# Anonymous KYC But not possible to remove duplicates

About 100 validators are selected through conviction voting to perform KYC. For each user, 5 validators are randomly chosen as jurors. KYC is conducted via end-to-end encrypted P2P chat and video conferencing by selected jurors.

## ZKP with validators

Jurors are provided with the user's name, address, and a hash. The user must prove that they know the secret corresponding to the hash.

```rust
use risc0_zkvm::{default_prover, ExecutorEnv};
use sha2::{Digest, Sha256};

let name = "Alice".to_string();
let address = "123 Main St".to_string();
let expiry = 1690000000u64; // Timestamp
let secret = "secret-password"

let env = ExecutorEnv::builder()
    .write(&name)
    .write(&address)
    .write(&secret)
    .write(&expiry)
    .build()?;

let hash = Sha256::digest(format!("{name}:{address}:{secret}").as_bytes());

assert_eq!(receipt.journal.bytes[..32], hash[..], name, address);
```

## Zero Knowledge proof in Blockchain

Hash is stored in blockchain.

```rust
fn main() {
    let (name, address, secret, expiry): (String, String, String, u64) = env::read();
    let commitment = Sha256::digest(format!("{name}:{address}:{secret}").as_bytes());

    env::commit(&commitment); // Privacy-preserving
    env::commit(&expiry);
}
```

```rust
use risc0_zkvm::{default_prover, ExecutorEnv};
use sha2::{Digest, Sha256};

let name = "Alice".to_string();
let address = "123 Main St".to_string();
let expiry = 1690000000u64; // Timestamp
let secret = "secret-password"
```

```rust
8
9  let env = ExecutorEnv::builder()
10     .write(&name)
11     .write(&address)
12     .write(&secret)
13     .write(&expiry)
14     .build()?;
15
16 let hash = Sha256::digest(format!("{name}:{address}:{secret}").as_bytes());
17
18 assert_eq!(receipt.journal.bytes[..32], hash[..]);
```

```rust
1 #[pallet::storage]
2 pub type Commitments<T: Config> = StorageMap<_, Blake2_128Concat, [u8; 32],
  (T::AccountId, u64)>;
3 // Maps (commitment) => (owner, expiry)
```

```rust
1  #[pallet::call]
2  impl<T: Config> Pallet<T> {
3      #[pallet::weight(10_000)]
4      pub fn register_kyc(
5          origin: OriginFor<T>,
6          proof: Vec<u8>,
7          hash: [u8; 32],
8          expiry: u64
9      ) -> DispatchResult {
10         let who = ensure_signed(origin)?;
11
12         // Verify Risc0 proof
13         risc0::verify_proof(&proof, &hash, &expiry)?;
14
15         ensure!(
16             !KycHashes::<T>::contains_key(&hash),
17             Error::<T>::KycAlreadyRegistered
18         );
19
20         KycHashes::<T>::insert(&hash, (who.clone(), expiry));
21         Ok(())
22     }
23
24     #[pallet::weight(10_000)]
25     pub fn extend_kyc(
26         origin: OriginFor<T>,
27         proof: Vec<u8>,
28         hash: [u8; 32],
29         new_expiry: u64
30     ) -> DispatchResult {
```

```rust
31          let who = ensure_signed(origin)?;
32
33          risc0::verify_proof(&proof, &hash, &new_expiry)?;
34
35          KycHashes::<T>::try_mutate(&hash, |entry| {
36              let (account, expiry) =
                    entry.as_mut().ok_or(Error::<T>::NotRegistered)?;
37              ensure!(*account == who, Error::<T>::NotAuthorized);
38              *expiry = new_expiry;
39              Ok(())
40          })
41      }
42
43      #[pallet::weight(10_000)]
44      pub fn governance_remove(
45          origin: OriginFor<T>,
46          hash: [u8; 32]
47      ) -> DispatchResult {
48          T::GovernanceOrigin::ensure_origin(origin)?;
49          KycHashes::<T>::remove(&hash);
50          Ok(())
51      }
52  }
```

## Store the name, address and photo by validator

```rust
1   /// Struct to hold identity information                              ⊕ Rust
2   #[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug, TypeInfo,
    MaxEncodedLen)]
3   pub struct IdentityData {
4       pub name: BoundedVec<u8, ConstU32<100>>,
5       pub address: BoundedVec<u8, ConstU32<200>>,
6       pub photo_hash: BoundedVec<u8, ConstU32<200>>,
7       pub validator_approved: bool,
8   }
9
10  // Stored and approved by validator
11  #[pallet::storage]
12  #[pallet::getter(fn identity_data)]
13  pub type IdentityStore<T: Config> = StorageValue<_, IndentityData, ValueQuery>;
```

Users must ensure that their name, address, and photo are added to the blockchain by a validator, or they can appeal to the governance.