Protecting data vs systems

Practicality, performance, and problems solved

Happy Star Wars day

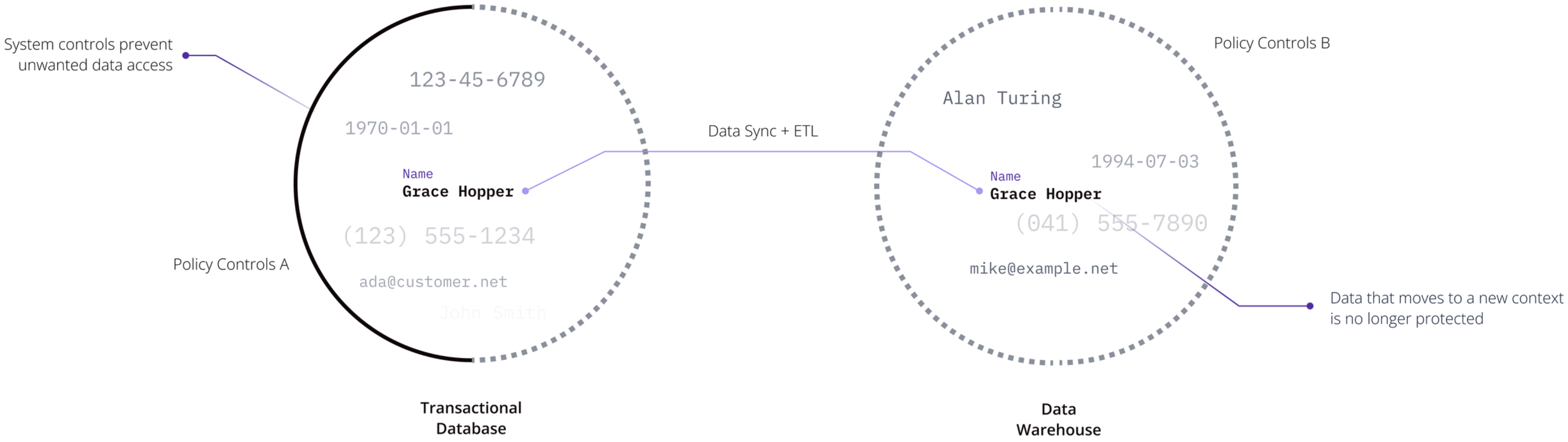
May the fourth be with you

About me

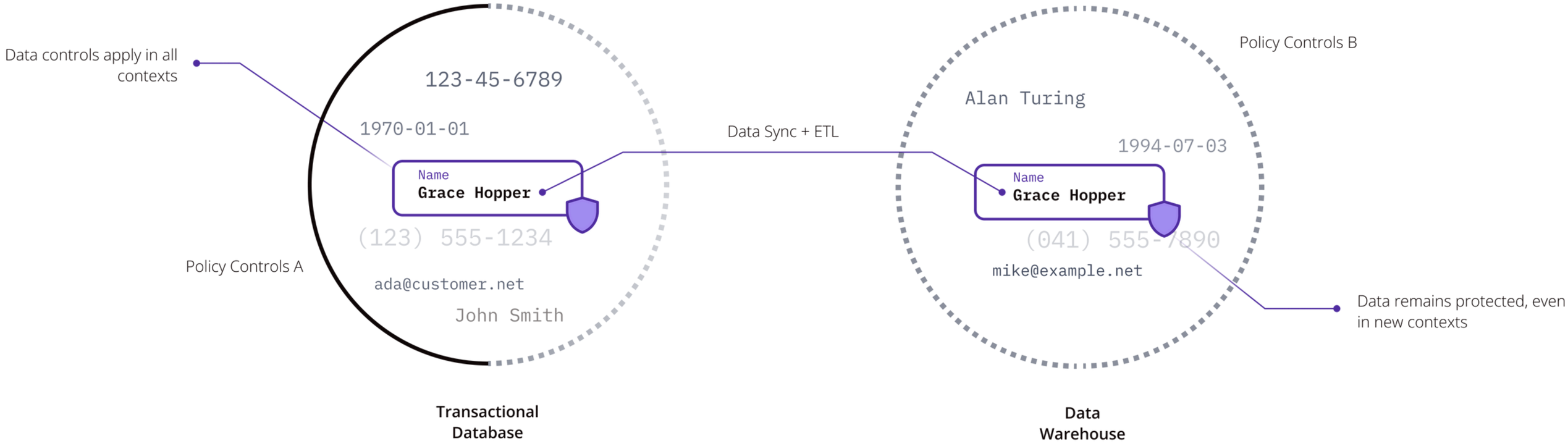
Founder & CEO @ CipherStash

Software and cryptography engineer
Take my coffee with milk but no sugar
Australian but hate the beach

Protecting systems



VS protecting data



GOAL:

Granular, deny-by-default access that works everywhere

Traditional approaches to protecting data directly

File on a filesystem

Use file permissions

```
chown dan file.txt
chgrp admins file.txt
chmod 640 file.txt
```

Row in a database

Use row and column-level controls

```
ALTER TABLE passwd ENABLE ROW LEVEL SECURITY;

CREATE POLICY user_mod ON passwd FOR UPDATE
  USING (current_user = user_name)
  WITH CHECK (
    current_user = user_name AND
    shell IN ('/bin/bash','/bin/zsh')
  );
```

Traditional approaches to protecting data directly

IAM

Police the ARNs

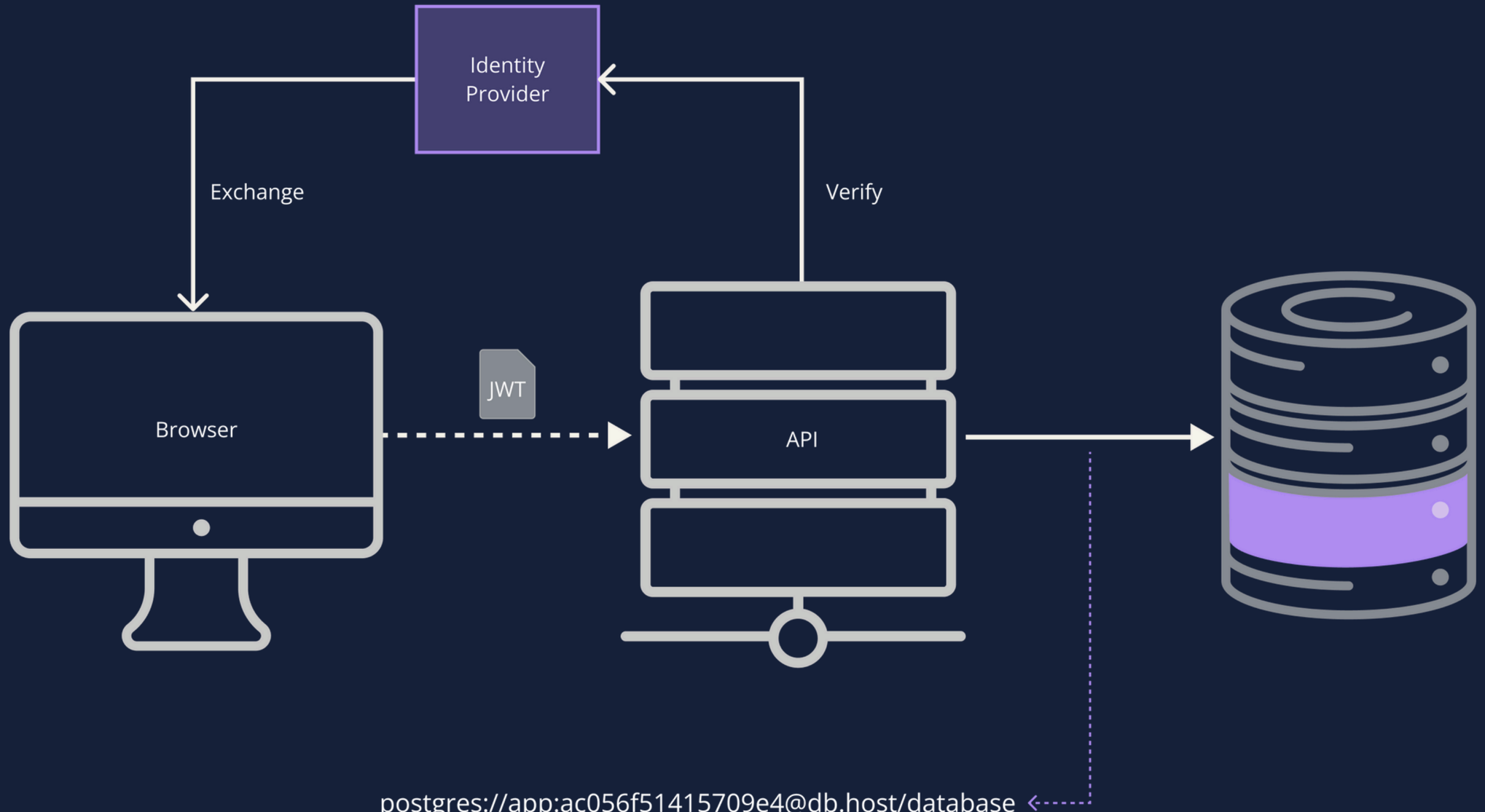
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "iam:AttachUserPolicy",
            "Resource": "arn:aws:iam::*:user/*"
        }
    ]
}
```

Challenge

Systems must be identity aware

ID	user_name	amount
1	dan	100
2	dan	150
3	jill	230

```
CREATE POLICY owned_transactions
ON transactions
USING (current_user = user_name);
```


Identity Provider
Exchange
Verify
Browser
JWT
API
postgres://app:ac056f51415709e4@db.host/database

Challenge

Controls are not universal

id	email	cust_id
7	dan@cipherstash.com	1
23	person@example.net	2

```
CREATE POLICY tenant_policy ON users
FOR SELECT TO cust_1
USING (cust_id = 1);
```

```
CREATE ROW ACCESS POLICY
tenant_policy AS (cust_id NUMBER)
RETURNS boolean ->
      'cust_id' = get_cust_id(current_role())
;
```

Postgres

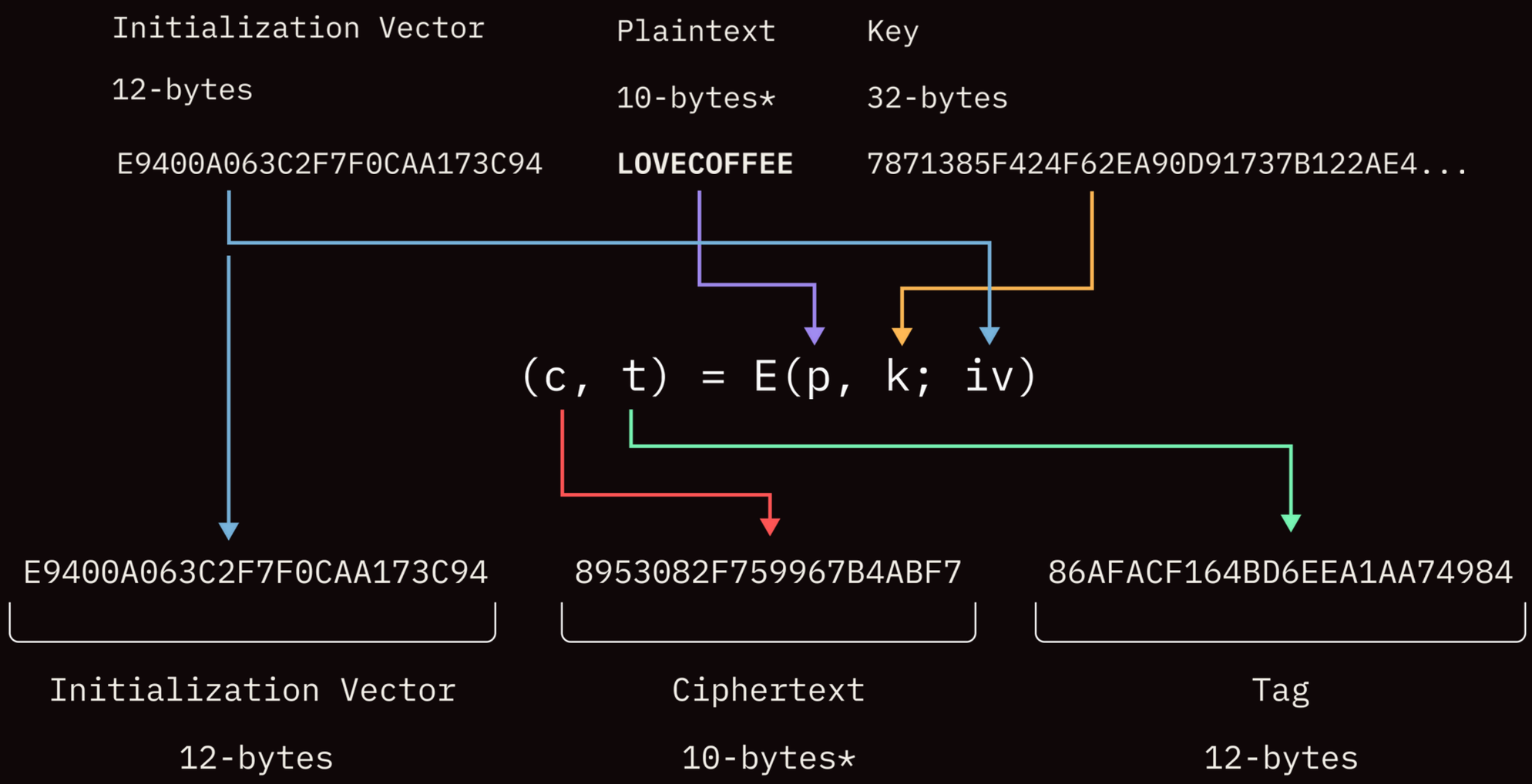
Snowflake

Can encryption be used to protect data directly?

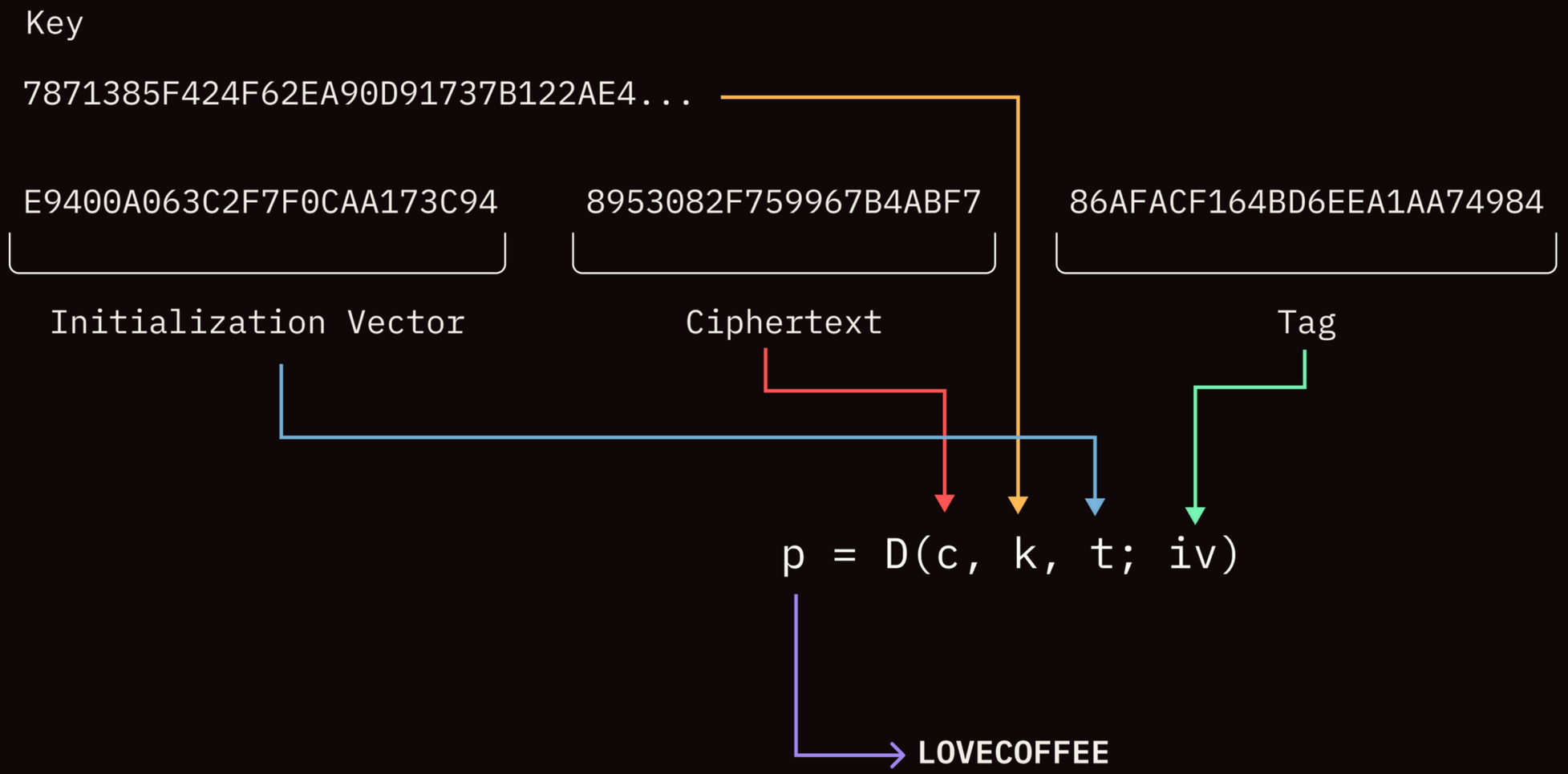
```
psql> SELECT name, email FROM users;
```

name	email
4fea1a8d30305ed3...	1394b5bca5a3b0d8...
8827c6e15fc43900...	3e395338bd8c856e...
fcfae2d53b840df9...	41f69f8479d57c5d...

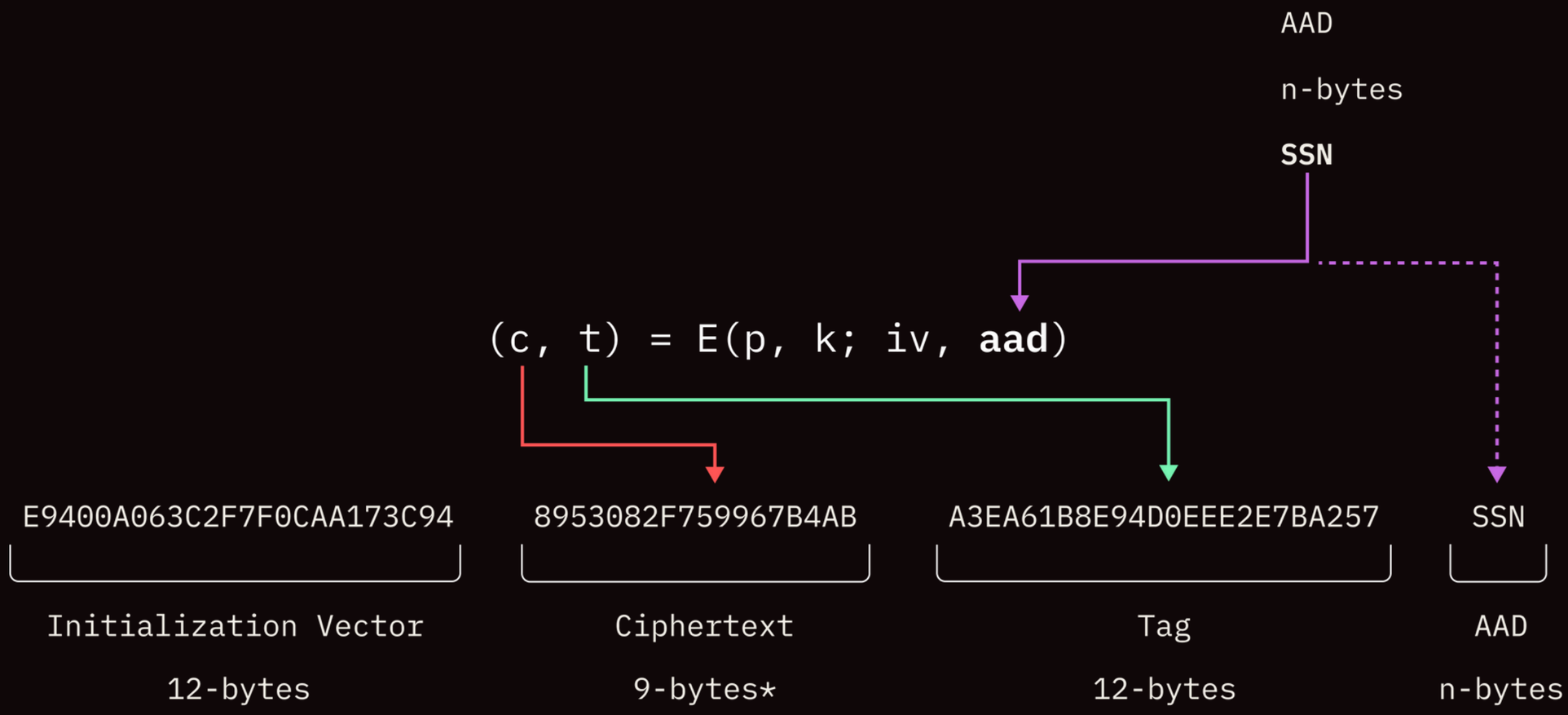
AES-GCM encryption



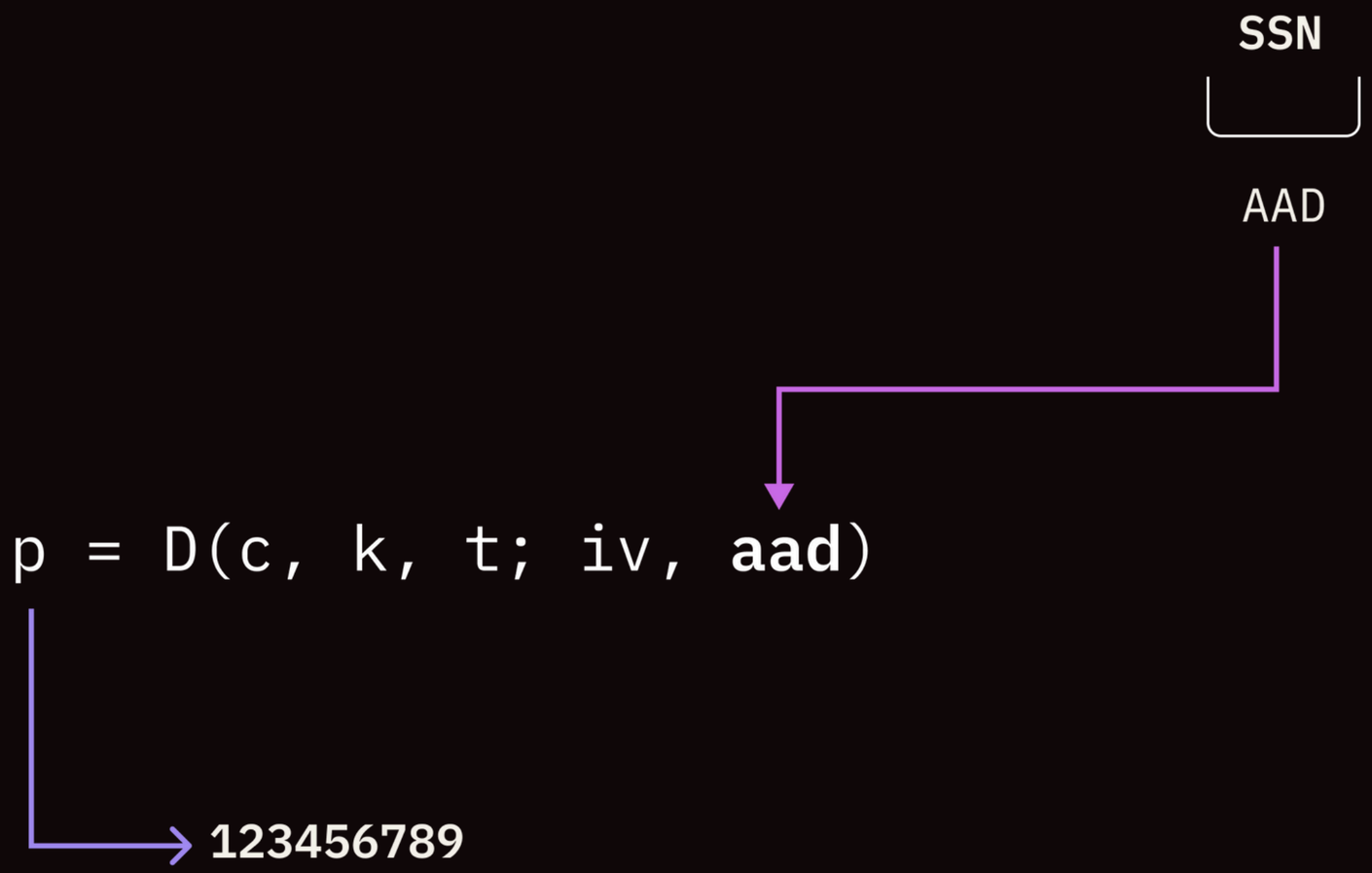
**Ciphertext length = plaintext length for GCM*


Key
7871385F424F62EA90D91737B122AE4...
E9400A063C2F7F0CAA173C94
8953082F759967B4ABF7
86AFACF164BD6EEA1AA74984
Initialization Vector
Ciphertext
Tag
p = D(c, k, t; iv)
LOVECOFFEE

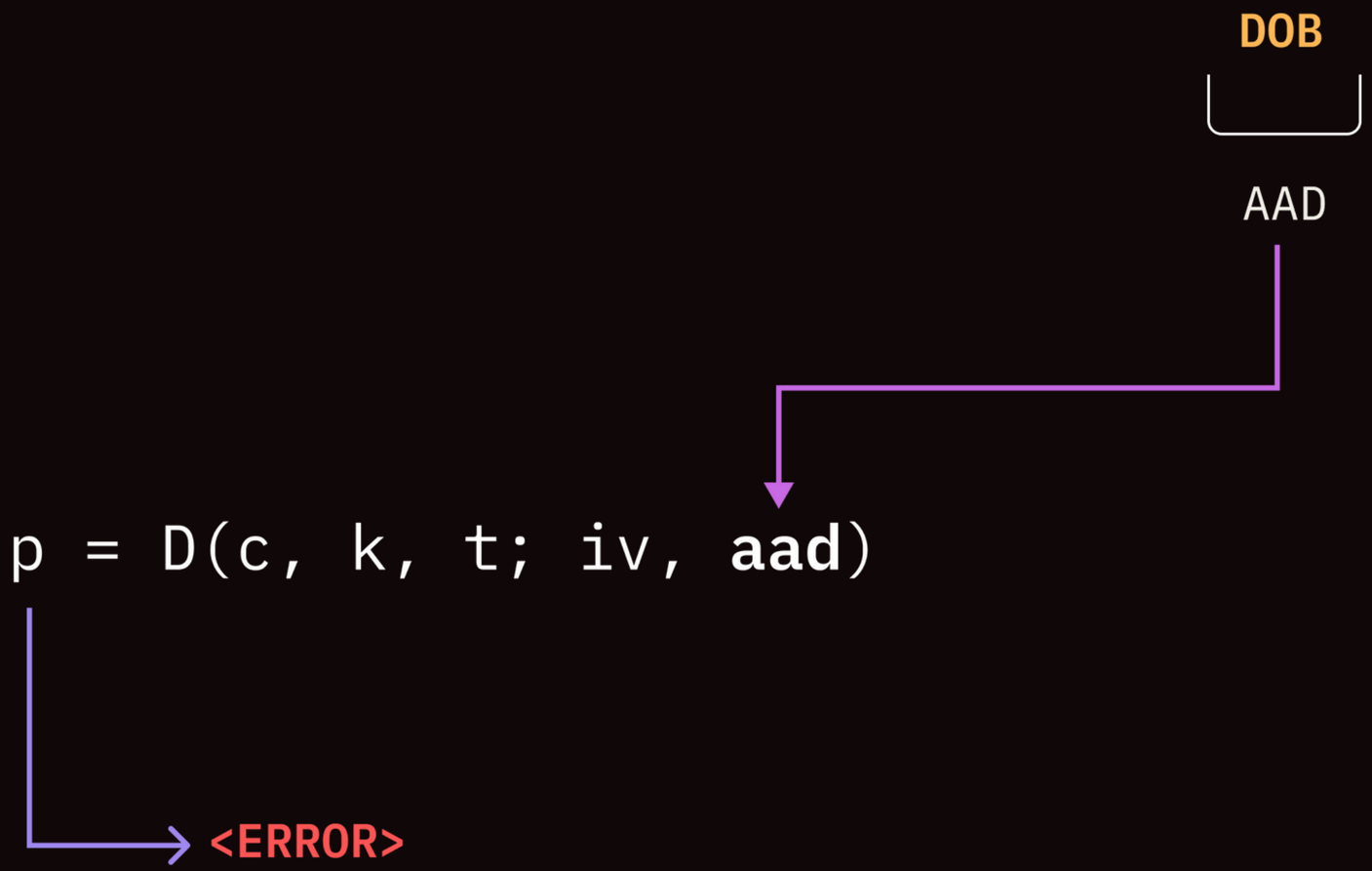
Authenticated Associated Data (AAD)



Decryption with AAD



Correct AAD must be provided



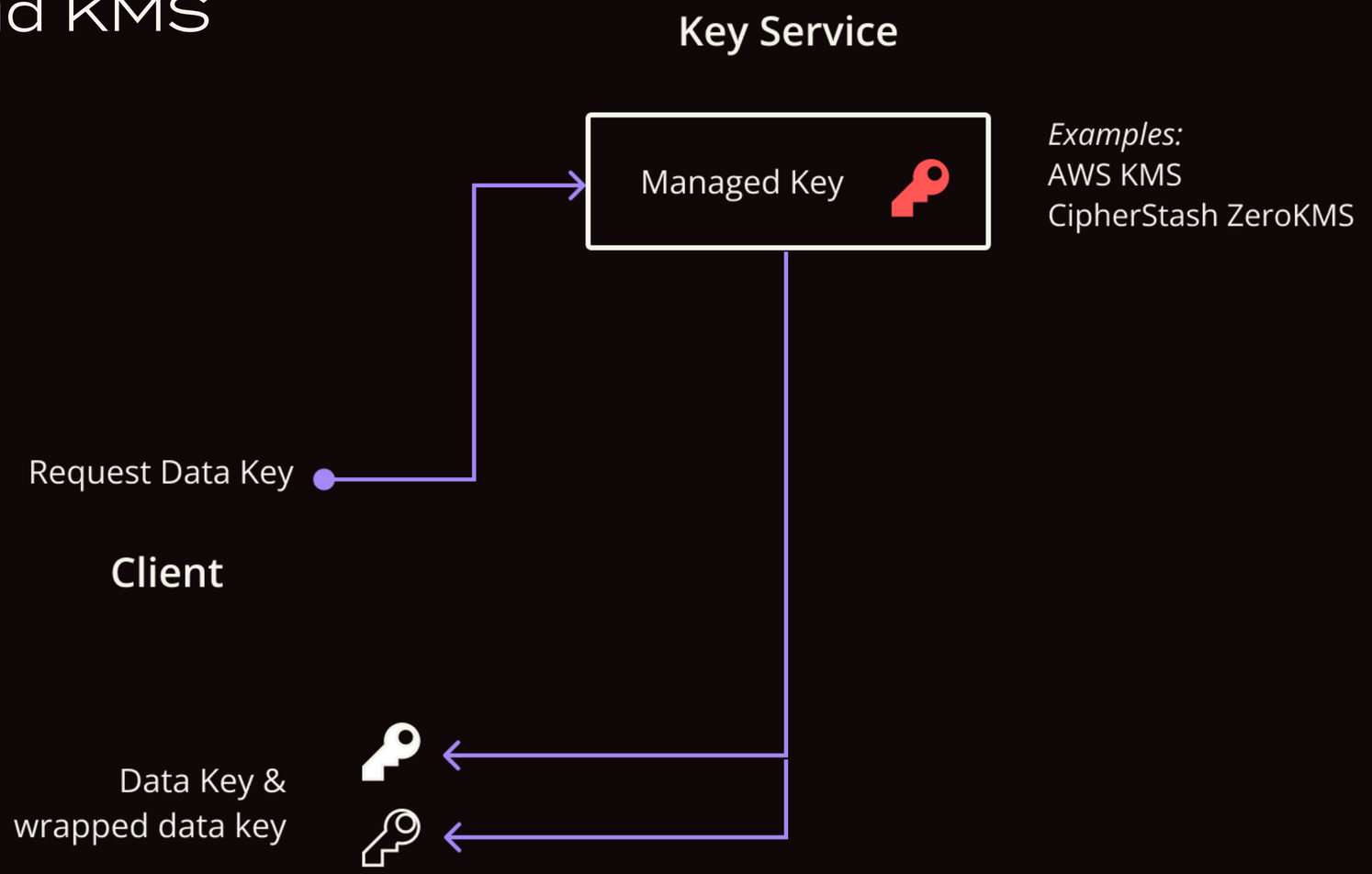
We now have a key management problem!

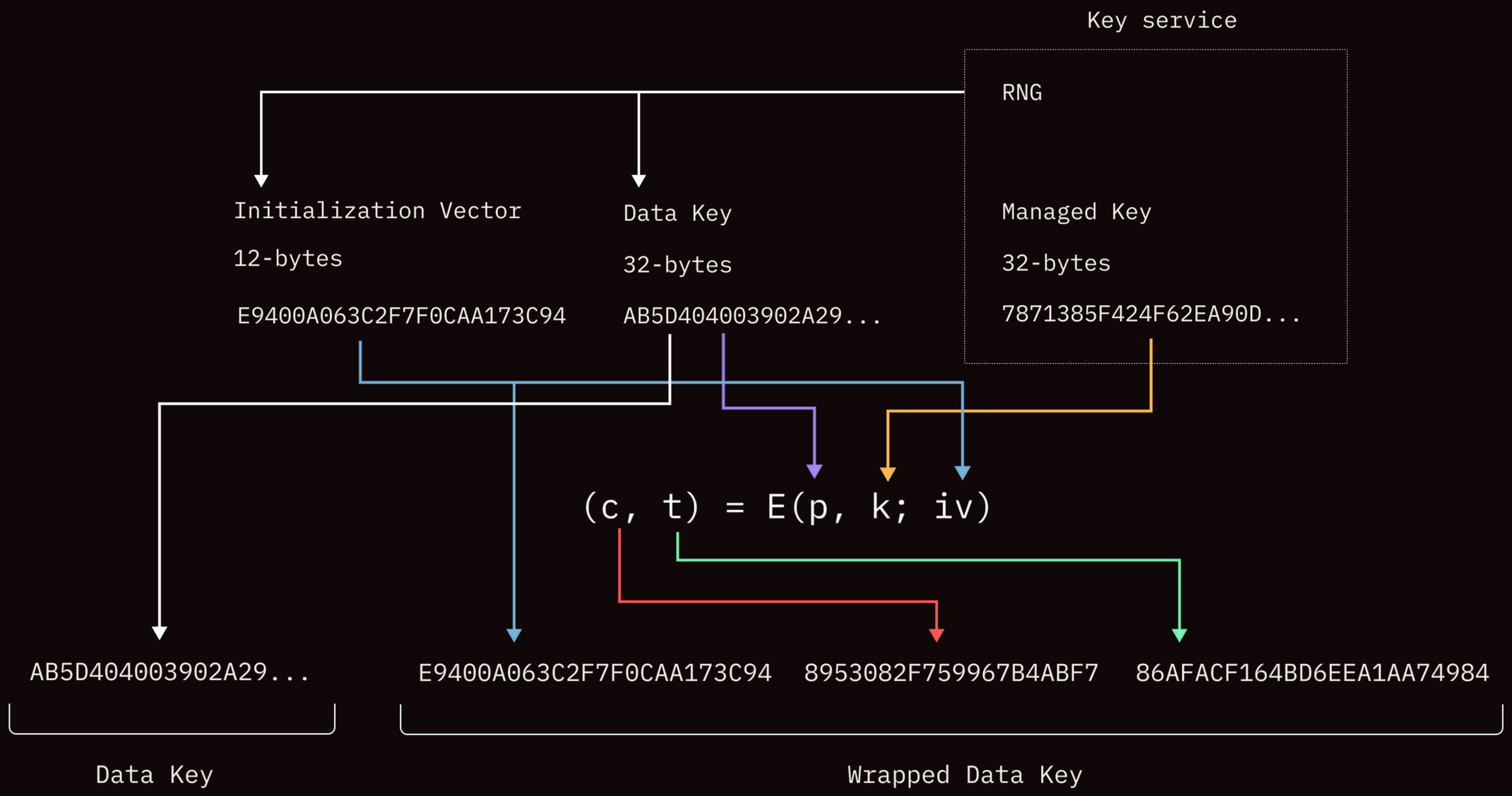
```
psql> SELECT decrypt(name, k), decrypt(email, k) FROM users;
```

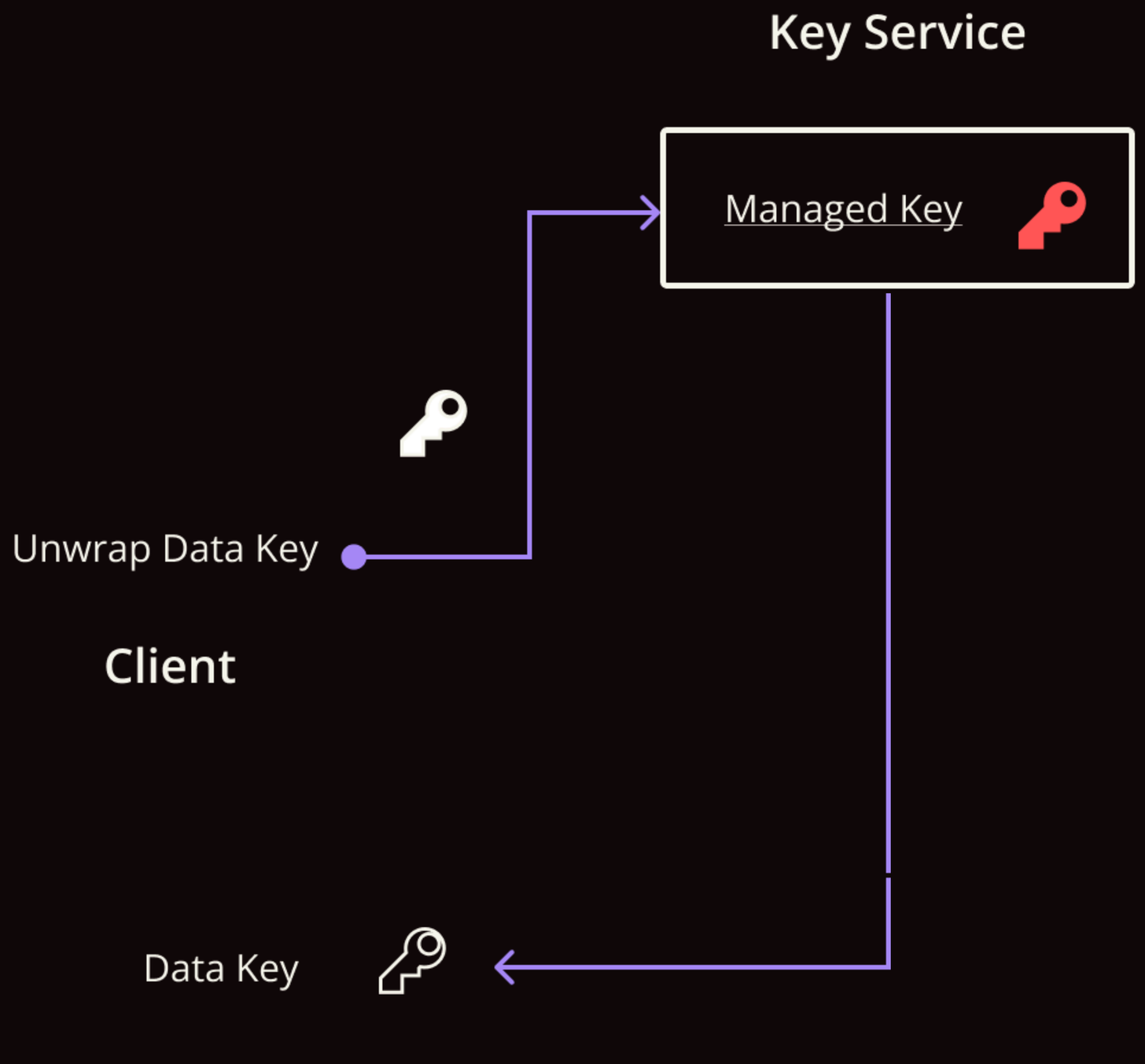
name	email
4fea1a8d30305ed3...	1394b5bca5a3b0d8...
8827c6e15fc43900...	3e395338bd8c856e...
fcfae2d53b840df9...	41f69f8479d57c5d...

Envelope encryption

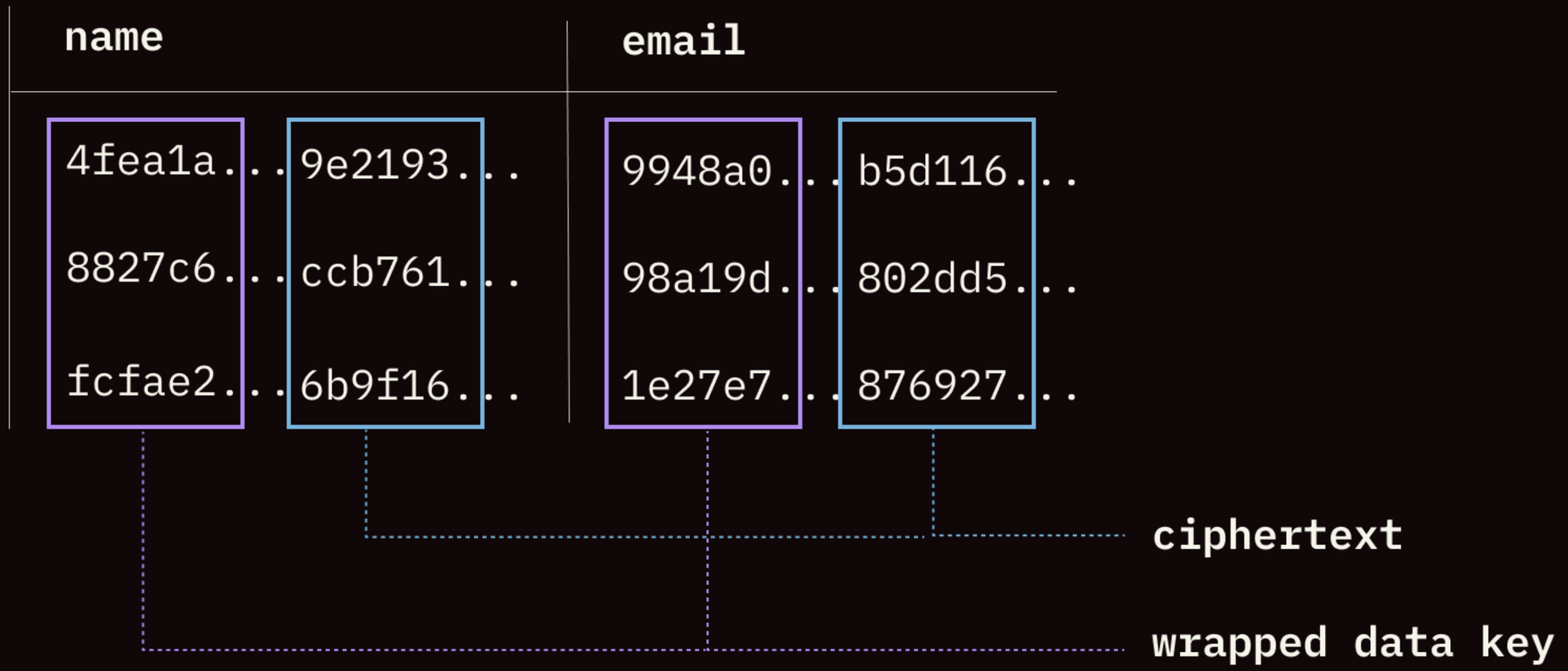
Managing keys with a Cloud KMS



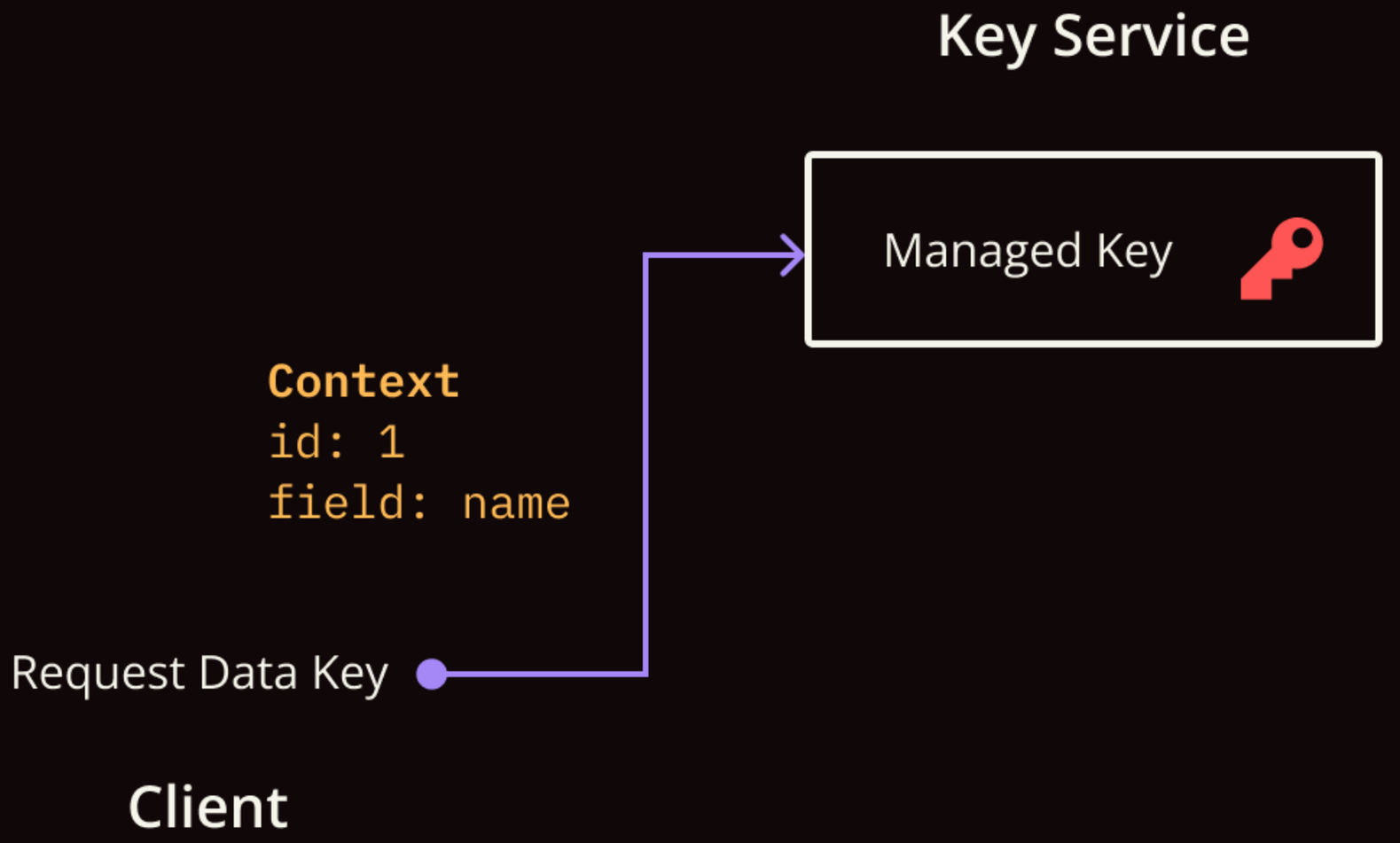

Key service
RNG
Managed Key
32-bytes
7871385F424F62EA90D...
Initialization Vector
12-bytes
E9400A063C2F7F0CAA173C94
Data Key
32-bytes
AB5D404003902A29...
(c, t) = E(p, k; iv)
AB5D404003902A29...
E9400A063C2F7F0CAA173C94
8953082F759967B4ABF7
86AFACF164BD6EEA1AA74984
Data Key
Wrapped Data Key

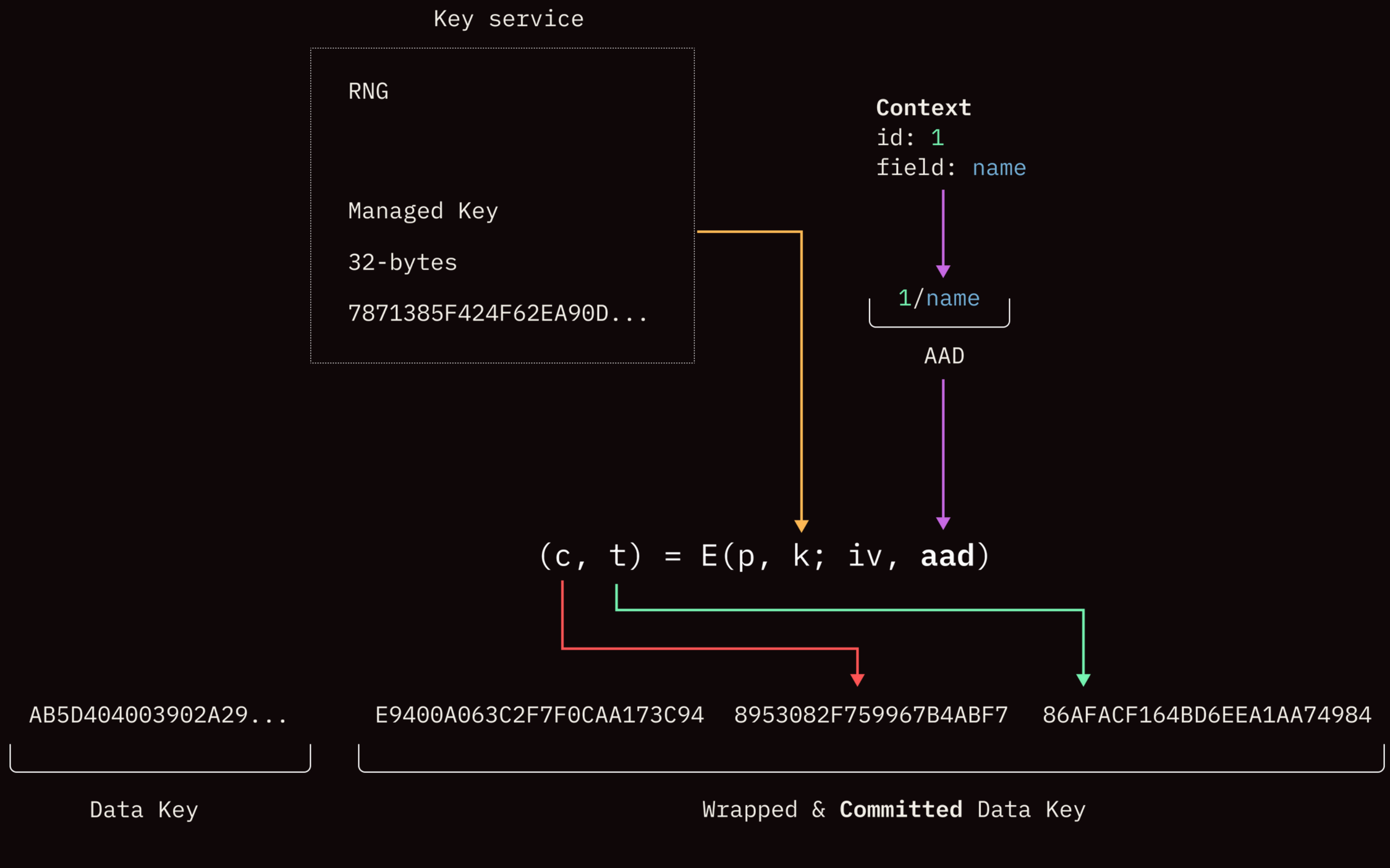

Key Service
Managed Key
Unwrap Data Key
Client
Data Key

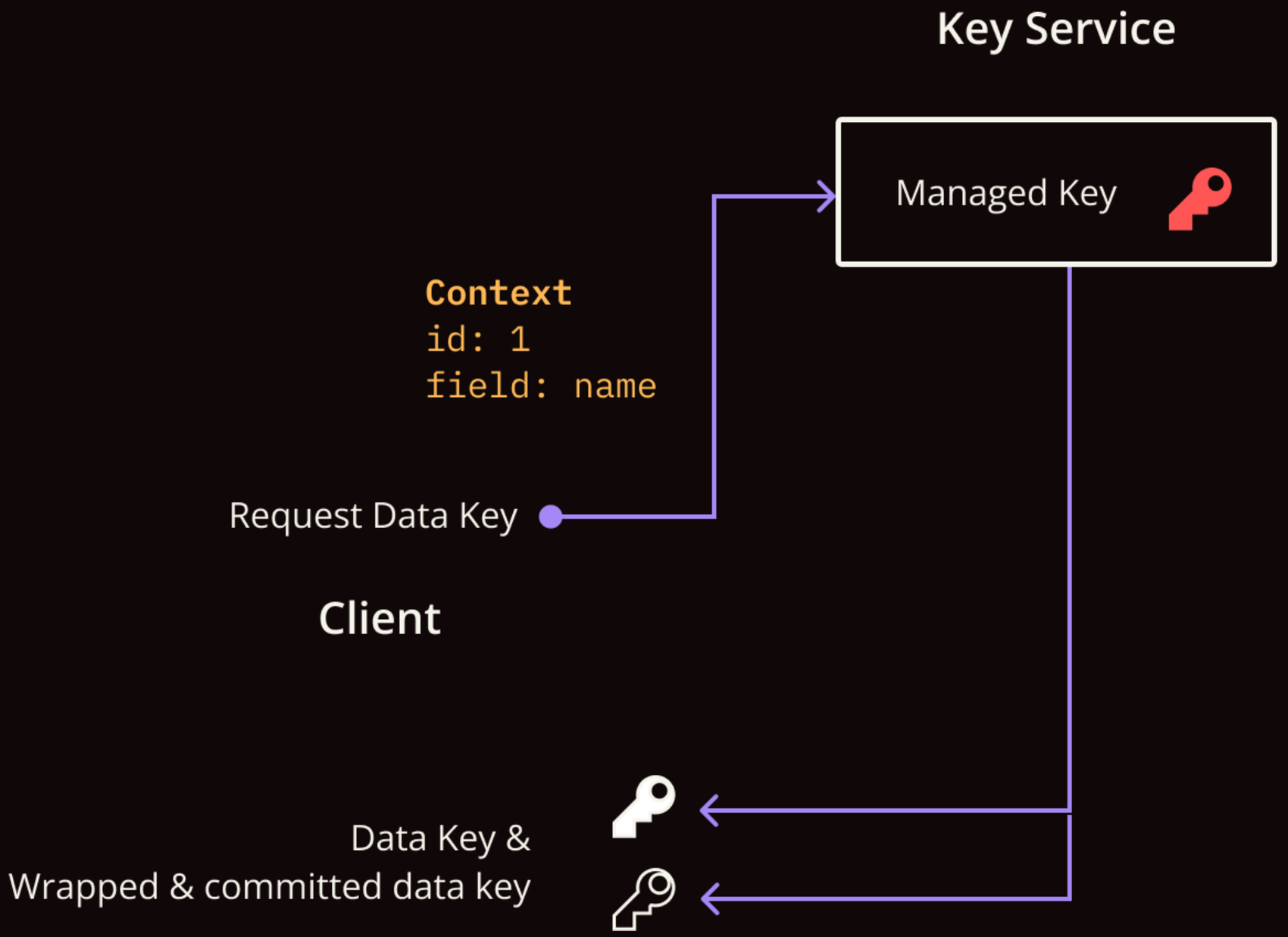
```
psql> SELECT name, email FROM users;
```



Key commitment


Key Service
Managed Key
Context
id: 1
field: name
Request Data Key
Client

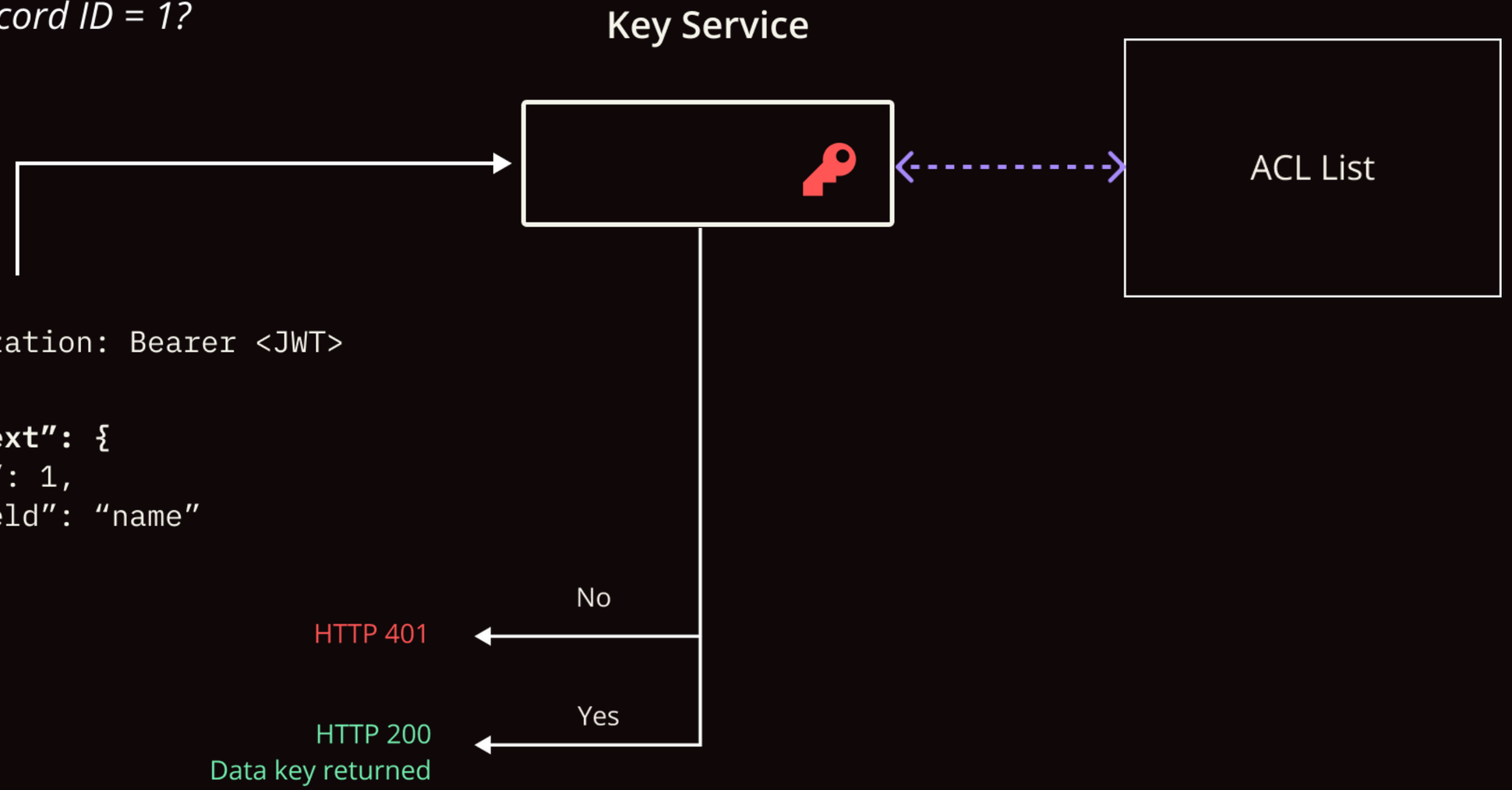

Key service
RNG
Managed Key
32-bytes
7871385F424F62EA90D...
Context
id: 1
field: name
1/name
AAD
(c, t) = E(p, k; iv, aad)
AB5D404003902A29...
E9400A063C2F7F0CAA173C94
8953082F759967B4ABF7
86AFACF164BD6EEA1AA74984
Data Key
Wrapped & Committed Data Key


Key Service
Managed Key
Context
id: 1
field: name
Request Data Key
Client
Data Key &
Wrapped & committed data key

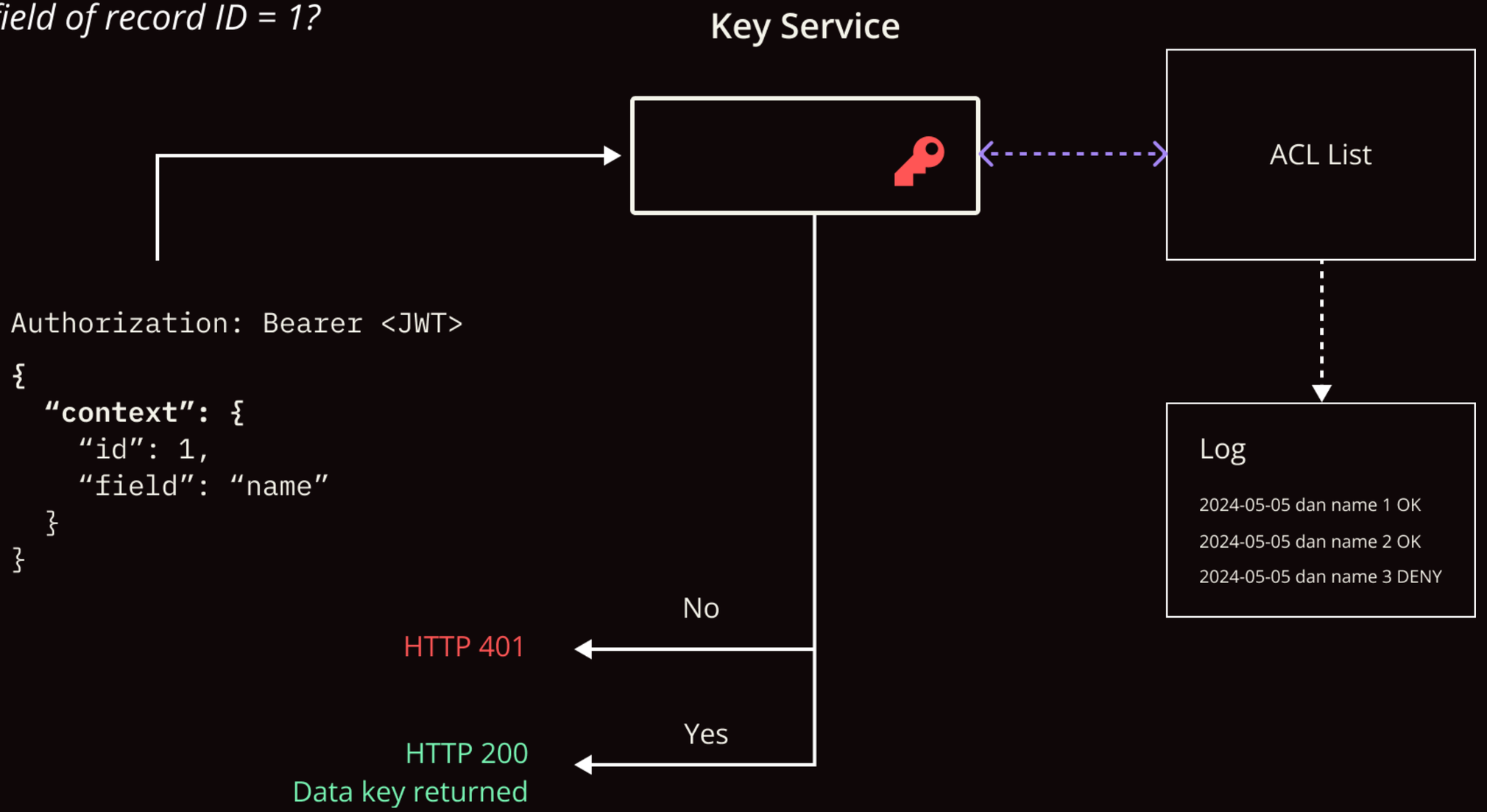
Requested Context	Data key AAD	Result
`1/name`	`1/name`	Data key decrypts Value can be decrypted
`2/name`	`1/name`	Data key decryption fails
`1/email`	`1/name`	Data key decryption fails

Access control using selective data-key decryption

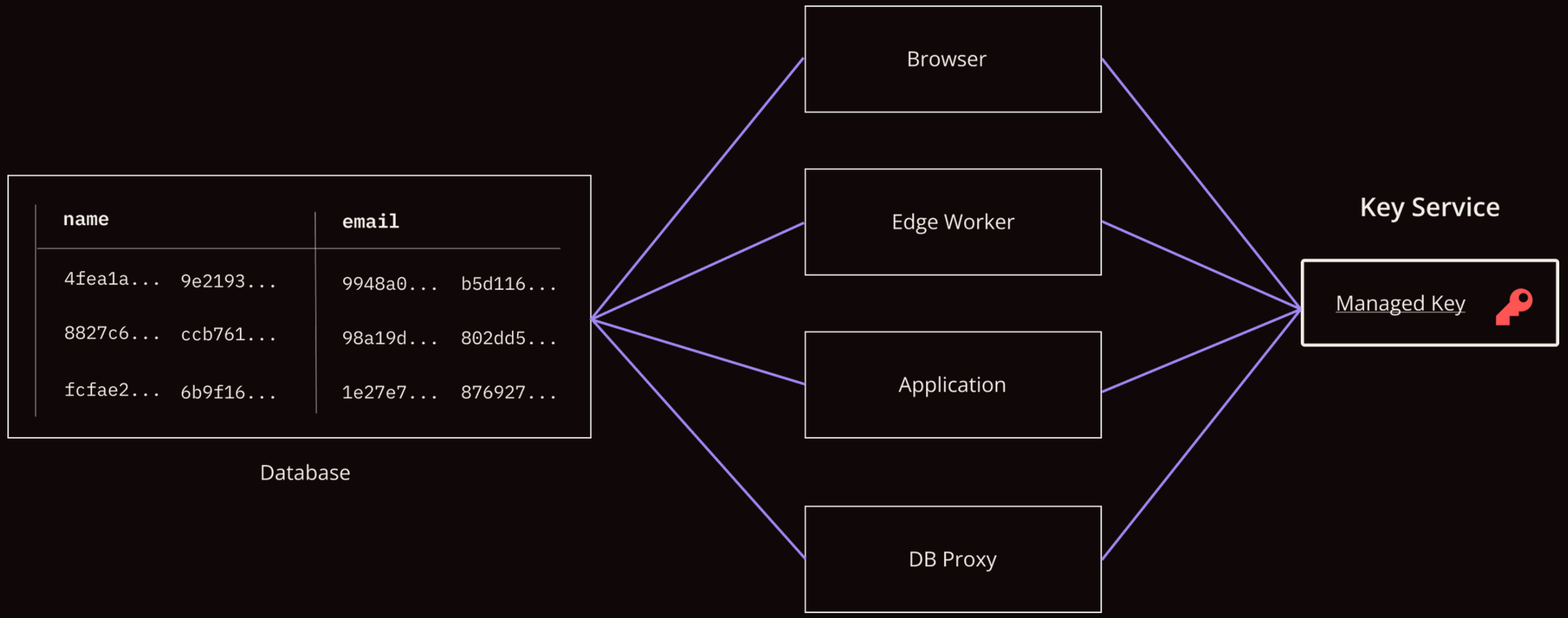
Can I decrypt the “name” field of record ID = 1?



Can I decrypt the “name” field of record ID = 1?



Where do we decrypt?


name
email
4fea1a... 9e2193...
9948a0... b5d116...
8827c6... ccb761...
98a19d... 802dd5...
fcfae2... 6b9f16...
1e27e7... 876927...
Database
Browser
Edge Worker
Application
DB Proxy
Key Service
Managed Key

Universal interface
1 way to check permissions: talk to the key service

Deny by default
Works outside configured systems or if data is leaked

Supports “untrusted” clients
Decryption can take place in the browser
(Compare to doing permission checks in a front-end app)

What about queries?

```
psql> SELECT name FROM users WHERE name = 'Dan Draper';
psql> SELECT name FROM users WHERE name LIKE 'Dan%';
psql> SELECT name FROM users WHERE dob > '2000-01-01';
psql> SELECT name FROM users ORDER BY name;
```

name	email	dob
4fea1a8d30305ed3...	1394b5bca5a3b0d8...	bae79718c82c07...
8827c6e15fc43900...	3e395338bd8c856e...	5a5a0ea59d2751...
fcfae2d53b840df9...	41f69f8479d57c5d...	d9e44836144fc2...

Homomorphic Encryption?

Example: THFE-rs

Benchmarks

```
Results from running on an AMD Ryzen 9 3900x:

tfhe/encrypt            time:   [1.9684 ms 1.9721 ms 1.9755 ms]
tfhe/a == b             time:   [110.00 ms 111.21 ms 112.40 ms]
tfhe/a > b              time:   [188.50 ms 191.92 ms 195.34 ms]
tfhe/a < b              time:   [186.07 ms 189.58 ms 193.16 ms]
tfhe/a >= a             time:   [43.571 ms 43.980 ms 44.389 ms]
tfhe/a <= a             time:   [44.045 ms 44.498 ms 44.945 ms]
```

Example: THFE-rs

User table contains 100,000 records

Query time

```
psql> SELECT name FROM users WHERE dob > ‘2000-01-01’;
```

Using B-Tree: ~3 seconds

Sequential scan: 5.5 hours!

Order-Revealing Encryption

Example: ore.rs

```
Results from running on an AMD Ryzen 9 3900x:

tfhe/encrypt            time:   [1.9684 ms 1.9721 ms 1.9755 ms]
tfhe/a == b             time:   [110.00 ms 111.21 ms 112.40 ms]
tfhe/a > b              time:   [188.50 ms 191.92 ms 195.34 ms]
tfhe/a < b              time:   [186.07 ms 189.58 ms 193.16 ms]
tfhe/a >= a             time:   [43.571 ms 43.980 ms 44.389 ms]
tfhe/a <= a             time:   [44.045 ms 44.498 ms 44.945 ms]

ore/encrypt             time:   [48.028 µs 48.154 µs 48.280 µs]
ore/a == b              time:   [238.17 ns 238.41 ns 238.68 ns]
ore/a > b               time:   [237.78 ns 238.03 ns 238.31 ns]
ore/a < b               time:   [239.17 ns 240.28 ns 241.57 ns]
ore/a >= a              time:   [221.17 ns 221.31 ns 221.46 ns]
ore/a <= a              time:   [225.81 ns 226.83 ns 227.72 ns]
```

> 800,000x faster!

Example: ore.rs

User table contains 100,000 records

Query time

```
psql> SELECT name FROM users WHERE dob > '2000-01-01';
```

Using B-Tree: ~0.004 ms

Sequential scan: ~24 ms

Searchable encryption

Order Revealing Encryption (ORE)

Symmetric Searchable Encryption

Structured Encryption

Encrypted Bloom Filters

Resources

https://github.com/cipherstash/bsidessf

dan@cipherstash.com