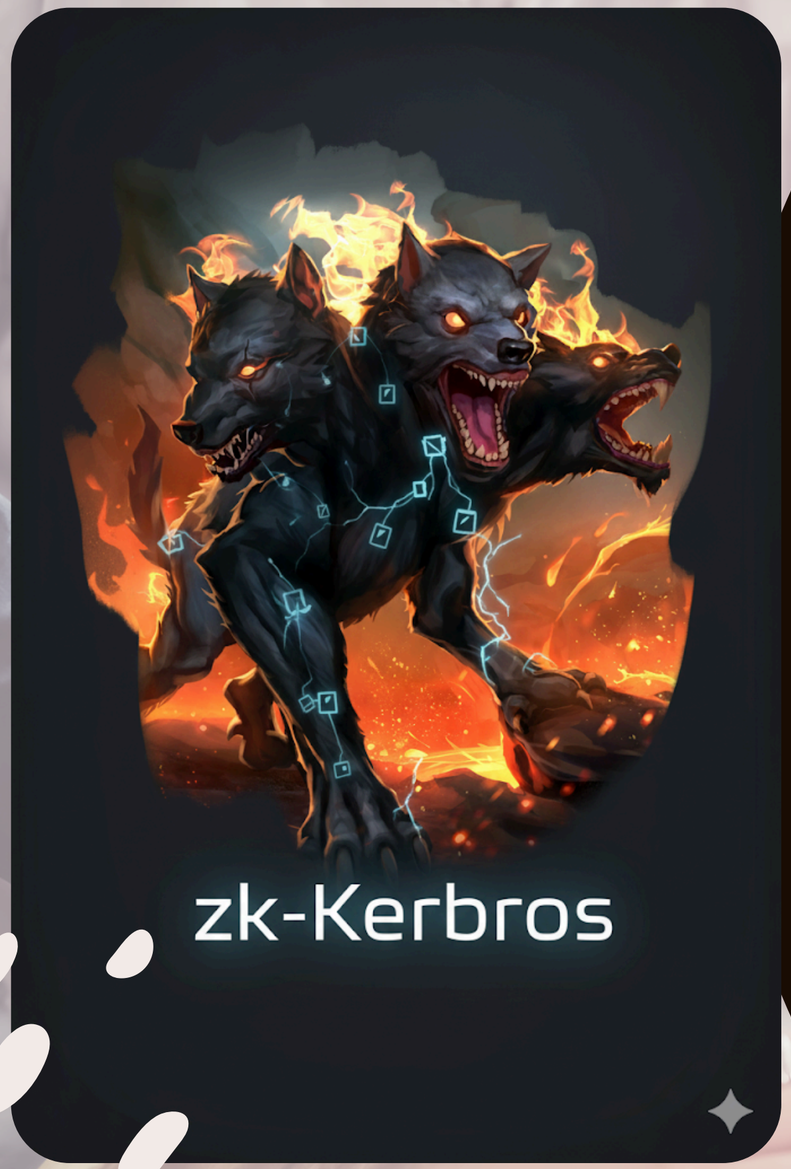
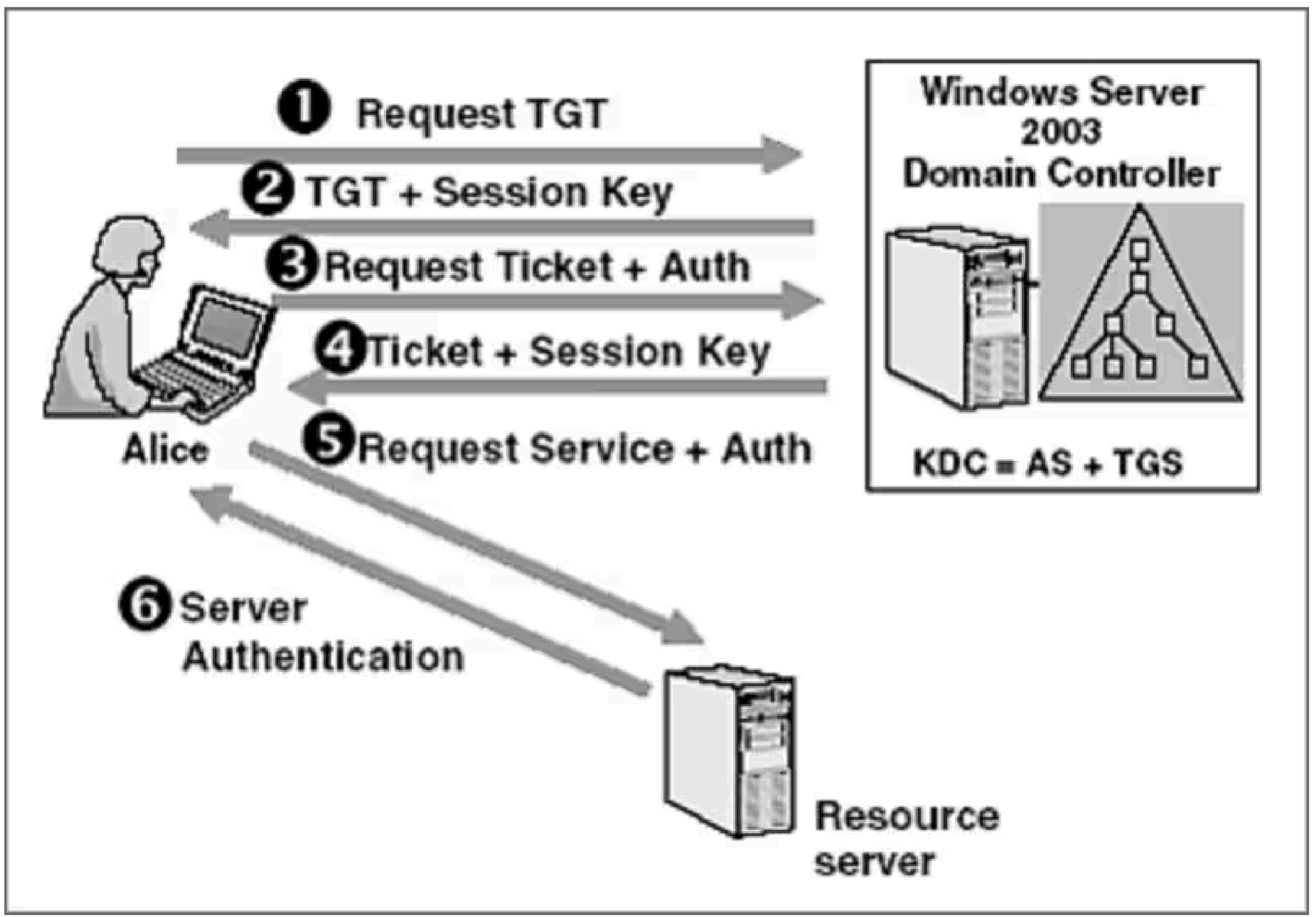


ZK kerberos

“ I possess a valid (UserID, ServiceID) pair that corresponds to one of the hashes in the public database, but I will not reveal which one.”

Using MoPro for mobile proving,
Filecoin for database

What is kerberos ?



Current Uses of kerberos

Colleges and Universities

1. Campus-wide Single Sign-On (SSO)
2. Eduroam / Campus Wi-Fi Network Access
3. Email Systems:
4. Learning Management Systems (LMS)
5. Student Information Systems
6. Library Services:
7. Departmental Servers & Lab Access:
8. Virtual Desktop Infrastructure (VDI):
9. VPN Access
10. Administrative Systems:

Kerberos is the default and primary authentication protocol for virtually all services and user logins within an Active Directory domain, including:

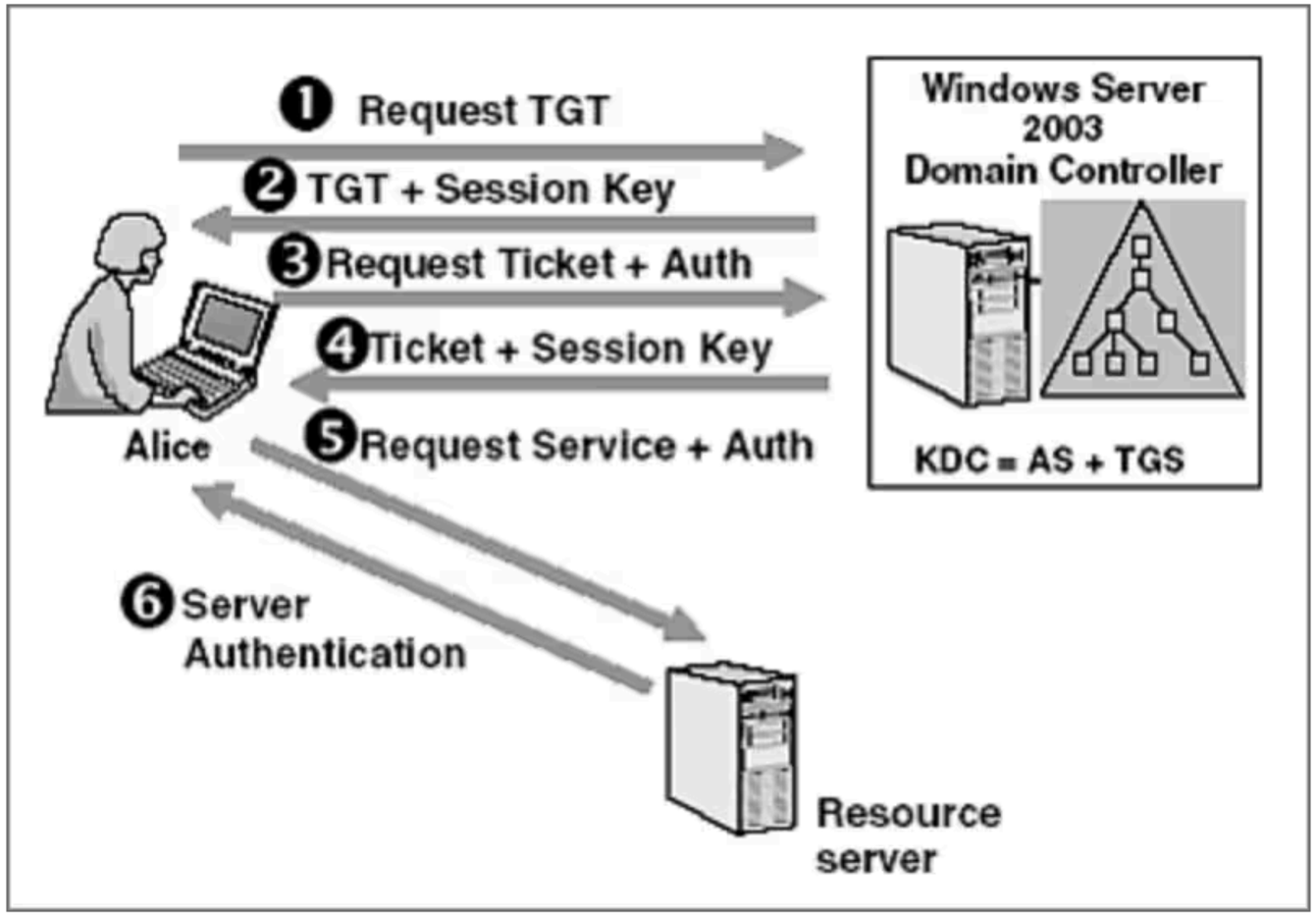
- Logging into Windows PCs and Servers.
- Accessing file shares (SMB/CIFS) on Windows servers.
- Authenticating to Exchange Server (email), SharePoint, SQL Server, and other Microsoft applications.

1. Linux/Unix Environments (e.g., FreeIPA, MIT Kerberos):
   - SSO
   - NFS
   - SSH
   - LDAP
2. Large-Scale Distributed Systems

Government & Defense:

Issues with old kerberos

- lot of communicatio steps, high latency
- Issue of privacy, kerberos gains knowledge of UserID and session he wants to avail.
- Centralised Data server

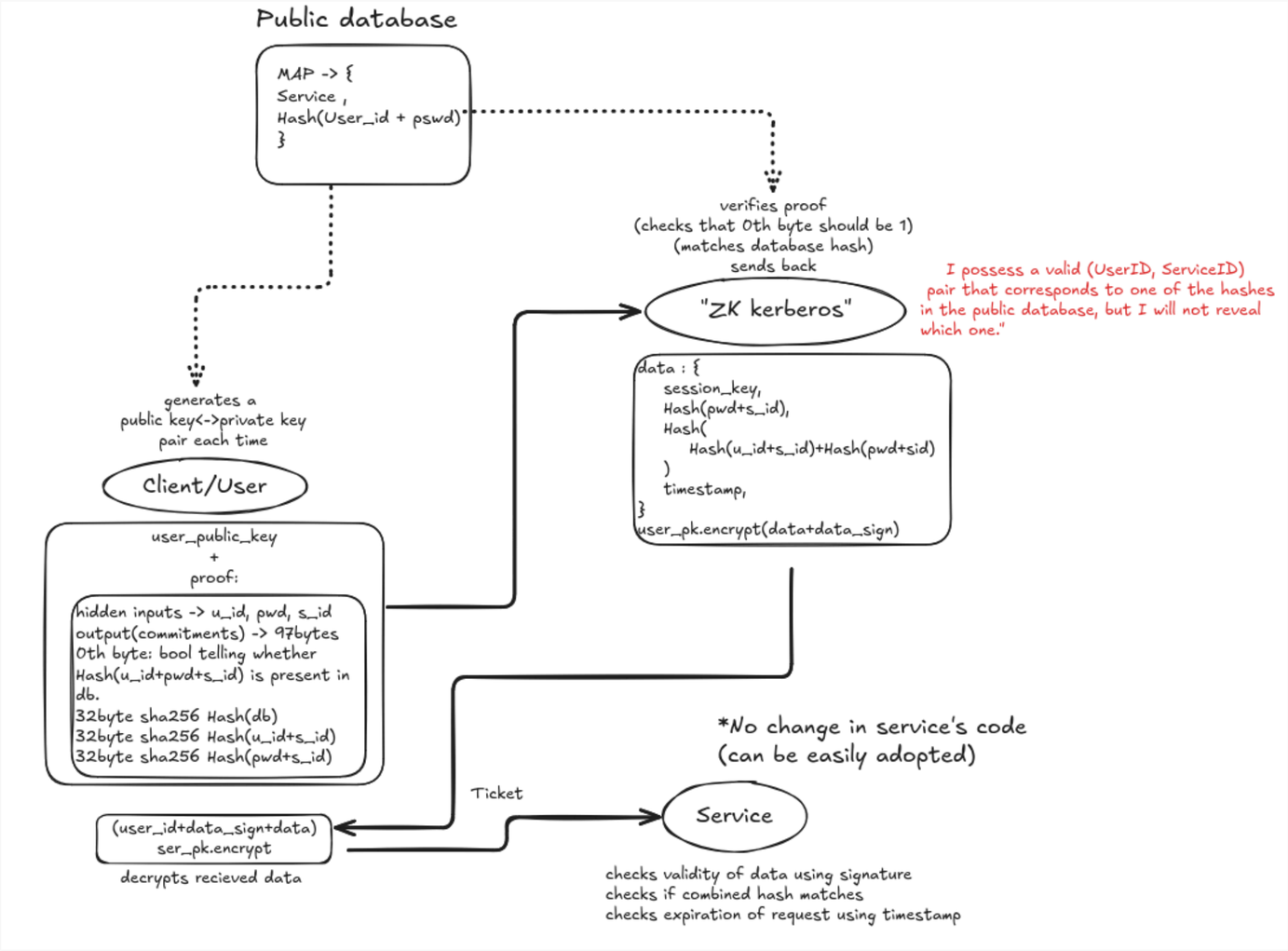


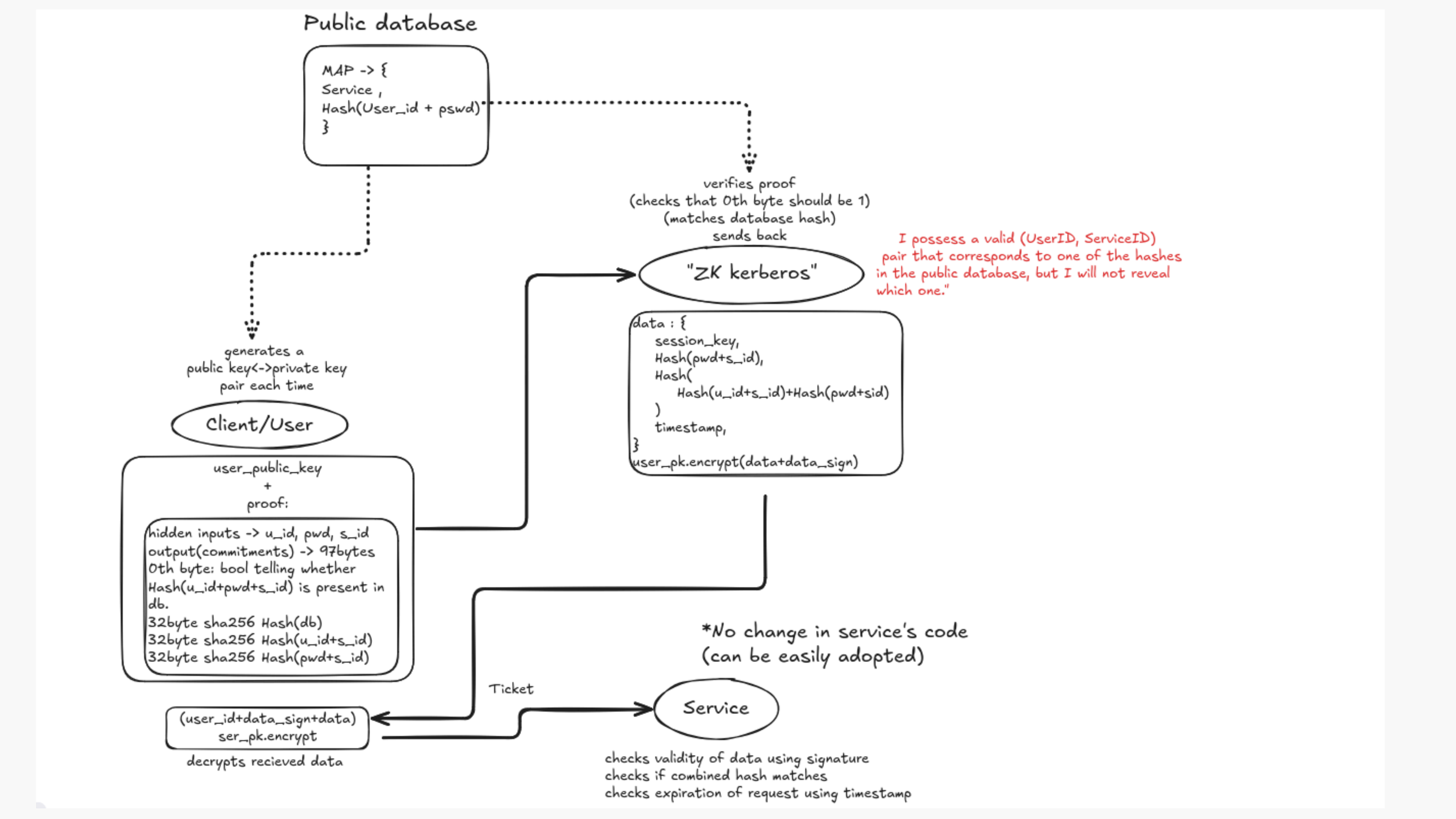
New zk-kerberos

- No issue of privacy,

“ I possess a valid (UserID, ServiceID) pair that corresponds to one of the hashes in the public database, but I will not reveal which one.”

- Much much faster, as verification of can be done in O(log n)
- Publically non backtracable database.
- Precomputed proofs, and finger print for better user experience.




Public database
MAP -> {
Service ,
Hash(User_id + pswd)
}
verifies proof
(checks that 0th byte should be 1)
(matches database hash)
sends back
"ZK kerberos"
I possess a valid (UserID, ServiceID) pair that corresponds to one of the hashes in the public database, but I will not reveal which one."
data : {
session_key,
Hash(pwd+s_id),
Hash(
Hash(u_id+s_id)+Hash(pwd+sid)
)
timestamp,
}
user_pk.encrypt(data+data_sign)
generates a
public key<->private key
pair each time
Client/User
user_public_key
+
proof:
hidden inputs -> u_id, pwd, s_id
output(commitments) -> 97bytes
0th byte: bool telling whether
Hash(u_id+pwd+s_id) is present in
db.
32byte sha256 Hash(db)
32byte sha256 Hash(u_id+s_id)
32byte sha256 Hash(pwd+s_id)
*No change in service's code
(can be easily adopted)
Ticket
(user_id+data_sign+data)
ser_pk.encrypt
decrypts recieved data
Service
checks validity of data using signature
checks if combined hash matches
checks expiration of request using timestamp