پروژه درس امنیت و حریم خصوصی داده

محمد مهدی سوری

در بخش اول پروژه الگوریتم رمزنگاری با قابلیت جستجو برای پایگاه داده های رابطه ای را از مقاله زیر بیان کرده و با استفاده از کد نحوه پیاده سازی آن را بررسی می کنیم. سپس برای بخش های دوم و سوم پایگاه داده ای را ایجاد کرده و داده هایی درون آن قرار می دهیم سپس با استفاده از الگوریتم های رمزنگاری آن پایگاه داده را با انتخاب ستون های دلخواه از جدول رمزگذاری کرده و پایگاه داده جدید را می سازیم و سپس پایگاه داده جدید ساخته شده را نیز می توانیم رمزگشایی کرده و پایگاه داده جدید متن ساده را ایجاد کنیم.

الگوریتم SSE (Symmetric Searchable Encryption)

هدف از رمزنگاری با قابلیت جستجو این است که پایگاه داده مان را می خواهیم به سرویس های ابری برون سپاری کنیم اما لازم داریم که محرمانگی و حریم خصوصی داده هایمان را نیز حفظ کنیم از طرف دیگر نیاز داریم که به پایگاه داده مان که در فضای ابری وجود دارد درخواست زده و رکورد های مورد نیاز را دریافت کنیم. بدیهی است که راه حل این موضوع استفاده از رمزنگاری با قابلیت جستجو است.

این الگوریتم از پیاده سازی چهار تابع زیر تشکیل می شود.

تابع KeyGen

این تابع به عنوان ورودی کلید اصلی (Master Key) را گرفته و با توجه به آن سه کلید دیگر تولید می کند. دو تا از این سه کلید در جایگشت های شبه رندوم و کلید سوم در تابع شبه رندوم استفاده می شود.

جایگشت شبه رندوم بدین صورت است که لیستی از اعداد را به صورت مرتب دریافت کرده و با توجه به seed به حالت خاصی بر می زند که شبیه رندوم می شود.

تابع شبه رندوم بدین صورت است که seed را دریافت کرده bitstring شبه رندوم را تولید می کند.

تابع BuildIndex

این تابع به عنوان ورودی پایگاه داده را به همراه سه کلید دریافت کرده و در نهایت باید ایندکسی را بسازد که این ایندکس به فضای ابری ارسال شده و در فضای ابری باید بتوان با استفاده از این ایندکس و trapdoor (در آینده توضیح آن می آید) عملیات جستجو روی پایگاه داده رمز شده را انجام دهد.

ابتدا باید به ازای هر کلیدواژه ای که داریم مشخص کنیم که کدام رکورد ها با چه مشخصه ای (id) دارای آن کلیدواژه است و آن ها را درون یک dictionary ذخیره کنیم. کلیدواژه ها را به صورت نام ستون=مقدار در نظر می گیریم تا مقدارهای مشابه در چند ستون باعث تعارض نشود.

حال باید لیست پیوندی دو بعدی ای را بسازیم که به ازای هر کلیدواژه ای که داریم یک لیست درون خودش دارد که اندازه آن لیست برابر مقادیر منطبق با آن در پایگاه داده است و درون آن از Node هایی تشکیل شده است که ترکیب این سه اطلاعات هستند. 1- نام id 2- کلید تصادفی برای رمز کردن گره بعدی در همان لیست 3- جایگاه گره بعدی در آرایه A (توضیح آن می آید) که این جایگاه با استفاده از تابع جایگشت شبه رندوم اول به دست می آید.

سپس باید آرایه ای را ایجاد کنیم به اسم A که اندازه آن برابر است با مجموع تعداد رکورد های منطبق با هر کلیدواژه باشد. این آرایه را با استفاده از لیست پیوندی دو بعدی ای که ساخته ایم پر می کنیم بدین صورت که به ازای هر کلیدواژه یک کلید رندوم صفر ایجاد می کنیم سپس گره اول را با این کلید و همینطور گره های بعدی را با کلید های گره فعلی رمزگذاری کرده و با استفاده از تابع جایگشت شبه رندوم اول در آرایه A پر می کنیم.

سپس آرایه ای را به نام T ایجاد می کنیم که اندازه آن برابر با کلیدواژه هایمان است و ایندکس عنصر هایش با استفاده از تابع جایگشت شبه رندوم دوم پر می شود. عناصر این آرایه یک رشته بیت دودویی است که از xor کردن تابع شبه رندوم و آدرس گره اول هر کلیدواژه در آرایه A به همراه کلید رندوم صفر مربوط به آن کلیدواژه تشکیل می شود.

ایندکسی که به پایگاه داده ابری می رود در واقع همان A و T است.

تابع Trapdoor

این تابع یک کلیدواژه را به همراه سه کلید دریافت کرده و در آخر باید یک توکن بسازد. این توکن شامل دو بخش است 1- خروجی شبه رندوم دوم برای این کلیدواژه (برای به دست آوردن آدرس در آرایه T) 2- خروجی تابع شبه رندوم برای این کلیدواژه (با xor کردن آن با عنصر متناسب در T می توانیم آدرس گره اول در A و کلید صفر را بیابیم)

تابع Search

این تابع در پایگاه داده ابری اجرا شده و بدین صورت است که ایندکس را که همان A و T است را دارد و یک توکن را نیز در هنگام جستجو دریافت کرده است و در نهایت باید شناسه های مرتبط با آن کلیدواژه را برگرداند.

ابتدا با توجه به مقدار اول توکن می آید و مقدار مربوطه را در T به دست می آورد سپس آن را با مقدار دوم توکن xor کرده تا مجموعه آدرس گره اول و کلید صفر را دریافت کند سپس این دو را از هم جدا کرده و با توجه به آدرس و کلید، گره مربوطه را در A رمز گشایی می کنیم. در آن گره نیز آدرس و کلید گره بعدی وجود دارد پس همین کار را تا زمانی که آدرس گره بعدی null باشد ادامه می دهیم. در هر گره شناسه نیز وجود دارد، این شناسه ها را در یک لیست ریخته و بر می گردانیم.

برای پیاده سازی این الگوریتم ابتدا در postgres جدولی مطابق با مثال مقاله ایجاد می کنیم.

```sql
SELECT "ID", "Name", "Surname", "Age", "Gender", "Occupation"
    FROM public.data_table;
```

Data Output | Messages | Notifications

	ID [PK] text	Name text	Surname text	Age text	Gender text	Occupation text
1	001	David	Smith	38	M	Farmer
2	002	Mary	Grant	27	F	Lawyer
3	003	Julie	David	19	F	student
4	004	Daniel	Farmer	45	M	Lawyer

تابع KeyGen به صورت زیر پیاده سازی شده است.

```python
def KeyGen(k):
    random_class = random.Random(k)
    first_k = random_class.randint(1, 100)
    second_k = random_class.randint(1, 100)
    third_k = random_class.randint(1, 100)
    return first_k, second_k, third_k
```

تابع شبه رندوم و تابع شبه رندوم به صورت زیر پیاده سازی شده است.

```python
def pseudo_random_permutation(key, list, index):
    random_class = random.Random(key)
    shuffle_list = random_class.sample(list, len(list))
    if index is None:
        return None
    return shuffle_list[index]
```

```python
def pseudo_random_function(key, word, number_of_bits):
    random_class = random.Random(key)
    number = 0
    for char in word:
        number += ord(char)
    int_random_number = random_class.getrandbits(number)
    binary_random_number = bin(int_random_number)[2:]
    if len(binary_random_number) >= number_of_bits:
        return binary_random_number[:number_of_bits]
    else:
        return binary_random_number + ("0" * (number_of_bits - len(binary_random_number)))
```

گره ها نیز به صورت یک کلاس پیاده سازی شده اند که دارای دو متد رمزگذاری و رمزگشایی هستند. به دلیل استفاده از mode CTR الگوریتم AES نیاز به ذخیره سازی nonce نیز هستیم.

چون می خواهیم که شناسه ها هر بار به صورت یکسان رمز نشوند لازم است که از mode هایی استفاده کنیم که چنین قابلیتی را برایمان فراهم آورند.

```python
class Node:
    def __init__(self, record_id, key, permutation_output, nonce=None):
        self.nonce = nonce
        self.record_id = record_id
        self.key = key
        self.permutation_output = permutation_output

    def encrypt(self, key):
        cipher = AES.new(key, AES.MODE_CTR)
        cipher_text1 = cipher.encrypt(self.record_id.encode("utf8"))
        cipher_text2 = cipher.encrypt(self.key)
        cipher_text3 = cipher.encrypt(str(self.permutation_output).encode("utf8"))
        self.nonce = cipher.nonce
        return Node(cipher_text1, cipher_text2, cipher_text3, self.nonce)

    def decrypt(self, key):
        decrypt_cipher = AES.new(key, AES.MODE_CTR, nonce=self.nonce)
        plain_text1 = decrypt_cipher.decrypt(self.record_id).decode("utf8")
        plain_text2 = decrypt_cipher.decrypt(self.key)
        plain_text3 = decrypt_cipher.decrypt(self.permutation_output).decode("utf8")
        if plain_text3 != "None":
            plain_text3 = int(plain_text3)
        else:
            plain_text3 = None
        return Node(plain_text1, plain_text2, plain_text3, self.nonce)
```

کلاسی نیز برای کار با پایگاه داده ایجاد شده است.

```python
class PostgresHelper:
    def __init__(self, database_name):
        self.conn = psycopg2.connect(
            host="localhost",
            database=database_name,
            user="postgres",
            password="admin")
        self.conn.autocommit = True

    def get_all_data_in_table(self, table_name, attribute_list):
        cursor = self.conn.cursor()
        attributes = ",".join(attribute_list)
        cursor.execute(f"SELECT {attributes} from {table_name}")
        return cursor.fetchall()
```

تابع استخراج کلیدواژه ها از پایگاه داده در شکل زیر مشاهده می شود.

```python
def extract_words(table_name):
    PH = PostgresHelper(database_name="postgres")
    attribute_list = ["\"ID\"", "\"Name\"", "\"Surname\"", "\"Age\"", "\"Gender\"", "\"Occupation\""]
    rows = PH.get_all_data_in_table(table_name, attribute_list)
    words = []
    for row in rows:
        ID = row[0]
        Name = row[1]
        Surname = row[2]
        Age = row[3]
        Gender = row[4]
        Occupation = row[5]
        words.append(f"Name={Name}")
        words.append(f"Surname={Surname}")
        words.append(f"Age={Age}")
        words.append(f"Gender={Gender}")
        words.append(f"Occupation={Occupation}")
    return words
```

تابع ساخت دیکشنری ای که به ازای هر کلیدواژه لیست شناسه ها را داشته باشد در شکل زیر پیاده سازی شده است. این دیکشنری فقط توسط دارنده اطلاعات قابل دسترسی است. اگر پایگاه داده

ابری همه ی توکن ها را داشته باشد باید بتواند شبیه همین دیکشنری را بسازد که ما در آخر برای اثبات صحت کد از همین نکته استفاده می کنیم.

```
def create_dictionary(words, table_name):
    PH = PostgresHelper(database_name="postgres")
    attribute_list = ["\"ID\"", "\"Name\"", "\"Surname\"", "\"Age\"", "\"Gender\"", "\"Occupation\""]
    rows = PH.get_all_data_in_table(table_name, attribute_list)
    dic = {}
    for word in words:
        search_key = -1
        id_list = []
        split_word = word.split("=")
        if split_word[0] == "Name":
            search_key = 1
        elif split_word[0] == "Surname":
            search_key = 2
        elif split_word[0] == "Age":
            search_key = 3
        elif split_word[0] == "Gender":
            search_key = 4
        elif split_word[0] == "Occupation":
            search_key = 5
        for row in rows:
            if row[search_key] == split_word[1]:
                id_list.append(row[0])
        dic[word] = id_list
    return dic
```

در شکل زیر تابعی که لیست پیوندی دو بعدی را به همراه لیست کلیدهای صفر را می سازد را مشاهده می کنید.

```python
def create_linked_list_first_keys(words, dictionary, first_key):
    Lists = []
    counter_number = get_number_of_counter(words, dictionary)
    ctr = 0
    first_keys_dic = {}
    first_key_list = []
    for word in words:
        one_list = []
        key = get_random_bytes(16)
        first_keys_dic[word] = key
        first_key_list.append(key)
        for i, id in enumerate(dictionary[word]):
            key = get_random_bytes(16)
            if i == len(dictionary[word]) - 1:
                permutation_output = pseudo_random_permutation(first_key, range(0, counter_number), None)
            else:
                permutation_output = pseudo_random_permutation(first_key, range(0, counter_number), ctr + 1)
            new_node = Node(record_id=id, key=key, permutation_output=permutation_output)
            one_list.append(new_node)
            ctr += 1
        Lists.append(one_list)
    return Lists, first_keys_dic, first_key_list
```

در شکل زیر تابع ساخت آرایه A به همراه لیست آدرس های گره های اول را مشاهده می کنید.

```python
def create_array_A_first_addresses(words, dictionary, Lists, first_key, first_key_list):
    counter_number = get_number_of_counter(words, dictionary)
    ctr = 0
    A = [None] * counter_number
    first_addresses = []
    for i in range(len(Lists)):
        for j in range(len(Lists[i])):
            key = None
            if j == 0:
                key = first_key_list[i]
            else:
                key = Lists[i][j - 1].key
            new_node = Lists[i][j].encrypt(key)
            permutation_output = pseudo_random_permutation(first_key, range(0, counter_number), ctr)
            if j == 0:
                first_addresses.append(permutation_output)
            A[permutation_output] = new_node
            ctr += 1
    return A, first_addresses
```

در تابع زیر نیز آرایه T ساخته می شود

```python
def create_lookup_table_T(words, second_key, third_key, first_keys_dic, first_addresses):
    counter_number = len(words)
    T = [None] * counter_number
    for i, word in enumerate(words):
        permutation_output = pseudo_random_permutation(second_key, range(0, counter_number), i)
        key_integer = int.from_bytes(first_keys_dic[word], byteorder='big')
        key_bit_string = bin(key_integer)[2:].zfill(128)
        address_bit_string = bin(first_addresses[i])[2:].zfill(128)
        random_function_output = pseudo_random_function(third_key, word, 256)
        T[permutation_output] = xor_of_two_bitstring(key_bit_string + address_bit_string, random_function_output)
    return T
```

در تابع زیر نیز Trapdoor برای همه کلیدواژه ها ساخته می شود.

```python
def Trapdoor(words, second_key, third_key):
    Trapdoors = {}
    counter_number = len(words)
    for i, word in enumerate(words):
        permutation_output = pseudo_random_permutation(second_key, range(0, counter_number), i)
        random_function_output = pseudo_random_function(third_key, word, 256)
        Trapdoors[word] = (permutation_output, random_function_output)
    return Trapdoors
```

تابع Search نیز به صورت زیر پیاده سازی شده است.

```python
def Search(A, T, one_trapdoor):
    ids_list = []
    bitstring = T[one_trapdoor[0]]
    address_and_key = xor_of_two_bitstring(one_trapdoor[1], bitstring)
    address = address_and_key[128:]
    address = address.lstrip('0')
    key = address_and_key[:128]
    key_byte_object = int(key, 2).to_bytes((int(key, 2).bit_length() + 7) // 8, 'big')
    decrypted_node = A[int(address, 2)].decrypt(key_byte_object)
    ids_list.append(decrypted_node.record_id)
    while decrypted_node.permutation_output is not None:
        cipher_node = A[decrypted_node.permutation_output]
        key = decrypted_node.key
        decrypted_node = cipher_node.decrypt(key)
        ids_list.append(decrypted_node.record_id)
    return ids_list
```

در شکل زیر الگوریتم یکبار اجرا می شود و نتایج در شکل بعدی آن می آید.

```
words = extract_words("data_table")
dic = create_dictionary(words, "data_table")
first_key, second_key, third_key = KeyGen(5)
linked_lists, first_keys_dic, first_keys_list = create_linked_list_first_keys(words, dic, first_key)
A, first_addresses = create_array_A_first_addresses(words, dic, linked_lists, first_key, first_keys_list)
T = create_lookup_table_T(words, second_key, third_key, first_keys_dic, first_addresses)
trapdoors = Trapdoor(words, second_key, third_key)
result_dic = {}
for i in range(len(words)):
    result_dic[words[i]] = Search(A, T, trapdoors[words[i]])

pprint(dic)
pprint(result_dic)
```

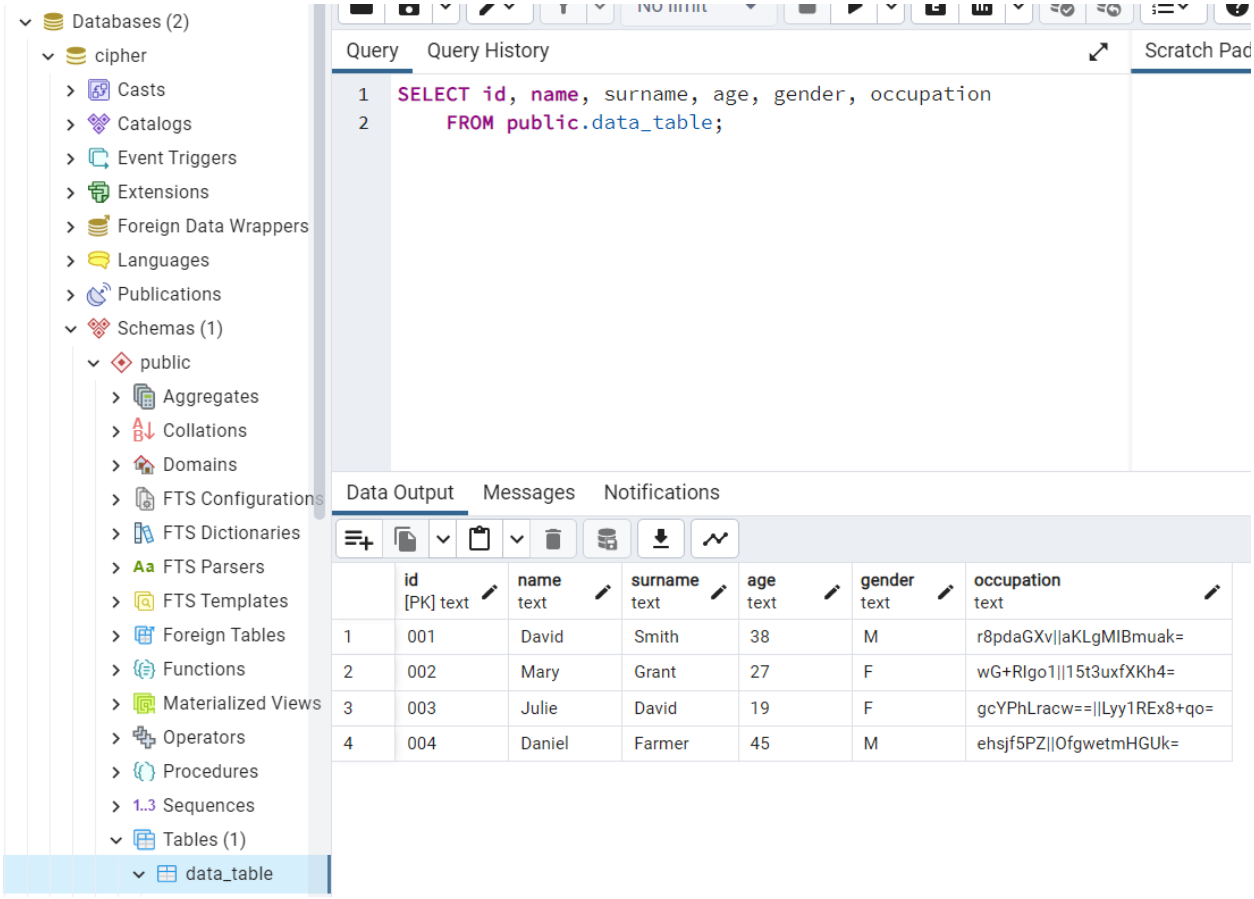
```
{'Age=19': ['003'],
 'Age=27': ['002'],
 'Age=38': ['001'],
 'Age=45': ['004'],
 'Gender=F': ['002', '003'],
 'Gender=M': ['001', '004'],
 'Name=Daniel': ['004'],
 'Name=David': ['001'],
 'Name=Julie': ['003'],
 'Name=Mary': ['002'],
 'Occupation=Farmer': ['001'],
 'Occupation=Lawyer': ['002', '004'],
 'Occupation=student': ['003'],
 'Surname=David': ['003'],
 'Surname=Farmer': ['004'],
 'Surname=Grant': ['002'],
 'Surname=Smith': ['001']}
{'Age=19': ['003'],
 'Age=27': ['002'],
 'Age=38': ['001'],
 'Age=45': ['004'],
 'Gender=F': ['002', '003'],
 'Gender=M': ['001', '004'],
 'Name=Daniel': ['004'],
 'Name=David': ['001'],
 'Name=Julie': ['003'],
 'Name=Mary': ['002'],
 'Occupation=Farmer': ['001'],
```

```
'Occupation=Farmer': ['001'],
'Occupation=Lawyer': ['002', '004'],
'Occupation=student': ['003'],
'Surname=David': ['003'],
'Surname=Farmer': ['004'],
'Surname=Grant': ['002'],
'Surname=Smith': ['001']}
```

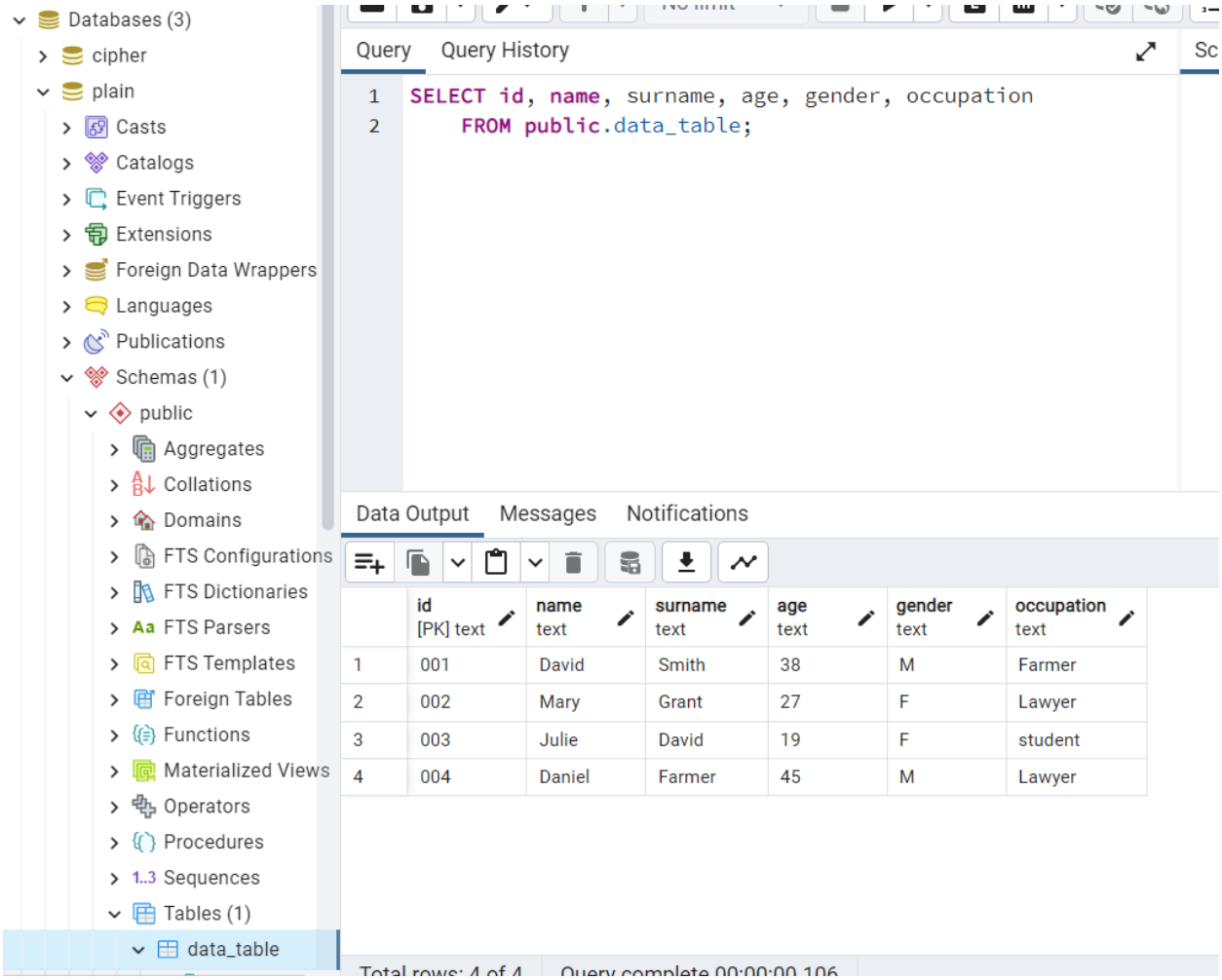
برای بخش دوم و سوم نیز برنامه لیست ستون هایی که کاربر می خواهد رمز کند را از او گرفته و آن را رمز کرده و در پایگاه داده cipher قرار می دهد و در مرحله بعد کاربر می تواند برای برنامه مشخص کند که همان ستون ها را از cipher گرفته و رمزگشایی کرده و در پایگاه داده plain قرار دهد.

در شکل های زیر دو مثال نشان داده شده است.

```
if you want to encrypt database input 1 and if you want to decrypt database input 2
1
which attribute do you want to encrypt ? (Name, Surname, Age, Gender, Occupation)
(if you want to encrypt more than one attribute, you should input like this "Name, Surname")
Occupation
```

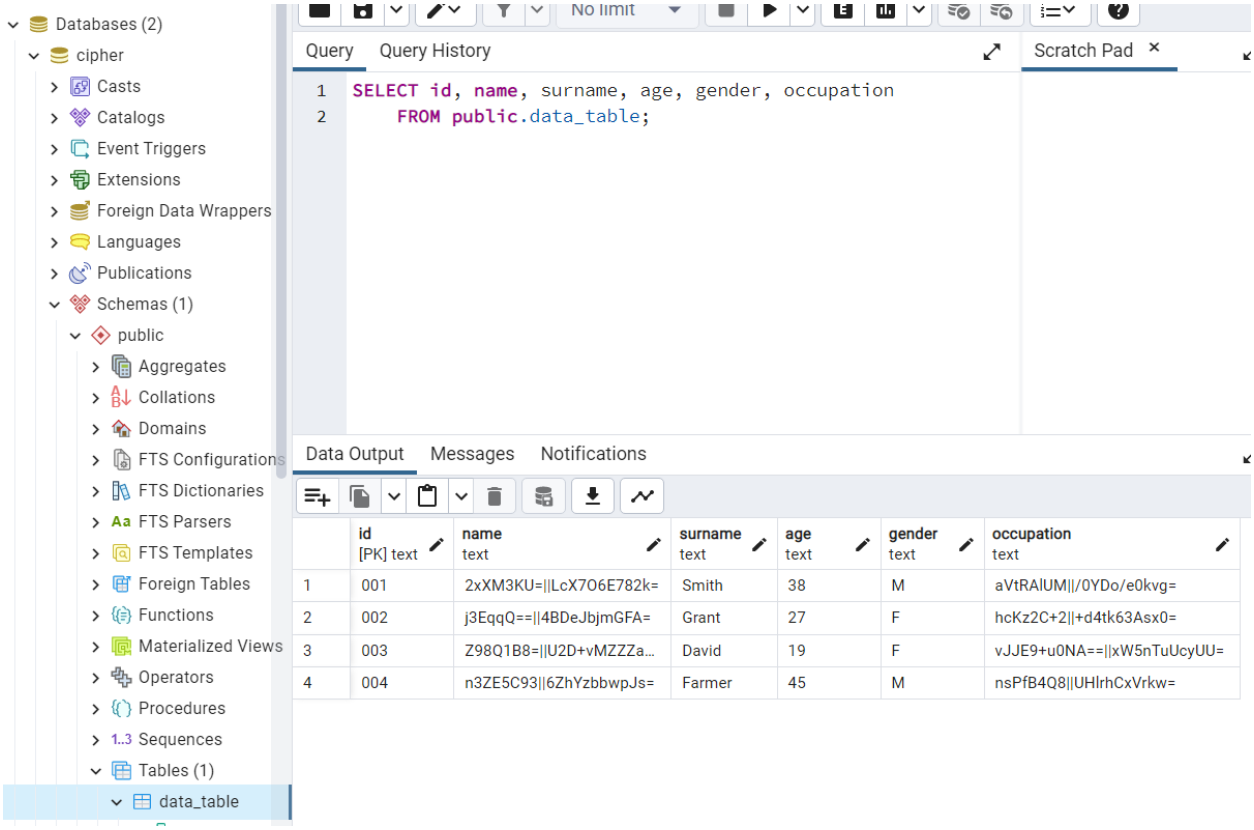



```
if you want to encrypt database input 1 and if you want to decrypt database input 2
1
which attribute do you want to encrypt ? (Name, Surname, Age, Gender, Occupation)
(if you want to encrypt more than one attribute, you should input like this "Name, Surname")
Occupation
if you want to encrypt database input 1 and if you want to decrypt database input 2
2
which attribute do you want to decrypt ? (Name, Surname, Age, Gender, Occupation)
(if you want to decrypt more than one attribute, you should input like this "Name, Surname")
Occupation
```

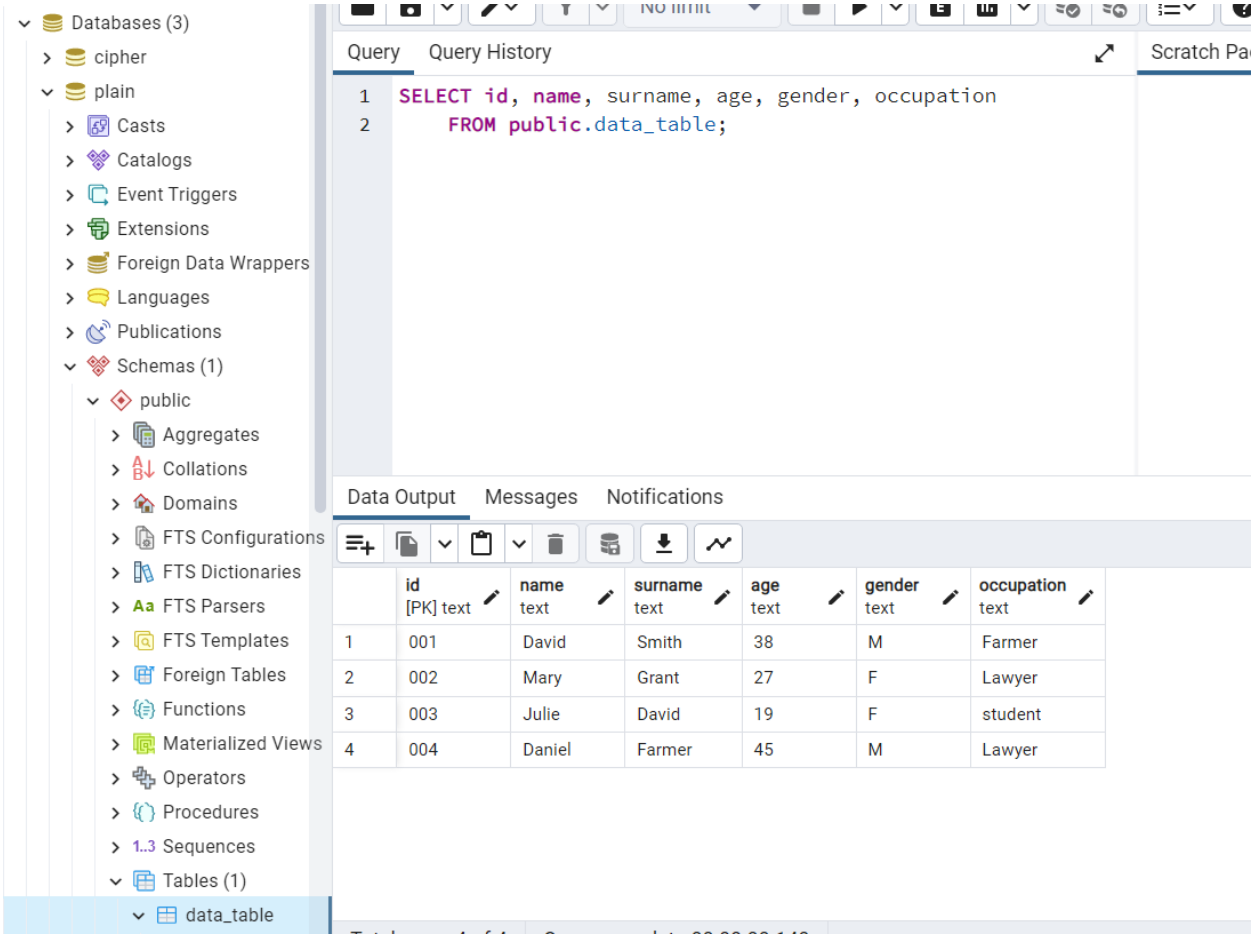


مثال دوم

```
if you want to encrypt database input 1 and if you want to decrypt database input 2
1
which attribute do you want to encrypt ? (Name, Surname, Age, Gender, Occupation)
(if you want to encrypt more than one attribute, you should input like this "Name, Surname")
Name, Occupation
```




```
if you want to encrypt database input 1 and if you want to decrypt database input 2
1
which attribute do you want to encrypt ? (Name, Surname, Age, Gender, Occupation)
(if you want to encrypt more than one attribute, you should input like this "Name, Surname")
Name, Occupation
if you want to encrypt database input 1 and if you want to decrypt database input 2
2
which attribute do you want to decrypt ? (Name, Surname, Age, Gender, Occupation)
(if you want to decrypt more than one attribute, you should input like this "Name, Surname")
Name, Occupation
```


Databases (3)
cipher
plain
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas (1)
public
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (1)
data_table
Query
Query History
Scratch Pa
1 SELECT id, name, surname, age, gender, occupation
2 FROM public.data_table;
Data Output
Messages
Notifications
id [PK] text
name text
surname text
age text
gender text
occupation text
1 001 David Smith 38 M Farmer
2 002 Mary Grant 27 F Lawyer
3 003 Julie David 19 F student
4 004 Daniel Farmer 45 M Lawyer