Hold the Encrypted Messages:

#Key-1

key = Fernet.generate\_key()

fernet = Fernet(key)

#Key-2

key2 = Fernet.generate\_key()

fernet2 = Fernet(key2)

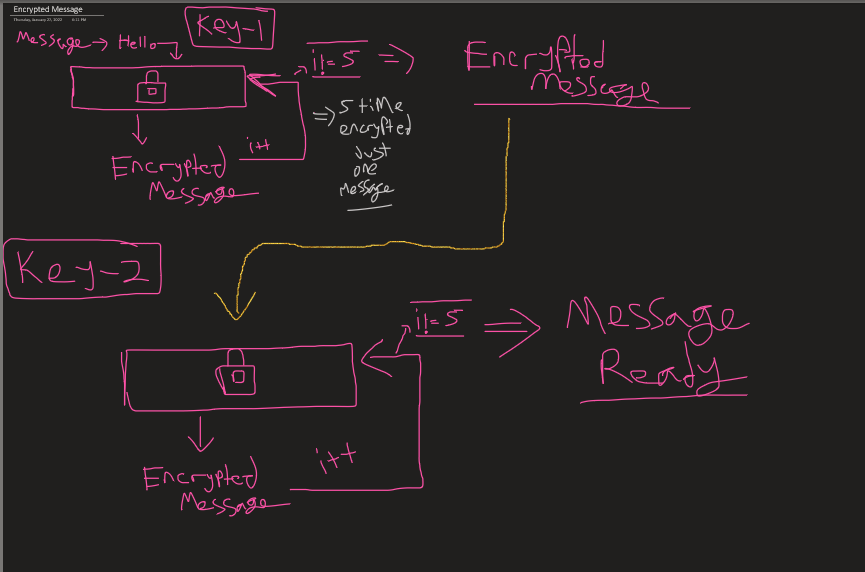
#List

Ciphers = []

#Calling Flask Method Creating Flask Object For Server Methods

app = Flask(\_\_name\_\_)

1. Encrypted Message:



**Code:**

@app.route('/encrypted',methods = ['POST'])

def encrypted():

text = request.get\_json()

keys = ['message']

if not all(key in text for key in keys):

return "Json key issue!",404

#Key-1 Encrypted to User message

i=0

enc = None

enc = fernet.encrypt(text['message'].encode())

while(True):

if i==5:

break

enc = fernet.encrypt(str(enc).encode())

i+=1

enc = fernet2.encrypt(str(enc).encode())

i=0

while(True):

if i==5:

break

enc = fernet2.encrypt(str(enc).encode())

i+=1

sha = hashlib.sha256(enc).hexdigest()

ciphers.append(str(enc.decode("utf-8"))+","+str(sha)+","+str(datetime.datetime.now()))

keys = []

for i in range(len(ciphers)):

key = ciphers[i].split(',')

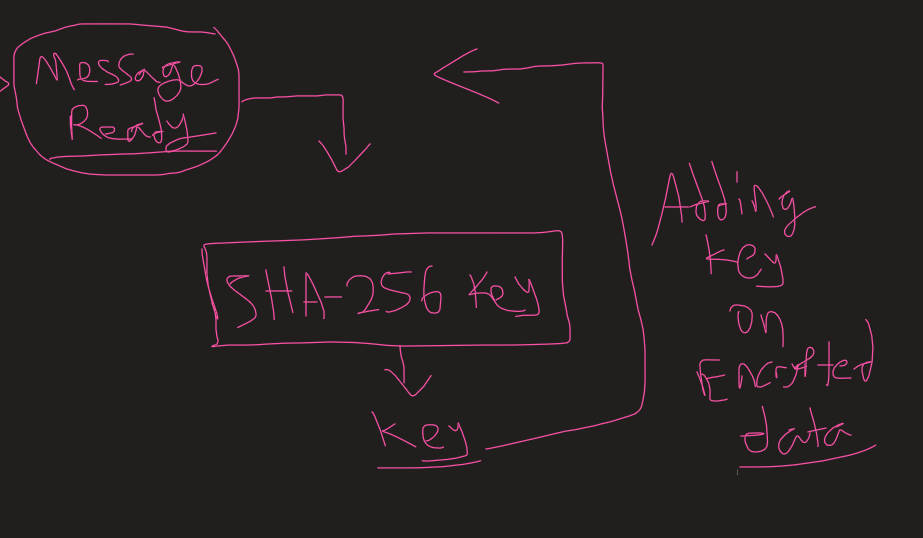
keys.append(key[1]+","+key[len(key)-1])

random.shuffle(keys) #Every request changing the order randomly of keys

response = {'ciphers':keys}

return jsonify(response),201

1.1)Giving the Sha256 Id On Encrypted Data:



sha = hashlib.sha256(enc).hexdigest() **#Encrypted data convert to hash with Sha256**

result = str(enc.decode("utf-8"))+","+str(sha) **#String data**

ciphers.append(result) **# we create a string list so we add that string data**

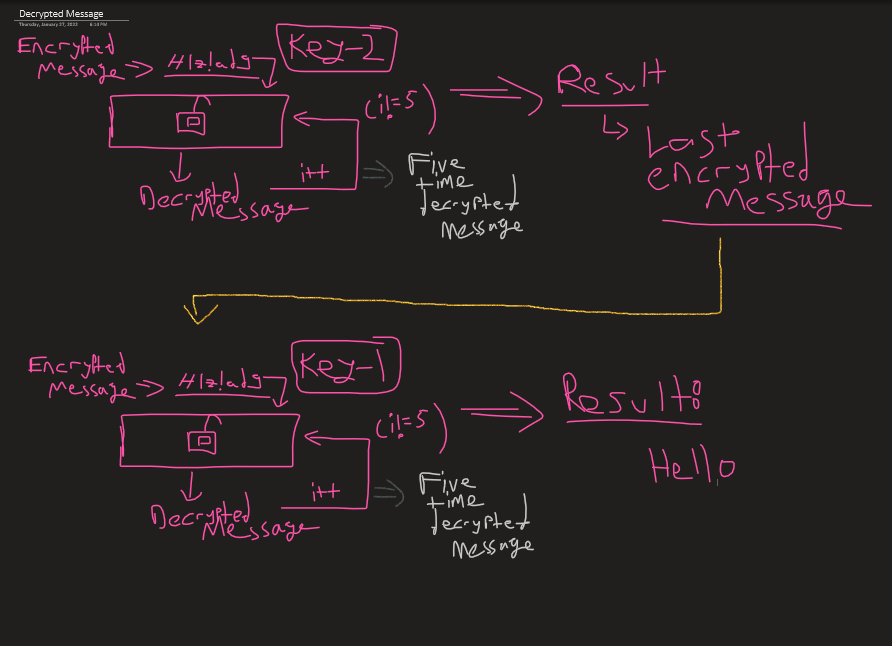
**Example of Encrypted Data:**

gAAAAABh8-9dv5mdivAueb5oZ\_\_8x2dh1JTBqr5c6HVFQR9wGwUYdEMXJNQcgRS5zZXhHO4YqhcBSiWRSFn6k0FczpPU9z\_IlbUhDX4IDa9GfN8C4EZJxrjraWe086xpD8m9uKr4qfUFUyfKSYPqt2y7iAFfRDcyLlLh\_kCXlwjRJPjlf8sfPs-cAT11Yxgvt3BEHMHYdBBgZCCLvCDx4de8eGbXRBbNGRZ-wWqnjsC1uhvX2aOFjZEfzIPSTw\_6vzse6qfTNEhWvrZTPAozGa5cKOY9JVcMXTdQ9n0WIWDHzJNYSTw3aLN9sWvugyX-8g-TJXsASyVrQQDAiuRhsVpIr43zVYILwKdyXxOhs42qcz3hHZruAVBAFJyX2eH1NbM5YgZCi29r6y6iOpLkWyYOsxI9Tv8ddbeQTpx6ummZONCAwjWU\_KM3USCB7YICWs3lIPtHf9GW\_ESredG\_6KGl1VGZjlrw84tvB5nCxCPXEYDclr-xpHlpUPVUWxkQMnnEB2HB8ciQgPFKRlUS5aKkq95KiXi-4LBOBwE87gNKrfD2-enSayKae8f8soEy2pu7fSg9ogofR6f6TJs-JTfcX5stMtXdF9oZwV7nYupOTgM1Mfsi7MxoxlQO8de3-jnrJ5pNnggJS1MZ\_fbKXrE8RVVPA5xof1gNuLWymkTIsYVjcYsA6QeDBWIfqC9Ph2N9c79N1xgFanAKiuvTtJbr5\_j4mdTieQAOlQI2kM5rIaq9Z8HNIV3JLfQLVC6pz\_QZU2cQ8-CuCuZ30GM05aKKFWysFBF4tq810UcO0dfiIaeXUn5SM6FYbUb7XGUknIPAbUFIC5m6bwuAQia5\_hNXaFd7eoZCYW1pjQ9-7WdxG4hfgJXkn1VTA2LA9K5kWVKuLg2j4cUddmCcWwam15E1i9Uvzwbo9XeKKe2AzKIUv4lDMQEdozyxy2-8bwWA42w1ZTb8yiksS4-NrB3f2Tuo-ZE0y70z0S9XCEQJ9baWeaIbKNO4cjcuGCYFn4GiJm8AtqJa-bDvaUwgPRGSvESZ4fz-MUAk-XWfT7RQMIoojFVZ8aU\_9qg1bymbiUUg5\_dRgdPUpd-eCJzahb8JebiAafRVBh3CPDKfjMClZRwC5FLGrNDt1hlEDxO\_1cj8a2bBPwX1heprEvkZqQun4n7LS2uV0Uc9KqZZfJ3TSWDe2okD-6BK-Pxm7Rg0b-jtLQAss6GvFsOb9bhxzkTApsSbeN8wwT7CteN5mcp58qziE5I-WMxPxK-RM8J41aPIpXv1uV9ojaDJJA-ZnM6Ca9gwpMf23C4McQRrSt7Rwk\_aqesitRhkriibb7hxhr4RsqaPy77F-osP7EjqKwlgk0n3A5gu-JjqQzQ1jF0oExm8eplKEwJqtIaGxamiRGFs7e\_LjWTid-57LpaXE3\_jqjoz8urav7CcvgFrAVi3XvMBpNoWdA8FR3-9E1WCBNPYSDrgl62KDzII5h28ly9GcsTbjWJuco0-wiu7-lukc9ORpCIU-wYCv\_yTK-jwVduf3hcokZXQLsqEdqMWQf8NeLHccFG00LrkiQsN-n3KqsKsPcZicv4JWfItdWTQ1vO69vmCFBrS2fvFneDtTwFpy7Yt-BYqRd9SFwP1RIDYL3eCqMZf650EDPkaKUwXFn\_MSJ3e4q3JUq1q9KYPsS\_3W-7t12Fm0xby2mk7UfDOGvpzTzRbCmB9ADaO-RDn4Ugu20yA8BgC2RoLMaBGBnrdJnhA9zz4yK5jastWBySUSQjcdkC5Oe3YNYkBG1oQWPnvzL7V3ZEg7UdW-i3V0vDZH-oqhyfx2MrKhNwTkm9HJXW2nA\_DU9fCKMIpmAgG05LMT2C-sknmCXnxsaZE-bLqlgYcwWbEKp26iki2K9XbcvEzsE0fgDqsVl47x7Z1S8kJAeRlRK0forMa2wvw6QNR-Ta\_B5YDYnaipiR9l527Xyh96EgEaA\_1FQWVgQoDlBTnaRnZ3\_pzm6iMpxCu5WTJcUMMDegW50UOPzEYazJotipDAoc4\_jdiWHakCZFwtYXv7NqZIqboBBCA7-JF-F0nsAP3X\_1FK-7xiwmLJMaoG9p62yYdvXnM2mc1SfDzgh7DUI0dhV0ZssDxUv2joXswTLe5pZJKEg\_nnnjBQ8k1zBlaDKwygFNFv2jHpHy3NlUHatGhNklWEr6fPXgJiW0oSJQlQSEqH\_sQanGT67bMuR8L71\_2BtwgiWDkjPMGeKDf5Gy\_hnv93\_0QpL7h3y59J-NX-\_VkzCu9cb50SDVQykZ3TwjATWVfh-rIp4r6c6g2b6tzSMNugoWlywOuG1Vw6bs9GedhzpF-UQoQFjo4kRXYaS5yfpauDIrUjnWnUkVRFzx51U20jU38etCDw3ibPL79WO0QUOTK9beYI2XT-J6RvS1ZjLl3CStrZdFJXZA63aQG0whTSyWUMwoFXM82nv5mJNJjdrepeQjZbdzTulxs\_f-q3jYWy5H4m85kI7vxfHlhFtC2FV9IVN4DJ5zgBgNg-aDgIkQ1Gl7evBI8CBIuUgqOMNogkyBj79UkpnPyBYTImc3pQkz\_FW6WUmSKZ8FC7RlSd6qUQKmq5ti53hm0OrRlBBbGyOXV2SxY0tRQvq6Evvt9aP9QJsCawz8ypEVpPcrnv7YEPlTrD1Zhcw4dv735xxrTsH6ynuRpE5Xv5Qx-NnPzOaNLKjaqVgBj0ReeFffM5mXyjj6OihfvmDmhgaSXNXgXb4NYt9A5VyVPJWiWWIUFIQV8nZ9xxxTdetShqtqdwnDCtdNCOOXzYdSLRLo0slVe5aHZb9qnI23TAVGaIplDnqtoD7by59f4rWvJwRfCYtMamsQyFtafQ5Z9VNPIaoRX8Hq2ngLaBeiwV1qpTVIP3QvtBtCwknqSfAfZRf5gX2ezBzLjRdSelGDo1gpRDP-kR9Jt4wVAlZrnWf0A-zsg0FZdHZgTMp35Ppl9p28i7IGHkCgtTe-fBDnzFgasl\_YTEJjoQe0xcTA30LqAgYD\_-G3yM3phsrC7ZbhB0YZ\_KNIVzlIwZpKKefDSlUis\_KkEEReh1ykFZ6\_4SIqcASp5-5\_sxgGl3zNEptrStc5FWlxljcahsHGC\_9Cro7zCfRmz9Of2mcQ9AL1P0zfIIkl1E8qbUF1tGrsrtykA8gpVUFOYlVjnyXFXuaCKsu-PiZwzmtjpYTq\_DpM1BmGAQyY8VtXdBJ1tq4ACzZOf7YoG6ZOcvbLvN2nfja3iNpswNazcIXoSBzNWW\_9tDXH58hTETn8dKZbyNJY9ttyfL7kmQOHZKvis2VePVpIE8tsGkd6U60TLvCq6cMZ6FFdX9Ts0qSWbmPYSI7DnyRdnlDEjVL5TM4ZhBx0QB3job7QXgA-QKMDEPPHKPX942UpFwV5EtFk1kn8rfcuseGfgNqghYIab3EOaX8z7eUKcMgAI\_uP3ypoEp0HJo8reE4KcDAkve3\_lbJkr3hA0qSNhKJmaoC6QK9pEUsKfk7iK\_Hpy-apPy\_QyCpm-83ID6VWqXuK0N480vsBUlCDPd\_5ECOn\_OLWj2Ci6N\_Ej1si8QYhXSy9zbGvAmg7lm4s1xOg0BDskBjvwd6Ri0zucrOc3SjwAd-6GkW6sS59b1hoVmCgKe1jf3cqpRWYsBKnUZoPnym0XU4uvdvfs2atuYVAD4pettJIWdoq\_Q4HaWGI8CUwR7PPKmv69F9q5wQ7v-y7veqJeYoZiNzq6iQlyzSBnybsc5A51aMpTKxJbby8r8N64tmCApVl6sOGnsWlqgR7-YsDR8NU78jfDLBWPYFMbUZNfAEUB0uvVoZrpmaVDLMCUmsB7AOmIFWJvc8n19ElKkAab\_f9Nh4fSxDoS7Bh38g43PQoX6ALR41bZgsCFDQ81VdkHc55Wq-JPug7wqRzkY7LABeITQmgy15ATIsYBYpLBCqrju0vgHa2unwG4Eu-U08V9ye6GX6465GuXxlqD0tS8VMKiYvRcqx2w7Miqnt4kj1Dr9AFVZhS2BTQncHCh5MQ9N\_cor8Mb8J1X8BFs19H1QGT90KqnU1ZmlRWgosbpF-Nqz1dUqzV-FTF6fYKme3Q8TVwBPo-HNOt6grrRWuArL2vF1aBhqeWr2erxRimnqjWLnjZhJWHF1OxoORQrdv9oIXab0v-h6acUjOqEvWjYQEEhv5Qaj6VH2nvtTMpicyyO932iljKUeRiFqOqmFRwzsYcwOmbGnCdry6gAKxTnYPbAkrZ4mr9AQeLil\_Jghp3IW-5iTvnW5laweWEUKbxN2yuPoLZjvammK5lQmq8gxN6\_bKcdkGQR00SGtLHNhIF-s1-Tp8xyhZcZ2Ti2Tz9NC39dRPvnUA8glhd5nS\_vsCau80d\_07YnmTQgnfZQog8bct6KDEv5HXGe6TXkqYHcH6-yHoQ8freFPC7WRNmJ85huEpoKv8wyDAwN0diQiw7mi\_-I6dHMspcMZl9Q7C4JQLl8YHM2AL3Zacfu9XtCyxqSSKtuKCO0XXHOZ9JOdH1dUCnnBwbkEkDxEruG8lhl-5vEfDBzBiphpF7lyOKKk81JHglH4bYvZzMvKuAdSCCRVJXeS3o22gJ0FhZrGN1XP9ibSxfhHzcs7f0KSIHU4p1UxaU3HAx\_nReMxYmTDiKYs27PKTAMwB7NsOfdfbORUtA5bwptd88pA8-cfMNzJUwh-IWs7pCAHvqwyyeU-s7tTeDoSvjwf6Sk3OYi2d7GUmnsg55HlG9rTx4zlMHnRHTcTx-g7f-ajdyFavZCmWIxrcZSVYkKsfGiqVFbbGLyWc6yGfmmgZcMvh3K6amfyT\_jnVHYZfgfGU2dD6V9hzJlJ1gnWQBG4jbCmH10C8jvtJ9Ar7UnvRjShnYgVCdKTS3ynGPFJLw9PnDXDvXb\_v3EdlkQ9ZFxsUVbKV8gsKTaMe4T-cF2Gdja6tuXpdI\_Jra5LfLcTRy5MTTX\_NcLZmlIAfyni71FAiWII2o7MoilqmNDNnRqeKX6q6Hy4e-ySkofEmqUcQmUG8EMnVeec3lzZTrNtu\_mQNg86d1G6p-VdJAQP5NWXJfNl2SQQM3zB9jCMZh4NPhlP4tpyEcEsPj-G4Iolta62Df9WiOew\_2qZBFy5GVe21bAUJCMqtc13rISCAjpjXmlQJX9ekNAQi9T0Yb2ubzvGkmfEaunWCSOmTvllYYfxL\_-h9\_as3vlPmzxFgjkA3e0We3Us7PHrxG8JuNJAH2iMg-aX5ph8GgNfwRt5hLMlUWVJHt6hD9F6PHzFtbgA-PyM3VpCJxa6xSMXVvWrL9ChThQSaU6BTz\_DjypC-SAPUli41OfuQ9Pt22DKENsRSIh7nyhmKEGEyePkHFZ7SDrBzIIWFWCAoE12H8kj9v5cyjfT\_-IepUcCgiOQixzRF68tRz0TQKX-Hh0N6VoJ4i1ArO5jYsHjAM45VR7vL6yqFxfC-46iGOtTiZ9W6KmA6f-qtX6c8Q5rutCnMq3QS0XXWbQ6HXb\_BLsvVNUNuAMWD2DBQjBttkeSdctm1rIAyzLXP8mNwIGU2AK-pMecYDncVm-l1tMcqZaVtT9r0JZwjjdNV5d6pzJtdeWNAnLJ6RV1Jcs7uS\_Xb\_1hmllajnoLs0r2P9y5g2ueFexAFsP5gU3irtcXch1HlHhg6B2v\_HABU7NFoRc27OXg4Y-8vbEmtUcKm6ElUjWq1TU95CTV3Y3ttSAvwFEqZ0njApLYD8wSpFTA7buYUaO\_ooxrK\_kBFcpQ0zH28F2KLWwGuUVr-S\_lrJbfXFRLUdZcJoZhHcftgC5yVDleyerkBCmMKSeFhErjE0T9FYdtO4-jgDSITIbjdV8w4t\_Uq1ySnOFihWWAMXkTAHcUfZOLi63OE-HAWS5o6mRkR0VBydNXqu5z0A-2mkCdo9Ouz-e85-aw7mToO3iQsa5ftpWs9XAXB5QjV8bGxe7Nqu8bYzP5vTuabHgiPldWR2uMaOfbEBUmfb\_JPQkLl-ePp3jMok\_Te8Su9qf2HeskGHTHj0fa7BAMk6R5FJUIIJuJGyy-ZbZ0inHIK4LGpuubhvE5t\_eNzA7ppG7DbH-Q8L7iDMd3BuyAAnINgMBMhh45HUnyB6IPm5KI-tTmvhA25s17I25SmkVGk9tkZQ4DXzBeb7h06tZrUVyINUplLOR8xob4CojoJwN4WRLKpaUhVyEPOkFLiJu43AyjxiSDJ7jyC7DWokrztB4pzXgl5l3rDng9FsToSKaqrR-B3dUMF3XtfUBdAlGvkCCWCfQYtY1xengVMnRC9uyt\_RjePYnT53MmNqvac8WGB5DF7X\_oKSkEyOP0GHj9e4T8TZTGtqtCzlYqF3qnq5u\_rGXvtuIhWPhF9xe8\_knjEtAAZGIZrpd8\_zcHZB-j44OwSnBxSrTs6RjyMIrobaC\_64wmWSIo-nqcP\_juM76SOOckpFjFmjcWmc9PBMoan7MOhHkkuuDp0StFg2BJ422fU2csnJ-J4GfwNFMcKTr0qHqfIfflGXyxdi1hb5KdabUkUZO8MMJ6nU72NpxldS8UpgHG15Abspcyl\_9X93-Le922p7RfqbYPS33YOOADZIdRtppNMf8Wb6\_0ESfYGvFYit258k3yxA5gUEF0Qx7HLnctB6WuKRIh8piW3lpy57bAM7w45rCd4PSXmFv36fAM5VzTXbBRv3VyjI0tzsPn914gzhxfIp7JkNZemTo3Fqf\_xZl9L3OHVR1y4YhinKtyVAm1fcnGpl2O1kW5D7pPOwpNCPVxPjVWuQANJ-Dz7fpr4kQdzmKCKgiXeXPcdcebEag1IdD-\_wlVR3iDj1svNHV9N8im1VUhCl4xRScVGlzoD-c8-C25w1imVk\_en3IbKd\_swSm5TG4S3gAnW9cZASJGR3GBGKQa4fvHcjUKyomvGvvmH0JOdLfxUI208-G1AQKI2C0h9FUdIjqWPgEgZH5sWBN6lPer03F4dwbRARzgCgAcz2wKoAmpe7E6AdgZx27qcLX3As1JpEUDarDsU7KdTuxdNYryUINb3Pq\_SvEbPDdG3yzXTPxLK0WG\_TMbQsR\_3B-IDvKpoisXRwWgFzAOwOrRMlgft-wM0MdMStnQJDBx2oKne0pMdYWE9IBcBc93AEvlpL1zQ\_lql2-vqW7m9Pntjh\_B8WIdnuaZMFDonv8PC1qkFp1wSOOtbj2YMN1DPthyGxV3Gtz9cmTT8ze98B1X-m8oXVwKE3gMd9RX4-bFV6BnoVCQv2DTxcVpG3OuSEgAeSI1AiZrG4iRkeKDg5ylq\_vfFL1pgRuHLYx6IwMmWrwSTmkGoO74n\_0JwdwS7IjH12CjquvLr\_l\_Koyivir0v4kqw0t4d4NzxHvn1YMQbR\_ZZWxXdU19-e-zLuAFEHYVlN-9jjzgKypNLrWLGCfmPbP0QHygZb5IDQPuHKuji-ZeFcroJUGWDgG4L0c99nUpqyGru0iYwkEvqdVSWkNf6aa-TSVuxcHpvia7MrvIs5Kd2zwoDGBubrT-J3TCAzV-FEm7N0U7C4fH071jw9JWJu8GQYDh8F7tnRsg\_9BYCNt8FDKWnzmkq4SiboIr4xFWKb4gTj7zWaG1cx2i7c4l7psLPkh1MJjQfY8pXXJBjwf1e7VvhygE\_r3X56KzjNJIWT\_wCENgg7gHlil1iWdHQCTQVrbzhNdLDQYgNF-9Zcn3Yttf45M3xcfLhZO3zyi8P-PGUr\_ANXqzQoly4YqWAXTiVQvCOw6wLd-dDJqxF5Xtr7RX1qiyfPWk7x0u8QquZYa3hF8YjXlTZiWFS0jRREHpeCuPrE7cOyRlGjBclDRKUguzA6JkPMBli4nUq\_\_Px8L2f6\_p55KTfb7oiZFFI9bJ\_XPvtD7NFvPxrqM24Ktc53pa4j9boJ8Tz92yO9fvVRWt3z5OmkM2T5rOmkQHOxxFvMT2qVihfBk4veVcstiv5aFPWk13JELfqLg6Zc8Ed9EcomnR6IBzMJOtiOUWkkNs4AeFyBtHi6ESp5lGsCFlDXznPAOjU9pPaaMzqnNW7LOd64toLLMfGYoYrnFbxFDxxgBMJGDgqTl0GP9ceso6qq0MngPE9V0PQ9vf4-tJjOtWCxQ-LaqMUpfb2RrbKE375Chu4u7JDn7MmKh3sa\_Xn3fQFvhYU1ee6dgIwD60w2wRuAnNsO6HC9SYwHme-IMFbWtit55rMbXAlCf5mOYqjiUZMJttYonyCbHEoL0MMzA9Q95Wr6X\_fBKPOgSiUFJdqiZ757ckgnv2hGk4Ll-Lu6vsNAP29BYbWP6T\_K6AqB1QcBcXZUkLRDeUMsMZ9heCPa2hGxB1sCAA9hVLnUEkG0Wh97VFDFyah8PvA3tnMoRS\_UMKRSUnSPoLHhWQkg2y9Nz72\_2yK973V4UDIdunEEA5SIQvGi4CxsSal86MRmioGVwl-2Ya-g30aMc0etuBQ==

Data is So long so this data should has unique id and this id is given from Sha256 key. Users should use like this Sha256 key for decrypted the cipher message. When every encrypted message user get like this Sha256 key.

**Ex:** 6f0ab6676ef590d82fe6cec14b56527c30938faa1685d47dc152913669117073

2)Decrypted Message:



**Code:**

@app.route('/decrypted',methods = ['POST'])

def decrypted():

try:

text = request.get\_json()

keys = ['encrypted']

if not all(key in text for key in keys):

return "Json key issue!",404

#Key-2 Decrypted Key-2 encrypted message

i=0

key = text['encrypted']

cipher = ""

index =0

for i in range(len(ciphers)):

data = ciphers[i].split(',')

if data[len(data)-2] == key:

cipher = data[0]

index =i

break

text = bytes(cipher, 'utf-8')

dec = fernet2.decrypt(text).decode()

while(True):

if i ==(5+index):

break

dec = dec[1:]

dec = dec.replace(" ' ",' ')

dec = bytes(dec,'utf-8')

dec = fernet2.decrypt(dec).decode()

i+=1

i=0

dec = dec[1:]

dec = dec.replace(" ' ",' ')

dec = bytes(dec,'utf-8')

dec = fernet.decrypt(dec).decode()

while(True):

if i ==(5):

break

dec = dec[1:]

dec = dec.replace(" ' ",' ')

dec = bytes(dec,'utf-8')

dec = fernet.decrypt(dec).decode()

i+=1

except:

dec = "Encrypted message is not valid!"

result = f'{dec}'

response = {'result':result}

return jsonify(response),201

1. How many we have cipher?

**Code:**

@app.route('/encrypted\_data\_length',methods = ['GET'])

def encrypted\_data\_length():

response = {"total\_len":len(ciphers)}

return jsonify(response),200

1. Get the last cipher time?

**Code:**

@app.route('/last\_cipher\_time',methods = ['GET'])

def last\_cipher\_time():

dates = []

for i in range(len(ciphers)):

cipher = ciphers[i].split(',')

dates.append(cipher[len(cipher)-1])

dates.sort()

response = {'date':str(dates[len(dates)-1])}

return jsonify(response),200

1. Remove the key

Code:

@app.route('/remove\_key',methods = ['POST'])

def remove\_key():

text = request.get\_json()

json\_key = ['key']

isFoundKey = False

response = None

if not all(key in text for key in json\_key):

return "Json key issue!",404

for i in range(len(ciphers)):

cipher = ciphers[i].split(',')

if cipher[len(cipher)-2] == text['key']:

ciphers.pop(i)

isFoundKey=True

break

if isFoundKey:

response = {'result':'Key is successfuly remove!'}

else:

response = {'result':'Key is not valid!'}

return jsonify(response),201

**Conclusion:**

It’s not possible to crack the Bitloop encrypted data. But system must be work on server and if server down all encrypted values can not decrypted the system. Main goal of this technique, secret message can not showing other people. Privacy is one of the best significant thing on Bitloop.

**References:**

<https://cryptography.io/en/latest/fernet/>

<https://csrc.nist.gov/publications/detail/sp/800-90a/archive/2012-01-23>

<https://www.youtube.com/watch?v=ulg_AHBOIQU&ab_channel=Numberphile>

<https://en.wikipedia.org/wiki/Cryptography>

<https://flask.palletsprojects.com/en/2.0.x/>

<https://blog.malwarebytes.com/threat-analysis/2018/03/encryption-101-how-to-break-encryption/>

<https://www.nsa.gov/portals/75/documents/resources/everyone/csfc/capability-packages/dar-cp.pdf>