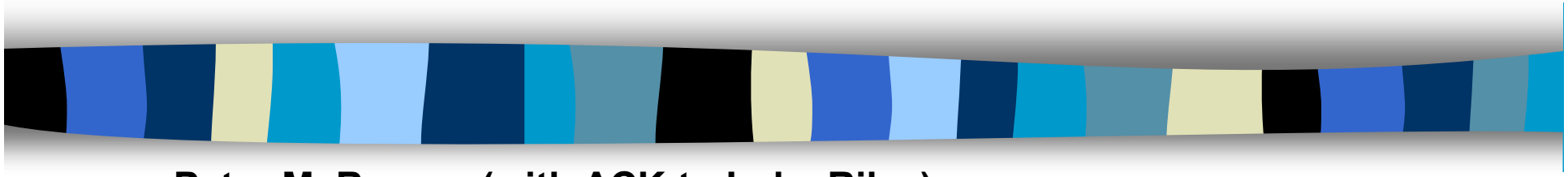
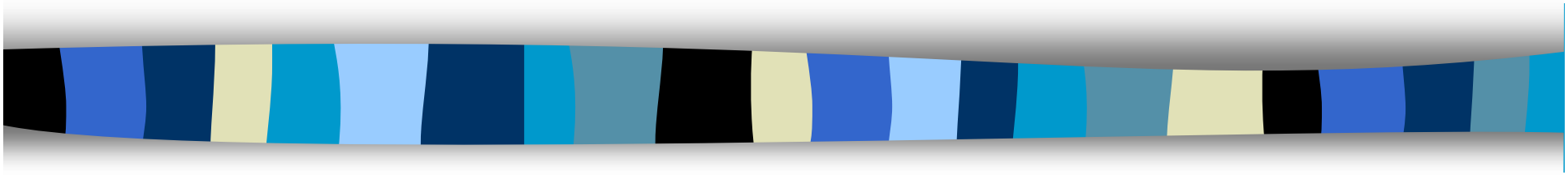
7CCSMDLC: Distributed Ledgers & Cryptocurrencies

Lecture 9: Smart Contracts

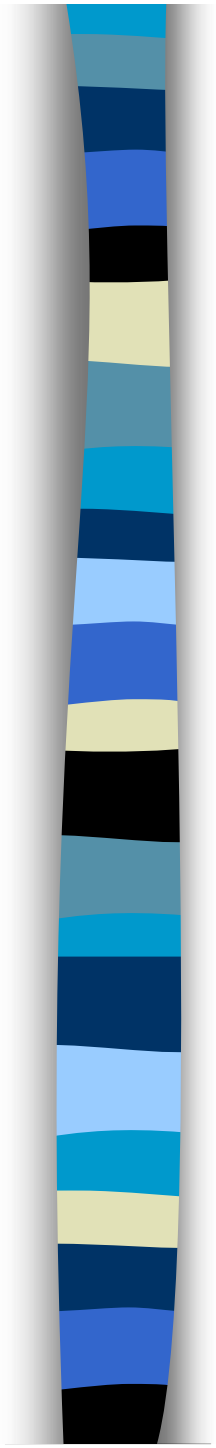


Peter McBurney (with ACK to Luke Riley)
Professor of Computer Science
Department of Informatics
King's College London

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.

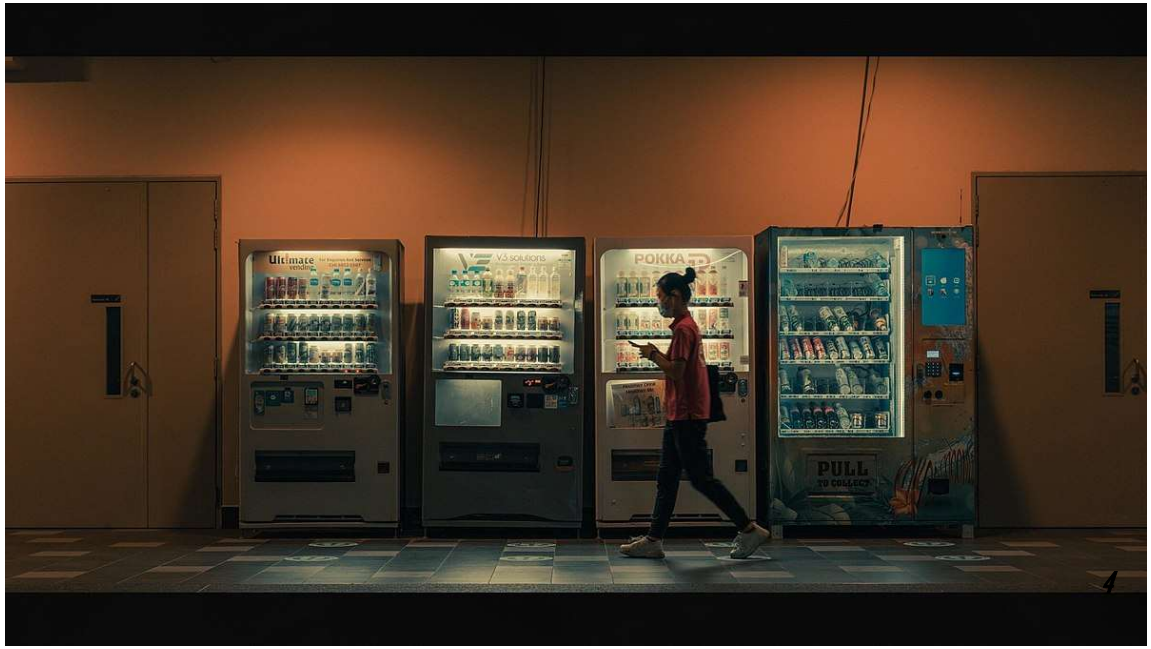
Smart contracts were invented in 1994

'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 & Ethereuem

- 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.

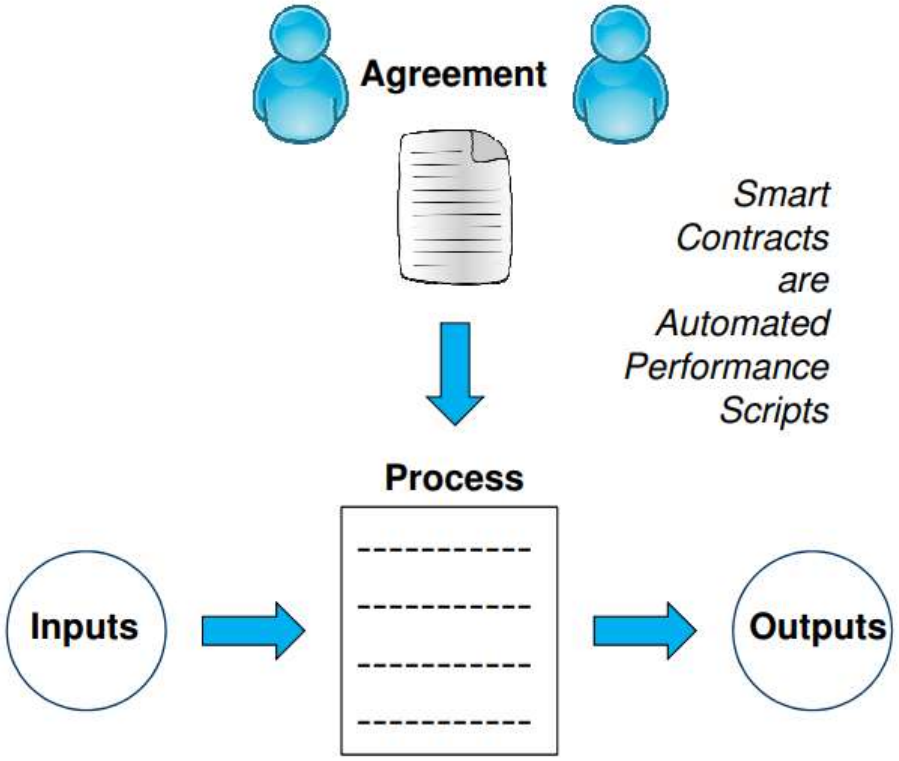
SC: What is in a name?

- **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

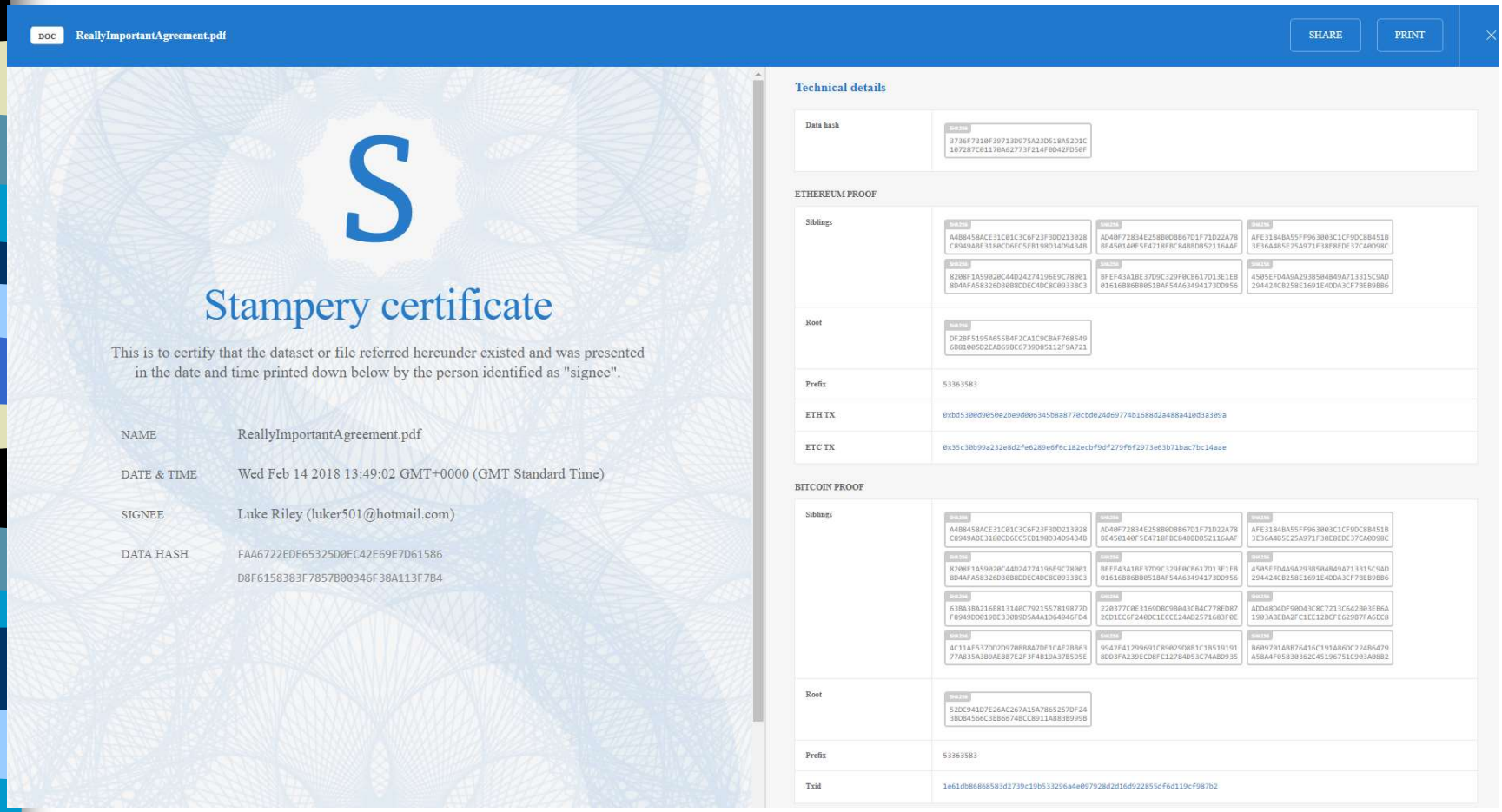
A Stampery is a service that stamps a document to certify its existence at a particular date and time.

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.

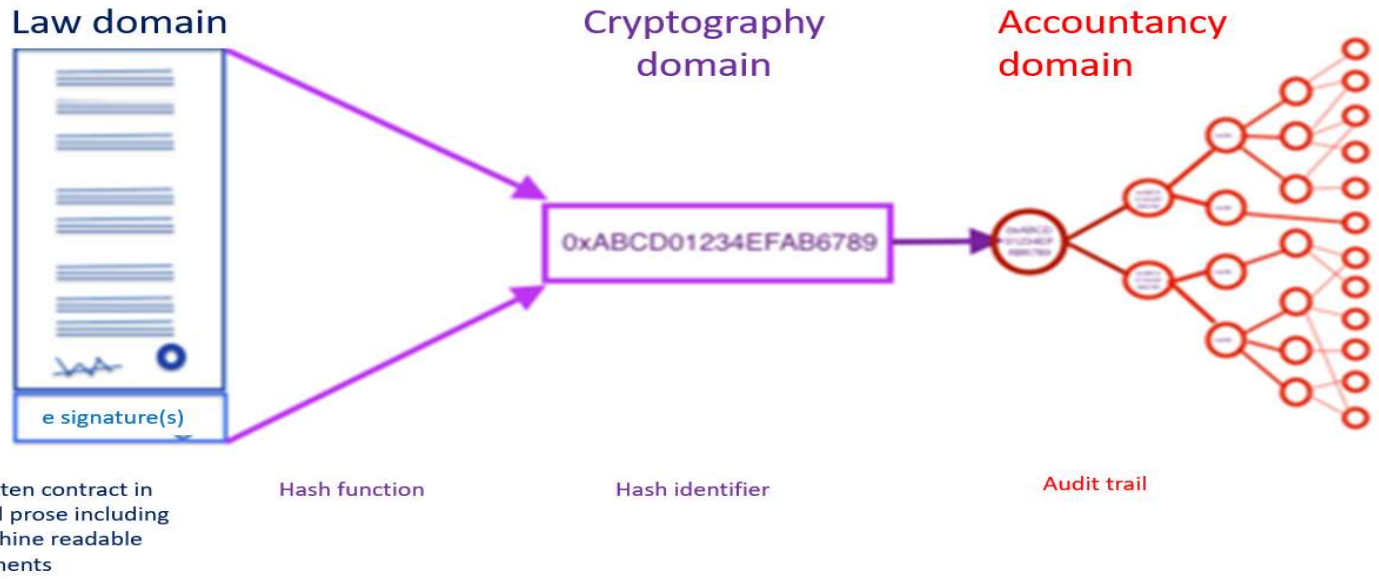
- Subsequently, you can prove the document has been unedited by:
  - Producing your original signed document
  - Running the hash algorithm on the original
  - Showing the hash value generated by the original equals the hash value posted to the distributed ledgers.

Example of a Stampery certificate



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.

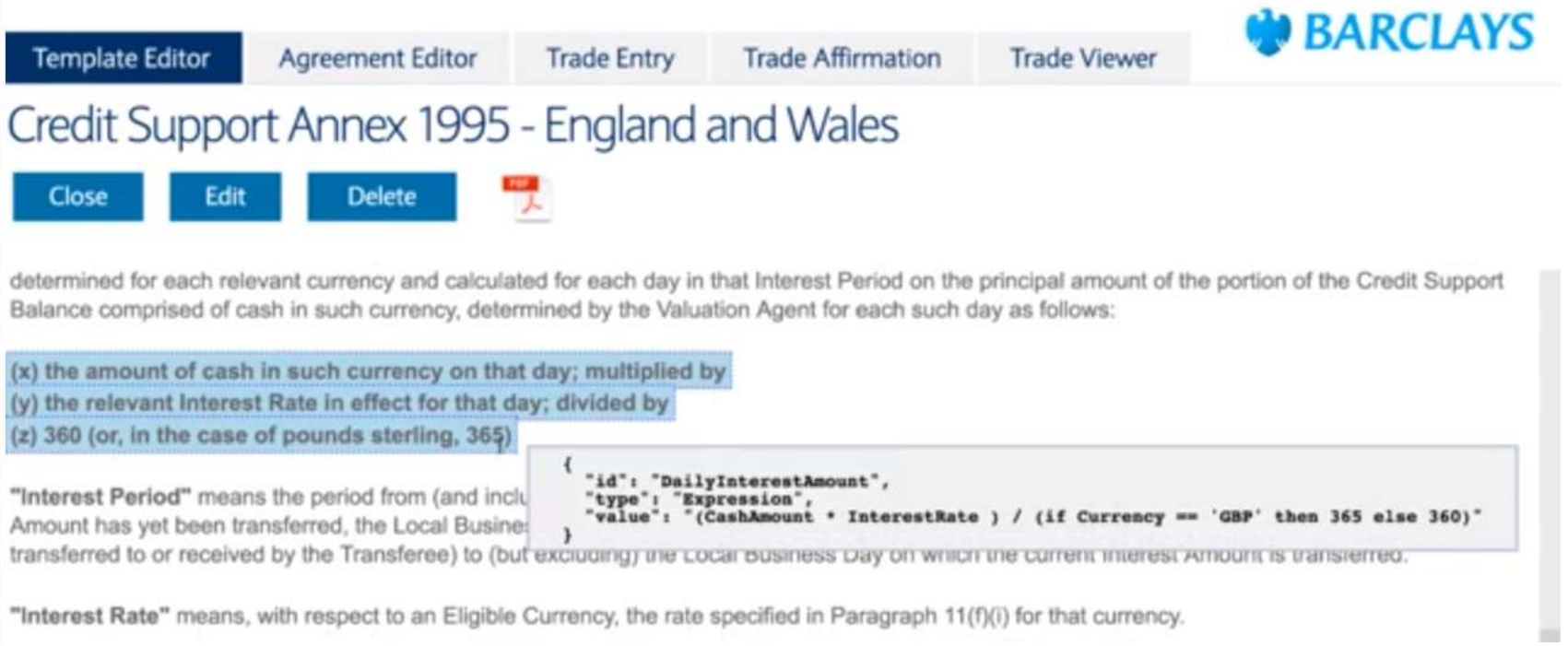
Picture source: https://en.wikipedia.org/wiki/Ricardian_contract#/media/File:RC-bow-tie.png

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



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)**

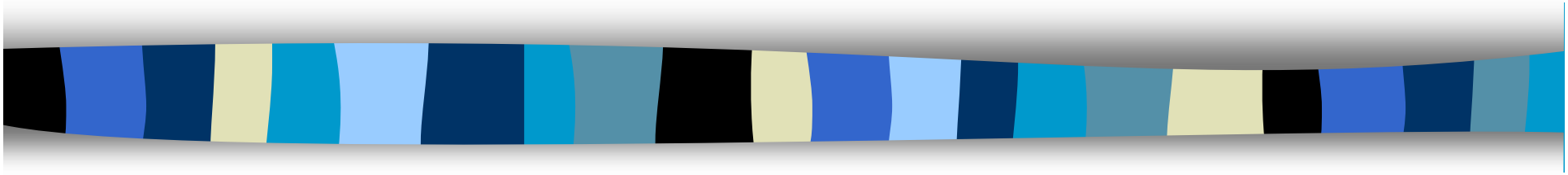
The official development environment for programming in Solidity is:

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