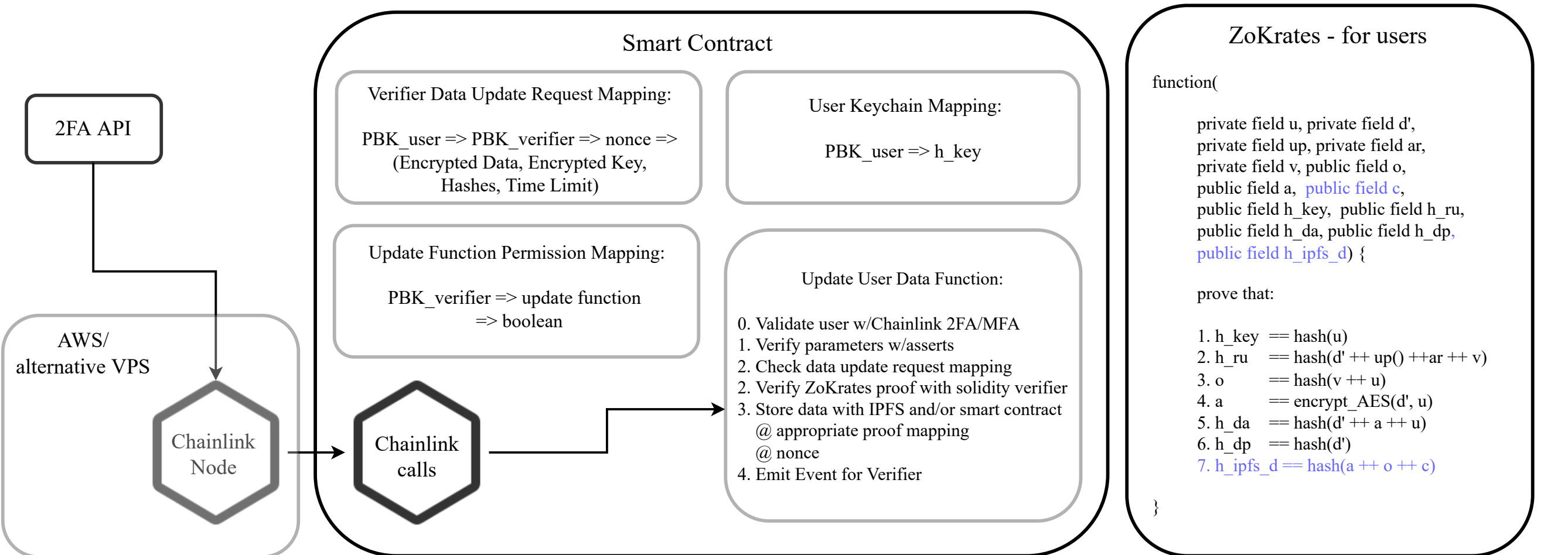


# Secure On-boarding of Data To Smart Contract While Maintaining Privacy in Solidity



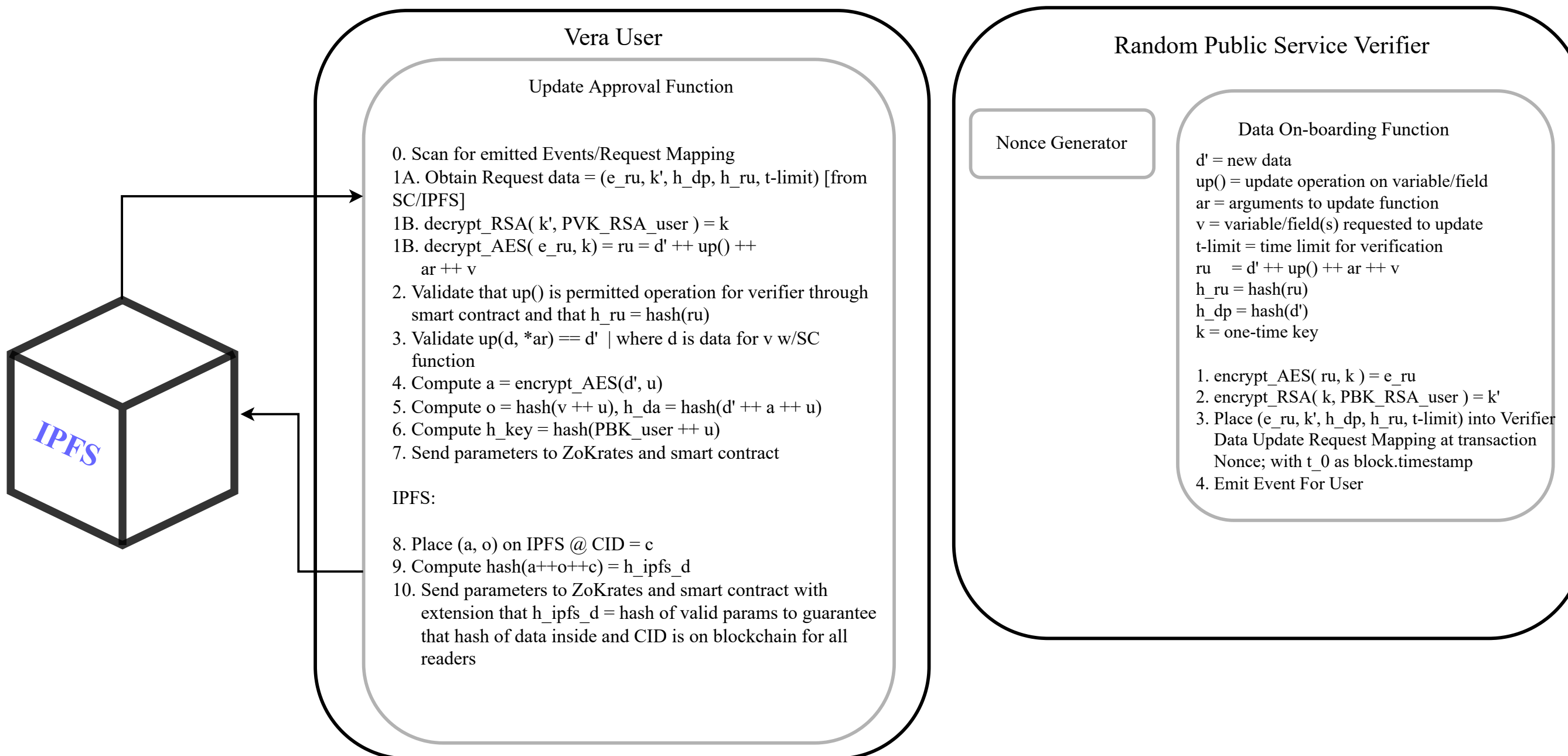
In the smart contract:

- a. Assert that on-chain hash == h\_key computed off-chain provided to smart contract by user when onboarding in User Keychain Mapping for PBK\_user
- b. Assert that h\_ru == hash submitted in request by public service verifier
- c. Assert that block.timestamp <= t\_0 + t\_limit [verify times are feasible when requests are onboarded by verifiers through smart contract] [store t\_0 as block.timestamp when request is placed into smart contract]
- d. Assert that on-chain hash == h\_dp computed off-chain provided to smart contract by verifier

In ZoKrates:

1. Verifies that u is valid
2. Verifies that d' and v are valid
3. Verifies o is valid
4. Verifies a is valid
5. Verifies provided hash to contract for (d' ++ a ++ u) is valid (helpful later when proving ownership of data)
6. Three separate ZKP circuits are used for this function to lower computational cost - one for conditions 1/2/3/5, one for 1/4/6, and one for 7. Common parameters are matched between these functions in the smart contract to verify that in combination they are equivalent to the pseudocode outlined. Redundancies are present because only some variables are public, others are hidden.

7. Proves that provided hash h\_ipfs\_d of content at CID c uploaded on IPFS is valid

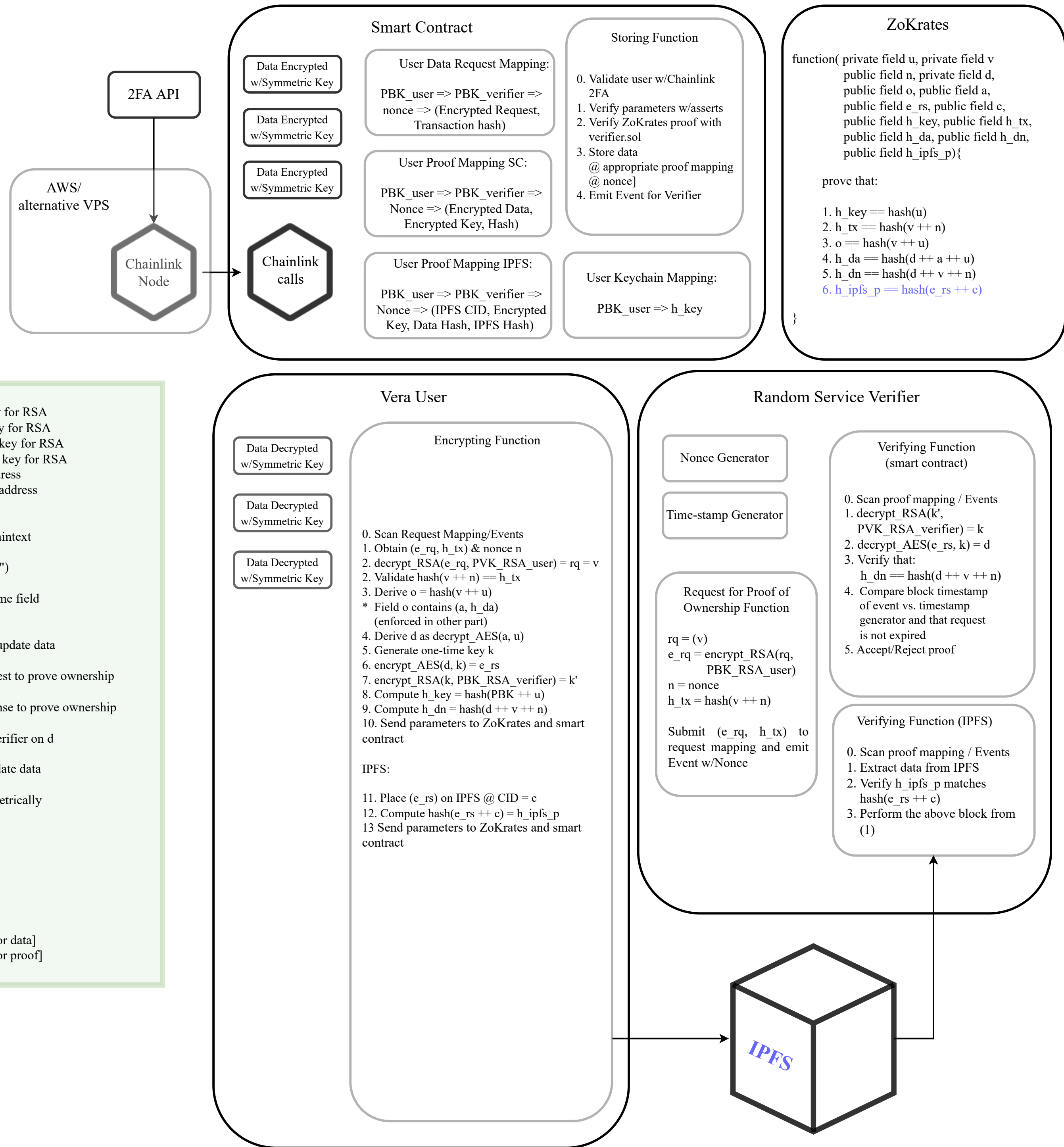


PBK\_RSA\_user := user's public key for RSA  
PVK\_RSA\_user := user's private key for RSA  
PBK\_RSA\_verifier := verifier's public key for RSA  
PVK\_RSA\_verifier := verifier's private key for RSA  
PBK\_user := user's public address  
PBK\_verifier := verifier's public address

u := symmetric key of user  
v := variable field requested ("age"), in plaintext  
n := transaction nonce (10 byte string)  
o := obfuscated field ("age" -> "v2m0t97y")  
d := user data stored at some field  
d' := user data requested to be stored at some field  
a := d' encrypted symmetrically with AES  
c := IPFS content identifier (URI/URL)  
e\_ru := symmetrically encrypted request to update data  
ru := request to update data  
e\_rq := asymmetrically encrypted request to prove ownership  
rq := request to prove ownership  
e\_rs := symmetrically encrypted response to prove ownership  
rs := response to prove ownership  
up() := update function requested by verifier on d  
ar := arguments to update function  
t\_limit := the time limit for request to update data  
k := one-time key for encryption  
k' := one-time key encrypted asymmetrically

h\_ru := hash(d' ++ up() ++ ar ++ v)  
h\_dp := hash(d')  
h\_key := hash(u)  
h\_tx := hash(v ++ n)  
h\_dn := hash(d ++ v ++ n)  
o := hash(v ++ u)  
h\_da := hash(d' ++ a ++ u)  
h\_ipfs\_d := hash(a ++ o ++ c) [d for data]  
h\_ipfs\_p := hash(e\_rs ++ c) [p for proof]

# Proving Ownership of Encrypted Data In Smart Contract While Maintaining Privacy in Solidity



In the smart contract:

- a. Assert that on-chain hash == h\_key computed off-chain provided to smart contract by user when onboarding in User Keychain Mapping for PBK\_user
- b. Assert that on-chain hash == h\_tx computed off-chain provided to smart contract by public service verifier
- c. Assert that User Data Mapping at (PBK\_user) => o => (a, h\_da)

In ZoKrates:

1. Therefore the symmetric key u provided is valid
2. Therefore v (and n) is valid and request is decrypted correctly [coupled with assertion (b)]
3. Therefore o is valid, the obfuscated field from which we are retrieving data
4. Therefore (d, a) is valid [one MUST onboard with valid hash h\_da]
5. Therefore h\_dn is a valid hash of d with v and n that can be used to verify that the encrypted response e\_rs is valid upon decryption to rs
6. Proves that provided hash h\_ipfs\_p of content at CID c uploaded on IPFS is valid

In the smart contract:

1. Place (e\_rs, k', h\_dn) into smart contract w/storing function
2. Place (IPFS hash [c], k', h\_dn, h\_ipfs\_p) into smart contract w/storing function

Off-chain:

1. a. Extract (e\_rs, k', h\_dn, h\_ipfs\_p) from smart contract/IPFS CID  
b. Verify hash(e\_rs ++ c) == h\_ipfs\_p
2. decrypt\_RSA(k', PVK\_RSA\_verifier) = k
3. a. decrypt\_AES(e\_rs, k) = d  
b. Verify that h\_dn == hash(d ++ v ++ n)  
c. Compare timestamps of proof
4. Decide to accept or reject proof

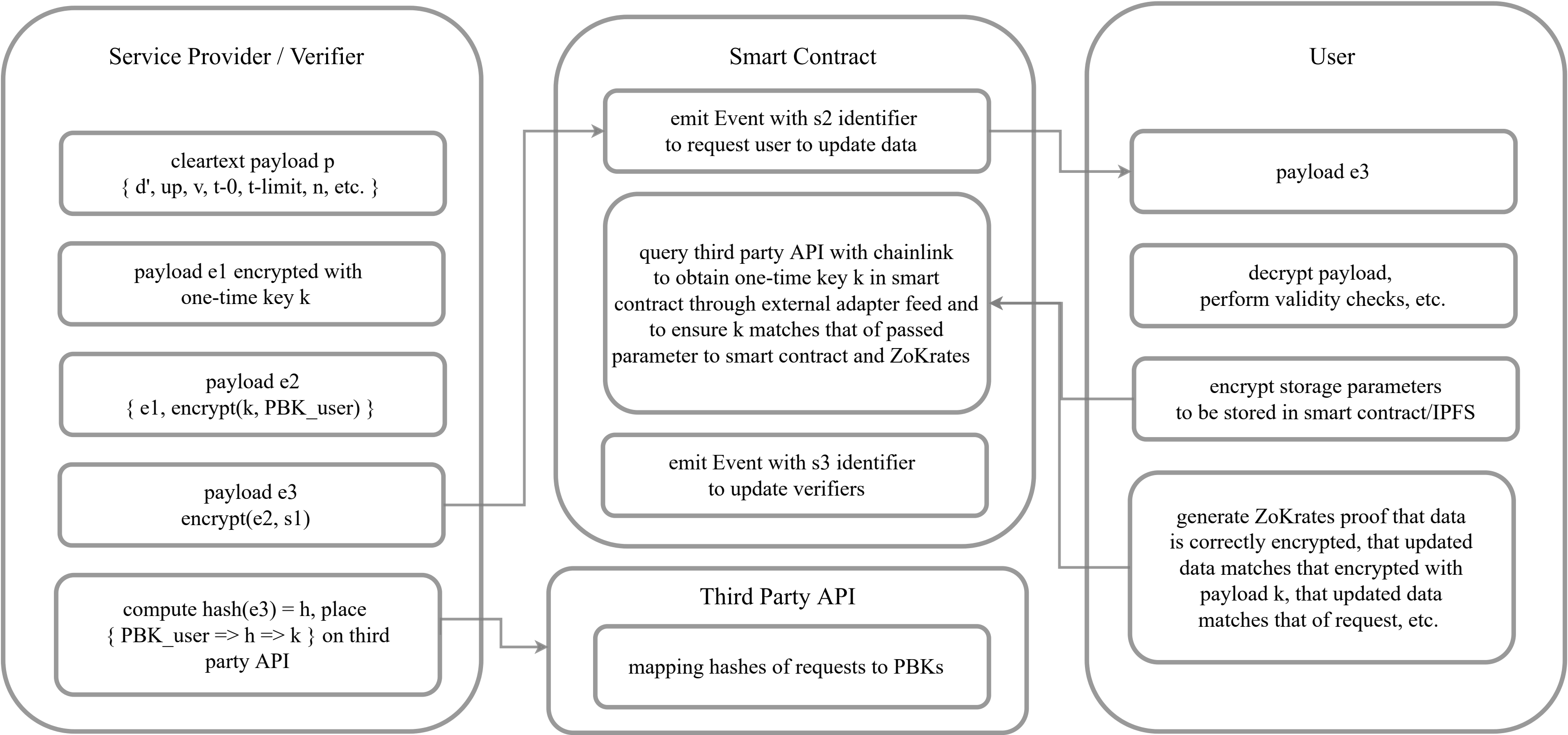
PBK\_RSA\_user := user's public key for RSA  
PVK\_RSA\_user := user's private key for RSA  
PBK\_RSA\_verifier := verifier's public key for RSA  
PVK\_RSA\_verifier := verifier's private key for RSA  
PBK\_user: := user's public address  
PBK\_verifier: := verifier's public address

u := symmetric key of user  
v := variable field requested ("age"), in plaintext  
n := transaction nonce (10 byte string)  
o := obfuscated field ("age" -> "v2m0t97y")  
d := user data stored at some field  
d' := user data requested to be stored at some field  
a := d' encrypted symmetrically with AES  
c := IPFS content identifier (URI/URL)  
e\_ru := symmetrically encrypted request to update data  
ru := request to update data  
e\_rq := asymmetrically encrypted request to prove ownership  
rq := request to prove ownership  
e\_rs := symmetrically encrypted response to prove ownership  
rs := response to prove ownership  
up() := update function requested by verifier on d  
ar := arguments to update function  
t\_limit := the time limit for request to update data  
k := one-time key for encryption  
k' := one-time key encrypted asymmetrically

h\_ru := hash(d' ++ up() ++ ar ++ v)  
h\_dp := hash(d')  
h\_key := hash(u)  
h\_tx := hash(v ++ n)  
h\_dn := hash(d ++ v ++ n)  
o := hash(v ++ u)  
h\_da := hash(d' ++ a ++ u)  
h\_ipfs\_d := hash(a ++ o ++ c) [d for data]  
h\_ipfs\_p := hash(e\_rs ++ c) [p for proof]

Accounting For Side-Channel Inference: An Implementable Extension With Chainlink and/or Asymmetric Encryption  
Verification Components To Hide Vera Request and Response Receivers For Proof of Valid Storage When On-boarding

*\*The requests, responses, and fields remain obfuscated regardless of whether this extension is present,*

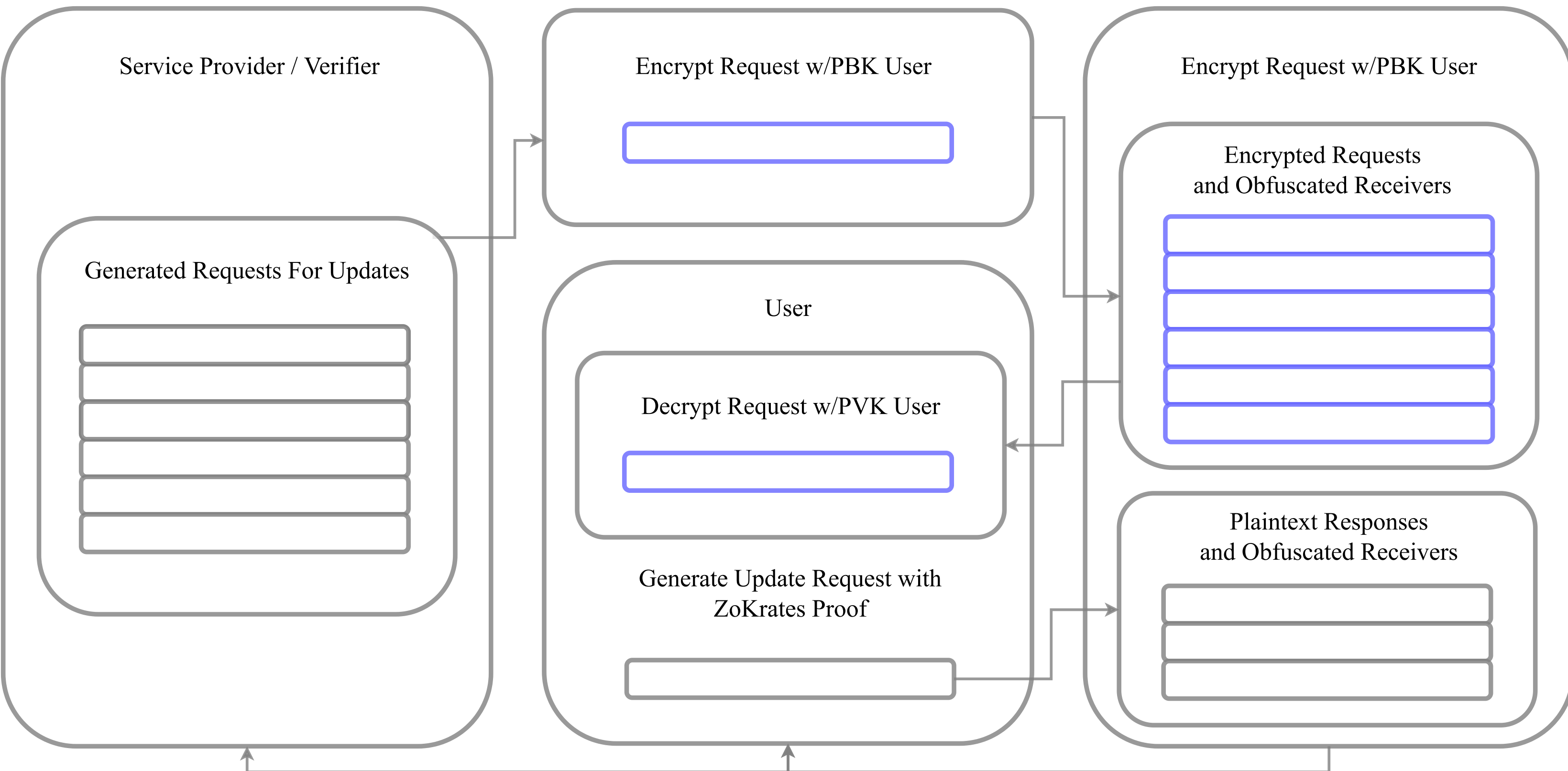


Assume a number of negotiated shared secrets { s1, s2, s3, ... } between user and verifier are available negotiated through DH key exchange, secondary secure channels, etc.



Accounting For Side-Channel Inference: An Implementable Extension With Diffie-Hellman Exchange And/Or  
Secondary Secure Channel To Hide Vera Request and Response Receivers For Proof of Ownership

*\*The requests, responses, and fields remain obfuscated regardless of whether this extension is present,*



Alert Each Other Of Available Request  
Through Secure Channel Or Through  
Emitting Events/Information With Shared  
Secret In Smart Contract Such as With  
SmartDHX Diffie-Hellman Key Exchange  
(Using A Separately Negotiated DH Key for  
Each Direction), etc.

Alternatively, The User/Verifier  
Periodically Monitors The Blockchain  
To Identify Matching Requests