



Accounts and transactions in Algorand


Lecture 5

Author: Olha Konnova



Plan

- 1) Account in the Algorand blockchain. Types of accounts
- 2) Wallets in Algorand. Types of crypto wallets
- 3) Transactions. Types of transactions in Algorand
- 4) Group transactions
- 5) Internal transactions
- 6) Creating transactions using the JS AlgoSDK

The background features a dark blue to black gradient with intricate, flowing, and swirling patterns of lighter blue lines and particles, creating a sense of dynamic movement and depth.

**Algorand is an account-
based blockchain platform**



Blockchain account

A **blockchain account** is a digital record that contains information about the owner's status in the blockchain network. Each account has a unique address that is used to identify the user and interact with the blockchain.

Users use a Blockchain account to:

- hold cryptocurrencies and tokens in the blockchain network;
- send and receive cryptocurrency or tokens;
- interact with blockchain applications;
- sign transactions.

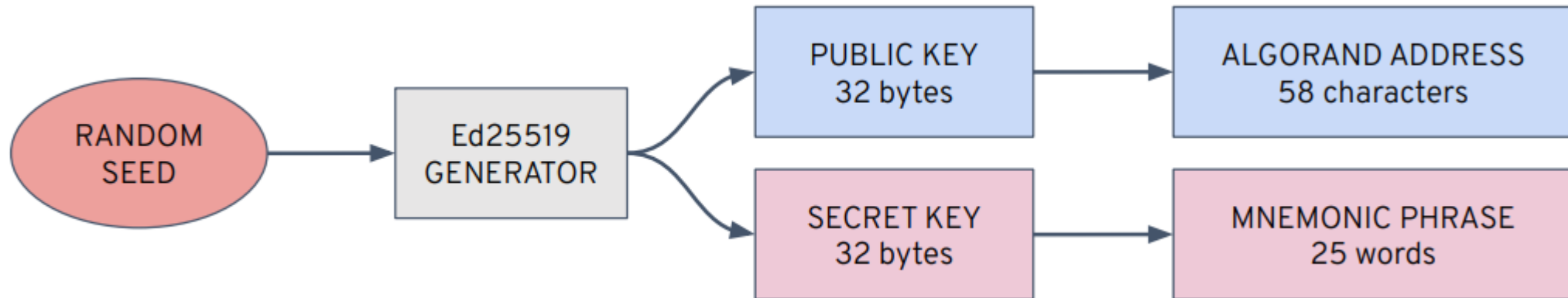
Account structure

When creating a new account, a pair of keys is generated: a private and a public one.

- **Public key** - a key that is generated from the private key. The public key is used to generate the account address and can be made publicly available.
- **Private key** - a secret key used to sign transactions. The private key must be kept confidential because anyone with access to it can control the account.

Mnemonic phrase

- This is a set of 25 BIP39 English words used to recover an account. The mnemonic phrase is generated based on the private key and can be used to recover an account in case of loss of access to it.
- It is used to back up and restore your account. Saving the mnemonic phrase is critical because losing access to it means losing access to your account.



Ed25519 is an algorithm used to generate keys and sign transactions in Algorand.

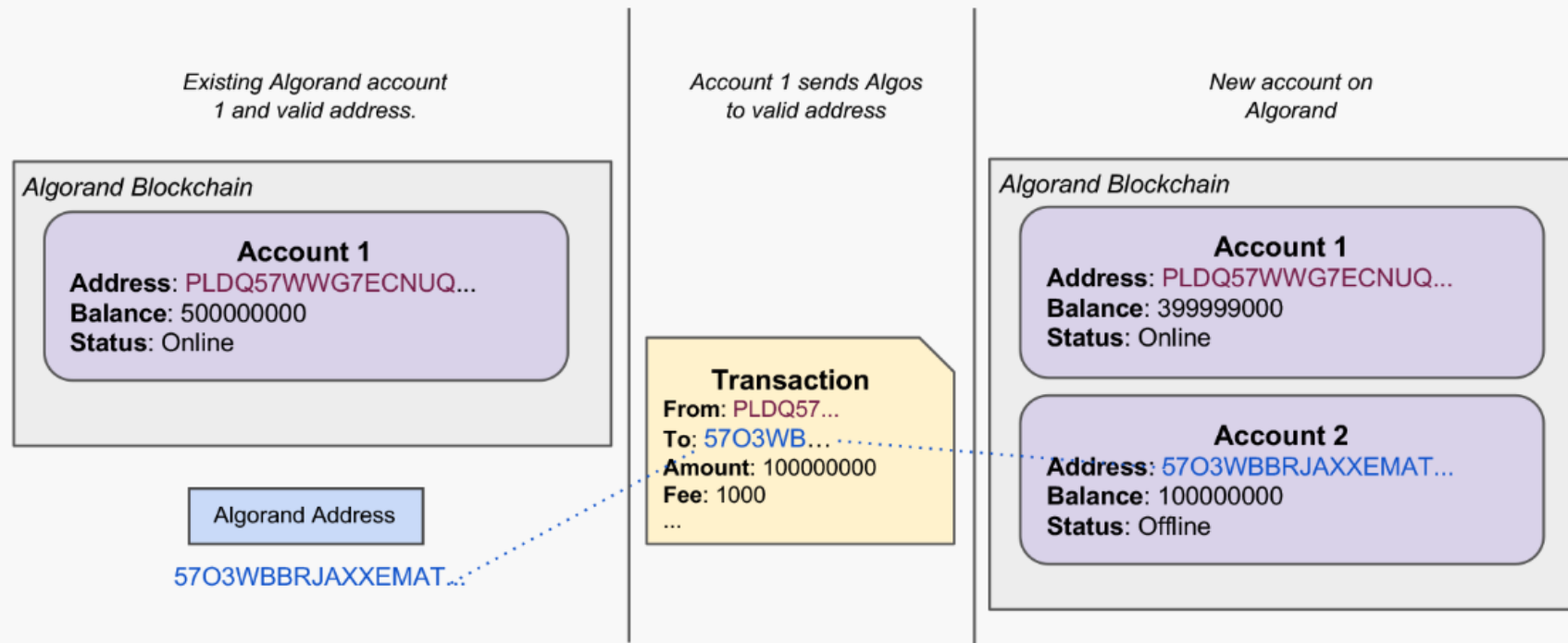
ADDRESS: the **public key** is converted to an **Algorand address** by adding a 4-byte checksum to the end of the public key and **encoding it in base32**.

MNEMONIC: a 25-word mnemonic is created by converting the private key bytes to 11-bit integers and then mapping these integers to the English word list bip-0039.

Algorand Accounts

Accounts are objects in the Algorand blockchain associated with a specific local state in the network, such as balance, created applications, and assets.

The **Algorand address** is a unique identifier for your Algorand account. It consists of 58 characters and is generated based on the public key.



After generating the private key and the corresponding address, sending Algos to the address in Algorand initialises its state in the Algorand blockchain.

Dappflow account information

Account address

Account : FXA6NGWY6MR2Y5Q7ZPFH7CC7OWHX5JAIED22235KH6RPK55GHUB5JH4O54

Show QR

Local states

View Raw JSON

Current balance in Algo

Offline

Balance999,999,995.971▲

Holding assets0

Created applications19

Minimum balance4.983▲








Created assets0

Opted applications0

Assets created by this account

TransactionsAssetsCreated assetsCreated applicationsOpted applications

Account transaction history

Txn ID	Block	Age	From	To	Amount	Fee	Type
 SJMLCJDCIK5456O2TN...	12828	1 day, 4 hours ago	FXA6NGWY6MR2Y5Q7ZPFH...	OUT Application: 1078		▲ 0.001	App call
 BCODZMWSER76J2EJAL...	12793	1 day, 4 hours ago	FXA6NGWY6MR2Y5Q7ZPFH...	OUT Application: 1078		▲ 0.001	App call
 LOAKGN2K4JWSPFY5EK...	12646	1 day, 4 hours ago	FXA6NGWY6MR2Y5Q7ZPFH...	OUT Application: 1078		▲ 0.001	App call
  AG6SPP4DB232YIJJV...	10838	1 day, 10 hours ago	FXA6NGWY6MR2Y5Q7ZPFH...	OUT CWP3BUY5KM553OL...	▲ 2	▲ 0.001	Payment
  LDTHLAIMQJ3QFXA3E...	10838	1 day, 10 hours ago	FXA6NGWY6MR2Y5Q7ZPFH...	OUT Application: 1068		▲ 0.001	App call



Minimum balance for Algorand accounts

Each account on Algorand must have a **minimum** balance of **100,000 microAlgos**.

The minimum balance **increases** with each asset held in the account (regardless of whether the asset was created or belongs to the account), as well as with each application that the account has created or been included in.

Increasing the minimum balance

Assets:

- For each asset created or owned by the account, its minimum balance is **increased by 0.1 Algos** (100,000 microAlgos).
- An account can opt out an asset at any time. This means that the account will no longer hold the asset and the account will no longer be able to receive the asset. The minimum balance requirement is also **reduced** by **0.1 Algo**.

Smart contracts:

- When you create or connect to a smart contract, the minimum balance for your account will be increased. The amount by which it will be increased will depend on the amount of storage in the network that is used. (Lecture 2. Slide: Minimum balance requirements for a smart contract¶)

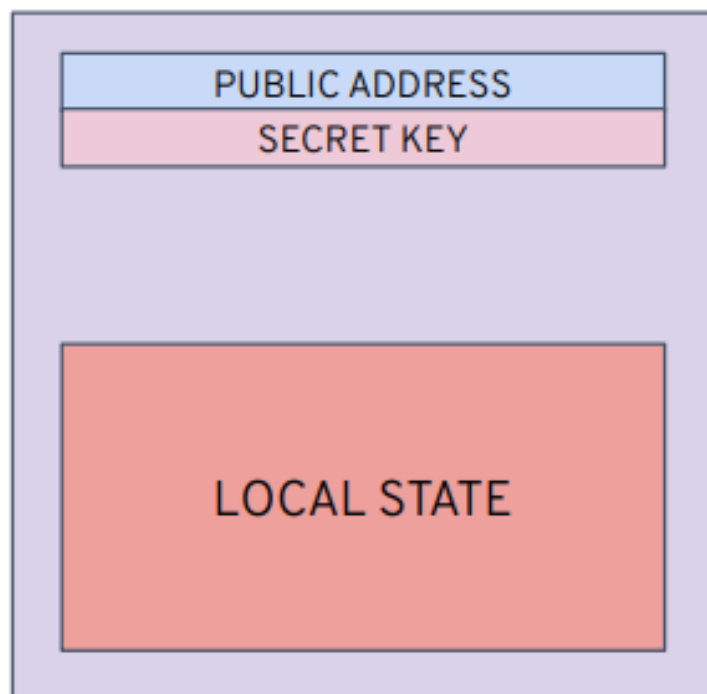
*If ever a transaction is sent that would result in a balance lower than the minimum, the transaction will fail.



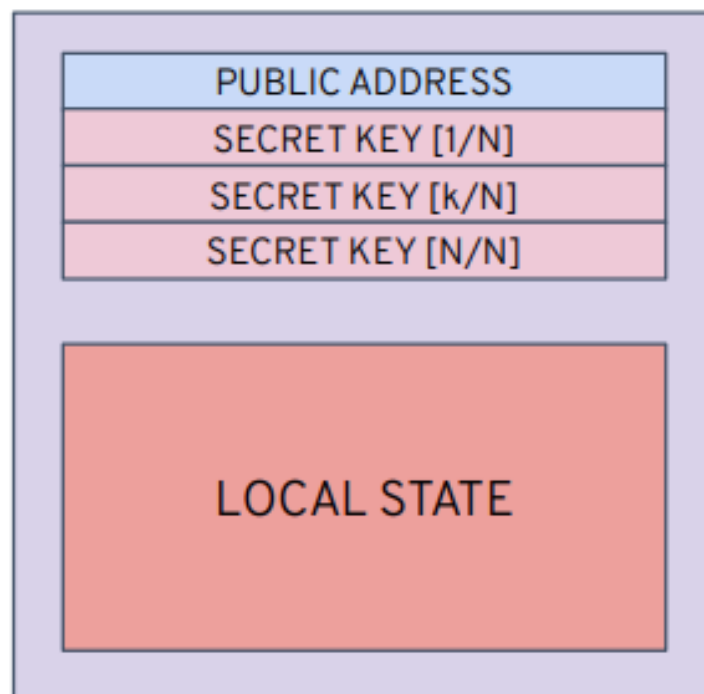
Types of accounts

- **Standard account:** used to store ALGOs and other assets, as well as to make transactions.
- **Multisignature Account:** requires multiple signatures before a transaction can be executed. They consist of a list of addresses, a version, and a threshold value that indicates how many signatures are required to validate a transaction.
- **Contract Account:** all deployed smart contracts have their own application account with an associated Algorand public address. These accounts are used for internal transactions within the smart contract.

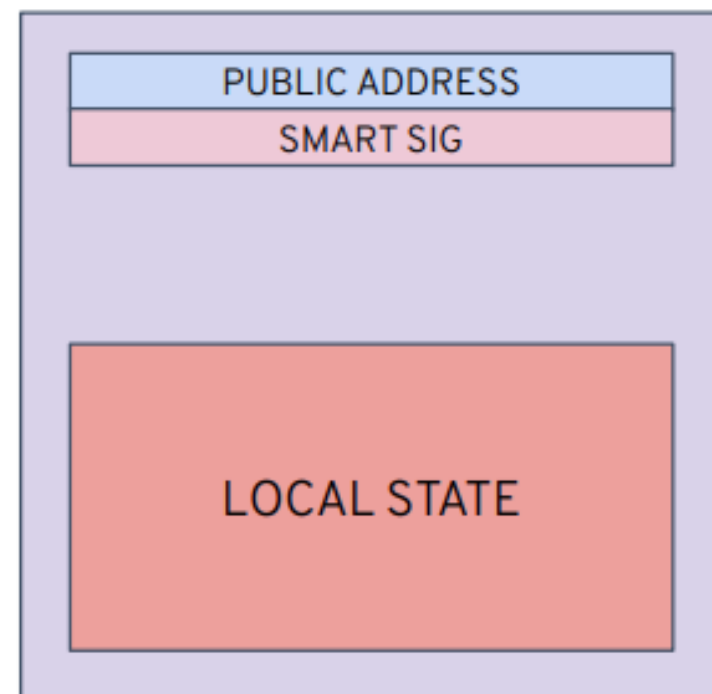
STANDARD ACCOUNT



MULTI SIGNATURE ACCOUNT



CONTRACT ACCOUNT



Multi-subscription account

01

Multisignature accounts can perform the same operations as other accounts, including sending transactions and participating in consensus.

02

A multi-subscriber account address is created by hashing a list of addresses, version, and threshold. The order of the addresses is important - changing the order will change the address.

03

The threshold determines how many signatures are required to process any transaction from this multi-signature account.

04

As with other accounts, to activate a multi-signature address on the blockchain, you need to send several Algos to it.

Smart contract account

01

To use the TEAL app as an account, you need to send Algos to its address to convert it into an Algorand account with a balance.

02

Externally, this account is no different from any other Algorand account, and anyone can send it Algos or Algorand Standard Assets to increase its balance.

03

The peculiarity of a contract account is that it is the logic of the smart contract that determines whether a transaction will be approved.

04

Contract accounts require Algos to pay transaction fees and participate in transactions. You can increase its balance by sending Algos to the contract account address.

A **crypto wallet** is an application that functions as a wallet for cryptocurrency. It is called a wallet because it is used similarly to a wallet in which you put cash and cards. Instead of storing these physical items, it stores the access keys used to sign cryptocurrency transactions and provides an interface that allows users to access their cryptocurrency.

Wallets

01

Blockchain wallets do not contain real currency. The wallets are used to store private keys and maintain the balance of transactions. The real data or currency is stored in blocks on the blockchain.

02

Wallets contain the address and private keys needed to sign transactions. Anyone who knows the private key can control the coins associated with that address.

Types of cryptocurrency wallets

Depending on the principle of operation, there are hot (software) and cold (hardware) crypto wallets.

- **Hot wallets** include mobile, desktop, and web applications.
- **Cold wallets** have a physical form, they are special devices that can be connected to a PC. They store the private keys to the user's cryptocurrency offline.

According to the type of data management, wallets are divided into two groups: custodial and non-custodial.

- **Custodial services**, such as cryptocurrency exchanges, store public and private keys on their own servers.
- **Non-custodial wallets** allow the user to store the keys on the device themselves. This type of wallet does not rely on a third party or custodian to ensure the security of the user's cryptocurrency. When creating an account in such a wallet, the user specifies a recovery phrase, so they can access the cryptocurrency even if the keys are lost.



An Algorand wallet is software or hardware that securely stores private and public keys, allowing users to store, access, send and receive Algo and other assets



Software wallets in Algorand

The main characteristics of the wallet:

- **Multiple accounts:** one wallet can manage multiple accounts.
- **Key management:** the wallet securely stores and manages cryptographic keys (private and public) for each account.
- **User Interface:** wallets offer interfaces (web, desktop or mobile apps) that allow users to interact with the blockchain, send and receive coins, view transaction history and much more.

Algorand wallets



Android - iOS - Browser extension -
Desktop



Android - iOS



Hardware wallet



Android - iOS - Web



Transactions. Changing the state of the blockchain

A transaction is the basic element of blocks that determines the evolution of the state of the distributed ledger. It is a record that is added to the blockchain ledger and includes details about the transfer of digital assets between participants or other actions on the network (creating an application, creating assets, calling an application, etc.).

There are seven types of transactions in the Algorand protocol:

- [Payment](#)
- [Registration Key](#)
- [Asset Configuration](#)
- [Asset Freeze](#)
- [Asset Transfer](#)
- [Application Call](#)
- [State Proof](#)

Types of transactions

- **Payment transaction** (Payment): sends Algos from one account to another.
- **Asset Transfer**: used to agree to receive a certain type of ASA (Algorand Standard Asset), transfer ASA from one account to another.
- **Asset Configuration transaction**: assetConfigTx is used to create an asset, change certain asset parameters, or destroy an ASA.
- **Asset Freeze transaction**: allows you to freeze or unfreeze assets for a specific account. When an asset is frozen for an account, the account cannot send or receive that asset.
- **Key Registration transaction**: used to register participation keys and online status for an Algorand account. This transaction is important for users who want to participate in the Algorand consensus mechanism (Pure Proof of Stake) by becoming a validator.

Types of transactions

Application Call transaction: sent to the network with an AppId and an OnComplete method. The AppId identifies which application is to be called, and the OnComplete method is used in the contract to determine which branch of logic is to be executed.

Application call transactions can contain other fields required by the logic, for example:

- ApplicationArgs - for passing arbitrary arguments to a smart contract
- Accounts - for transferring accounts that may require some balance checking or connection status
- ForeignApps - for transferring applications and allowing access to the state of an external application
- ForeignAssets - passing ASA to check parameters
- Boxes - to pass links to application boxes so that AVM can access the content

An example of a payment transaction

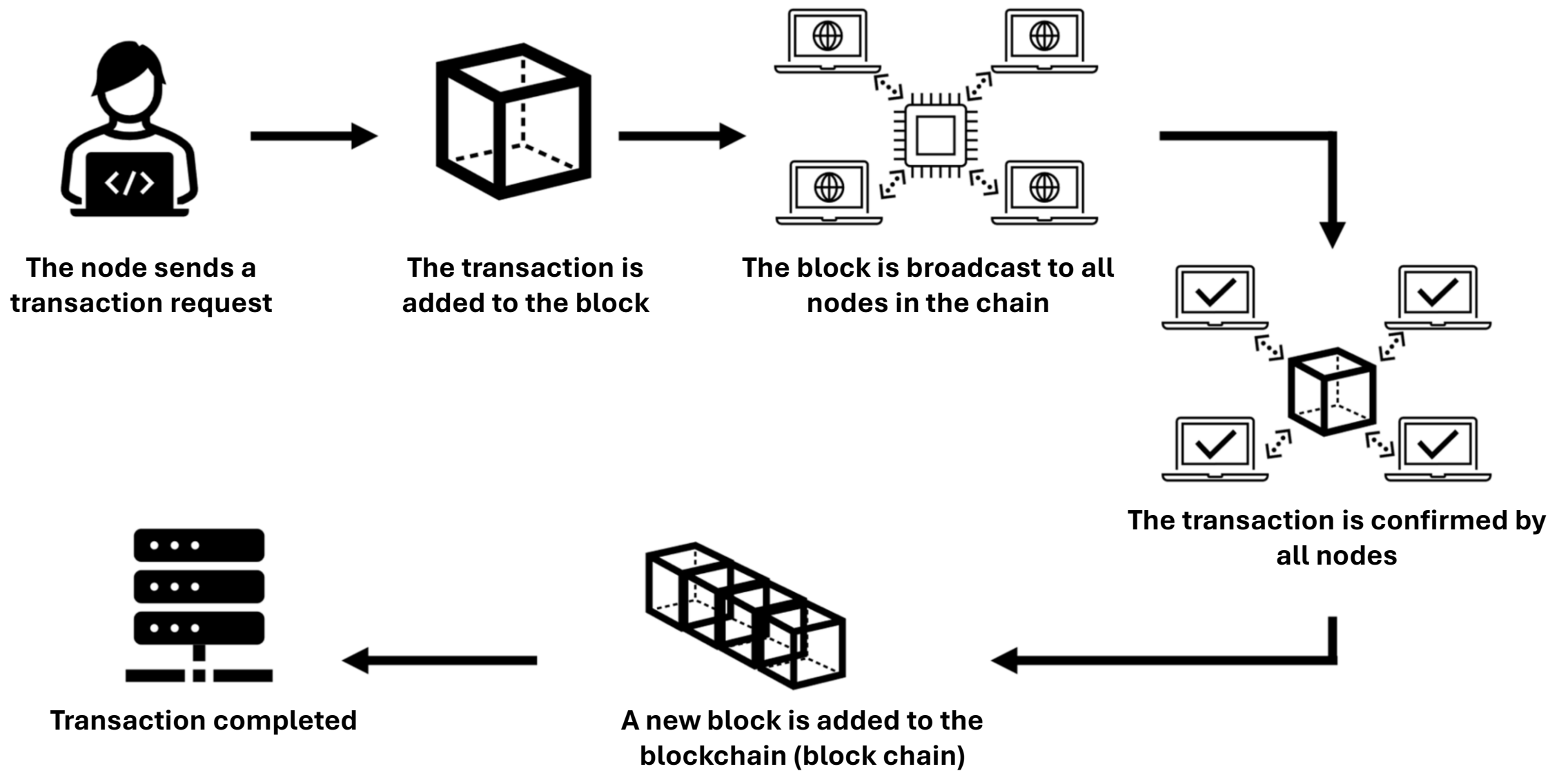
Example of a transaction that sends 5 ALGOs from one account to another in MainNet

```
{
  "txn": {
    "amt": 5000000,
    "fee": 1000,
    "fv": 6000000,
    "gen": "mainnet-v1.0",
    "gh": "wGHE2Pwvdvd7S12BL5Fa0P20EGYesN73ktiC1qzkkit8=",
    "lv": 6001000,
    "note": "SGVsbG8gV29ybGQ=",
    "rcv": "GD64YIY3TWGDMCNPP553DZPPR6LDUSFQ0IJVFDPPXWEG3FVOJCCDBBHU5A",
    "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCLIHZU6TBEOC7XRSBG4",
    "type": "pay"
  }
}
```

First valid: Number of the first block in which the transaction can be included

Genesis ID: The identifier of the genesis block that defines the network to which this transaction belongs

Note: Additional information related to the transaction is encoded in Base64





Atomic transactions

An atomic transaction in Algorand is a way to group multiple transactions together so that they execute as a single unit. If any transaction in the group fails, all transactions in the group fail. This ensures that all actions are completed successfully or none.



Application cases

Atomic transactions can be used in such cases:

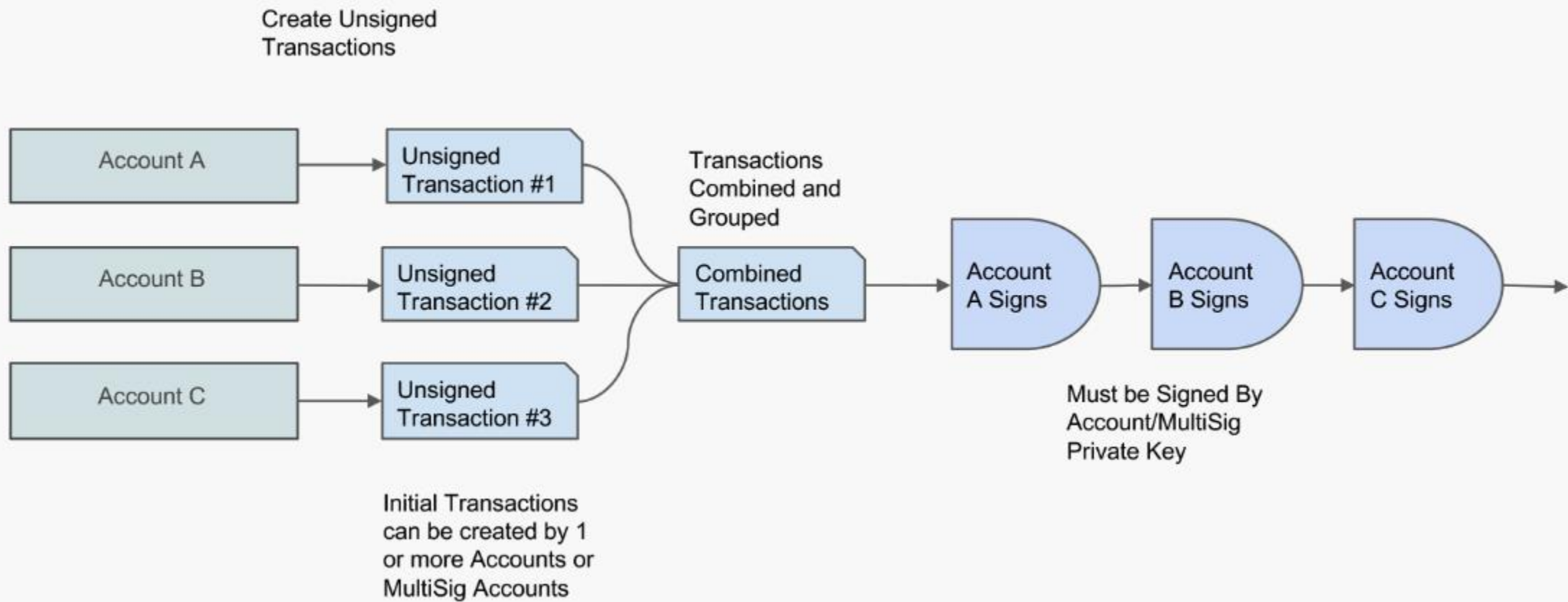
- **Group payments:** everyone pays or no one pays.
- **Distributed payments:** payments to multiple recipients. When you need to make payments to multiple recipients at the same time, ensuring that all recipients receive their payments or none of them do.
- **Aggregate transaction fees:** one transaction pays for the fees of the others. One party to the transaction can cover the fees for all transactions in the group, ensuring that all transactions are executed simultaneously without disruption due to insufficient funds to pay the fees.
- **Asset swap:** The exchange of one asset for another between two or more parties. Group transactions ensure that the exchange only occurs if all transactions associated with the exchange are successfully completed.



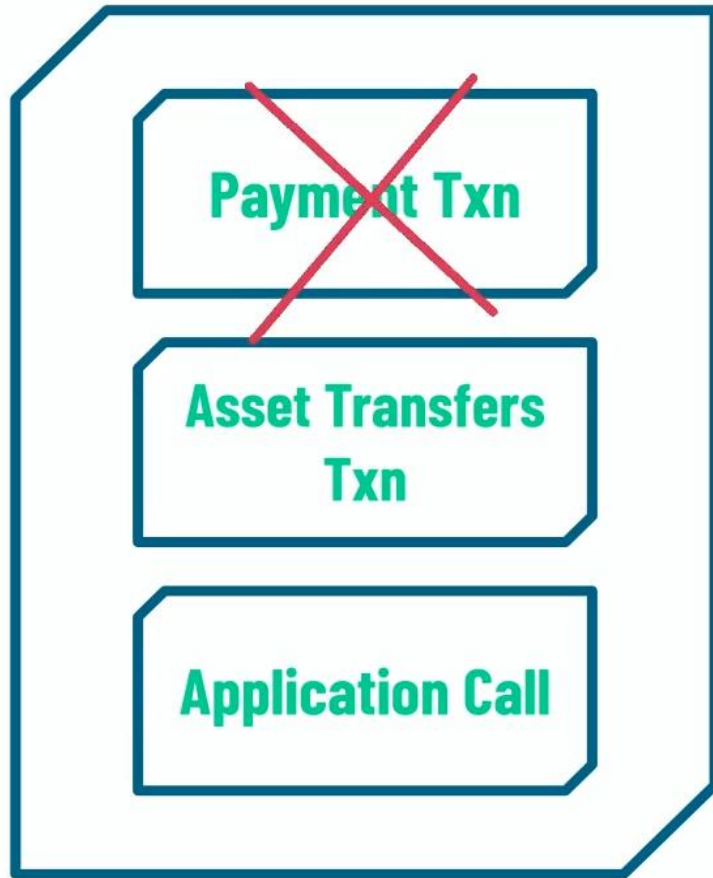
Group transactions

Group transactions are an implementation of atomic transactions in Algorand. Here's how they work and what they include:

1. **Creating group transactions:** to create a group of transactions, you need to specify several individual transactions (up to 16) that you want to include in the group. Each transaction will have a unique identifier.
2. **Group ID assignment:** all transactions in a group are assigned a common group ID. This identifier ensures that the transactions are linked together and should be processed together.
3. **Validation and sending:** when a group of transactions is sent to the Algorand network, the network validates the entire group. If any transaction in the group is invalid or fails for any reason, the entire group is rejected.



Atomic Group



Atomic Group





Internal transactions

Internal transactions in Algorand are transactions that are created and executed by smart contracts (or "apps" in Algorand terminology) during their execution. They allow smart contracts to initiate asset transfers or perform other actions without the need for external intervention or the creation of separate transactions outside the contract.

A smart contract can execute up to 256 internal transactions in a single call.

If any of these transactions fails, the smart contract will also fail.

Internal transaction fees are paid by the smart contract and are automatically set at the minimum transaction fee. Internal transaction fees can be combined into a **fee pool** like any other transaction. This allows either the application call or any other transaction in the transaction group to pay the internal transaction fee.

Internal transactions are evaluated during the execution of the AVM, allowing changes to be visible to the contract. For example, if the "balance" transaction code is used before and after the "payment" transaction is sent, the change in balance will be visible to the executing contract.

The recipient of the internal transaction must be in the Accounts array of the application call transaction.

Application cases

- A smart contract can serve as an escrow account, holding funds and then transferring them when certain conditions are met, as defined by the smart contract logic.
- Smart contracts can interact with each other by using internal transactions to call functions of other smart contracts or transfer assets to them.
- A smart contract can create new assets and automatically distribute them to users. This can be used to issue new tokens or NFTs.

Create an internal transaction

The `itxn_begin`, `itxn_field`, `itxn_next`, and `itxn_submit` operation codes are used to create an internal transaction.

- `itxn_begin` means the beginning of an internal transaction.
- `itxn_field` is used to set certain properties of the transaction.
- `itxn_next` moves to the next transaction in the same group as the previous one, and the transaction code
- `itxn_submit` is used to send a transaction or a group of transactions.

Example

The following is a typical payment transaction that transfers 5 Algo from the smart contract account to the account of the designated recipient.

```
# ...  
InnerTxnBuilder.Begin(),  
InnerTxnBuilder.SetFields(  
    {  
        TxnField.type_enum: TxnType.Payment,  
        TxnField.amount: Int(5000),  
        TxnField.receiver: Txn.sender(),  
    }  
)  
InnerTxnBuilder.Submit(),
```

The sender of the transaction by default is the smart contract account.

JavaScript AlgoSDK





Algorand SDKs

Algorand provides SDKs for various programming languages, allowing developers to choose the most convenient tool for their needs. The main Algorand SDKs include:

- JavaScript SDK
- Python SDK
- Java SDK
- Go SDK

JavaScript AlgoSDK is a library that provides a user-friendly interface for interacting with the Algorand blockchain. Using the AlgoSDK, developers can create, sign, and send transactions, work with smart contracts, create and manage assets, and perform many other operations

Installing and configuring the AlgoSDK

To start working with AlgoSDK, you need to install the library via npm (Node Package Manager):

```
bash
```

```
npm install algosdk
```

After installing the library, you can import it into the project:

```
javascript
```

```
const algosdk = require('algosdk');
```


Connecting to Algorand Node

To interact with the Algorand blockchain, you need to connect to a node. You can use both your own node and third-party services.

Connection to the Algorand Sandbox:

```
const algodToken = 'a'.repeat(64);
const algodServer = 'http://localhost';
const algodPort = 4001;

const algodClient = new algosdk.Algodv2(algodToken, algodServer, algodPort);
```

- **token:** Authentication token for accessing the sandbox node. The value "a" * 64 is usually used for local sandbox nodes.
- **server:** The address of the sandbox node. For a local sandbox, <http://localhost> is used.
- **port:** The port on which your sandbox node is running. For a local sandbox, port 4001 is usually used.

Create an account

To interact with the Algorand blockchain, you need to have a funded account. You can create a test account as follows:

```
const generatedAccount = algosdk.generateAccount();  
const passphrase = algosdk.secretKeyToMnemonic(generatedAccount.sk);  
console.log(`My address: ${generatedAccount.addr}`);  
console.log(`My passphrase: ${passphrase}`);
```

Check your account balance

```
const acctInfo = await algodClient.accountInformation(acct.addr).do();  
console.log(`Account balance: ${acctInfo.amount} microAlgos`);
```

Create a payment transaction

```
const suggestedParams = await algodClient.getTransactionParams().do();
const ptxn = algosdk.makePaymentTxnWithSuggestedParamsFromObject({
  from: acct.addr,
  suggestedParams,
  to: acct2.addr,
  amount: 10000,
  note: new Uint8Array(Buffer.from('hello world')),
});
```

- **getTransactionParams()**: this method calls the API to get the recommended parameters for a new transaction, such as the current minimum fee, the last block (first valid round), etc.
- **algosdk.makePaymentTxnWithSuggestedParamsFromObject**: this function creates a new payment transaction using the recommended parameters.
- **from**: The address of the sender of the transaction. This is the person who initiates the transaction.
- **suggestedParams**: The transaction parameters received from the previous step (suggestedParams).
- **to**: Address of the recipient of the transaction. This is the person who receives Algos.
- **amount**: The amount of Algos to be sent in the transaction. In this case, it is 10000 microAlgos (0.01 Algos).
- **note**: A note to the transaction that may contain additional information in byte format. In this case, it is 'hello world' converted to Uint8Array.

Signing a transaction

Before a transaction is considered valid, it must be signed with the sender's private key:

```
const signedTxn = ptxn.signTxn(acct.privateKey);
```

Sending a transaction

The signed transaction can now be sent to the network. `WaitForConfirmation` is called after the transaction is sent to wait for the transaction to be transferred to the Algorand blockchain and confirmed.

```
const { txId } = await algodClient.sendRawTransaction(signedTxn).do();
const result = await algosdk.waitForConfirmation(algodClient, txId, 4);
console.log(result);
console.log(`Transaction Information: ${result.txn}`);
console.log(`Decoded Note: ${Buffer.from(result.txn.txn.note).toString()}`);
```


Recover an account from a mnemonic phrase

```
const mnemonic =  
  'creek phrase island true then hope employ veteran rapid hurdle above liberty tissue connect alcohol timber idle ten frog bulb embody crunch taxi abstract month';  
const recoveredAccount = algosdk.mnemonicToSecretKey(mnemonic);  
console.log('Recovered mnemonic account: ', recoveredAccount.addr);
```

Defining a schema for global and local states

```
// define uint64s and byteslices stored in global/local storage
const numGlobalByteSlices = 1;
const numGlobalInts = 1;
const numLocalByteSlices = 1;
const numLocalInts = 1;
```

Determining the source TEAL from a string or file

```
const approvalProgram = fs.readFileSync(  
  path.join(__dirname, '/application/approval.teal'),  
  'utf8'  
);  
const clearProgram = fs.readFileSync(  
  path.join(__dirname, '/application/clear.teal'),  
  'utf8'  
);
```

Compiling the application

```
const approvalCompileResp = await algodClient
  .compile(Buffer.from(approvalProgram))
  .do();
```

```
const compiledApprovalProgram = new Uint8Array(
  Buffer.from(approvalCompileResp.result, 'base64')
);
```

```
const clearCompileResp = await algodClient
  .compile(Buffer.from(clearProgram))
  .do();
```

```
const compiledClearProgram = new Uint8Array(
  Buffer.from(clearCompileResp.result, 'base64')
);
```

Application creation transaction

Create a transaction with defined values, sign, send and confirm the transaction:

```
const appCreateTxn = algosdk.makeApplicationCreateTxnFromObject({
  from: creator.addr,
  approvalProgram: compiledApprovalProgram,
  clearProgram: compiledClearProgram,
  numGlobalByteSlices,
  numGlobalInts,
  numLocalByteSlices,
  numLocalInts,
  suggestedParams,
  onComplete: algosdk.OnApplicationComplete.NoOpOC,
});

// Sign and send
await algodClient
  .sendRawTransaction(appCreateTxn.signTxn(creator.privateKey))
  .do();
const result = await algosdk.waitForConfirmation(
  algodClient,
  appCreateTxn.txID().toString(),
  3
);
// Grab app id from confirmed transaction result
const appId = result['application-index'];
console.log(`Created app with index: ${appId}`);
```

Additional materials

- [Learn How to Manage Private / Public Algorand Account Keys \[Accounts Explained #2\]](#)
- [The 4 A's of Algorand](#): we will discuss the level 1 primitives of the Algorand blockchain. Developers will learn what the 4 A's of Algorand (Accounts, Assets, Atomic Transactions and Applications) are, how they are interconnected, and how they are built into reliable decentralised solutions.
- [Algorand JavaScript SDK Examples](#): The Algorand JavaScript SDK repository contains examples of how to use the Algorand JavaScript SDK.
- [Interact with smart contracts](#): Algorand's official documentation on how to interact with smart contracts using the AlgoSDK.