Encrypting database:

Within models.py, we imported the django_cryptography.fields and modified the Card object's data attribute using the following command:

data = encrypt(models.BinaryField(unique = True))

The resulting data in the database is now a blob and not stored as plaintext anymore.

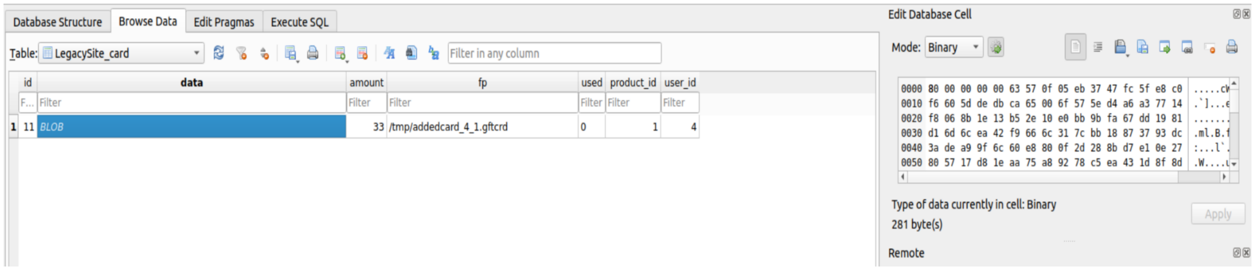

Table: LegacySite_card

	id	data	amount	fp	used	product_id	user_id
	F...	Filter	Filter	Filter	Filter	Filter	Filter
1	11	*BLOB*	33	/tmp/addedcard_4_1.gftcrd	0	1	4

Mode: Binary

```
0000 80 00 00 00 00 63 57 0f 05 eb 37 47 fc 5f e8 c0  .....cW
0010 f6 60 5d de db ca 65 00 6f 57 5e d4 a6 a3 77 14  .`]...e
0020 f8 06 8b 1e 13 b5 2e 10 e0 bb 9b fa 67 dd 19 81  .......
0030 d1 6d 6c ea 42 f9 66 6c 31 7c bb 18 87 37 93 dc  .ml.B.f
0040 3a de a9 9f 6c 60 e8 80 0f 2d 28 8b d7 e1 0e 27  :...l`.
0050 80 57 17 d8 1e aa 75 a8 92 78 c5 ea 43 1d 8f 8d  .W....u
```

Type of data currently in cell: Binary

281 byte(s)

Secret key management:

The first and most basic step for key management is restricting access. Encryption keys should only be accessed by those who need it, denying access to all others. All accesses to the key as well as any operations done on it should be logged to allow for easier investigation if the key was to be compromised at all.

We can also have a certain lifespan for the keys and exchange the older keys for new ones when the lifespan is over. Those old keys should also be permanently deleted to reduce usage of unused keys and give more protection to the system.

The keys are currently kept in settings.py, but we could also alternatively keep them in their own database which is separate from the Card.data that they are used to encrypt/decrypt.