

7CCSMDLC: Distributed Ledgers & Cryptocurrencies

Lecture 9: Smart Contracts



Peter McBurney (with ACK to Luke Riley)

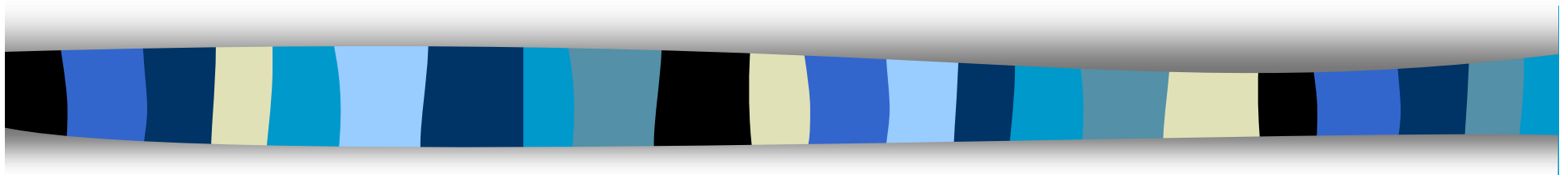
Professor of Computer Science

Department of Informatics

King's College London

Email: peter.mcburney@kcl.ac.uk

Bush House Central Block North – Office 7.15



Smart Contracts



Introduction

- In this lecture, we give a high-level overview of smart contracts.
- We don't go into detail about programming with smart contracts in this course, for several reasons:
 - The details differ from one distributed ledger to another
 - Because the technology is still immature, software development is still messy and cumbersome
 - And there are still not available yet easy-to-use development environments and software engineering tools.

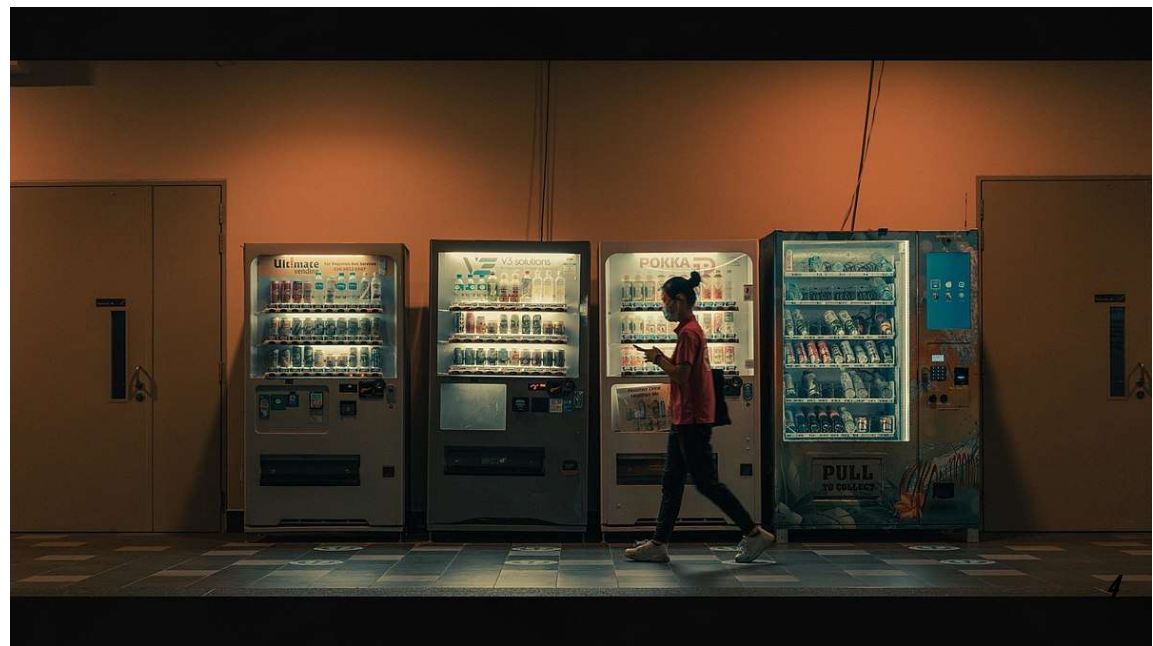
Smart contracts were invented in 1994

'A smart contract is a computerized transaction protocol that executes the terms of a contract. The general objectives are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries.'

Nick Szabo, 1994

Example: Vending machines

Photo credit: Gizer Esigues





Smart contracts

- What are they?
 - An automated computer programme or script.
- First described: In 1994 by Nick Szabo.
- Why are they on distributed ledgers?
 - This is a neutral platform enabling sharing of data in a tamper-proof way.
- If a smart contract represents an agreement between 2 or more parties, then who should host it?
 - Neither party may trust the other parties to host it.
 - They could ask a third party to host it, but most 3rd parties wish to be paid.
 - A third-party host could tamper with it themselves.



Bitcoin Blockchain & Ethereum

- A Bitcoin script is a basic smart contract.
 - They are conditions coded into a transaction stating the requirements for spending the output of the transaction.
 - Bitcoin Script language designed deliberately not to be too powerful.
- Ethereum designed to enable smart contracts:
 - Each smart contract to be programmed using a Turing complete language (Solidity); and
 - Each smart contract to add to and modify its associated data store in the ledger.
- A Turing complete language allows the developer full programming power (eg, recursion, loops, etc).



Modifying data storage

- Allowing a smart contract to modify its associated data storage, in any manner described in its implementation, provides the developer with flexibility over the smart contract's storage model.
- This storage model can now include basic distributed ledger information (eg how much cryptocurrency does this smart contract hold), as well as user-defined parameters
 - eg, strings, integers, classes . . .
- Smart contract code is usually placed in a transaction in a block, it has certain computational restrictions.
 - So even if you can code your smart contract using a Turing complete language, you still have to consider the computational restrictions of the distributed ledger you will deploy your smart contract code onto.



Smart contracts and trust

- A smart contract:
 - Adds trust to the system by allowing:
 - logic and (any associated) state to be independently verified
 - verification to be completed in near real time.
 - Automates code.
- Deploying code as a smart contract onto a distributed ledger increases trust in this code because:
 - The smart contract code cannot be modified.
 - If the smart contract has any associated data storage, this data can only be modified by satisfying the conditions encoded within the smart contract.
 - The smart contract code will always run when invoked by another transaction of the distributed ledger.
 - The distributed ledger will record all interactions with this smart contract.



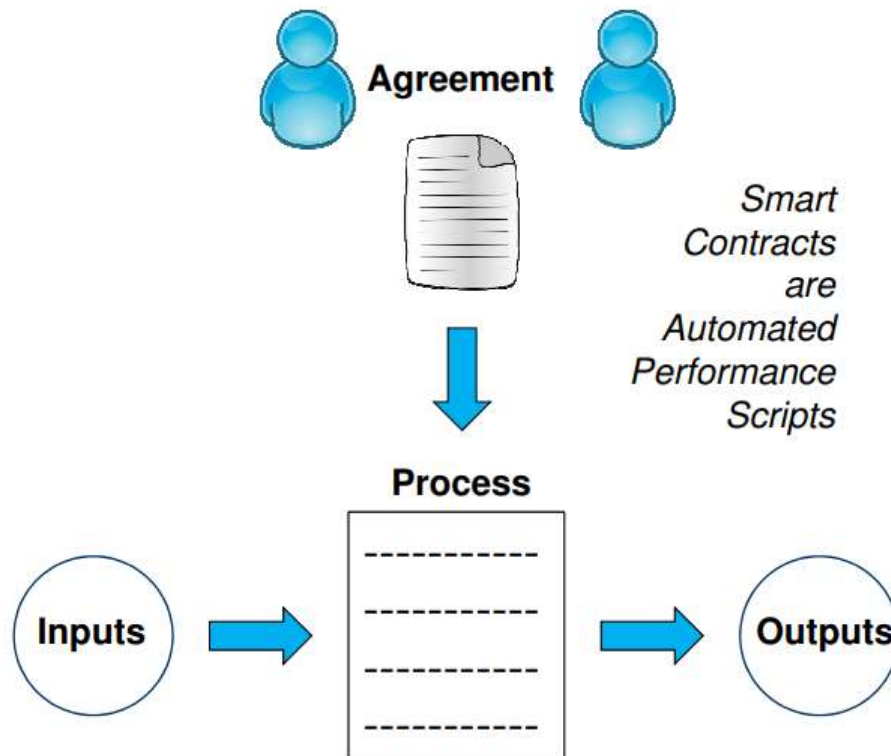
SC: What is in a name?

- **They are not smart**
 - A smart contract is only a collection of scripts/functions placed on the distributed ledger, as well as possibly some additional data storage elements.
- **They may be only crude contracts**
 - They are publicly readable and verifiable. So, if you invoke a smart contract's script or function, it is assumed that you agree to any of the circumstances that occur from its execution.
- **They are not legally binding contracts**
 - When you want to link smart contracts to legal text, you need to embed them in another model, such as Ricardian contracts
- **They may not represent agreement between two parties**
 - They may created by a developer to enable or support some other function.

What are agreements?

A smart contract . . .

. . . is an automated process, usually based on agreement between two or more parties, that autonomously executes at a trigger



But what are agreements in the context of distributed ledgers?

1. Agreements based on human readable text (e.g. legal contracts)
2. Agreements based on computation (e.g. function calls, protocols)



Using DLT to prove agreements exist

A Stampery is a service that stamps a document to certify its existence at a particular date and time.

This provides a **proof-of-agreement service**

- You upload your manually-signed document to the Stampery, which
 - Hashes it (normally with a specified hash algorithm),
 - Combines the hash with others in a Merkle Tree, and
 - Posts the Merkle Tree root to one or more distributed ledgers.
- Subsequently, you can prove the document has been unedited by:
 - Producing your original signed document
 - Running the hash algorithm on the original
 - Showing the hash value generated by the original equals the hash value posted to the distributed ledgers.

Example of a Stampery certificate

DOC ReallyImportantAgreement.pdf

SHAREPRINT

S

Stampery certificate

This is to certify that the dataset or file referred hereunder existed and was presented in the date and time printed down below by the person identified as "signee".

NAME	ReallyImportantAgreement.pdf
DATE & TIME	Wed Feb 14 2018 13:49:02 GMT+0000 (GMT Standard Time)
SIGNEE	Luke Riley (luka501@hotmail.com)
DATA HASH	FAA6722EDE65325D0EC42E69E7D61586 D8F6158383F7857B00346F38A113F7B4

Technical details

Data hash

3736f7318f397130975a230518a52d1c187287c81378a62773f216f40a2f058f

ETHEREUM PROOF

Siblings

A488458ACE31C81C3C6F23F3D0213628C8949ABE3180C06C5E8198D34D94348

A48F72834E25880C86701F71D22A78BE458148F5E4718F8C94880852116A4F

AFE3184BA55FF963803C1CF90C88451B3E36A885E25A973F38E8EDE37C8D98C

R288F1A59020C4D242741968C788018D4FA58326D38800EC4DC8C9338C3

BFEF43A18E3709C329F8C9617013E1E801E1688888518AF54A634941730D956

4505EFD4A9A2938584849A713315C3AD29442AC258E1691E4DDA3CF78E89866

Root

DF2BF5195A655B4F2CA1C9CBAF76854968B1805D2EAB69BC73D85112F9A721

Prefix

53363583

ETH TX

0xb5308d0650e2ebd065345b8a778cb024d59774b1688d2a488a418d3a3e9a

ETC TX

0x35c30b99a232e8d2f6e289e6fc182e2cf9df279f6f293663b71bac7bc14aae

BITCOIN PROOF

Siblings

A488458ACE31C81C3C6F23F3D0213628C8949ABE3180C06C5E8198D34D94348

A48F72834E25880C86701F71D22A78BE458148F5E4718F8C94880852116A4F

AFE3184BA55FF963803C1CF90C88451B3E36A885E25A973F38E8EDE37C8D98C

R288F1A59020C4D242741968C788018D4FA58326D38800EC4DC8C9338C3

BFEF43A18E3709C329F8C9617013E1E801E1688888518AF54A634941730D956

4505EFD4A9A2938584849A713315C3AD29442AC258E1691E4DDA3CF78E89866

638A3BA216E81188C7921557819877DFA84000108E3180D05A4A1D640404

228377C8E3169D8C98843C84C778ED873C01C8F248DC1ECCE384D2571683F8E

ADD48D4DF96D41C8C7211C642803E86A3193ABEBAJFC1E138CF662877A6C8

AC11AE5370D2D97888BA70E1CAE3886377A835A389AE87E2F3F4B19A3785D5E

9943F41299691C892508B1C1851919180D9FA239EC08F12784D53C74A8D935

B609781AB876416C191A80C22A86479A584AF85838362C45196751C903A8882

Root

52DC941D7E26AC267A15A78652570F24380B45663E866748CC8911A88389998

Prefix

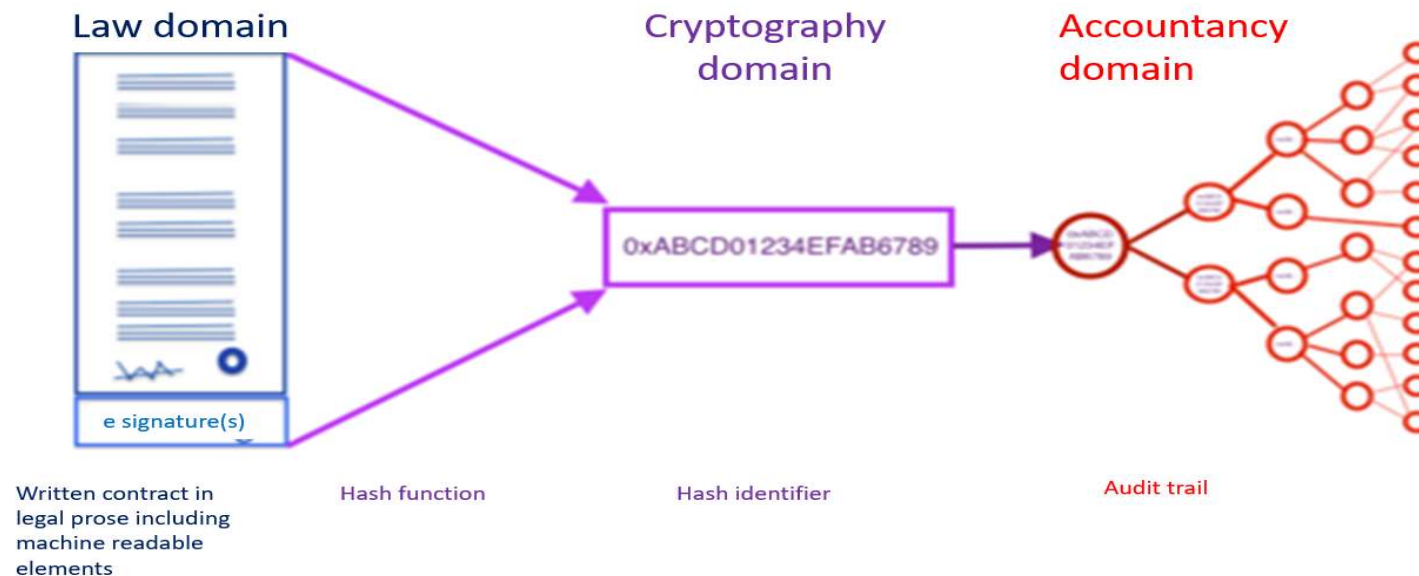
53363583

Txid

1a61db8688583d3739c19b532964ae097928d2d16d922855df6d119c-f987b2

Picture source: stamp.io

Ricardian Contracts: linking legal agreements to code



To convert a legal contract into computation, we can use a Ricardian contract design:

- The agreement is written as legal prose (with matching machine readable code for the critical sections)
- All of the participants digitally sign the Ricardian contract
- A hash identifier of this digitally signed contract is generated
- This hash identifier is placed somewhere safe (eg, on a blockchain) for auditing purposes.



Barclays' Ricardian Contract PoC

- In 2016, Barclays developed a Proof-of-Concept (PoC) for various interbank transactions that relied on a series of smart contract templates
 - The templates are in the form of Ricardian contracts
- Legal template documents were provided, where users fill in certain variables (eg, bank names).
- The code of the critical parts of the contract is provided
- When all counterparties sign it digitally, the code is placed on the distributed ledger to self execute when its conditions are met.
 - For the Proof-of-Concept, Barclays used Corda (the first public demonstration of the Corda platform).
- See Barclays Video (link provided on KEATS):

<https://www.youtube.com/watch?v=YIH4MJf6kH8&t=237s>

Barclays' Ricardian Contract example


Template Editor

Agreement Editor

Trade Entry

Trade Affirmation

Trade Viewer




Credit Support Annex 1995 - England and Wales

Close

Edit

Delete



determined for each relevant currency and calculated for each day in that Interest Period on the principal amount of the portion of the Credit Support Balance comprised of cash in such currency, determined by the Valuation Agent for each such day as follows:

(x) the amount of cash in such currency on that day; multiplied by

(y) the relevant Interest Rate in effect for that day; divided by

(z) 360 (or, in the case of pounds sterling, 365)

```
{
  "id": "DailyInterestAmount",
  "type": "Expression",
  "value": "(CashAmount * InterestRate) / (if Currency == 'GBP' then 365 else 360)"
}
```

"Interest Period" means the period from (and including) the date on which the first Cash Amount has yet been transferred, the Local Business Day on which the first Cash Amount is transferred to or received by the Transferee) to (but excluding) the Local Business Day on which the current Interest Amount is transferred.

"Interest Rate" means, with respect to an Eligible Currency, the rate specified in Paragraph 11(f)(i) for that currency.



Agreements using smart contracts

- Smart contracts on a distributed ledger are readable by others
 - If the ledger is permissioned (not open), then only those entities with access permission may see it.
- By calling a function on a smart contract you implicitly agree to the execution of that function and any consequences that arise from its execution.
- Functions can be grouped into:
 - **Getters** - Get functions return the value of some current parameter
 - **Setters** - Set functions change the state of the blockchain and can change the agreements between individuals.
 - Other users can be alerted to the change of state of an agreement by either using a get function or subscribing to events that can be automatically fired from functions.



Smart Contract Languages

Smart contracts for distributed ledgers can be written in many new or existing programming languages, eg:

- **Script (based on the Forth language):** Bitcoin, Bitcoin Cash, Litecoin, ...
- **Solidity:** Ethereum, Quorum, Hyperledger Burrow, the Counterparty protocol, ...
- **Java:** Hyperledger Fabric, Corda, . . .
- **C++:** EOS, . . .

Therefore, how you implement your smart contract depends on the distributed ledger you choose.

- **Solidity Integrated Development Environment (IDE)**

The official development environment for programming in Solidity is:

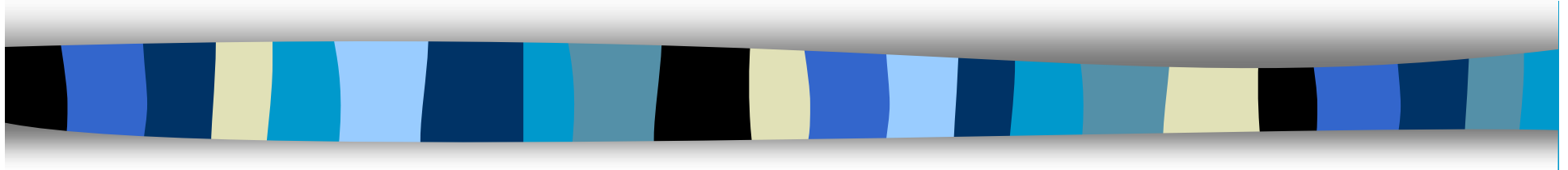
<https://remix.ethereum.org/>



Conclusions

- Smart contracts are automated programs that run over distributed ledgers.
- They are executed at every full node.
 - They act to change the state of a virtual machine which is sitting on each node.
- They may or may not represent an agreement between two or more parties in the real world outside the ledger.
 - Many SCs are created by developers to implement some desired functionality in a complex business process.
 - See the Appendix slides for an example developed by Dr Luke Riley.
- They need to run at every node, and may run at slightly different times.
 - Therefore they cannot rely on inputs that off the blockchain, as these inputs may alter. Their inputs have to be either on the blockchain or in another SC (eg, a variable access by a getter function).
 - They cannot be random algorithms – they are all deterministic.

Thank you!



peter.mcburney@kcl.ac.uk