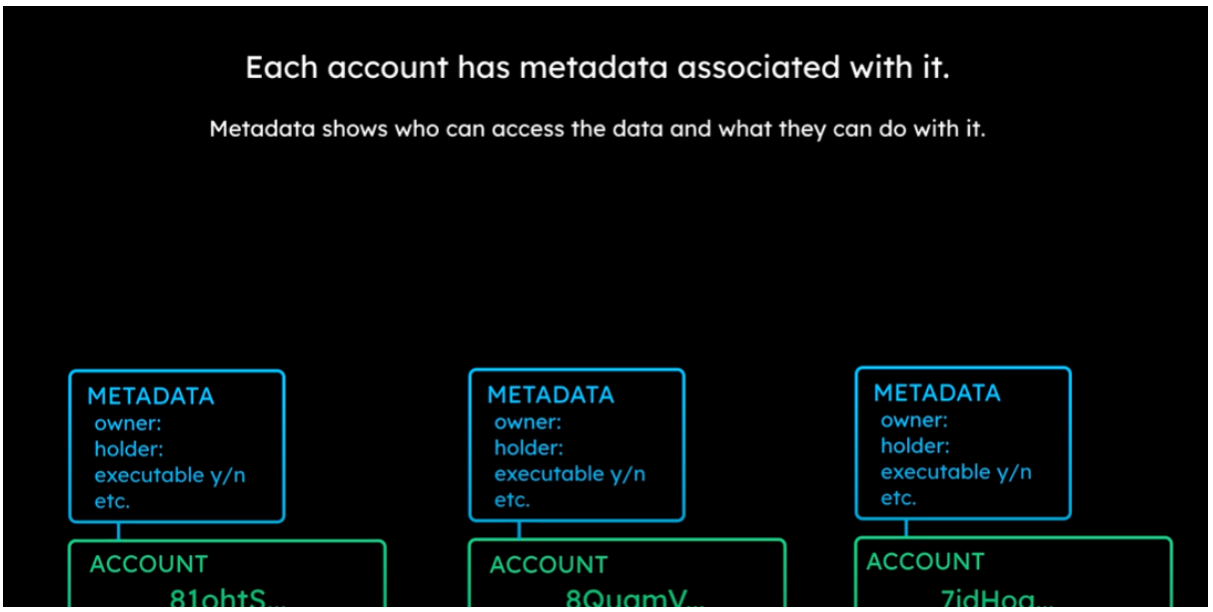
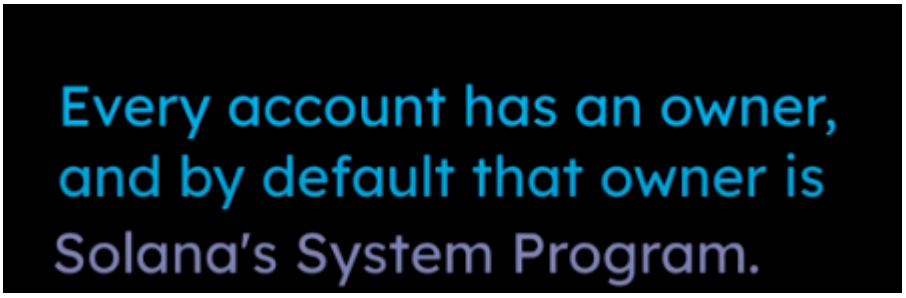
Core Concepts- Solana Account System

Solana uses a system of accounts. All Solana accounts can be identified 256-bit public key. There are two types of accounts in Solana. Account 1 stored programming code that is executable. Account 2 stores other valuable data such as variables or assets of the user, not pure programming values. Accounts are held in the validator's memory. Each account has different metadata associated with it and programs use metadata to determine who can access data and what can they do with it.

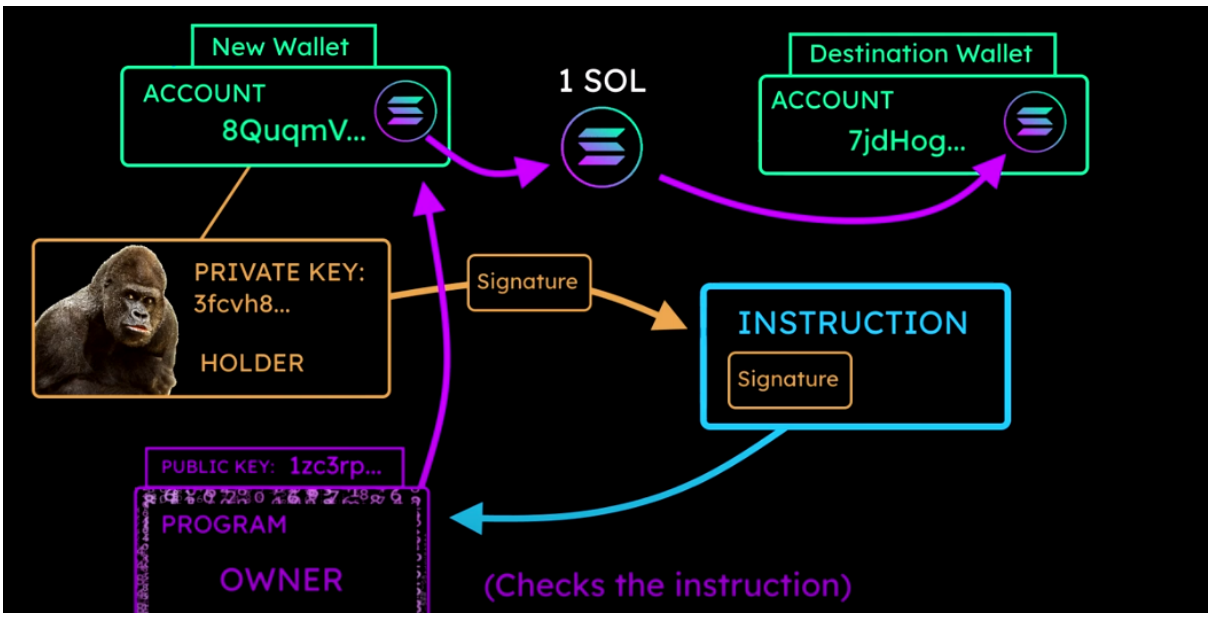
Metadata structure



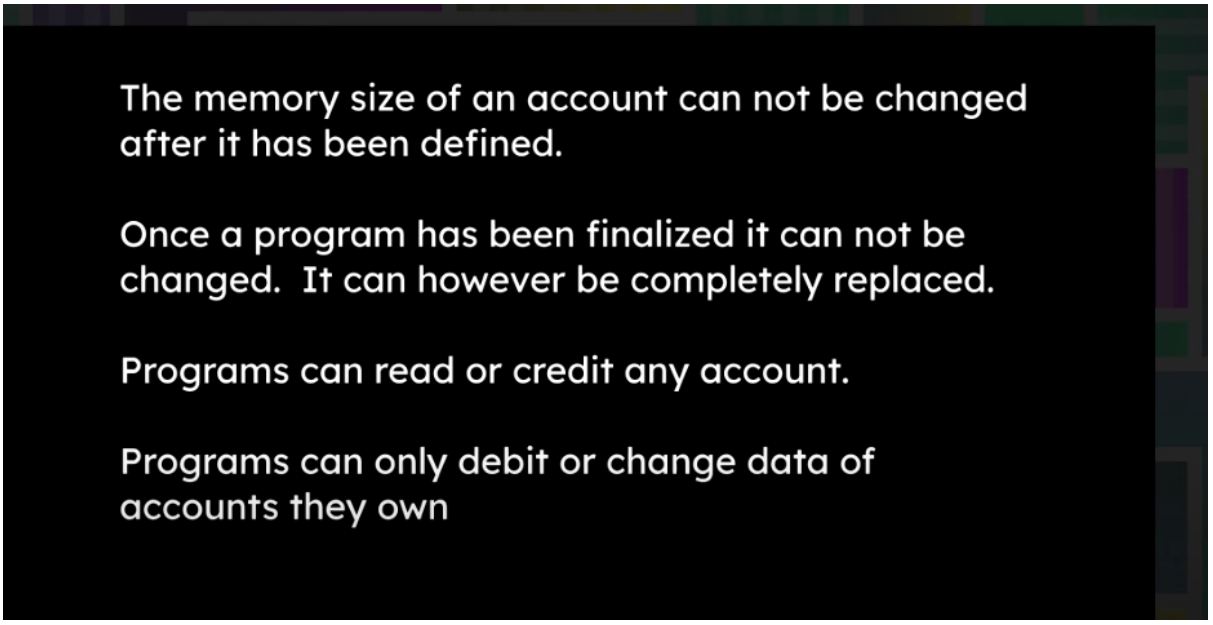
One field is called the owner. The public key of the program is to change data on the account. Every account on Solana has an owner.



Solana's System program does transfer ownership to other programs. A person who holds a private account is called a holder.



How the transaction system works on the Solana network.



The system program can reassign ownership of the account. It is the only program that can happen once per account. Once ownership is assigned by the system it is finalized and not changeable.

Since Solana programs cannot be changed after being finalized, data is stored in a separate non-executable storage account. And the system set the owner as the program. When data is needed, it pulls the data from the storage account.

Validity of Programs

Verifying validity of unmodified, reference-only accounts

For security purposes, it is recommended that programs check the validity of any account it reads, but does not modify.

This is because a malicious user could create accounts with arbitrary data and then pass these accounts to the program in place of valid accounts. The arbitrary data could be crafted in a way that leads to unexpected or harmful program behavior.

The security model enforces that an account's data can only be modified by the account's `Owner` program. This allows the program to trust that the data is passed to them via accounts they own. The runtime enforces this by rejecting any transaction containing a program that attempts to write to an account it does not own.

If a program were to not check account validity, it might read an account it thinks it owns, but doesn't. Anyone can issue instructions to a program, and the runtime does not know that those accounts are expected to be owned by the program.

To check an account's validity, the program should either check the account's address against a known value, or check that the account is indeed owned correctly (usually owned by the program itself).

One example is when programs use a sysvar account. Unless the program checks the account's address or owner, it's impossible to be sure whether it's a real and valid sysvar account merely by successful deserialization of the account's data.

So as we can see for malicious purposes, only the owner of the program can change and modify the program. As we can see in the second paragraph, created accounts can reach the program and harm it. The runtime enforces rejects the program that is not owned.

Transaction on Solana

Program execution starts with being submitted to the cluster. Solana runtime process the instructions in transactions, serially and atomically. The message header contains unsigned 8-bit values. Each instruction specifies a single program that is a subset of the transaction account that passed to the program. If an error occurs system totally fails automatically. The transaction can include instructions in any order so that a sequence would be blocked by malicious intent.

```
pub fn create_account(
    from_pubkey: &Pubkey,
    to_pubkey: &Pubkey,
    lamports: u64,
    space: u64,
    owner: &Pubkey,
) -> Instruction {
    let account_metas = vec![
        AccountMeta::new(*from_pubkey, true),
        AccountMeta::new(*to_pubkey, true),
    ];
    Instruction::new_with_bincode(
        system_program::id(),
        &SystemInstruction::CreateAccount {
            lamports,
            space,
            owner: *owner,
        },
        account_metas,
    )
}
```