



Prinzipien von Blockchain-Systemen

Konzepte der Ethereum-Programmierung mit Solidity - Workshop

Michael Fröwis

Agenda

A. Wiederholung: Ethereum

B. Tools und Ressourcen

C. Smart Contract-Programmierung mit Solidity anhand von Beispielen

- Solidity Überblick
- Token-System und ERC20
- Kryptographischer Münzwurf

D. Hausaufgabe

E. (Wenn noch Zeit übrig) Interaktion, DAPPS, Fallstricke und Best Practices

UTXO-Modell versus Kontenmodell

Jargon: TX = Transaction, TXO = Transaction Output, UTXO = Unspent TXO

UTXO-Modell

- „Rechnet in Transaktionen“, Münzanalogie
- Benötigt Wechselgeld
- Outputs gliedern Transaktionen in kleinere, fest definierte Einheiten.
- Client-Datenhaltung optimiert zur Bestimmung der Neuheit von Outputs, jedoch keine Salden
- Wert ergibt sich aus Rückverfolgbarkeit jeder Einheit zur Coinbase-Transaktion.
- Nicht fungibel

bei Bitcoin im Einsatz

Kontenmodell

- „Rechnet in Konten“, Giralgeldanalogie
- Kein Wechselgeld nötig
- Transaktionen ähneln Nachrichten mit Empfängern und Parametern.
- Client schreibt Systemzustand fort und prüft Salden.
- Zähler zur Bestimmung der Neuheit.
- Wert ergibt sich aus Korrektheit aller Zustandsübergänge.
- Fungibilität möglich

bei Ethereum im Einsatz

Ethereum Virtual Machine (EVM)

Architektur

Jeder Ethereum-Knoten implementiert eine Turing-vollständige virtuelle Maschine mit

- stapelbasierter Architektur und
- Wortbreite 32 Byte (256 Bit).

Einbindung in die Transaktionslogik

Im Gegensatz zum UTXO-Modell von Bitcoin kann die EVM

- Zustandsübergänge im Rahmen der Nebenbedingungen frei definieren und
- den Zustandsraum erweitern.

Programmierung

Für die EVM existieren Hochsprachen: Solidity, LLL, Serpent (nicht empfohlen)

Warum wollen „Normalnutzer“ für die EVM programmieren [können]?

Kontentypen bei Ethereum

Ethereum unterscheidet zwei Typen von Konten (engl. *account*).

Parteien „besitzen“:

Externally Owned Account (EOA)

gesteuert durch Signierschlüssel

- Adresse
- Saldo (*balance*)
- Transaktionszähler (*nonce*)

Transaktionen referenzieren:

Code Account (CA)

gesteuert durch Programmlogik

- Adresse
- Bytecode
- Saldo (*balance*)
- Lokale Variablen (Zustand)
- Kind-CA-Zähler (*nonce*)

„Inter-Konten-Kommunikation“

Ethereum nutzt ein Nachrichtensystem zur Kommunikation zwischen Konten. Abhängig vom Kontotyp werden die Nachrichten unterschiedlich bezeichnet.

EOAs lösen aus:

Transaktionen

- Empfänger
- Betrag
- Daten
- Gebühreninformation
- Digitale Signatur



in der Blockchain gespeichert

CAs lösen aus:

Message Calls

- Empfänger
- Betrag
- Daten
- Gebühreninformation



deterministische Folge von Transaktionen

Lebenszyklus von Code Accounts

Erstellen

- Die EVM interpretiert die Daten von Transaktionen ohne Empfängerangabe als Bytecode, führt ihn aus und speichert die Ausgabe als neuen Code Account.
- Deployment-Konvention: (Initialisierungs-Code||Code||Parameter)

Aufrufen

- Über Transaktionen und Message Calls, Parameter werden im Datenfeld übergeben

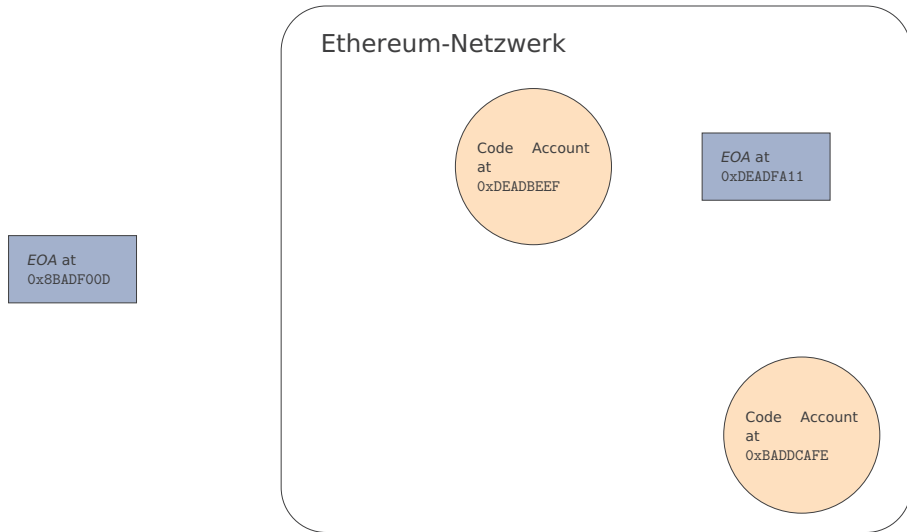
Aktualisieren

- Nicht vorgesehen → Workaround über Proxy-Pattern, **falls erwünscht!**

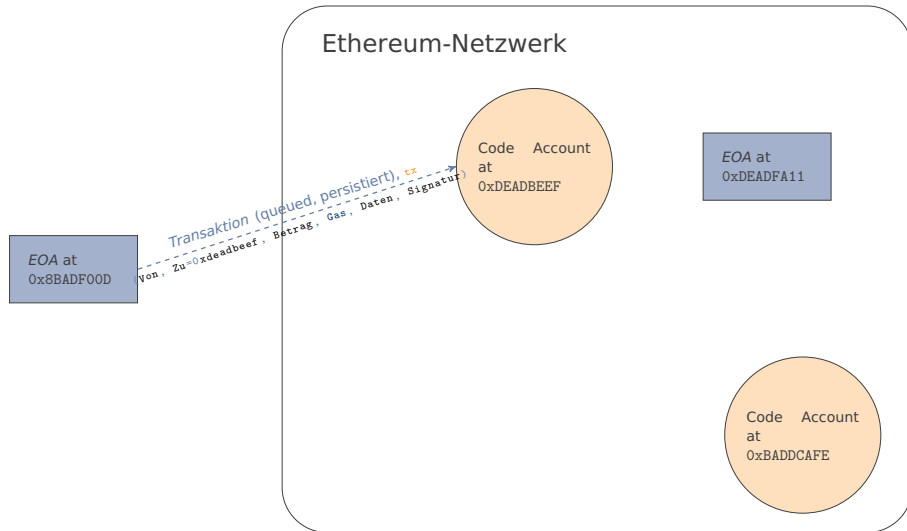
Löschen

- Nur über die EVM-Instruktion SELFDESTRUCT
- Danach erscheint die Adresse leer → Aufrufe geben TRUE ohne Seiteneffekt zurück!

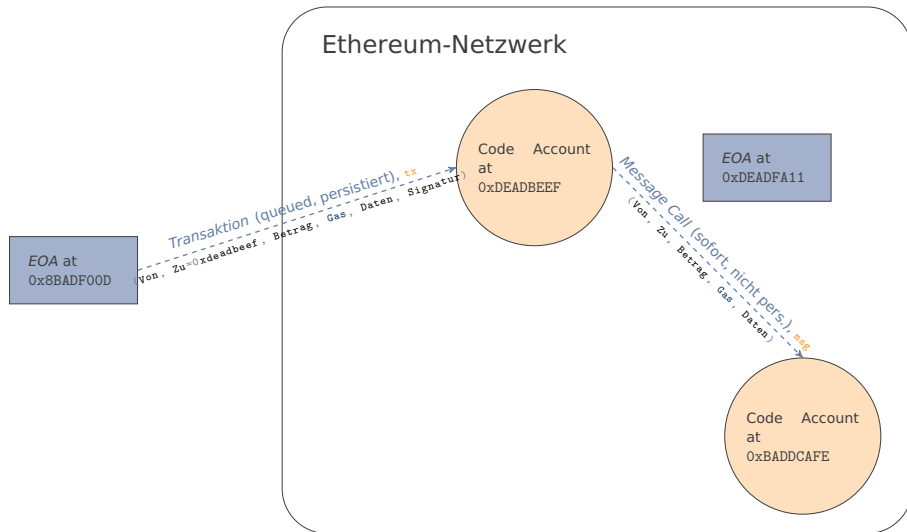
Ethereum Überblick – Entwicklersicht



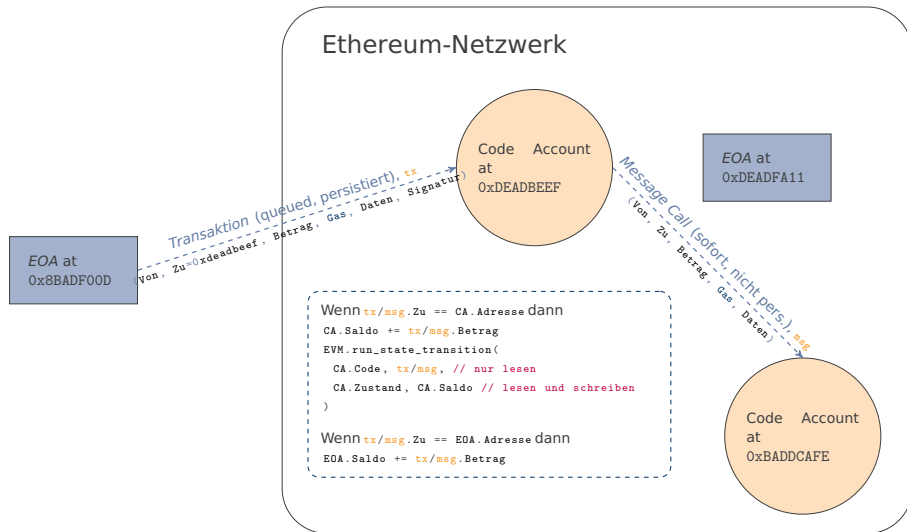
Ethereum Überblick – Entwicklersicht



Ethereum Überblick – Entwicklersicht



Ethereum Überblick – Entwicklersicht



Agenda

A. Wiederholung: Ethereum

B. Tools und Ressourcen

C. Smart Contract-Programmierung mit Solidity anhand von Beispielen

- Solidity Überblick
- Token-System und ERC20
- Kryptographischer Münzwurf

D. Hausaufgabe

E. (Wenn noch Zeit übrig) Interaktion, DAPPS, Fallstricke und Best Practices

Ressourcen

Workshop Git-Repository:

https://github.com/soad003/solidity_workshop

Solidity-Dokumentation:

[Solidity](#), [Ethereum](#), [Web3.js](#)

Design Patterns, Vorlagen und Beispiele:

[Open-Zeppelin](#)

Fragen and die Community:

[Solidity Gitter Chat](#)

Entwicklerwerkzeuge:

- [Remix IDE github](#), [Remix IDE online](#)
- Frameworks: [Truffle](#), [Embark](#)
- [Solidity Compiler](#)

Node-Implementierungen:

[Parity](#), [Go-Ethereum](#)

Agenda

A. Wiederholung: Ethereum

B. Tools und Ressourcen

C. Smart Contract-Programmierung mit Solidity anhand von Beispielen

- **Solidity Überblick**
- Token-System und ERC20
- Kryptographischer Münzwurf

D. Hausaufgabe

E. (Wenn noch Zeit übrig) Interaktion, DAPPS, Fallstricke und Best Practices

Solidity

- Populärste EVM-Sprache (*Ethereum Virtual Machine*)
- Stark und statisch typisiert
- Balance zwischen Gas-Effizienz und Abstraktionsgrad
- Syntaktisch verwandt mit Java, C++, JavaScript, ...
(*Kontrollstrukturen, Syntax, Abstraktionen etc.*)

Solidity

- Populärste EVM-Sprache (*Ethereum Virtual Machine*)
- Stark und statisch typisiert
- Balance zwischen Gas-Effizienz und Abstraktionsgrad
- Syntaktisch verwandt mit Java, C++, JavaScript, ...
(*Kontrollstrukturen, Syntax, Abstraktionen etc.*)
- **Analogien:**

Solidity

- Populärste EVM-Sprache (*Ethereum Virtual Machine*)
- Stark und statisch typisiert
- Balance zwischen Gas-Effizienz und Abstraktionsgrad
- Syntaktisch verwandt mit Java, C++, JavaScript, ...
(*Kontrollstrukturen, Syntax, Abstraktionen etc.*)

- **Analogien:**

Class Contract, definiert in Solidity

Instanz Code Account auf der Blockchain

RPC-Aufruf Transaktion/Message Call mit Code Account Empfänger
(*Datenfeld spezifiziert Funktion und Parameter*)

→ **Notiz:** die folgenden Codespiele wurden mit `solc v0.6.4` getestet.

Struktur eines Contracts

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.6.4;
contract Coin {
    address public minter;
    mapping (address => uint) public balances;

    event Sent(address indexed from, address indexed to, uint amount);

    constructor(address _minter) public { minter = _minter; }

    function mint(address receiver, uint amount) public {
        if (msg.sender != minter) return;
        balances[receiver] += amount;
    }

    function send_to(address receiver, uint amount) public returns (bool) {
        if (balances[msg.sender] < amount) return false;
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        Sent(msg.sender, receiver, amount);
        return true;
    }

    fallback() external { revert(); }
}
```

Strukturiertes Contracting

Version pragma:
verwende Compiler-Version

```
// Solidity Compiler Version
pragma solidity ^0.6.4;
contract Coin {
    address public minter;
    mapping (address => uint) public balances;

    event Sent(address indexed from, address indexed to, uint amount);

    constructor(address _minter) public { minter = _minter; }

    function mint(address receiver, uint amount) public {
        if (msg.sender != minter) return;
        balances[receiver] += amount;
    }

    function send_to(address receiver, uint amount) public returns (bool) {
        if (balances[msg.sender] < amount) return false;
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        Sent(msg.sender, receiver, amount);
        return true;
    }

    fallback() external { revert(); }
}
```

Struktur eines Contracts

Contract Definition

```
contract Coin {  
    address public minter;  
    mapping (address => uint) public balances;  
  
    event Sent(address indexed from, address indexed to, uint amount);  
  
    constructor(address _minter) public { minter = _minter; }  
  
    function mint(address receiver, uint amount) public {  
        if (msg.sender != minter) return;  
        balances[receiver] += amount;  
    }  
  
    function send_to(address receiver, uint amount) public returns (bool) {  
        if (balances[msg.sender] < amount) return false;  
        balances[msg.sender] -= amount;  
        balances[receiver] += amount;  
        Sent(msg.sender, receiver, amount);  
        return true;  
    }  
  
    fallback() external { revert(); }  
}
```

Struktur eines Contracts

Zustandsvariablen

Log-Definitionen: Events

Typ-Definitionen: Enums, Structs

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.4.18;

contract Minter {

    address public minter;
    mapping (address => uint) public balances;

    event Sent(address indexed from, address indexed to, uint amount);

    constructor(address _minter) public { minter = _minter; }

    function mint(address receiver, uint amount) public {
        if (msg.sender != minter) return;
        balances[receiver] += amount;
    }

    function send_to(address receiver, uint amount) public returns (bool) {
        if (balances[msg.sender] < amount) return false;
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        Sent(msg.sender, receiver, amount);
        return true;
    }

    fallback() external { revert(); }
}
```

Struktur eines Contracts

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.6.4;
contract Coin {
    address public minter;
    mapping (address => uint) public balances;

    event Sent(address sender, address receiver, uint amount);

    constructor(address _minter) public { minter = _minter; }

    function mint(address receiver, uint amount) public {
        if (msg.sender != minter) return;
        balances[receiver] += amount;
    }

    function send_to(address receiver, uint amount) public returns (bool) {
        if (balances[msg.sender] < amount) return false;
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        Sent(msg.sender, receiver, amount);
        return true;
    }

    fallback() external { revert(); }
}
```

Konstruktor:
wird beim Deployment ausgeführt

Struktur eines Contracts

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.6.4;
contract Coin {
    address public minter;
    mapping (address => uint) public balances;

    event Sent(address sender, address receiver, uint amount);

    constructor(address _minter) {
        minter = _minter;
    }

    function mint(address receiver, uint amount) public {
        if (msg.sender != minter) return;
        balances[receiver] += amount;
    }

    function send_to(address receiver, uint amount) public returns (bool) {
        if (balances[msg.sender] < amount) return false;
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        Sent(msg.sender, receiver, amount);
        return true;
    }

    fallback() external { revert(); }
}
```

*Funktionsdefinitionen:
selektiert anhand tx/msg.Daten*

Struktur eines Contracts

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.6.4;
contract Coin {
    address public minter;
    mapping (address => uint) public balances;

    event Sent(address indexed from, address indexed to, uint amount);

    constructor(address _minter) public { minter = _minter; }

    function mint(address receiver, uint amount) public {
        if (msg.sender != minter) return;
        balances[receiver] += amount;
    }

    function send_to(address receiver, uint amount) public returns (bool) {
        if (balances[msg.sender] < amount) return false;
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        Sent(msg.sender, receiver, amount);
        return true;
    }

    fallback() external { revert(); }
}
```

Fallback-Funktion:
Wenn keine Funktion in tx/msg
spezifiziert oder Funktion nicht
vorhanden

Umgebungsvariablen und vordefinierte Funktionen

Block: `block.blockhash(n)`, `block.coinbase`, `block.difficulty`,
`block.gaslimit`, `block.number`, `block.timestamp` = `now`

Umgebungsvariablen und vordefinierte Funktionen

Block: `block.blockhash(n), block.coinbase, block.difficulty,
block.gaslimit, block.number, block.timestamp = now`

Message: `msg.data, msg.sig, msg.gas, msg.sender, msg.value`

Umgebungsvariablen und vordefinierte Funktionen

Block: `block.blockhash(n)`, `block.coinbase`, `block.difficulty`,
`block.gaslimit`, `block.number`, `block.timestamp` = `now`

Message: `msg.data`, `msg.sig`, `msg.gas`, `msg.sender`, `msg.value`

Transaktion: `tx.gasprice`, `tx.origin`

Umgebungsvariablen und vordefinierte Funktionen

Block: `block.blockhash(n), block.coinbase, block.difficulty,
block.gaslimit, block.number, block.timestamp = now`

Message: `msg.data, msg.sig, msg.gas, msg.sender, msg.value`

Transaktion: `tx.gasprice, tx.origin`

Contract: `this, selfdestruct(<address>)`

Umgebungsvariablen und vordefinierte Funktionen

Block: `block.blockhash(n), block.coinbase, block.difficulty,
block.gaslimit, block.number, block.timestamp = now`

Message: `msg.data, msg.sig, msg.gas, msg.sender, msg.value`

Transaktion: `tx.gasprice, tx.origin`

Contract: `this, selfdestruct(<address>)`

Math: `addmod(x, y, k), mulmod(x, y, k), sha3(...) = keccak256(...)`

Umgebungsvariablen und vordefinierte Funktionen

Block: `block.blockhash(n), block.coinbase, block.difficulty,
block.gaslimit, block.number, block.timestamp = now`

Message: `msg.data, msg.sig, msg.gas, msg.sender, msg.value`

Transaktion: `tx.gasprice, tx.origin`

Contract: `this, selfdestruct(<address>)`

Math: `addmod(x, y, k), mulmod(x, y, k), sha3(...) = keccak256(...)`

Validierung/Fehler: `require(<condition>), assert(<condition>), revert()`

Umgebungsvariablen und vordefinierte Funktionen

Block: `block.blockhash(n)`, `block.coinbase`, `block.difficulty`,
`block.gaslimit`, `block.number`, `block.timestamp = now`

Message: `msg.data`, `msg.sig`, `msg.gas`, `msg.sender`, `msg.value`

Transaktion: `tx.gasprice`, `tx.origin`

Contract: `this`, `selfdestruct(<address>)`

Math: `addmod(x, y, k)`, `mulmod(x, y, k)`, `sha3(...)` = `keccak256(...)`

Validierung/Fehler: `require(<condition>)`, `assert(<condition>)`, `revert()`

Pseudo-Contracts: `ecrecover(hash, v, r, s)`, `ripemd160(...)`, `sha256(...)`

Funktionen und Modifier

Beispiel:

```
//function <ident>?(<param>*) <sichtbarkeit | annotationen | modifier> returns (<typ>*) {}

modifier only(address owner) {
    if(msg.sender == owner) _;      // _; wird mit Funktionsdefinition ersetzt
}

function withdraw() only(0xdeadbeef) { owner.transfer(this.balance); }
function withdraw(uint v) only(0xdeadbeef) { owner.transfer(this.balance) } //
    Ueberladen

function foo(uint bar, uint baz) public pure returns (uint bar_p, uint baz_p){
    return (bar / baz, bar % baz); // = bar_p = bar / baz; baz_p = bar % baz;
}
```

Funktions-Annotationen

Modifier payable oder modifier definiert im contract

Sichtbarkeit public, private, external, internal

Annotationen Forciert payable, nicht forciert pure, view, constant = view

Variablen, Datentypen und Parameterübergabe

Beispiel:

```
bool x = true && false;
uint y = 0xa;
bytes32 d = 0xdeadbeef;
uint z = y + address(x); // Fehler! Cast bool -> address nicht erlaubt ...
uint z = uint(d[0]); // Mit explizitem Cast
address a = address(z); // Cast uint -> address
```

Wertetypen

Boolean `bool`, Operatoren: `!`, `&&`, `||`, `==`, `!=`

Integer `uint8`, `uint16`, ..., `uint248`, `uint256` = `uint`, `int8`, ..., `int256` = `int`
Operatoren: `<=`, `<`, `==`, `!=`, `>`, `=>`, `&`, `|`, `^`, `~`, `+`, `-`, `*`, `/`, `%`, `**`, `<<`, `>>`

bytesN `byte` = `bytes1`, `bytes2`, ..., `bytes32`, fixed-size Array mit byte-weise Zugriff

Operatoren: `<=`, `<`, `==`, `!=`, `>`, `=>`, `&`, `|`, `^`, `~`, `<<`, `>>`, `[N]`

Member: `x.length`

Variablen, Datentypen und Parameterübergabe

Beispiel:

```
bool x = true && false;
uint y = 0xa;
bytes32 d = 0xdeadbeef;
uint z = y + address(x); // Fehler! Cast bool -> address nicht erlaubt ...
uint z = uint(d[0]); // Mit explizitem Cast
address a = address(z); // Cast uint -> address
```

Wertetypen

Boolean `bool`, Operatoren: `!`, `&&`, `||`, `==`, `!=`

Integer `uint8`, `uint16`, ..., `uint248`, `uint256` = `uint`, `int8`, ..., `int256` = `int`
Operatoren: `<=`, `<`, `==`, `!=`, `>`, `=>`, `&`, `|`, `^`, `~`, `+`, `-`, `*`, `/`, `%`, `**`, `<<`, `>>`

bytesN `byte` = `bytes1`, `bytes2`, ..., `bytes32`, fixed-size Array mit byte-weise Zugriff
Operatoren: `<=`, `<`, `==`, `!=`, `>`, `=>`, `&`, `|`, `^`, `~`, `<<`, `>>`, `[N]`
Member: `x.length`

→ Keine Unterstützung für Gleitkommazahlen, Festkomma-typ ab v0.5.0: `fixedMxN`

Variablen, Datentypen und Parameterübergabe

Beispiel:

```
contract BurnShareKeep {
    enum Action {share, keep, burn}
    Action public take = Action.share;
    address payable private constant shareit = address(0xdeadbeef);

    fallback() external payable {
        if(take == Action.share && msg.value > 1 ether)
            shareit.transfer(1000 finney);
        else if(take == Action.burn)
            address(0x0).transfer(msg.value);
    }
}
```

Wertetypen

Address `address`, 160-Bit Ethereum Adresse, Operatoren: `<=`, `<`, `==`, `!=`, `>`, `=>`

Member: `x.transfer`, `x.balance`, `x.send`, `x.call`, `x.callcode`, `x.delegatecall`

Enums `enum` User definierter Datentyp, intern Darstellung als `uint`

Variablen, Datentypen und Parameterübergabe

Beispiel:

```
contract BurnShareKeep {  
    enum Action {share, keep, burn}  
    Action public take = Action.share;  
    address payable private constant shareit = address(0xdeadbeef);  
  
    fallback() external payable {  
        if(take == Action.share && msg.value > 1 ether)  
            shareit.transfer(1000 finney);  
        else if(take == Action.burn)  
            address(0x0).transfer(msg.value);  
    }  
}
```

Wertetypen

Address `address`, 160-Bit Ethereum Adresse, Operatoren: `<=`, `<`, `==`, `!=`, `>`, `=>`

Member: `x.transfer`, `x.balance`, `x.send`, `x.call`, `x.callcode`, `x.delegatecall`

Enums `enum` User definierter Datentyp, intern Darstellung als `uint`

Variablen, Datentypen und Parameterübergabe

Beispiel:

Modifizierer für Zustandsvariablen:

```
contract BurnShare {
    enum Action {share, burn}
    Action public take = Action.share;
    address payable private constant shareit = address(0xdeadbeef);

    fallback() external payable {
        if(take == Action.share && msg.value > 1 ether)
            shareit.transfer(1000 finney);
        else if(take == Action.burn)
            address(0x0).transfer(msg.value);
    }
}
```

Wertetypen

Address `address`, 160-Bit Ethereum Adresse, Operatoren: `<=`, `<`, `==`, `!=`, `>`, `=>`

Member: `x.transfer`, `x.balance`, `x.send`, `x.call`, `x.callcode`, `x.delegatecall`

Enums `enum` User definierter Datentyp, intern Darstellung als `uint`

Variablen, Datentypen und Parameterübergabe

Beispiel:

```
contract BurnShareKeep {
    enum Action {share, keep, burn}
    Action public take = Action.share;
    address payable private constant shareit = address(0xdeadbeef);

    // ...

    (Von=this, Zu=0xdeadbeef, Betrag=10^18 wei, Gas=2300, Daten="")
    shareit.transfer(1000 finney);
    else if (take == Action.burn)
        address(0x0).transfer(msg.value);
}
}
```

Erstellt Message Call:

(Von=this, Zu=0xdeadbeef, Betrag=10^18 wei, Gas=2300, Daten="")

shareit.transfer(1000 finney);

else if (take == Action.burn)

address(0x0).transfer(msg.value);

}

}

Wertetypen

Address `address`, 160-Bit Ethereum Adresse, Operatoren: `<=`, `<`, `==`, `!=`, `>`, `=>`

Member: `x.transfer`, `x.balance`, `x.send`, `x.call`, `x.callcode`, `x.delegatecall`

Enums `enum` User definierter Datentyp, intern Darstellung als `uint`

Variablen, Datentypen und Parameterübergabe

Beispiel:

```
contract BurnShareKeep {
    enum Action {share, keep, burn}
    Action public take = Action.share;
    address payable private constant shareit = address(0xdeadbeef);

    fallback() external payable {
        if(take == Action.share && msg.value > 1 ether)
            address(0x0).transfer(msg.value);
    }
}
```

Transaktion an 0x0, „verbrennt“ Ether.

Wertetypen

Address `address`, 160-Bit Ethereum Adresse, Operatoren: `<=`, `<`, `==`, `!=`, `>`, `=>`

Member: `x.transfer`, `x.balance`, `x.send`, `x.call`, `x.callcode`, `x.delegatecall`

Enums `enum` User definierter Datentyp, intern Darstellung als `uint`

Variablen, Datentypen und Parameterübergabe

Beispiel:

```
contract BurnShareKeep {
    enum Action {share, keep, burn}
    Action public take = Action.share;
    address payable private constant shareit = address(0xdeadbeef);;

    fallback() external payable {
        if(take == Action.share && msg.value > 1 ether)
            shareit.transfer(1000 finney);
        else if(take == Action.burn)
            address(0x0).transfer(msg.value);
    }
}
```

Wertetypen

Address `address`, 160-Bit Ethereum Adresse, Operatoren: `<=`, `<`, `==`, `!=`, `>`, `=>`

Member: `x.transfer`, `x.balance`, `x.send`, `x.call`, `x.callcode`, `x.delegatecall`

Enums `enum` User definierter Datentyp, intern Darstellung als `uint`

→ `send`, `call`, `delegatecall` bieten mehr Kontrolle über *Message Call* als `transfer`.

Variablen, Datentypen und Parameterübergabe

Beispiel (Mappings und Strukturen):

```
contract Registry {
    enum State {active, reserved, free}
    struct Entry { State stat; address where; }
    mapping(string => Entry) r;

    function Lookup_address(string memory s) public view returns (address) {
        return r[s].where;
    }

    function Reg(string memory s, address na) public {
        r[s] = Entry(State.active, na);
    }
}
```

Referenztypen

Strukturen `struct`, zusammengesetzter Datentyp

Key/Value `mapping`, alle keys mit `_` initialisiert, nicht iterierbar, keine Member

Variablen, Datentypen und Parameterübergabe

Beispiel (Mappings und Strukturen):

```
contract Registry {  
    enum State {active, reserved, free;  
    struct Entry { State stat; address where; }  
    mapping(string => Entry) r;  
  
    function Lookup_address(string memory s) public view returns (address) {  
        return r[s].where;  
    }  
  
    function Reg(string memory s, address na) public {  
        r[s] = Entry(State.active, na);  
    }  
}
```

Struktur mit zwei Feldern

Referenztypen

Strukturen `struct`, zusammengesetzter Datentyp

Key/Value `mapping`, alle keys mit `_` initialisiert, nicht iterierbar, keine Member

Variablen, Datentypen und Parameterübergabe

Beispiel (Mappings und Strukturen):

```
contract Registry {
    enum State {active, reserved, free}
    struct Entry { State stat; address where; }
    mapping(string => Entry) r;

    function Lookup_address(string memory s) public view returns (address) {
        return r[s].where;
    }

    function Reg(string memory s, address na) public {
        r[s] = Entry(State.active, na);
    }
}
```

Referenztypen

Strukturen `struct`, zusammengesetzter Datentyp

Key/Value `mapping`, alle keys mit `_` initialisiert, nicht iterierbar, keine Member

Variablen, Datentypen und Parameterübergabe

Beispiel (Mappings und Strukturen):

```
contract Registry {
    enum State {active, reserved, free}
    struct Entry { State stat; address where; }
    mapping(string => Entry) r;

    function Lookup_address(string memory s) public view returns (address) {
        return r[s].where;
    }

    function Reg(string memory s, address na) public {
        r[s] = Entry(State.active, na);
    }
}
```

Referenztypen

Strukturen `struct`, zusammengesetzter Datentyp

Key/Value `mapping`, alle keys mit `_` initialisiert, nicht iterierbar, keine Member

Variablen, Datentypen und Parameterübergabe

Beispiel (Mappings und Strukturen):

```
contract Registry {
    enum State {active, reserved, free}
    struct Entry { State stat; address where; }
    mapping(string => Entry) r;

    function Lookup_address(string memory s) public view returns (address) {
        return r[s].where;
    }

    function Reg(string memory s, address na) public {
        r[s] = Entry(State.active, na);
    }
}
```

Referenztypen

Strukturen `struct`, zusammengesetzter Datentyp

Key/Value `mapping`, alle keys mit `_` initialisiert, nicht iterierbar, keine Member

Variablen, Datentypen und Parameterübergabe

Beispiel (Mappings und Strukturen):

```
contract Registry {
    enum State {active, reserved, free}
    struct Entry { State stat; address where; }
    mapping(string => Entry) r;

    function Lookup_address(string memory s) public view returns (address) {
        return r[s].where;
    }

    function Reg(string memory s, address na) public {
        r[s] = Entry(State.active, na);
    }
}
```

Referenztypen

Strukturen `struct`, zusammengesetzter Datentyp

Key/Value `mapping`, alle keys mit `_` initialisiert, nicht iterierbar, keine Member

Variablen, Datentypen und Parameterübergabe

Beispiel (Arrays):

```
contract Registry2 {
    address[] participants;

    function Enroll() payable public {
        if(msg.value > 1000 wei)
            addToArray(participants, msg.sender);
    }

    function addToArray(address[] storage arr, address item) internal {
        arr.push(item);
    }
}
```

Referenztypen

Arrays `bytes`, `string`, `<typ>[N]`

Member: `x.length`, `x.push`

Variablen, Datentypen und Parameterübergabe

Beispiel *Erstellt dynamisches storage Array.*

statisch: `address[8] participants`

```
contract MyStorage {
    address[] participants;

    function Enroll() payable public {
        if(msg.value > 1000 wei)
            addToArray(participants, msg.sender);
    }

    function addToArray(address[] storage arr, address item) internal {
        arr.push(item);
    }
}
```

Referenztypen

Arrays `bytes`, `string`, `<typ>[N]`

Member: `x.length`, `x.push`

Variablen, Datentypen und Parameterübergabe

Beispiel (Arrays):

```
contract Registry2 {  
    address[] participants;  
  
    function Enroll() payable  
        if(msg.value > 1000 wei,  
            addToArray(participants, msg.sender);  
}  
  
function addToArray(address[] storage arr, address item) internal {  
    arr.push(item);  
}  
}
```

Aufruf mit storage Referenz.
Keine Message Call, lokaler Aufruf!

Referenztypen

Arrays `bytes`, `string`, `<typ>[N]`

Member: `x.length`, `x.push`

Variablen, Datentypen und Parameterübergabe

Beispiel (Arrays):

```
contract Registry2 {  
    address[] participants;  
  
    function Enroll() payable public {  
        if(msg.value > 1000 wei)  
            addToArray(participants, msg.sender);  
    }  
}
```

Storage Qualifizierer um Kopie zu verhindern.

```
function addToArray(address[] storage arr, address item) internal {  
    arr.push(item);  
}
```

Referenztypen

Arrays `bytes`, `string`, `<typ>[N]`

Member: `x.length`, `x.push`

Variablen, Datentypen und Parameterübergabe

Beispiel (Arrays):

```
contract Registry2 {
    address[] participants;

    function Enroll() payable public {
        if(msg.value > 1000 wei)
            addToArray(participants, msg.sender);
    }

    function addToArray(address[] storage arr, address item) internal {
        arr.push(item);
    }
}
```

Referenztypen

Arrays `bytes`, `string`, `<typ>[N]`

Member: `x.length`, `x.push` (push nur bei storage Arrays!)

→ Storage und Memory Qualifizierer nur bei Referenztypen.

Fehlerbehandlung

Exceptions ...

- ... setzen Code Account-Zustand vor der *Transaktion*/dem *Message Call* zurück.

Fehlerbehandlung

Exceptions ...

- ... setzen Code Account-Zustand vor der *Transaktion*/dem *Message Call* zurück.
- ... können nur vom Aufrufer behandelt werden.

Fehlerbehandlung

Exceptions ...

- ... setzen Code Account-Zustand vor der *Transaktion*/dem *Message Call* zurück.
- ... können nur vom Aufrufer behandelt werden.
- ... können vor v0.6.0 nicht behandelt werden.
(*Ausnahme über low-level calls!*)

Fehlerbehandlung

Exceptions ...

- ... setzen Code Account-Zustand vor der *Transaktion*/dem *Message Call* zurück.
- ... können nur vom Aufrufer behandelt werden.
- ... können vor v0.6.0 nicht behandelt werden.
(Ausnahme über low-level calls!)
- ... können ab v0.6.0 mit `try/catch` behandelt werden.

Exception-Typen

assert für interne Fehler, Zustände die nie erreicht werden können/dürfen
(Exception braucht Gas auf)

```
assert(balances[msg.sender] >= 0)
```

require für Eingabevalidierung etc.
(Gas wird nicht verbraucht, Ab Byzantium HF, Okt 17)

```
require(msg.sender == owner), throw, revert()
```

Agenda

A. Wiederholung: Ethereum

B. Tools und Ressourcen

C. Smart Contract-Programmierung mit Solidity anhand von Beispielen

- Solidity Überblick
- **Token-System und ERC20**
- Kryptographischer Münzwurf

D. Hausaufgabe

E. (Wenn noch Zeit übrig) Interaktion, DAPPS, Fallstricke und Best Practices

Was ist ein Token-System ?

Token: Jargon für handelbares virtuelles Gut, fungibel oder nicht-fungibel.

Was ist ein Token-System ?

Token: Jargon für handelbares virtuelles Gut, fungibel oder nicht-fungibel.

Was ist ein Token-System ?

Token: Jargon für handelbares virtuelles Gut, fungibel oder nicht-fungibel.

Token-System: CA verwaltet Besitzverhältnisse und Überweisungen.

Was ist ein Token-System ?

Token: Jargon für handelbares virtuelles Gut, fungibel oder nicht-fungibel.

Token-System: CA verwaltet Besitzverhältnisse und Überweisungen.

Use-cases: Virtuelle Währungen, Crowdfunding, Anteilsscheine, Wahlstimmen . . .

Wie funktioniert ein Token-System ?

Alice (EOA)
Balance: 3 ETH

My Token (CA)

State

Balance:	0 ETH
Alice:	2
Bob:	0

Rules

```
IF   data = 'send';to;val
AND from ≥ val
THEN
    SUB val FROM from
    ADD val TO to
```

Wie funktioniert ein Token-System ?

Alice (EOA)
Balance: 3 ETH

My Token (CA)

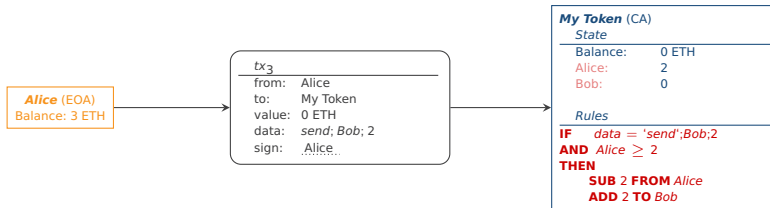
State

Balance:	0 ETH
Alice:	2
Bob:	0

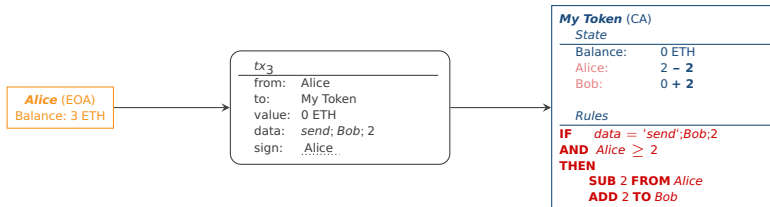
Rules

```
IF   data = 'send';to;val
AND from ≥ val
THEN
      SUB val FROM from
      ADD val TO to
```

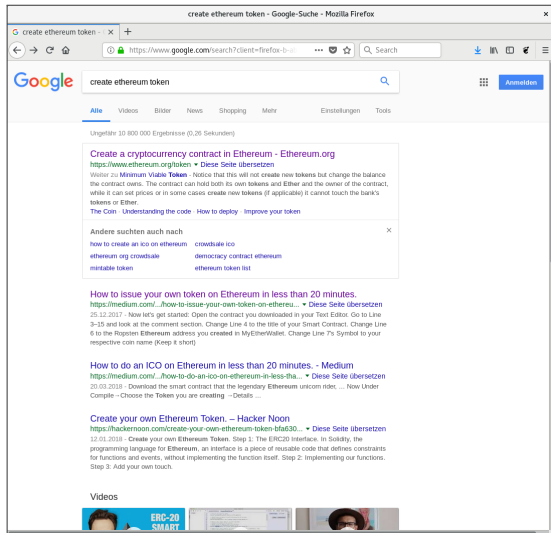
Wie funktioniert ein Token-System ?



Wie funktioniert ein Token-System ?



Wie erstellt man ein Token-System?



Wie erstellt man ein Token-System?

Create a cryptocurrency contract in Ethereum - Mozilla Firefox

create ethereum token

ETHereum - Create your own crypto-currency

The Coin

We are going to create a digital token. Tokens in the Ethereum ecosystem can represent any fungible tradable good: coins, loyalty points, gold certificates, IOUs, in-game items, etc. Since all tokens implement some basic features in a standard way, this also means that your token will be instantly compatible with the Ethereum wallet and any other client or contract that uses the same standards.

MINIMUM VIABLE TOKEN

The standard token contract can be quite complex. But in essence a very basic token boils down to this:

```
pragma solidity ^0.4.20;

contract MyToken {
    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;

    /* Initializes contract with initial supply tokens to the creator of the contract */
    function MyToken(
        uint256 initialSupply
    ) public {
        balanceOf[msg.sender] = initialSupply;          // Give the creator all initial tokens
    }

    /* Send coins */
    function transfer(address _to, uint256 _value) public returns (bool success) {
        require(balanceOf[msg.sender] >= _value);        // Check if the sender has enough
        require(balanceOf[_to] + _value >= balanceOf[_to]); // Check for overflows
        balanceOf[msg.sender] -= _value;                  // Subtract from the sender
        balanceOf[_to] += _value;                          // Add the same to the recipient
        return true;
    }
}
```

THE CODE

But if you just want to copy paste a more complete code, then use this:

OPEN CHAT

Wie erstellt man ein Token-System?

The screenshot shows a web browser window with the URL <https://www.ethereum.org/token>. The page is titled "Create a cryptocurrency contract in Ethereum - Mozilla Firefox". It features a "HOW TO DEPLOY" section with instructions on how to create a new contract. Below the text, there is a "SOLIDITY CONTRACT SOURCE CODE" section with a code editor showing a Solidity script for creating a token. The code includes comments and function definitions for initializing the contract, sending coins, and updating balances. To the right of the code, there is a "CONTRACT BYTE CODE" section and a "SELECT CONTRACT TO DEPLOY" dropdown menu. The dropdown menu is currently set to "MyToken". Below the dropdown, there are "CONSTRUCTOR PARAMETERS" fields for "_supply", "_name", "_symbol", and "_decimals". The "_supply" field is set to "10000", "_name" is "My DAO Shares", "_symbol" is "%", and "_decimals" is "2". At the bottom right of the page, there is a green "OPEN CHAT" button.

HOW TO DEPLOY

If you aren't there already, open the Ethereum Wallet, go to the contracts tab and then click "deploy new contract".

Now get the token source from above and paste it into the "Solidity source field". If the code compiles without any error, you should see a "pick a contract" drop-down list on the right. Get it and select the "MyToken" contract. On the right column, you'll see all the parameters you need to personalize your own token. You can tweak them as you please, but for the purpose of this tutorial we recommend you to pick these parameters: 10,000 as the supply, any name you want, "%" for a symbol and 2 decimal places. Your app should be looking like this:

SOLIDITY CONTRACT SOURCE CODE

```
47 event Transfer(address indexed from, address indexed to, uint256 value);
48
49 /* Initializes contract with initial supply tokens to the creator of the
50 contract */
51 /* If supply not given then generate 1 million of the smallest unit (
52 1 ether) */
53
54 /* Unless you add other functions these variables will never change */
55
56 name = _name;
57 symbol = _symbol;
58
59 /* If you want a divisible token then add the amount of decimals the
60 token will have */
61 decimals = _decimals;
62
63 /* Send coins */
64 function transfer(address _to, uint256 _value) {
65     /* If the sender doesn't have enough balance then stop */
66     if (balanceOf[msg.sender] < _value) throw;
67     if (balanceOf[_to] + _value > balanceOf[_to]); throw;
68
69     /* Add and subtract new balances */
70     balanceOf[msg.sender] -= _value;
71     balanceOf[_to] += _value;
72
73     /* Notify anyone listening that this transfer took place */
74     Transfer(msg.sender, _to, _value);
75 }
76 }
```

CONTRACT BYTE CODE

SELECT CONTRACT TO DEPLOY

MyToken

CONSTRUCTOR PARAMETERS

_supply - 256 bits unsigned integer
10000

_name - String
My DAO Shares

_symbol - String
%

_decimals - 8 bits unsigned integer
2

OPEN CHAT

Wie erstellt man ein Token-System?

create ethereum token

Ethereum Project

Devcon 4 Takes place between Oct 30th and Nov 2nd, 2018.

The project was bootstrapped via an ether presale in August 2014 by fans all around the world. It is developed by the Ethereum Foundation, a Swiss non-profit, with contributions from great minds across the globe.

Smart money, smart wallet

The Ethereum Wallet is a gateway to decentralized applications on the Ethereum blockchain. It allows you to hold and secure ether and other crypto-assets built on Ethereum, as well as write, deploy and use smart contracts.

DOWNLOAD
Ethereum Wallet for Linux

See all versions

Easy template-based contract creation

Learn **Solidity**, a new language for smart

Wie erstellt man ein Token-System?

create ethereum to create a cryptocurrency

HOW TO DEPLOY

We are going to create a digital token. Tokens in the Ethereum ecosystem can represent any fungible tradable good: coins, loyalty points, gold certificates, IOUs, in-game items, etc. Since all tokens implement some basic features in a standard way, this also means that your token will be instantly compatible with the Ethereum wallet and any other client or contract that uses the same standards.

MINIMUM VIABLE TOKEN

The standard token contract can be quite complex. But in essence a very basic token boils down to this:

```
pragma solidity ^0.4.20;

contract MyToken {
    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;

    /* Initializes contract with initial supply tokens to the creator of the contract */
    function MyToken(uint256 initialSupply) public {
        uint256 initialSupply;
        balanceOf[msg.sender] = initialSupply; // Give the creator all initial tokens
    }

    /* Send coins */
    function transfer(address _to, uint256 _value) public returns (bool success) {
        require(balanceOf[msg.sender] >= _value); // Check if the sender has enough
        require(balanceOf[_to] + _value >= balanceOf[_to]); // Check for overflows
        balanceOf[msg.sender] -= _value; // Subtract from the sender
        balanceOf[_to] += _value; // Add the same to the recipient
        return true;
    }
}
```

THE CODE

But if you just want to copy paste a more complete code, then use this:

OPEN CHAT

Wie erstellt man ein Token-System?

The image is a composite of two screenshots. The left screenshot shows a web browser with multiple tabs open, displaying a tutorial titled 'HOW TO DEPLOY' for creating an Ethereum token. The tutorial includes instructions on how to create a token, how to issue tokens, and how to do an ERC-20 transfer. The right screenshot shows the Ethereum Wallet interface. The top bar displays 'Ethereum Wallet' and 'BALANCE 1.00 ETH'. The main area is titled 'Deploy contract' and shows the 'FROM' field set to 'Account 1 - 1.00 ETH'. The 'AMOUNT' field is set to '0.0' and the 'SEND EVERYTHING' checkbox is checked. The 'SOLIDITY CONTRACT SOURCE CODE' tab is active, showing the following code:

```
1 pragma solidity ^0.4.20;
2
3 contract MyToken {
4     /* This creates an array with all balances */
5     mapping (address => uint256) public balanceOf;
6
7     /* Initializes contract with initial supply tokens to the creator of the contract */
8     constructor()
9     {
10         uint256 initialSupply = 10000;
11         balanceOf[msg.sender] = initialSupply; // Give the creator all initial tokens
12     }
13
14     /* Send coins */
15     function transfer(address _to, uint256 _value) public returns (bool success)
16     {
17         require(balanceOf[msg.sender] >= _value); // Check if the sender has enough
18         balanceOf[msg.sender] -= _value; // Subtract from the sender
19         balanceOf[_to] += _value; // Add the same to the recipient
20         return true;
21     }
22 }
23
24
```

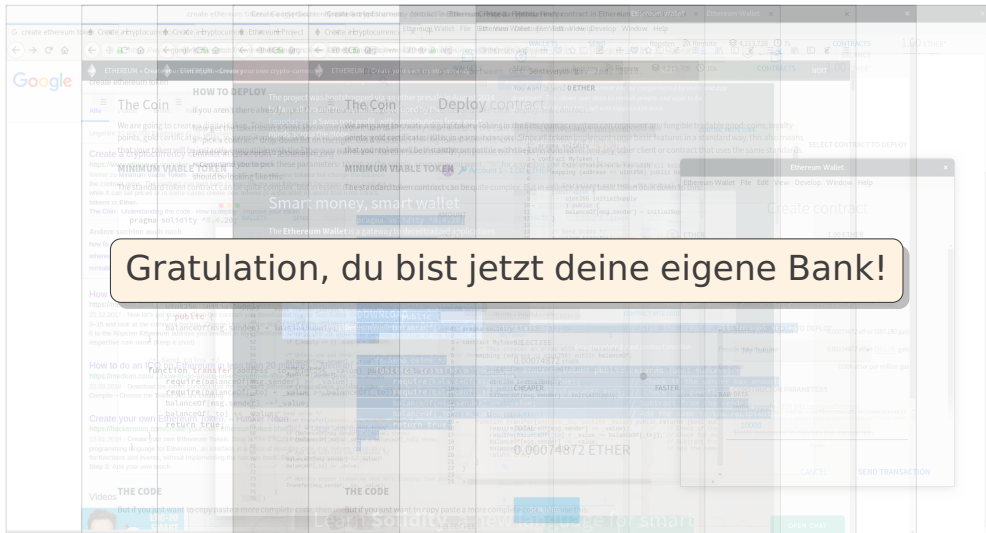
The 'CONTRACT BYTE CODE' tab is also visible, showing the compiled bytecode. The 'SELECT CONTRACT TO DEPLOY' dropdown is set to 'My Token'. The 'CONSTRUCTOR PARAMETERS' field is set to '10000'.

Wie erstellt man ein Token-System?

The image shows a composite of three elements related to creating an Ethereum token system:

- Background:** A web browser window displaying a tutorial titled "HOW TO DEPLOY" for "The Coin" project. The tutorial explains how to create a token on the Ethereum blockchain, mentioning the need to pick a contract and drop it on the network. It also shows a "Smart money, smart wallet" section with Solidity code snippets.
- Top Right:** The "Ethereum Wallet" interface. It shows the "WALLETS" tab with a balance of 4,213.728 ETH. The "CONTRACTS" tab is active, showing a balance of 1.00 ETH. A message says "You want to send 0.00074872 ETH".
- Bottom Right:** A "Create contract" dialog box. It shows a transaction of 0.00074872 ETH. The "Estimated fee consumption" is 0.00074872 ether (187,180 gas). The "Provide maximum fee" is 0.00114872 ether (287,140 gas). The "Gas price" is 0.004 ether per million gas. The "RAW DATA" section shows a long hexadecimal string. The "SEND TRANSACTION" button is highlighted.

Wie erstellt man ein Token-System?



Token Implementierung mit Remix

Agenda

A. Wiederholung: Ethereum

B. Tools und Ressourcen

C. Smart Contract-Programmierung mit Solidity anhand von Beispielen

- Solidity Überblick
- Token-System und ERC20
- **Kryptographischer Münzwurf**

D. Hausaufgabe

E. (Wenn noch Zeit übrig) Interaktion, DAPPS, Fallstricke und Best Practices

Beispiel

Alice und Bob schließen einen Vertrag:

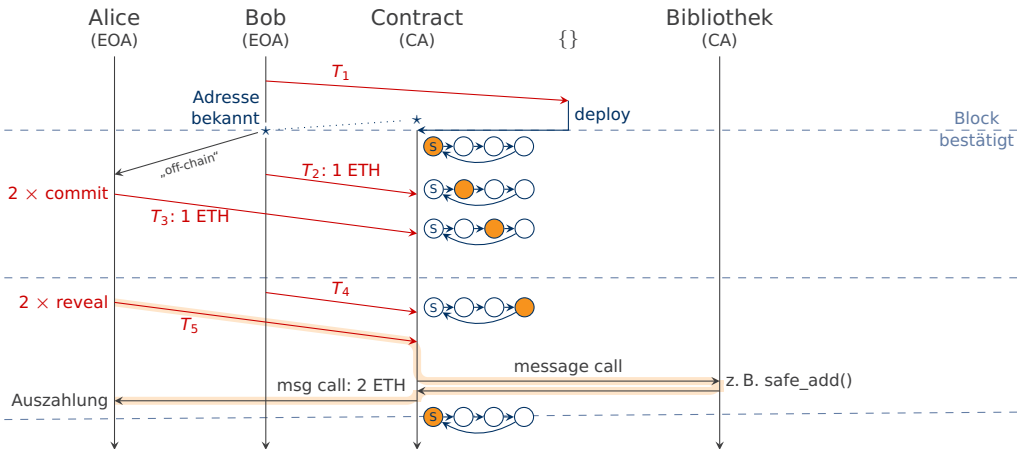
„Wir werfen die Münze. Der Gewinner bekommt vom Verlierer ein Geldstück.“

Bemerkungen

- Juristisch liegt ein Vertrag vor, sobald zwei Willenserklärungen abgegeben wurden. Ob dies nachweisbar oder durchsetzbar ist, spielt zunächst keine Rolle.
- Wir kennen eine Methode (kryptographische Commitments), um den Münzwurf gerecht und ohne dritte Partei über ein Rechnernetz durchzuführen.
- **Das reicht aber nicht aus um sicherzustellen, dass der Verlierer tatsächlich zahlt.**
- Dieser Vertrag lässt sich in einem Blockchain-System abbilden, sodass die Zahlung (in Einheiten eines virtuellen Guts) nach dem Wurf unabwendbar ist.

Beispiel

Alice und Bob möchten durchsetzbar eine Münze werfen.



Münzwurf Implementierung mit Remix

Agenda

A. Wiederholung: Ethereum

B. Tools und Ressourcen

C. Smart Contract-Programmierung mit Solidity anhand von Beispielen

- Solidity Überblick
- Token-System und ERC20
- Kryptographischer Münzwurf

D. Hausaufgabe

E. (Wenn noch Zeit übrig) Interaktion, DAPPS, Fallstricke und Best Practices

Demo

Hausaufgabe

Fragen?

Bitte nicht zögern.

Fragen?

Bitte nicht zögern.

→ Wer sich mehr mit dem Thema befassen will: Wir bieten Bachelorarbeiten an.

Agenda

A. Wiederholung: Ethereum

B. Tools und Ressourcen

C. Smart Contract-Programmierung mit Solidity anhand von Beispielen

- Solidity Überblick
- Token-System und ERC20
- Kryptographischer Münzwurf

D. Hausaufgabe

E. (Wenn noch Zeit übrig) Interaktion, DAPPS, Fallstricke und Best Practices

Appendix

Interaktion, DAPPS, Fallstricke

Interaktion – Kind-CAs

Beispiel:

```
import "./Coin.sol";
import "./Base.sol";

contract CoinFactory is Destructible, Payed{

    function CoinFactory(address _owner) public Ownable(_owner){}

    function CreateCoin() public payable fee(1 ether) returns (address) {
        return new Coin(msg.sender);
    }
}
```

Anmerkung

Gesamter Code von `Coin` wird in `CoinFactory` Code Account gespeichert. Potenziell hohe Gas-Kosten.

Interaktion – Kind-CAs

Beispiel:

```
import "./Coin.sol";
import "./Base.sol";

contract CoinFactory is Destructible, Payed{

    fun new erstellt Message Call
    (Von=this, Zu=null, ..., Daten = <Code of Coin> | owner)ress) {
        return new Coin(msg.sender);
    }
}
```

Anmerkung

Gesamter Code von `Coin` wird in `CoinFactory` Code Account gespeichert. Potenziell hohe Gas-Kosten.

Interaktion – Kind-CAs

Beispiel:

```
import "./Coin.sol";
import "./Base.sol";

contract CoinFactory is Destructible, Payed{

    function CoinFactory(address _owner) public Ownable(_owner){}

    function CreateCoin() public payable fee(1 ether) returns (address) {
        return new Coin(msg.sender);
    }
}
```

Anmerkung

Gesamter Code von `Coin` wird in `CoinFactory` Code Account gespeichert. Potenziell hohe Gas-Kosten.

Interaktion – Andere CAs aufrufen

Beispiel (Schlecht: low-level calls/assembly):

```
import "./Base.sol";
contract CoinWalletBad is Payed {
    mapping(address => address[]) coins;

    function CoinWalletBad(address _owner) public Ownable(_owner){}

    function RegCoin(address _new) fee(1 ether) payable public {
        bytes4 sig = bytes4(keccak256("minter()"));
        bool ok = false;
        address minter;
        assembly{
            let g := and(gas, 0xEFFFFFFF)
            let inout := mload(0x40)
            mstore(inout, sig)
            ok := call(g, _new, 0, inout, 4, inout, 32) // _new.call(bytes4(keccak256("minter()")));
            minter := mload(inout)
        }
        if(!ok) revert();
        address[] storage owners_coins = coins[minter];
        owners_coins.push(_new);
    }

    function Mint_all(address to, uint val) public {
        address[] storage owners_coins = coins[msg.sender];
        for(uint i=0; i < owners_coins.length; i++) {
            var ok = owners_coins[i].call.gas(30000) (bytes4(keccak256("mint(address,uint256)")), to, val);
            if(!ok) revert();
        }
    }
}
```

Interaktion – Andere CAs aufrufen

Beispiel (Schlecht: low-level calls/assembly):

```
import "./Base.sol";
contract CoinWalletBad is Payed {
    mapping(address => address[]) coins;

    function CoinWalletBad(address _owner) payable public {
        _owner = msg.sender;
    }

    function RegCoin(address _new) fee(1 ether) payable public {
        bytes4 sig = bytes4(keccak256("minter()"));
        bool ok = false;
        address minter;
        assembly{
            let g := and(gas, 0xEFFFFFFF)
            let inout := mload(0x40)
            mstore(inout, sig)
            ok := call(g, _new, 0, inout, 4, inout, 32) // _new.call(bytes4(keccak256("minter()")));
            minter := mload(inout)
        }
        if(!ok) revert();
        address[] storage owners_coins = coins[minter];
        owners_coins.push(_new);
    }

    function Mint_all(address to, uint val) public {
        address[] storage owners_coins = coins[msg.sender];
        for(uint i=0; i < owners_coins.length; i++) {
            var ok = owners_coins[i].call.gas(30000) (bytes4(keccak256("mint(address,uint256)")), to, val);
            if(!ok) revert();
        }
    }
}
```

Coins per Minter gruppieren und registrieren.

Interaktion – Andere CAs aufrufen

Beispiel (Schlecht: low-level calls/assembly):

```
import "./Base.sol";
contract CoinWalletBad is Payed {
    mapping(address => address[]) coins;

    function ...
    function ...
    function ...
    bytes ...
    bool ...
    address ...
    assembly {
        let g := and(gas, 0xEFFFFFFF)
        let inout := mload(0x40)
        mstore(inout, sig)
        ok := call(g, _new, 0, inout, 4, inout, 32) // _new.call(bytes4(keccak256("minter()")));
        minter := mload(inout)
    }
    if(!ok) revert();
    address[] storage owners_coins = coins[minter];
    owners_coins.push(_new);
}
function Mint_all(address to, uint val) public {
    address[] storage owners_coins = coins[msg.sender];
    for(uint i=0; i < owners_coins.length; i++) {
        var ok = owners_coins[i].call.gas(30000) (bytes4(keccak256("mint(address,uint256)")), to, val);
        if(!ok) revert();
    }
}
}
```

Lesen von low-level call
Rückgabewerten benötigt Assembly

Interaktion – Andere CAs aufrufen

Beispiel (Schlecht: low-level calls/assembly):

```
import "./Base.sol";
contract CoinWalletBad is Payed {
    mapping(address => address[]) coins;

    function CoinWalletBad(address _owner) public Ownable(_owner){}

    function RegCoin(address _new) fee(1 ether) payable public {
        bytes4 sig = bytes4(keccak256("minter()"));
        bool ok = false;
        address minter;
        assembly{
            let g := and(gas, 0xEFFFFFFF)
            let inout := mload(0x40)
            mstore(inout, sig)
            ok := call(g, _new, 0, inout, 4, inout, 32) // _new.call(bytes4(keccak256("minter()")));
            minter := mload(inout)
        }
        if(!ok) revert();
        address[] storage owners_coins = coins[minter];
        owners_coins.push(_new);
    }

    function Mint_all(address to, uint val) public {
        address[] storage owners_coins = coins[msg.sender];
        for(uint i=0; i < owners_coins.length; i++) {
            var ok = owners_coins[i].call.gas(30000) (bytes4(keccak256("mint(address,uint256)")), to, val);
            if(!ok) revert();
        }
    }
}
```

Interaktion – Andere CAs aufrufen

Beispiel (Schlecht: low-level calls/assembly):

```
import "./Base.sol";
contract CoinWalletBad is Payed {
    mapping(address => address[]) coins;

    function CoinWalletBad(address _owner) public Ownable(_owner){}

    function RegCoin(address _new) fee(1 ether) payable public {
        bytes4 sig = bytes4(keccak256("minter()"));
        bool ok = false;
        address minter;
        assembly{
            let g := and(gas, 0xEFFFFFFF)
            let inout := mload(0x40)
            mstore(inout, sig)
            ok := call(g, _new, 0, inout, 0x40, 0)
            minter := mload(inout)
        }
        if(!ok) revert();
        address[] storage owners_coins;
        owners_coins.push(_new);
    }

    function Mint_all(address _to, uint val) payable public {
        address[] storage owners_coins;
        for(uint i=0; i < owners_coins.length; i++) {
            var ok = owners_coins[i].call.gas(30000) (bytes4(keccak256("mint(address,uint256)")), to, val);
            if(!ok) revert();
        }
    }
}
```

Bietet volle Kontrolle über den
erstellten *Message Call* und
Fehlerbehandlung!
Ab v0.6.0 Interface und try/catch
verwenden!

Interaktion – Andere CAs aufrufen

Beispiel (Schlecht: low-level calls/assembly):

```
import "./Base.sol";
contract CoinWalletBad is Payed {
    mapping(address => address[]) coins;

    function CoinWalletBad(address _owner) public Ownable(_owner){}

    function RegCoin(address _new) fee(1 ether) payable public {
        bytes4 sig = bytes4(keccak256("minter()"));
        bool ok = false;
        address minter;
        assembly{
            let g := and(gas, 0xEFFFFFFF)
            let inout := mload(0x40)
            mstore(inout, sig)
            ok := call(g, _new, 0, inout, 4, inout, 32) // _new.call(bytes4(keccak256("minter()")));
            minter := mload(inout)
        }
        if(!ok) revert();
        address[] storage owners_coins = coins[minter];
        owners_coins.push(_new);
    }

    function Mint_all(address to, uint val) public {
        address[] storage owners_coins = coins[msg.sender];
        for(uint i=0; i < owners_coins.length; i++) {
            var ok = owners_coins[i].call.gas(30000) (bytes4(keccak256("mint(address,uint256)")), to, val);
            if(!ok) revert();
        }
    }
}
```

Interaktion – Andere CAs aufrufen

Beispiel (besser: Interfaces):

```
import "./Coin.sol";
import "./Base.sol";
contract CoinWallet is Payed {
    mapping(address => Coin[]) coins;
    function CoinWallet(address _owner) public Ownable(_owner){}

    function RegCoin(Coin _new) fee(1 ether) payable public {
        Coin[] storage owners_coins = coins[_new.minter()];
        owners_coins.push(_new);
    }
    function Mint_all(address to, uint val) public {
        Coin[] storage owners_coins = coins[msg.sender];
        for(uint i=0; i < owners_coins.length; i++) {
            owners_coins[i].mint. gas(30000)(to, val);
        }
    }
}
```

Anmerkungen

Einschränkung Interface: Vor v0.6.0 kann auf Fehler im Callee nicht reagiert werden!

→ Keine Typprüfung zur Laufzeit! Nur Typprüfung gegen Interface. Ab Solidity 0.4.0 wird geprüft, ob Adresse Code enthält.

Interaktion – Andere CAs aufrufen

Beispiel (besser: Interfaces):

```
import "./Coin.sol";
import "./Base.sol";
contract CoinWallet is Payed {
    mapping(address => Coin[]) coins;
    function CoinWallet(address _owner)

    function RegCoin(Coin _new) fee(1 ether, payable _fee) public {
        Coin[] storage owners_coins = coins[_new.minter()];
        owners_coins.push(_new);
    }
    function Mint_all(address to, uint val) public {
        Coin[] storage owners_coins = coins[msg.sender];
        for(uint i=0; i < owners_coins.length; i++) {
            owners_coins[i].mint. gas(30000)(to, val);
        }
    }
}
```

Compiler verwendet
Interface-Definition um *Message Call*
zu erstellen und den Rückgabewert
zu „parsen“.

Anmerkungen

Einschränkung Interface: Vor v0.6.0 kann auf Fehler im Callee nicht reagiert werden!

→ Keine Typprüfung zur Laufzeit! Nur Typprüfung gegen Interface. Ab Solidity 0.4.0 wird geprüft, ob Adresse Code enthält.

Interaktion – Andere CAs aufrufen

Beispiel (besser: Interfaces):

```
import "./Coin.sol";
import "./Base.sol";
contract CoinWallet is Payed {
    mapping(address => Coin[]) coins;
    function CoinWallet(address _owner) public Ownable(_owner){}

    function RegCoin(Coin _new) fee(1 ether) payable public {
        Coin[] storage owners_coins = coins[_new.minter()];
        owners_coins.push(_new);
    }
    function Minter(Coin _new) fee(1 ether) payable public {
        Coin[] storage owners_coins = coins[_new.minter()];
        for(uint i = 0; i < owners_coins.length; i++) {
            owners_coins[i].mint. gas(30000)(to, val);
        }
    }
}
```

Gas und Betrag können wie bei low-level calls gesteuert werden.

Anmerkungen

Einschränkung Interface: Vor v0.6.0 kann auf Fehler im Callee nicht reagiert werden!

→ Keine Typprüfung zur Laufzeit! Nur Typprüfung gegen Interface. Ab Solidity 0.4.0 wird geprüft, ob Adresse Code enthält.

Interaktion – Andere CAs aufrufen

Beispiel (besser: Interfaces):

```
import "./Coin.sol";
import "./Base.sol";
contract CoinWallet is Payed {
    mapping(address => Coin[]) coins;
    function CoinWallet(address _owner) public Ownable(_owner){}

    function RegCoin(Coin _new) fee(1 ether) payable public {
        Coin[] storage owners_coins = coins[_new.minter()];
        owners_coins.push(_new);
    }
    function Mint_all(address to, uint val) public {
        Coin[] storage owners_coins = coins[msg.sender];
        for(uint i=0; i < owners_coins.length; i++) {
            owners_coins[i].mint. gas(30000)(to, val);
        }
    }
}
```

Anmerkungen

Einschränkung Interface: Vor v0.6.0 kann auf Fehler im Callee nicht reagiert werden!

→ Keine Typprüfung zur Laufzeit! Nur Typprüfung gegen Interface. Ab Solidity 0.4.0 wird geprüft, ob Adresse Code enthält.

Decentralized Application (W)

DAPP = Benutzungsschnittstelle + Code Account + Wallet

Decentralized Application (W)

DAPP = Benutzungsschnittstelle + Code Account + Wallet

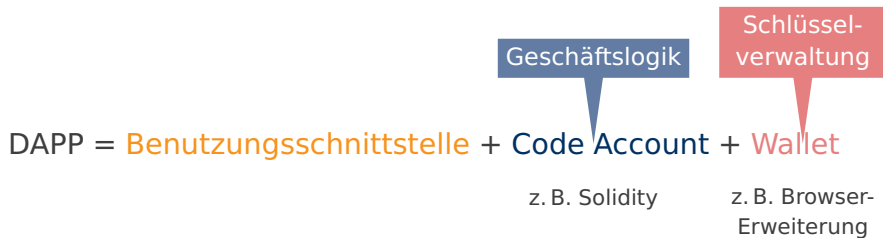
z. B. Solidity

The diagram illustrates the components of a Decentralized Application (DAPP). It is represented as a sum of three parts: 'Benutzungsschnittstelle' (User Interface), 'Code Account', and 'Wallet'. A blue callout box labeled 'Geschäftslogik' (Business Logic) points to the 'Code Account' component. Below the equation, it is noted that 'z. B. Solidity' (e.g., Solidity) is used for the code.

Vertrauensanker

- Der Code Account regelt die Sicherheitsinteressen aller.

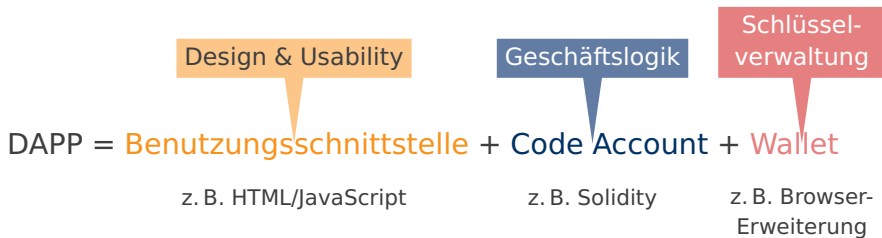
Decentralized Application (W)



Vertrauensanker

- Der Code Account regelt die Sicherheitsinteressen aller.
- Die Sicherheit des Wallets ist notwendig für die eigene Sicherheit.

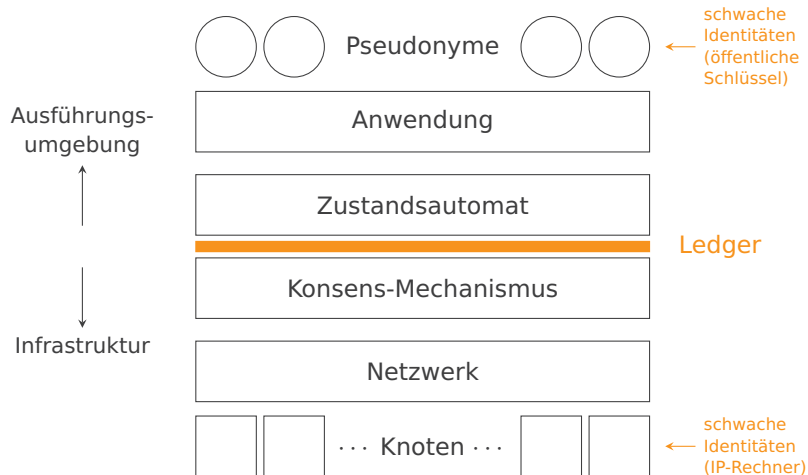
Decentralized Application (W)



Vertrauensanker

- Der Code Account regelt die Sicherheitsinteressen aller.
- Die Sicherheit des Wallets ist notwendig für die eigene Sicherheit.
- Die Benutzungsschnittstelle definiert die eigene Sicht auf die Interaktion. Sicherheitsrelevante Entscheidungen bedürfen korrekter Information.

Architektur für DAPPs auf Ethereum (W)



Interaktion mit der Außenwelt

Beispiel (Code Account):

```
import "./Base.sol";
contract GotCoins is Payed {
    uint public totalReceived = 0;

    event GotNewCoins(address indexed lastHop, address indexed signer, uint
        howMuch);

    function GotCoins(address _owner) public Ownable(_owner){}

    function() payable public {
        if(msg.value > 0) {
            emit GotNewCoins(msg.sender, tx.origin, msg.value);
            totalReceived += msg.value;
        }
    }
}
```

Anmerkungen

Event = Log, Handler für Events können im Node registriert werden. Über Web3.js auch in Webapps möglich.

Achtung: Web3.js in ständiger Entwicklung, Interface ändert sich!

Interaktion mit der Außenwelt

Beispiel (Code Account):

```
import "./Base.sol";  
contract GotCoins is Payed  
    uint public totalReceived
```

indexed
Nodes erstellen Index für dieses Feld.
Effizienter Zugriff auf historische
Logeinträge.

```
event GotNewCoins(address indexed lastHop, address indexed signer, uint  
    howMuch);
```

```
function GotCoins(address _owner) public Ownable(_owner){}
```

```
function() payable public {  
    if(msg.value > 0) {  
        emit GotNewCoins(msg.sender, tx.origin, msg.value);  
        totalReceived += msg.value;  
    }  
}
```

Anmerkungen

Event = Log, Handler für Events können im Node registriert werden. Über Web3.js auch in Webapps möglich.

Achtung: Web3.js in ständiger Entwicklung, Interface ändert sich!

Interaktion mit der Außenwelt

Beispiel (Code Account):

```
import "./Base.sol";
contract GotCoins is Payed {
    uint public totalReceived = 0;

    event GotNewCoins(address indexed lastHop, address indexed signer, uint
        howMuch);

    function GotCoins(address _owner) public Ownable(_owner){}

    function() payable {
        if(msg.value > 0) {
            emit GotNewCoins(msg.sender, tx.origin, msg.value);
            totalReceived += msg.value;
        }
    }
}
```

Schreibe Logeintrag.

Anmerkungen

Event = Log, Handler für Events können im Node registriert werden. Über Web3.js auch in Webapps möglich.

Achtung: Web3.js in ständiger Entwicklung, Interface ändert sich!

Interaktion mit der Außenwelt

Beispiel (Benutzerschnittstelle, JavaScript, Web3.js):

```
var web3 = require('web3');
var abi = [{ "name": "GotNewCoins", "type": "event", "inputs": [...], }, ...];
var gotcoinsContract = web3.eth.contract(abi).at('0xdeadbeef');

gotcoinsContract.methods.totalReceived.call().then(res => {
  if(!res.gte(web3.toBigNumber('2000000000000000000'))) return;

  var gotCoinsEvent = gotcoinsContract.GotNewCoins({},
    {fromBlock: 0, toBlock: 'latest'})
    ;

  gotCoinsEvent.watch((error, result) =>
    console.log("Got Coins!" + result + ":" + result.args));

  gotcoinsContract.transfer(web3.toWei(1, "ether"),
    { from: web3.eth.accounts[0], gas: 4000000 });
}).catch(e => console.log(e.message));
```

Anmerkungen

Event = Log, Handler für Events können im Node registriert werden. Über Web3.js auch in Webapps möglich.

Achtung: Web3.js in ständiger Entwicklung, Interface ändert sich!

Beispiel (Benutzerschnittstelle, JavaScript)

Event-Handler registrieren wenn

```
totalReceived >= 2 ether.
```

Keine *Transaktion* nötig!

Call fragt lokalen Zustand ab.

```
var web3 = require('web3');
var abi = [{ "name": "GotNewCoins", "type":
var gotcoinsContract = web3.eth.contract
```

```
gotcoinsContract.methods.totalReceived.call().then(res => {
    if(!res.gte(web3.toBigNumber('2000000000000000000'))) return;

    var gotCoinsEvent = gotcoinsContract.GotNewCoins({},
                                                {fromBlock: 0, toBlock: 'latest'});

    gotCoinsEvent.watch((error, result) =>
        console.log("Got Coins!" + result + ":" + result.args));

    gotcoinsContract.transfer(web3.toWei(1, "ether"),
        { from:web3.eth.accounts[0], gas:4000000});
}).catch(e=> console.log(e.message));
```

Anmerkungen

Event = Log, Handler für Events können im Node registriert werden. Über `Web3.js` auch in Webapps möglich.

Achtung: Web3.js in ständiger Entwicklung, Interface ändert sich!

Beispiel (Benutzerschnittstelle, JavaScript, Web3.js):

```
var web3 = require('web3');  
var abi = [{ "name": "GotNewCoins", "type": "event", "inputs": [...], }, ...];  
var gotcoinsContract = web3.eth.contract(abi);  
  
gotcoinsContract.methods.totalReceived().call((error, res) => {  
    if (!res.gte(web3.toBigNumber('200').toNumber())) return;  
  
    var gotCoinsEvent = gotcoinsContract.GotNewCoins({}, {fromBlock: 0, toBlock: 'latest'});  
  
    gotCoinsEvent.watch((error, result) => console.log("Got Coins!" + result + ":" + result.args));  
  
    gotcoinsContract.transfer(web3.toWei(1, "ether"), { from: web3.eth.accounts[0], gas: 4000000 });  
}).catch(e => console.log(e.message));
```

Anmerkungen

Event = Log, Handler für Events können im Node registriert werden. Über `Web3.js` auch in Webapps möglich.

Achtung: Web3.js in ständiger Entwicklung, Interface ändert sich!

Interaktion mit der Außenwelt

Beispiel (Benutzerschnittstelle, JavaScript, Web3.js):

```
var web3 = require('web3');
var abi = [{"name": "GotNewCoins", "type": "event", "inputs": [...], }, ...];
var gotcoinsContract = web3.eth.contract(abi).at('0xdeadbeef');

gotcoinsContract.methods.totalReceived.call().then(res => {
  if(!res.gte(web3.toBigNumber('200000000000000000'))) return;

  var gotCoinsEvent = gotcoinsContract.events.GotNewCoins({fromBlock: 0});

  gotCoinsEvent.watch((error, res) => {
    console.log('Got New Coins', res);
  });

  gotcoinsContract.transfer(web3.toWei(1, "ether"),
    { from: web3.eth.accounts[0], gas: 4000000 });
}).catch(e => console.log(e.message));
```

Transaktion erstellen, um Event auszulösen.

Transaktion nötig!

Änderung des geteilten Zustandes.

Anmerkungen

Event = Log, Handler für Events können im Node registriert werden. Über Web3.js auch in Webapps möglich.

Achtung: Web3.js in ständiger Entwicklung, Interface ändert sich!

→ Nebenläufigkeit beachten wenn Aktionen aufgrund von Zustand ausgeführt werden!

Interaktion mit der Außenwelt

Beispiel (Benutzerschnittstelle, JavaScript, Web3.js):

```
var web3 = require('web3');
var abi = [{"name": "GotNewCoins", "type": "event", "inputs": [...], }, ...];
var gotcoinsContract = web3.eth.contract(abi).at('0xdeadbeef');

gotcoinsContract.methods.totalReceived.call().then(res => {
  if(!res.gte(web3.toBigNumber('2000000000000000000'))) return;

  var gotCoinsEvent = gotcoinsContract.GotNewCoins({},
    {fromBlock: 0, toBlock: 'latest'})
    ;

  gotCoinsEvent.watch((error, result) =>
    console.log("Got Coins!" + result + ":" + result.args));

  gotcoinsContract.transfer(web3.toWei(1, "ether"),
    { from: web3.eth.accounts[0], gas: 4000000});
}).catch(e => console.log(e.message));
```

Anmerkungen

Event = Log, Handler für Events können im Node registriert werden. Über Web3.js auch in Webapps möglich.

Achtung: Web3.js in ständiger Entwicklung, Interface ändert sich!

→ Nebenläufigkeit beachten wenn Aktionen aufgrund von Zustand ausgeführt werden!

Fallstricke

Achtung!

Alle Berechnungen sind deterministisch und alle Informationen sind öffentlich.

Achtung!

Alle Berechnungen sind deterministisch und alle Informationen sind öffentlich.

Implikationen

- Als `private` gekennzeichnete Felder können nicht direkt von anderen Code Account gelesen werden, sind aber für jeden Node einsehbar.

(Kryptographische Commitment Protokolle!)

Achtung!

Alle Berechnungen sind deterministisch und alle Informationen sind öffentlich.

Implikationen

- Als `private` gekennzeichnete Felder können nicht direkt von anderen Code Account gelesen werden, sind aber für jeden Node einsehbar.
(Kryptographische Commitment Protokolle!)
- Da alle Berechnungen deterministisch sind existiert kein echter Zufall in Ethereum.
(Bootstrapping von Zufallszahlen schwierig. Oft werden TTP verwendet.)

Fallstricke

Achtung!

Miner können das Resultat der Code Account Ausführung beeinflussen.

Fallstricke

Achtung!

Miner können das Resultat der Code Account Ausführung beeinflussen.

Hintergrund

- `block.timestamp` kann zu einem gewissen Maße vom Miner gewählt werden.
(Verwendet um zeitabhängige Aktionen zu modellieren)

Fallstricke

Achtung!

Miner können das Resultat der Code Account Ausführung beeinflussen.

Hintergrund

- `block.timestamp` kann zu einem gewissen Maße vom Miner gewählt werden.
(Verwendet um zeitabhängige Aktionen zu modellieren)
- `block.blockhash(n)` kann zu einem gewissen Maße vom Miner beeinflusst werden.
(Verwendet z.b. als PRNG seed. *Nicht einfach voraussehbar, aber kein Zufall!!*)

Fallstricke

Achtung!

Miner können das Resultat der Code Account Ausführung beeinflussen.

Hintergrund

- `block.timestamp` kann zu einem gewissen Maße vom Miner gewählt werden.
(Verwendet um zeitabhängige Aktionen zu modellieren)
- `block.blockhash(n)` kann zu einem gewissen Maße vom Miner beeinflusst werden.
(Verwendet z.b. als PRNG seed. *Nicht einfach voraussehbar, aber kein Zufall!!*)
- Miner können eigene Transaktionen an beliebigen Stellen im Block einfügen, sie bestimmen die Ausführungsreihenfolge der Transaktionen.

Fallstricke

Achtung!

Miner können das Resultat der Code Account Ausführung beeinflussen.

Hintergrund

- `block.timestamp` kann zu einem gewissen Maße vom Miner gewählt werden.
(Verwendet um zeitabhängige Aktionen zu modellieren)
- `block.blockhash(n)` kann zu einem gewissen Maße vom Miner beeinflusst werden.
(Verwendet z.b. als PRNG seed. *Nicht einfach voraussehbar, aber kein Zufall!!*)
- Miner können eigene Transaktionen an beliebigen Stellen im Block einfügen, sie bestimmen die Ausführungsreihenfolge der Transaktionen.

Fallstricke

Achtung!

Miner können das Resultat der Code Account Ausführung beeinflussen.

Hintergrund

- `block.timestamp` kann zu einem gewissen Maße vom Miner gewählt werden.
(Verwendet um zeitabhängige Aktionen zu modellieren)
- `block.blockhash(n)` kann zu einem gewissen Maße vom Miner beeinflusst werden.
(Verwendet z.b. als PRNG seed. *Nicht einfach voraussehbar, aber kein Zufall!!*)
- Miner können eigene Transaktionen an beliebigen Stellen im Block einfügen, sie bestimmen die Ausführungsreihenfolge der Transaktionen.

Implikationen

- Miner können die Ausführung von CAs beeinflussen: gezieltes Anpassen von `block.timestamp`, `block.blockhash(n)` oder durch Einfügen von neuen Transaktionen bzw. Änderungen der Reihenfolge.

Fallstricke

Achtung!

Ether überweisen ist nicht trivial.

Fallstricke

Achtung!

Ether überweisen ist nicht trivial.

Hintergrund

- Code Account und *EOA* sollen nicht unterschieden werden. Das Verhalten bei Überweisungen ist jedoch unterschiedlich.

Achtung!

Ether überweisen ist nicht trivial.

Hintergrund

- Code Account und *EOA* sollen nicht unterschieden werden. Das Verhalten bei Überweisungen ist jedoch unterschiedlich.
- Standard-Überweisungen werden mit `x.transfer(value)` oder `x.send(value)` ausgeführt. *Message Call*: (Von=`this`, Zu=`x`, Betrag=`value`, Gas=2300, Daten="").

Achtung!

Ether überweisen ist nicht trivial.

Hintergrund

- Code Account und *EOA* sollen nicht unterschieden werden. Das Verhalten bei Überweisungen ist jedoch unterschiedlich.
- Standard-Überweisungen werden mit `x.transfer(value)` oder `x.send(value)` ausgeführt. *Message Call*: (Von=`this`, Zu=`x`, Betrag=`value`, Gas=`2300`, Daten=`""`).
- Ist `x` ein Code Account, dann wird die Fallback-Funktion ausgeführt. Dies schlägt z. B. fehl, wenn die Fallback-Funktion mehr als `2300` Gas benötigt.

Fallstricke

Achtung!

Ether überweisen ist nicht trivial.

Hintergrund

- Code Account und *EOA* sollen nicht unterschieden werden. Das Verhalten bei Überweisungen ist jedoch unterschiedlich.
- Standard-Überweisungen werden mit `x.transfer(value)` oder `x.send(value)` ausgeführt. *Message Call*: (Von=`this`, Zu=`x`, Betrag=`value`, Gas=2300, Daten="").
- Ist `x` ein Code Account, dann wird die Fallback-Funktion ausgeführt. Dies schlägt z. B. fehl, wenn die Fallback-Funktion mehr als 2300 Gas benötigt.

Implikationen

- Sender und/oder Empfänger können blockiert werden.
(Bis zur Handlungsunfähigkeit.)

Fallstricke

Beispiel (blockierter CA):

```
contract attacker{
    uint dummy;
    function() payable public {
        dummy = msg.value; // oder einfach revert(); SSTORE 5000/20000 Gas
    }
}

contract victim {
    address[] toPay;
    bool payedOut = false;

    function payout_push() public {
        if(!payedOut){
            for(uint i=0; i<toPay.length; i++){
                // Wenn ein transfer fehlschlaegt, schlagen alle fehl
                // Ein nicht trivialer fallback reicht um den CA zu blockieren
                toPay[i].transfer(1 ether);
            }
            payedOut = true;
        }
    }
}
```

Fallstricke

Beispiel (blockierter CA):

```
contract attacker{
    uint dummy;
    function() payable public {
        dummy = msg.value; // oder einfach revert(); SSTORE 5000/20000 Gas
    }
}

contract victim {
    address[] toPay;
    bool payedOut = false;

    function payout_push() public {
        if(!payedOut){
            for(uint i=0; i<toPay.length; i++){
                // Wenn ein transfer fehlschlaegt, schlagen alle fehl
                // Ein nicht trivialer fallback reicht um den CA zu blockieren
                toPay[i].transfer(1 ether);
            }
            payedOut = true;
        }
    }
}
```

Fallstricke

Beispiel (blockierter CA):

```
contract attacker{
    uint dummy;
    function() payable public {
        dummy = msg.value; // oder einfach revert(); SSTORE 5000/20000 Gas
    }
}

contract victim {
    address[] toPay;
    bool payedOut = false;

    function payout_push() public {
        if(!payedOut){
            for(uint i=0; i<toPay.length; i++){
                // Wenn ein transfer fehlschlaegt, schlagen alle fehl
                // Ein nicht trivialer fallback reicht um den CA zu blockieren
                toPay[i].transfer(1 ether);
            }
            payedOut = true;
        }
    }
}
```

Fallstricke

Beispiel (Wiedereintritt):

```
contract attacker{
    uint dummy;
    function() payable public {
        dummy = msg.value; // oder einfach revert(); SSTORE 5000/20000 Gas
    }
}

contract victim {
    address[] toPay;
    bool payedOut = false;

    function payout_push() public {
        if(!payedOut){
            for(uint i=0; i<toPay.length; i++){
                // Wenn ein transfer fehlschlaegt, schlagen alle fehl
                // Caller kann mehr Gas einsetzen um nicht triviale fallbacks zu
                // erlauben
                // revert() ist immer noch ein Problem
                var ok = toPay[i].call.value(1 ether)();
                if(!ok) revert();
            }
            payedOut = true;
        }
    }
}
```

Fallstricke

Beispiel (Wiedereintritt):

```
contract attacker{
    uint dummy;
    function() payable public {
        dummy = msg.value; // oder einfach revert(); SSTORE 5000/20000 Gas
    }
}

contract victim {
    address[] toPay;
    bool payedOut = false;

    function payout_push() public {
        if(!payedOut){
            for(uint i=0; i<toPay.length; i++){
                // Wenn ein transfer fehlschlaegt, schlagen alle fehl
                // Caller kann mehr Gas einsetzen um nicht triviale fallbacks zu
                // erlauben
                // revert() ist immer noch ein Problem
                var ok = toPay[i].call.value(1 ether)();
                if(!ok) revert();
            }
            payedOut = true;
        }
    }
}
```


Fallstricke

Beispiel (Wiedereintritt):

```
contract attacker{
    uint ctr=0;
    function() payable public {
        if(ctr > 1) return;
        ctr++;
        msg.sender.call(bytes4(keccak256("payout_push()"))); // Re-Enter Victim!
    }
}

contract victim {
    address[] toPay;
    bool payedOut = false;

    function payout_push() public {
        if(!payedOut){
            for(uint i=0; i<toPay.length; i++){
                var ok = toPay[i].call.value(1 ether)();
                if(!ok) revert();
            }
            payedOut = true;
        }
    }
}
```

Fallstricke

Beispiel (Wiedereintritt):

```
contract attacker{
  uint ctr=0;
  function() payable public {
    if(ctr > 1) return;
    ctr++;
    msg.sender.call(bytes4(keccak256("payout_push()"))); // Re-Enter Victim!
  }
}

contract victim {
  address[] toPay;
  bool payedOut = false;

  function payout_push() public {
    if(!payedOut){
      for(uint i=0; i<toPay.length; i++){
        var ok = toPay[i].call.value(1 ether)();
        if(!ok) revert();
      }
      payedOut = true;
    }
  }
}
```

Fallstricke

Beispiel (besser: checks → effects → interaction, pull):

```
contract attacker{
  uint ctr=0;
  function() payable public {
    if(ctr > 1) return;
    msg.sender.call(bytes4(keccak256("payout_push()"))); // Re-enter victim!
    ctr++;
  }
}

contract victim {
  enum State { NotRegistered, Pay, Payed }
  mapping(address => State) toPay;

  function payout_pull() public {
    var userState = toPay[msg.sender];
    require(userState == State.Pay);           // Checks
    toPay[msg.sender] = State.Payed;           // Effects
    msg.sender.transfer(1 ether);              // Interaction
  }
}
```

Fallstricke

Beispiel (besser: checks → effects → interaction, pull):

```
contract attacker{
  uint ctr=0;
  function() payable public {
    if(ctr > 1) return;
    msg.sender.call(bytes4(keccak256("payout_push()"))); // Re-enter victim!
    ctr++;
  }
}

contract victim {
  enum State { NotRegistered, Pay, Payed }
  mapping(address => State) toPay;

  function payout_pull() public {
    var userState = toPay[msg.sender];
    require(userState == State.Pay);           // Checks
    toPay[msg.sender] = State.Payed;           // Effects
    msg.sender.transfer(1 ether);              // Interaction
  }
}
```

Fallstricke

Beispiel (besser: checks → effects → interaction, pull):

```
contract attacker{
    uint ctr=0;
    function() payable public {
        if(ctr > 1) return;
        msg.sender.call(bytes4(keccak256("payout_push()"))); // Re-enter Victim!
        ctr++;
    }
}

contract victim {
    enum State { NotRegistered, Pay, Payed }
    mapping(address => State) toPay;

    function payout_pull() public {
        var userState = toPay[msg.sender];
        require(userState == State.Pay); // Checks
        toPay[msg.sender] = State.Payed; // Effects
        if(!msg.sender.call.value(1 ether)()) revert(); // Interaction
    }
}
```

Fallstricke

Beispiel (besser: checks → effects → interaction, pull):

```
contract attacker{
  uint ctr=0;
  function() payable public {
    if(ctr > 1) return;
    msg.sender.call(bytes4(keccak256("payout_push()"))); // Re-enter Victim!
    ctr++;
  }
}

contract victim {
  enum State { NotRegistered, Pay, Payed }
  mapping(address => State) toPay;

  function payout_pull() public {
    var userState = toPay[msg.sender];
    require(userState == State.Pay); // Checks
    toPay[msg.sender] = State.Payed; // Effects
    if(!msg.sender.call.value(1 ether)()) revert(); // Interaction
  }
}
```

Fallstricke

- Block Gas Limit
(z.b. beim löschen großer Arrays)

Fallstricke

- Block Gas Limit
(z.b. beim löschen großer Arrays)

- Solidity Type Inference

```
for (var i=0; i<2000; i++)
```

→ `typeof(i) == uint8` → Endlosschleife

Fallstricke

- Block Gas Limit

(z.B. beim löschen großer Arrays)

- Solidity Type Inference

```
for(var i=0; i<2000; i++)
```

- Callstack Limit

(max. 1024 Stack Frames)

→ `typeof(i) == uint8` → Endlosschleife

Fallstricke

- Block Gas Limit

(z.b. beim löschen großer Arrays)

- Solidity Type Inference

```
for(var i=0; i<2000; i++)
```

- Callstack Limit

(max. 1024 Stack Frames)

- Forced Ether Transfer

```
selfdestruct(0xdeadbeef)
```

→ `typeof(i) == uint8` → Endlosschleife

→ 0xDEADBEEF kriegt Saldo, kann nicht ablehnen

Fallstricke

- Block Gas Limit

(z.b. beim löschen großer Arrays)

- Solidity Type Inference

```
for (var i=0; i<2000; i++)
```

- Callstack Limit

(max. 1024 Stack Frames)

- Forced Ether Transfer

```
selfdestruct(0xdeadbeef)
```

- ...

→ `typeof(i) == uint8` → Endlosschleife

→ 0xDEADBEEF kriegt Saldo, kann nicht ablehnen

Best Practices

- Schleifen und nicht konstanten Gasverbrauch vermeiden → gegen blockierte Code Account

Best Practices

- Schleifen und nicht konstanten Gasverbrauch vermeiden → gegen blockierte Code Account
- Checks Effects Interaction Pattern → gegen Wiedereintritt

Best Practices

- Schleifen und nicht konstanten Gasverbrauch vermeiden → gegen blockierte Code Account
- Checks Effects Interaction Pattern → gegen Wiedereintritt
- Geld abheben: pull statt push → gegen blockierte Code Account
(bessere Isolation von Ressourcen)

Best Practices

- Schleifen und nicht konstanten Gasverbrauch vermeiden → gegen blockierte Code Account
- Checks Effects Interaction Pattern → gegen Wiedereintritt
- Geld abheben: pull statt push → gegen blockierte Code Account
(bessere Isolation von Ressourcen)
- Zustandsautomaten als Modellierungstool
(Von Transaktions/Message-übergreifenden Abläufen)

Best Practices (Forts.)

- Zugriffskontrolle

→ `Ownable`, etc.

Best Practices (Forts.)

- Zugriffskontrolle
- Plane den Worst-Case
(Wie behalte ich die Kontrolle wenn etwas schief läuft?)

→ `Ownable`, etc.

Best Practices (Forts.)

- Zugriffskontrolle → `Ownable`, etc.
- Plane den Worst-Case
(*Wie behalte ich die Kontrolle wenn etwas schief läuft?*)
 - Abschalten? → `Destructable`, etc.

Best Practices (Forts.)

- Zugriffskontrolle → `Ownable`, etc.
- Plane den Worst-Case
(*Wie behalte ich die Kontrolle wenn etwas schief läuft?*)
 - Abschalten? → `Destructable`, etc.
 - Fail-Safe-Modus?

Best Practices (Forts.)

- Zugriffskontrolle → `Ownable`, etc.
- Plane den Worst-Case
(*Wie behalte ich die Kontrolle wenn etwas schief läuft?*)
 - Abschalten? → `Destructable`, etc.
 - Fail-Safe-Modus?
 - Code „ändern“? → Proxy-Pattern

Best Practices (Forts.)

- Zugriffskontrolle → `Ownable`, etc.
- Plane den Worst-Case
(*Wie behalte ich die Kontrolle wenn etwas schief läuft?*)
 - Abschalten? → `Destructable`, etc.
 - Fail-Safe-Modus?
 - Code „ändern“? → Proxy-Pattern
(Achtung: Kann zu Vertrauensproblemen führen.)

Best Practices (Forts.)

- Zugriffskontrolle → `Ownable`, etc.
- Plane den Worst-Case
(*Wie behalte ich die Kontrolle wenn etwas schief läuft?*)
 - Abschalten? → `Destructable`, etc.
 - Fail-Safe-Modus?
 - Code „ändern“? → Proxy-Pattern
(Achtung: Kann zu Vertrauensproblemen führen.)
- Software-Testing, Bug-Bounties, formale Verifikation ...

Fragen ?