



Laurea Triennale in Informatica - Università di Salerno -
Sistema di autenticazione sulla rete Blockchain Ethereum



KRYPTOAUTH

Manuale delle Istruzioni KryptoAuth

Riferimento	
Versione	1.0
Data	29/07/2022
Presentato da	Montefusco Alberto



Sommario

Sommario	2
1. Introduzione	3
1.1 Scopo del Sistema	3
1.2 Scopo del Documento	3
2. Requisiti per l'installazione	3
2.1 Deploy Smart Contracts	4
2.2 Configurazione rete Metamask - Ganache	7
2.3 Installazione Web DApp	10



1. Introduzione

1.1 Scopo del Sistema

La realizzazione di KryptoAuth ha l'obiettivo di offrire maggiore sicurezza durante l'operazione di autenticazione sfruttando la tecnologia Blockchain Ethereum.

1.2 Scopo del Documento

Lo scopo di questo documento è di aiutare l'utente nell'installazione del sistema e, qualora mancanti, di tutte le componenti necessarie al suo funzionamento. In particolare, sarà mostrata la procedura di deploy dello Smart Contract sulla Blockchain di test Ganache, la creazione e la configurazione della rete per l'interfacciamento tra Metamask e Ganache e, infine, la traduzione dello Smart Contract scritto in Solidity in una classe Java mediante l'uso di solc.js e Web3j.

2. Requisiti per l'installazione

Requisiti lato client:

- web Browser;
- estensione Metamask;
- connessione ad Internet.

Requisiti lato server, necessari per l'uso di KryptoAuth:

- **Ganache**, requisito base per il funzionamento del sistema (**LINK:** [Ganache](#));
- **Web3j**, con il quale si potrà convertire lo Smart Contract in linguaggio Java ed interfacciare la Web DApp con Ganache (**LINK:** [Web3j](#));
- **Solc**, per generare l'abi e il bin dello Smart Contract (**LINK:** [solc.js](#));
- **Truffle**, per effettuare il deploy dello Smart Contract su Ganache (**LINK:** [Truffle](#)).

2.1 Deploy dello Smart Contract

Per effettuare il deploy dello Smart Contract bisogna dapprima aprire l'applicazione Ganache: all'avvio possiamo creare un workspace personalizzato oppure avviarne uno di default tramite la sezione "Quickstart".

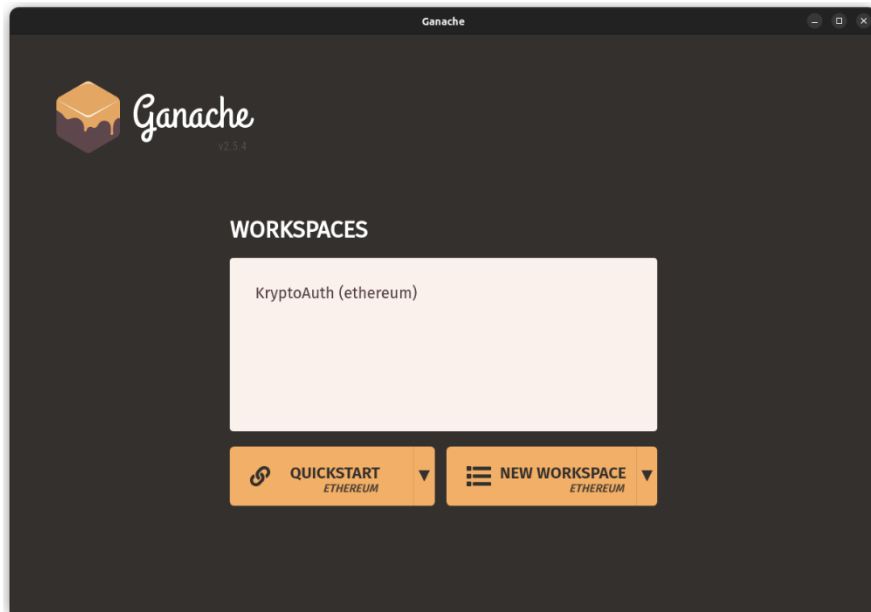


Figura 1. Homepage Ganache

Successivamente, importiamo lo Smart Contract all'interno di Ganache andando a specificare il suo path:

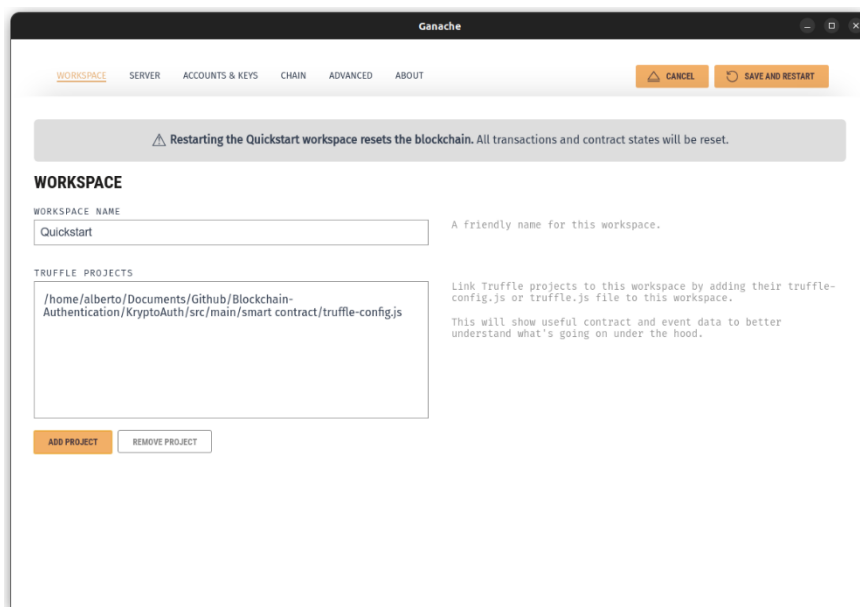
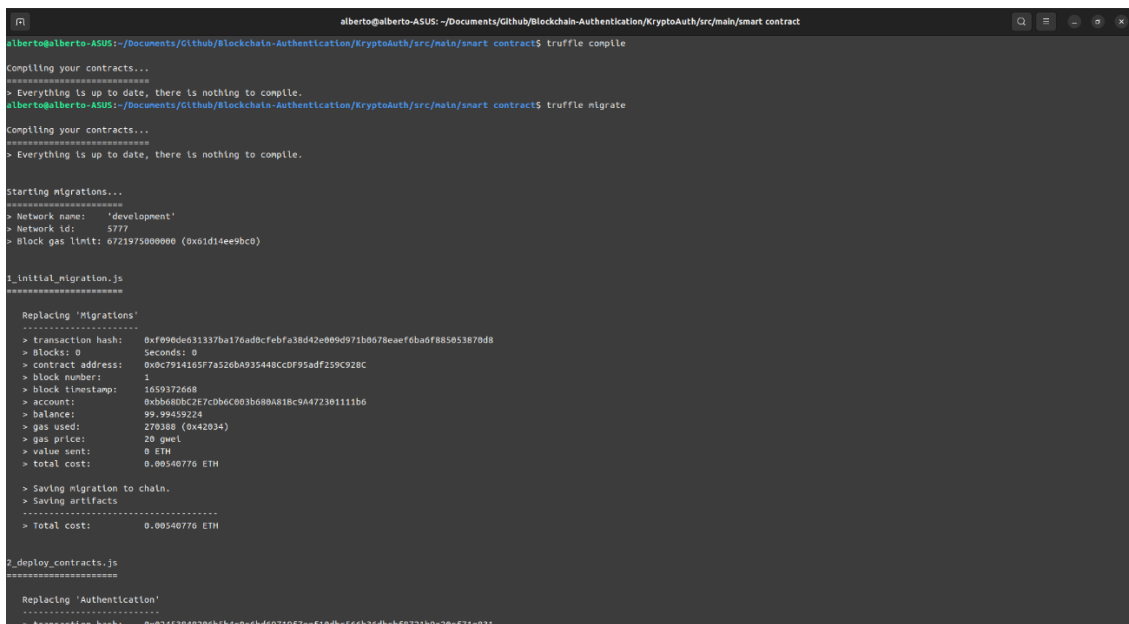


Figura 2. Aggiunta Smart Contract

Dopodiché, apriamo il nostro terminale e ci rechiamo nel package “smart contract” all’interno del progetto KryptoAuth. All’interno del terminale digitiamo:

- `truffle compile`, per verificare se ci sono errori sintattici all’interno dello Smart Contract;
- `truffle migrate`, per deployare lo Smart Contract (per effettuare un reset delle connessioni eseguiamo `truffle migrate --reset`).



```

alberto@alberto-ASUS: ~/Documents/Github/Blockchain-Authentication/KryptoAuth/src/main/smart contract
alberto@alberto-ASUS:~/Documents/Github/Blockchain-Authentication/KryptoAuth/src/main/smart contract$ truffle compile
Compiling your contracts...
Everything is up to date, there is nothing to compile.
alberto@alberto-ASUS:~/Documents/Github/Blockchain-Authentication/KryptoAuth/src/main/smart contract$ truffle migrate
Compiling your contracts...
Everything is up to date, there is nothing to compile.

Starting migrations...
> Network name: 'development'
> Network id: 5777
> Block gas limit: 672197500000 (0x0d14ee8bc0)

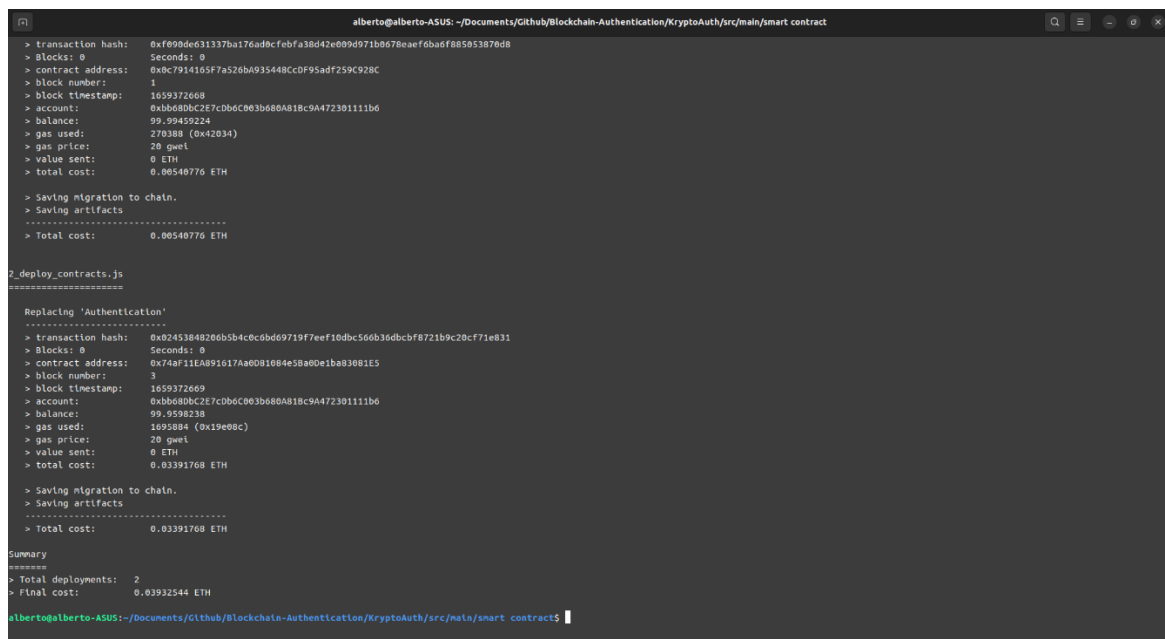
1_initial_migration.js
-----
Replacing 'Migrations'
-----
> transaction hash: 0xf090de631337ba176ad0cfeba38d42e09d971b0678eae6ba6f88505387bd8
> blocks: 0
> contract address: 0x0c7914165f7a520ba935448ccdf95adf239c928c
> block number: 1
> block timestamp: 1659372608
> account: 0xb6b80bc2e7c0b6c003b680a818c9a47230111b6
> balance: 99.99459224
> gas used: 270388 (0x42034)
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.00540776 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost: 0.00540776 ETH

2_deploy_contracts.js
-----
Replacing 'Authentication'
-----
> transaction hash: 0x02453848200b5b4cc0bd69719f7ee10dbc566b30dbcbf8721b9c20cf71e831

```

Figura 3. Compilazione Smart Contract



```

> transaction hash: 0xf090de631337ba176ad0cfeba38d42e09d971b0678eae6ba6f88505387bd8
> blocks: 0
> contract address: 0x0c7914165f7a520ba935448ccdf95adf239c928c
> block number: 1
> block timestamp: 1659372608
> account: 0xb6b80bc2e7c0b6c003b680a818c9a47230111b6
> balance: 99.99459224
> gas used: 270388 (0x42034)
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.00540776 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost: 0.00540776 ETH

2_deploy_contracts.js
-----
Replacing 'Authentication'
-----
> transaction hash: 0x02453848200b5b4cc0bd69719f7ee10dbc566b30dbcbf8721b9c20cf71e831
> blocks: 0
> contract address: 0x74af11Ea891017Aa00B1084e58a00e1ba03081E5
> block number: 3
> block timestamp: 1659372609
> account: 0xb6b80bc2e7c0b6c003b680a818c9a47230111b6
> balance: 99.9598238
> gas used: 1695804 (0x19e08c)
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.03391768 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost: 0.03391768 ETH

Summary
-----
> Total deployments: 2
> Final cost: 0.03932544 ETH

alberto@alberto-ASUS:~/Documents/Github/Blockchain-Authentication/KryptoAuth/src/main/smart contract$

```

Figura 4. Deploy Smart Contract

Una volta completata la procedura il nostro contratto sarà deployato sulla Blockchain Ganache. Infatti, nella Figura 3., possiamo notare che il primo address ha speso 0.03932544 ETH per effettuare la seguente transazione.

ADDRESS	BALANCE	TX COUNT	INDEX
0xbb68DbC2E7cDb6C003b680A81Bc9A47230111b6	99.96 ETH	4	0
0xB7a95e695F053922cb70824CfC2EF8a2F78c2573	100.00 ETH	0	1
0x2CFB9721ca83BeaAda30A2e32e407a17E55C7b6D	100.00 ETH	0	2
0xAFAd2b4798C4b3c80c4A48459E476116C8fdeA9A	100.00 ETH	0	3
0x38517e96E449b088E3406812e839bEfde93613C3	100.00 ETH	0	4
0x5bEF3ae25115C569DDf6b27eF0b7E699908E5eD3	100.00 ETH	0	5
0xe6F00954178e53f12C65D62E78063fcc78acD882	100.00 ETH	0	6

Figura 5. Accounts Ganache

Nella sezione “Contracts” ci sono gli indirizzi dei contratti deployati.

NAME	ADDRESS	TX COUNT	STATUS
AccessControl	Not Deployed	0	
Authentication	0x62259579BAf566d844DE7D00Cd67Ea68c9D82f80	0	DEPLOYED
Context	Not Deployed	0	
ERC165	Not Deployed	0	
IAccessControl	Not Deployed	0	
IERC165	Not Deployed	0	
Migrations	0x18D37f3308Ebc3736e7105b3047463Db8AD15239	1	DEPLOYED
Strings	Not Deployed	0	

Figura 6. Contracts Address Ganache



Una volta che il contratto sarà deployato, copiamo il contract address dello Smart Contract Authentication di Ganache e lo andremo ad inserire nel progetto nella classe BlockchainServiceImpl assegnandolo alla costante CONTRACT_ADDRESS, poiché quest'ultima avrà l'indirizzo del deploy precedente:

```
@Service
public class BlockchainServiceImpl implements BlockchainService {

    2 usages
    private final static Web3j web3j = Web3j.build(new HttpService( url: "HTTP://127.0.0.1:7545"));
    1 usage
    private final static String CONTRACT_ADDRESS = "0x62259579BAf566d8440E7D00Cd67Ea68c9D82f80";
    11 usages
    private static String privateKey;
    5 usages
    private static Authentication authentication = null;
```

Figura 7. Contracts Address Ganache in Java

2.2 Configurazione rete Metamask - Ganache

Per collegare l'estensione browser Metamask alla Blockchain Ganache i passaggi iniziali da seguire sono di installare e di registrarsi a Metamask; successivamente dalle impostazioni dell'estensione andiamo nella sezione “Aggiungi Rete” per creare una nuova rete.

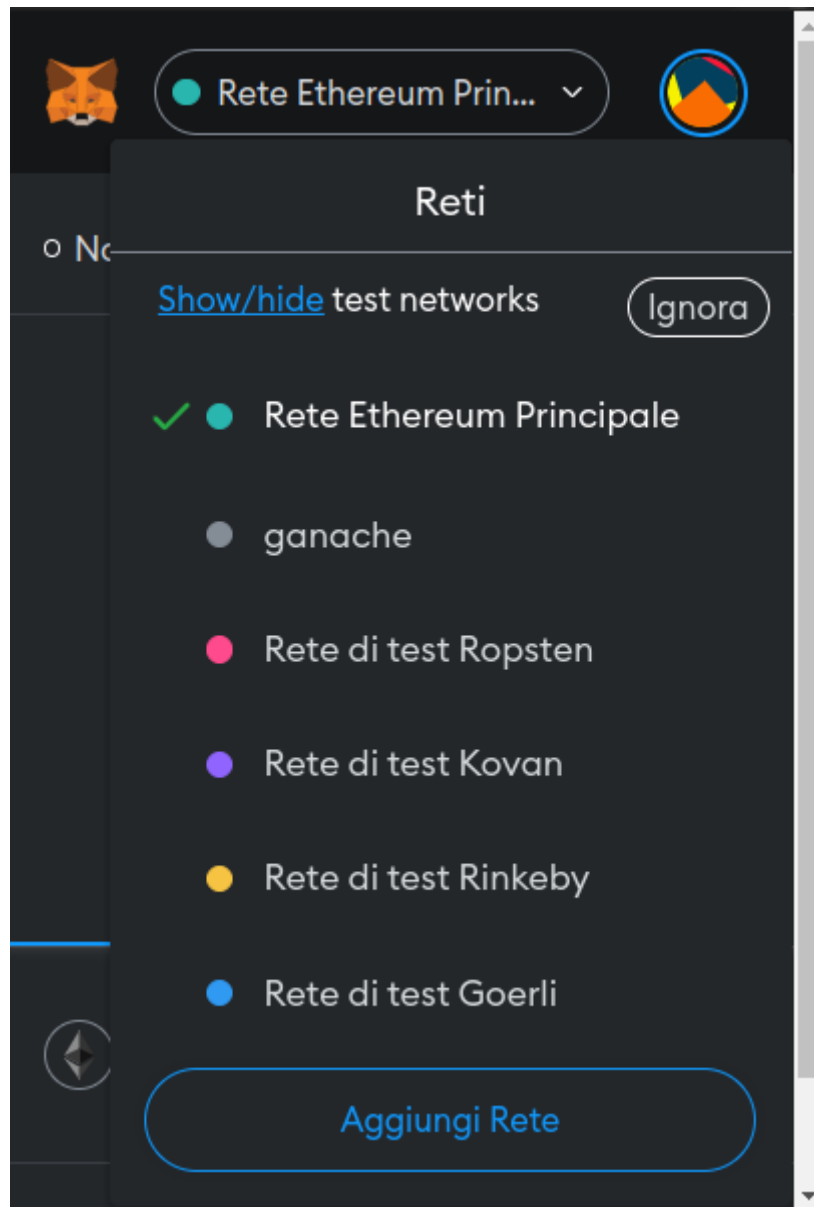


Figura 8. Aggiungi Rete

Le informazioni che devono essere inserite devono essere reperite da Ganache, in particolare:

- Nome rete: assegniamo il nome che vorremmo che abbia la nuova rete;
- Nuovo URL RPC: è l'indirizzo http di Ganache (default HTTP://127.0.0.1:7545);
- Chain ID: si deve inserire 1337 che corrisponde all'id di Ethereum;
- Currency Symbol: si deve inserire "ETH" se abbiamo una Blockchain Ethereum.

Figura 9. Configurazione Rete

Salviamo la nuova rete ed importiamo gli account da Ganache andando nella sezione “Importa Account” di Metamask.

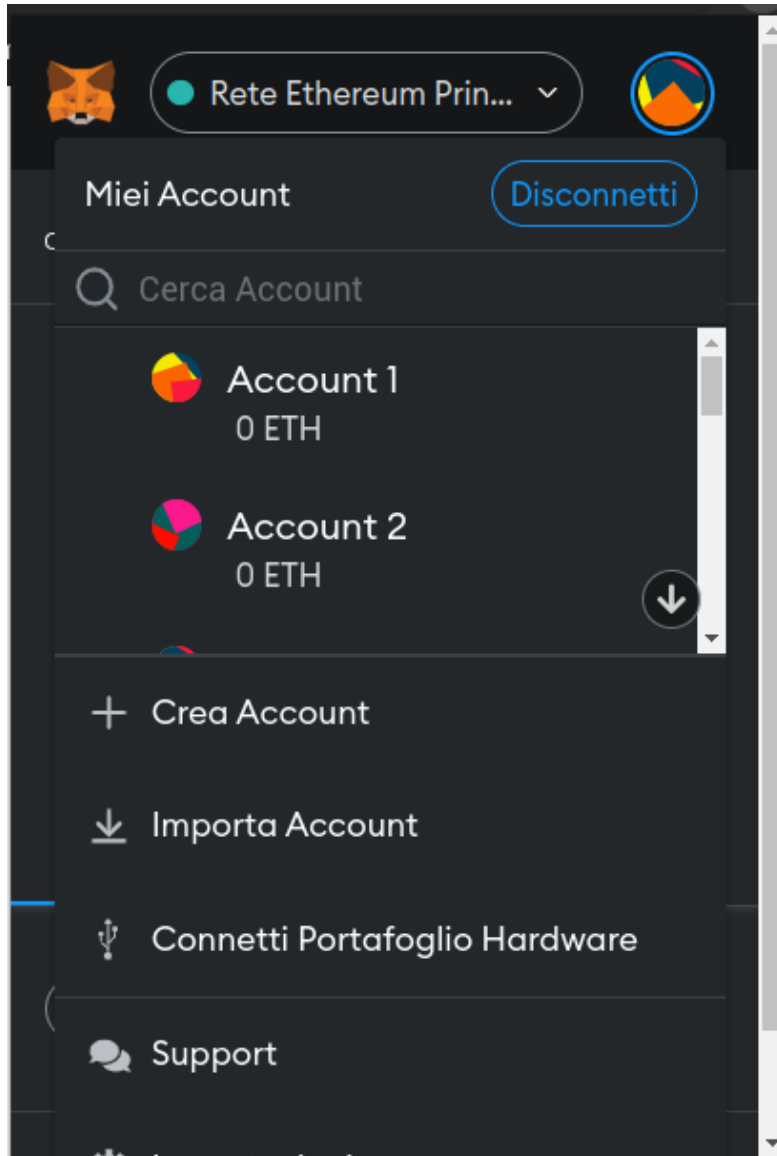


Figura 10. Importa Account

2.3 Installazione Web DApp

Per poter utilizzare la Web DApp KryptoAuth bisogna accedere dall'IDE JetBrains IntelliJ e avviare la classe main "KryptoAuth". Avviato il server Tomcat grazie a Spring Boot, accediamo al nostro Web Browser alla pagina <http://localhost:8080/kryptoauth>.

Nel caso in cui si volesse tradurre nuovamente lo Smart Contract scritto in Solidity in una classe Java, nel terminale digitiamo:

```
alberto@alberto-ASUS: ~  
alberto@alberto-ASUS:~$ solcjs /home/alberto/Documents/Github/Blockchain-Authentication/KryptoAuth  
/src/main/'smart contract'/contracts/Authentication.sol --bin --include-path node_modules/ --base-  
path . --abi --optimize -o /home/alberto/Documents/Github/Blockchain-Authentication/KryptoAuth/src  
/main/resources/solidity
```

Figura 11. Generazione .abi e .bin



```
alberto@alberto-ASUS: ~  
alberto@alberto-ASUS:~$ web3j generate solidity -b ./src/main/resources/solidity/Authentication.bi  
n -a ./src/main/resources/solidity/Authentication.abi -o ./src/main/java -p it.unisa.KryptoAuth.co  
ntracts
```

Figura 12. Generazione classe Java