Endorsement Delegation With Smart Contracts

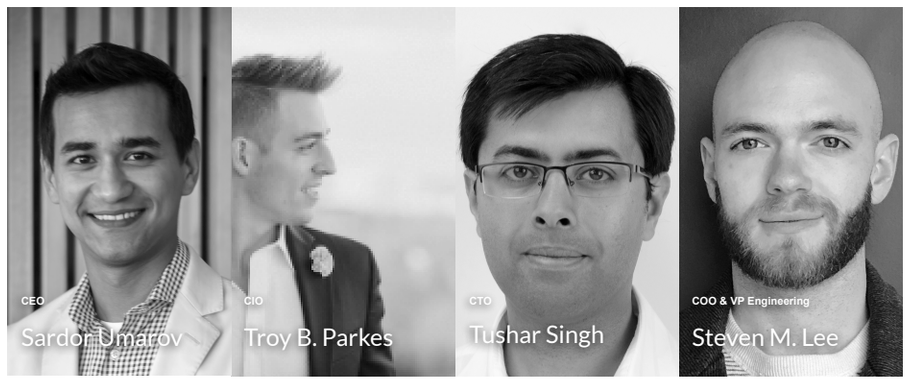
Tushar Singh
2018/09/22
EthAtlanta

Who am I?

Minute School

Tushar Singh, CEO/Founder of Minute School, ed-tech startup based out of Waterloo, Ontario. We’re in one of the world’s premier startup hubs with ~1000 startups in a region of half a million people.
This is my 10th startup and 4th ed-tech startup in the past 20 years.
Our focus is on mobile peer to peer education.

BookLocal

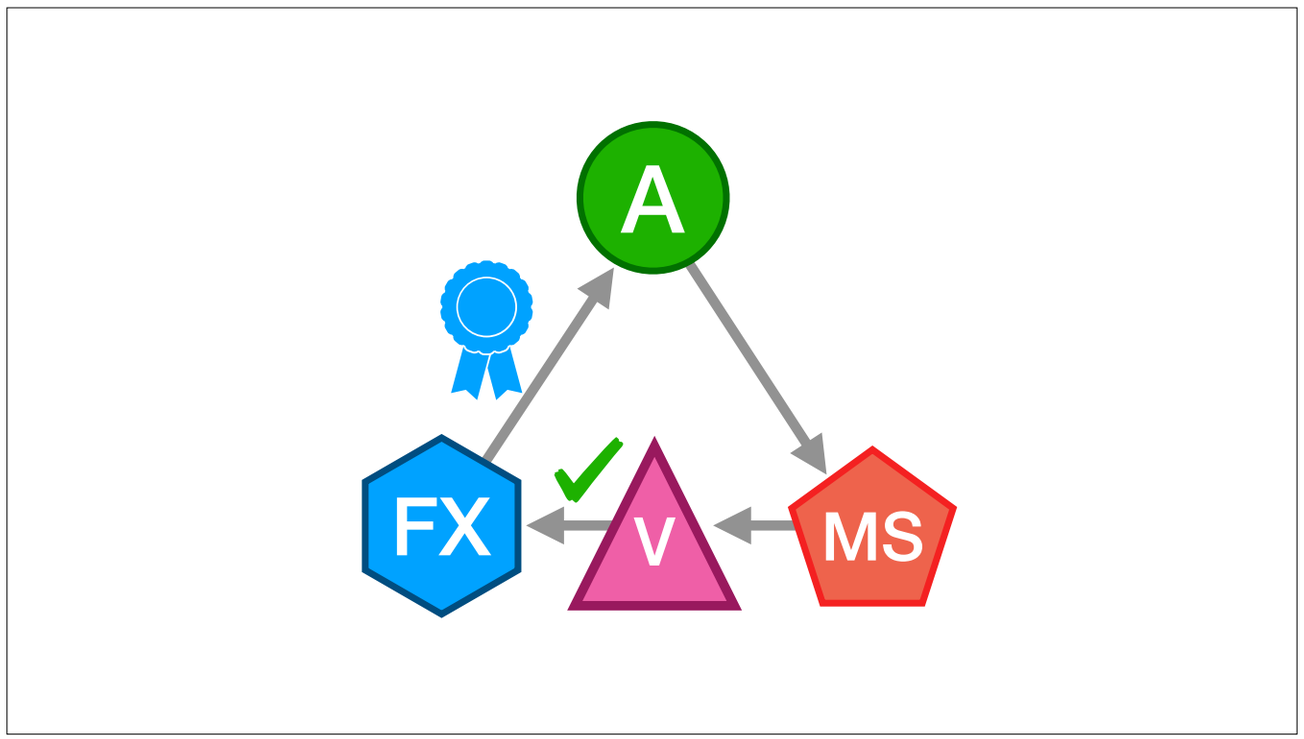


I am also the CTO for BookLocal which is the world’s first Direct Booking Aggregator. We enable hotels to tokenize, manage, and rent their room inventory with the security of the Ethereum network.

Four of the executive team are here at EthAtlanta and we were part of the team that helped deliver EthMemphis. We are a group that is passionate about blockchain technologies and how they already are and will continue to transform our lives.

What is endorsement delegation?

I did not want to use the word credentialing in the title for this presentation even though that is what is about. Rather, I want to speak about endorsement as it is far more pervasive and applicable in education, the hotel industry, governments, sports, etc. This process of endorsement and delegating endorsement is ubiquitous and is worth understanding through a few use cases.



For the first example let’s look at, simplistically, how a credential in education is awarded. Meet “A”, a student taking a blockchain development course from FX (perhaps from the FedEx Institute of Technology) on the MS platform.

Upon successful completion of the course FX issues a certificate to A. This is probably a very familiar process to many of you.

Seems pretty simple but let’s dig into this process a little bit more to understand the relationships.

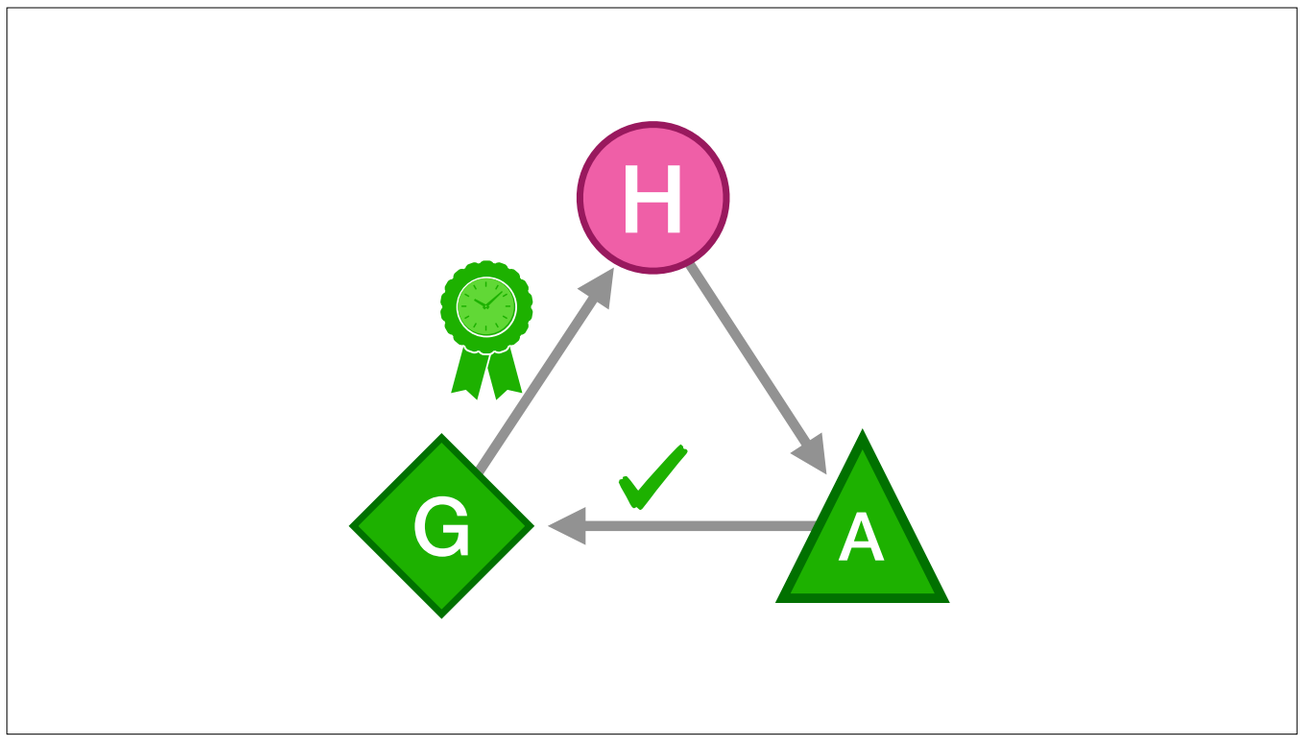
Let’s start backwards from the certificate. What does the certificate mean? At the base level a certificate or credential is an endorsement of the knowledge, skills, or abilities (KSA) of the recipient. FX endorses A for having knowledge of Solidity smart contracts, the skill to write Solidity smart contracts, and the ability to solve a problem using Solidity smart contracts.

This is important as A is able to present this credential to others and they will view it as an endorsement.

But there’s a problem here. We’re masking over a very important participant to this endorsement: who verified that A had the requirements for the certificate? Who evaluated A? Who told FX that it was ok to issue the credential?

These are important questions as FX does not issue a credential by itself, rather, it must be told by a trusted verifier that A meets the conditions for the credential.

Let’s add V to the process so we can better visualize the flow: A takes a course on MS, that activity is evaluated by V, who then verifies that the conditions for the credential are met, at which point FX issues the credential.



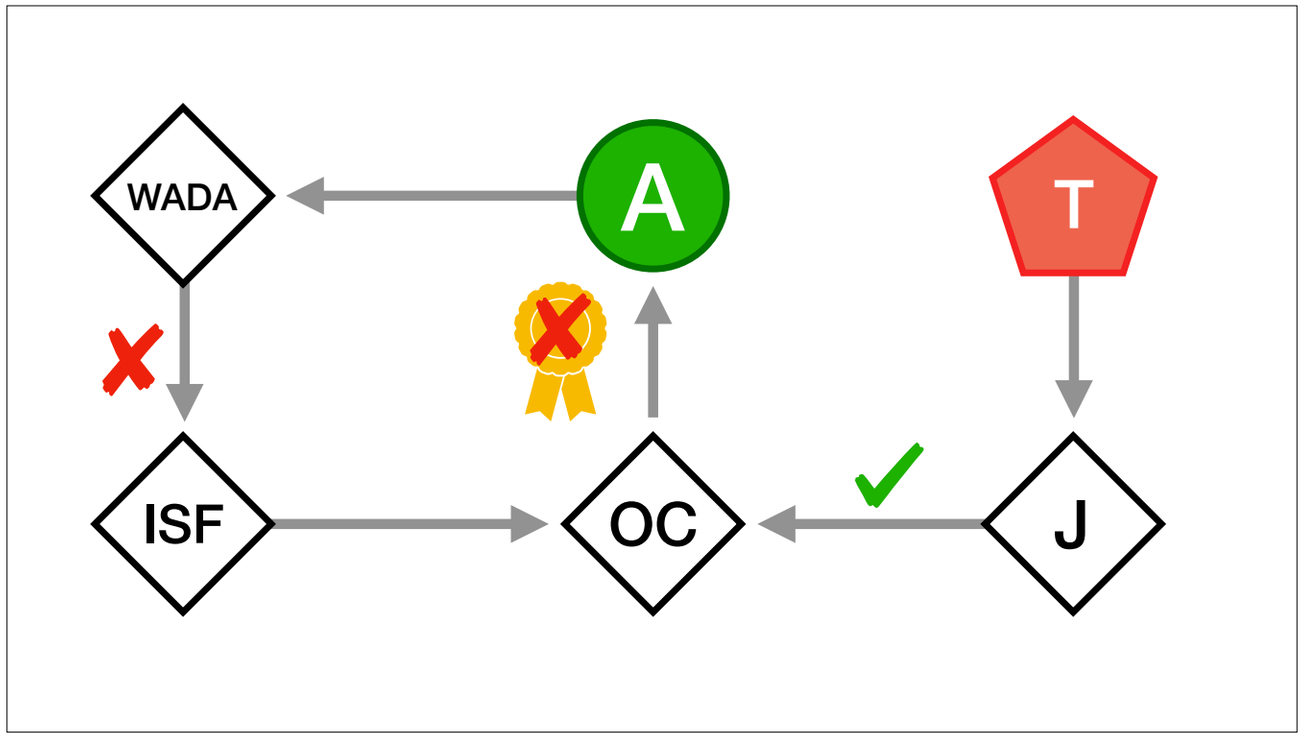
Let’s switch gears to the hotel and tourism industry.

Sustainable tourism is increasing in demand and it is important to tourists that hotels are eco-friendly/sustainable. There are several standards that a hotel can obtain, we’ll use one as an example and call it G.

In this scenario hotels receive certification if they meet 50% of criteria. In addition, if hotels remain certified for 5 years they receive gold status, and if 10 or more then they receive platinum status.

G does not directly verify hotels, rather they use trusted third party auditors who verify and pass on their assessments to G. If the standard is deemed to have been met by the auditors then G issues a certificate.

This certificate also has an expiry date associated with it. This requires annual auditing to ensure standards are met in order to maintain status and build up to the higher levels.



Let’s have a look at another example, something a bit more complex: sports competitions.

In particular let’s look at the 100m race. When the race starts a timer run by a third party starts, as athletes cross the line they receive times from the official timer, based upon that result the athletes receive medals.

Much like in other cases a trusted third party verifies times and pass it on to the judges from the IAAF who certify the result and pass that on to the Olympic Committee who are then actually the ones that award the medals.

However, that’s not the end of the story. Athletes are also tested constantly by the World Anti-Doping Agency (WADA). This is continuous verification and the results are passed onto the various International Sports Federations (ISFs). In case of a failure an ISF may retroactively disqualify athletes and change results which are, in most cases, implicitly endorsed by the Olympics.

This implies that endorsements can be withdrawn retroactively as well.

Baseline Understanding

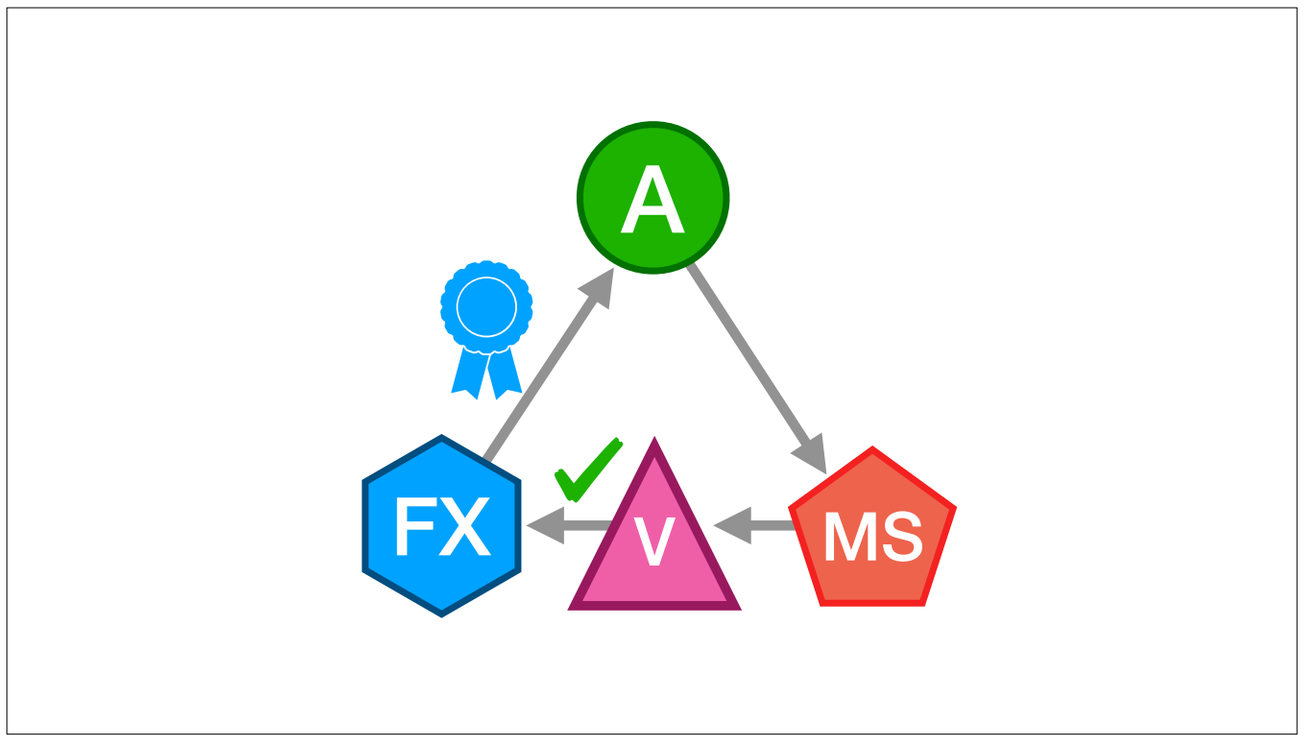
- Endorsement Delegation:
  - common
  - delegate verification
  - allows scaling
  - multiple verifiers
- Credentials:
  - awarded by endorser
  - can expire
  - can be revoked

We should all now have a reasonable understanding of endorsement and some requirements. Requirements are important for any product that we want to build in order to satisfy market needs.

Here are some key points on the patterns that we have seen:
* Endorsement delegation is quite common.
* It is used by credentialing entities to delegate verification.
* The resulting delegation allows scaling.
* There can also be multiple verifiers.

We should also recognize a few points about credentials:
* They are awarded by the endorser, not the verifier.
* They can expire.
* They can be revoked.

This forms our baseline for how to replicate this with smart contracts.



Let’s go back to our first use case with issuing a credential for course completion. With a smart contract we can implement much of that process directly within a single contract that some parties interact with.

```
pragma solidity ^0.5.0;

interface Endorsement {
    enum State{None, Endorsed, Expired, Revoked}
    enum RevocationReason{Preemptive, VerifierError, Cheating, Disassociation}

    function by() public pure returns(address);
    function description() public pure returns(string);
    function has(address who) public view returns(State,uint256);
    function grant(address who) public;
    function revoke(address who, RevocationReason why) public;
}
```

The endorsement contract is simple. Note that we're using solidity 0.5.0 here which has yet to be released. We're using it in this example as it allows us to include enumerators in interfaces.

```
pragma solidity ^0.5.0;

interface Endorsement {
    enum State{None, Endorsed, Expired, Revoked}
    enum RevocationReason{Preemptive, VerifierError, Cheating, Disassociation}

    function by() public pure returns(address);
    function description() public pure returns(string);
    function has(address who) public view returns(State,uint256);
    function grant(address who) public;
    function revoke(address who, RevocationReason why) public;
}
```

These first enum provides states for the endorsement while the second provide revocation reasons.

Our reasons include that we made errors: preemptive granting, that the verifier made an error.
Or that the receiver made bad decisions by cheating or untowardly behavior that the endorser does not want to be associated with.

```
pragma solidity ^0.5.0;

interface Endorsement {
    enum State{None, Endorsed, Expired, Revoked}
    enum RevocationReason{Preemptive, VerifierError, Cheating, Disassociation}

    function by() public pure returns(address);
    function description() public pure returns(string);
    function has(address who) public view returns(State,uint256);
    function grant(address who) public;
    function revoke(address who, RevocationReason why) public;
}
```

Moving on we have the set of functions that provide information.

Mainly, who is the endorser and a description of this endorsement.
The ‘has’ function returns whether a particular address has been endorsed and when that endorsement occurred. It can also return that the Endorsement was revoked.
Retrieving the reason for revocation is not included in this interface but it can be retrieved through other means.

```
pragma solidity ^0.5.0;

interface Endorsement {
    enum State{None, Endorsed, Expired, Revoked}
    enum RevocationReason{Preemptive, VerifierError, Cheating, Disassociation}

    function by() public pure returns(address);
    function description() public pure returns(string);
    function has(address who) public view returns(State,uint256);
    function grant(address who) public;
    function revoke(address who, RevocationReason why) public;
}
```

The final set of functions are those that allow granting and revocation of endorsement.

grant endorses ‘who’.
revoke removes endorsement from ‘who’ while also providing a reason.

This all seems simple and as an abstraction it should be simple. Delegation is dependent upon the implementation of the grant and revoke functions.

For example, we could have code that says if the caller is any of a particular set of verifier addresses then set the endorsed state.

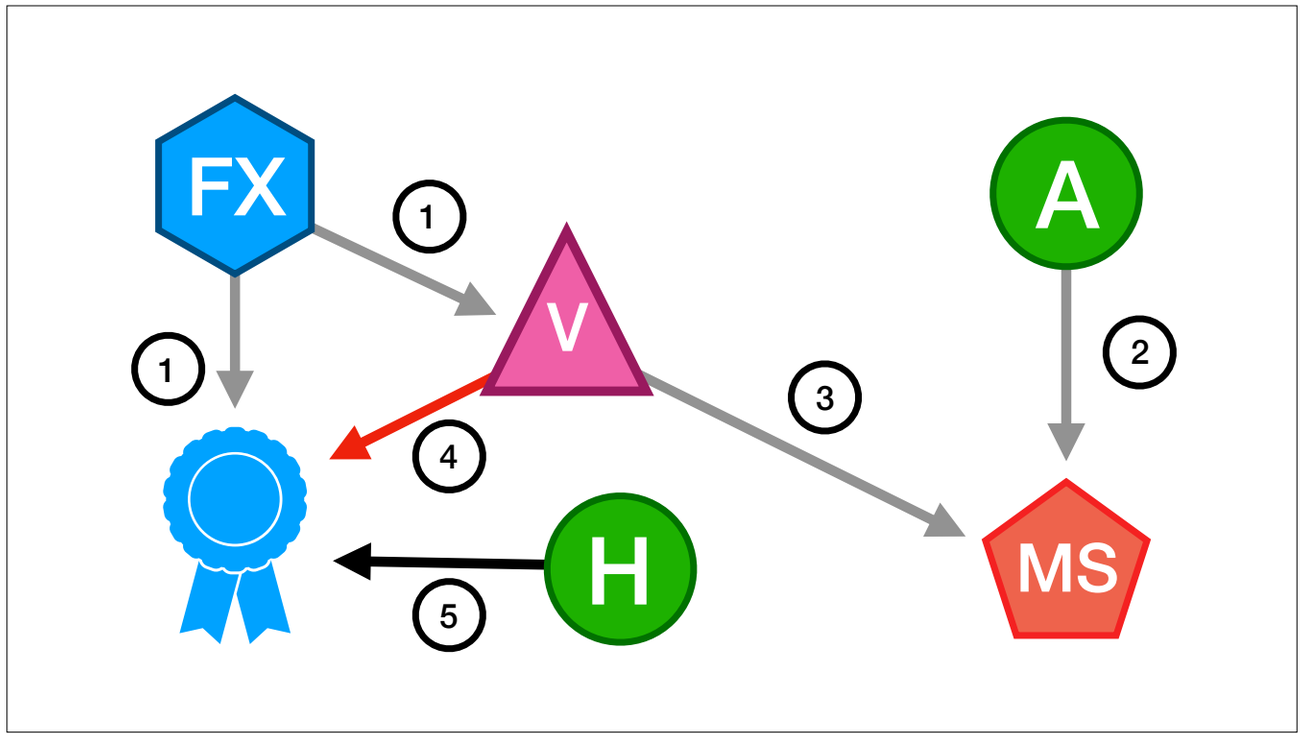
Alternatively we could say it has to be all verifiers.

Or 75% of verifiers.

The rules regarding granting are specific to the endorsement.

The same goes for revoke. You could have a set of verifiers that are only involved in revoking rather than granting. This is dependent upon the Endorser.

Now that we have an understanding of an Endorsement contract let’s look at our use case again.



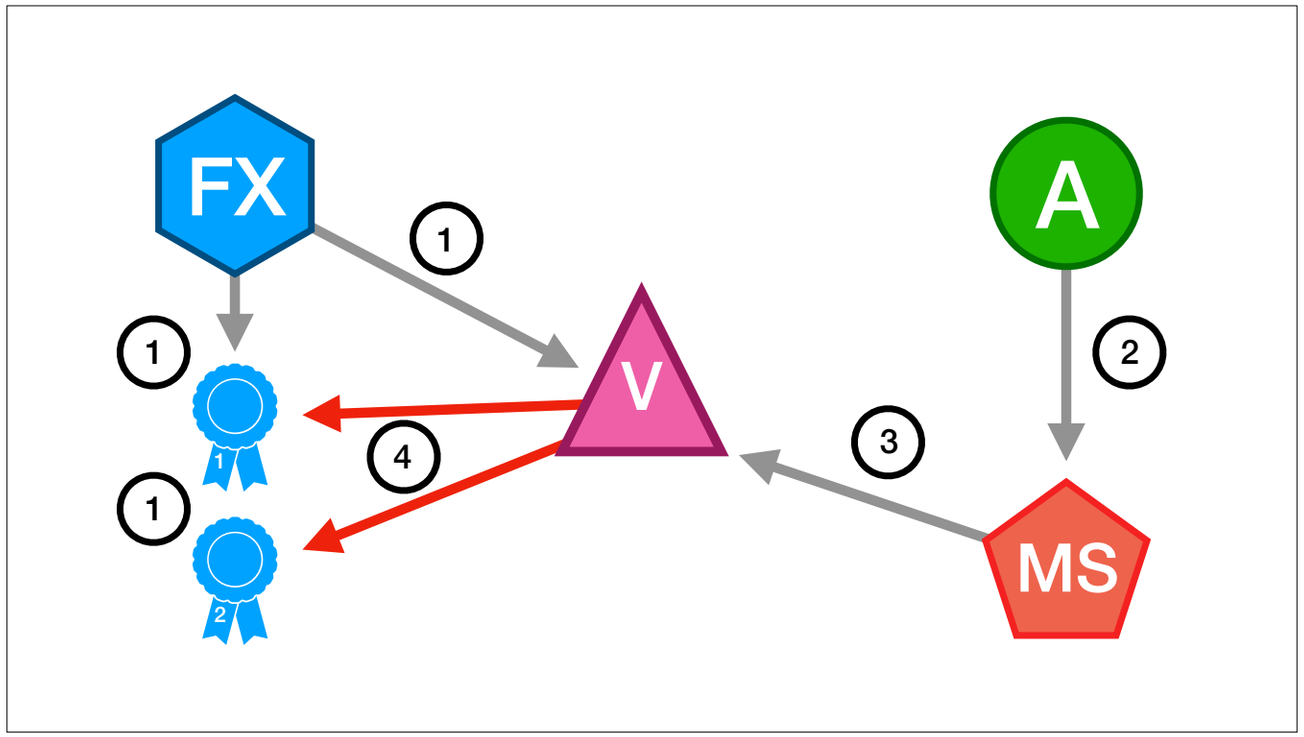
I have structured this to walk through the process:

1) FX creates a credential Endorsement contract that V can verify.
2) A engages with course material on MS.
3) V examines activity on MS to verify whether A meets FX's standards for the credential.
4) Satisfied that requirements have been met V calls the 'grant' function on the contract.
5) H, HR at A's company, checks the credential to see if A 'has' it and gives a raise accordingly.

This is a simple example. Larger more complex use cases have a few problems that need to be addressed.

1) How do you handle granting of multiple endorsements from a single course?
2) How do you handle granting of a degree or diploma that has multiple requirements?
3) How do you handle revocation?

Let's take a look at how this is structured.



FX creates two credentials and delegates granting to V. A takes the course on MS, V verifies A's KSA as it relates to both credentials and then grants them.

Seems pretty reasonable but what if FX adds a 3rd credential? Or a 4th? The onus is placed on V to keep up to date with that. This is also the simple case of this more complex scenario as there could be several verifiers. This is a logistical nightmare and we need to make it easier for V.

In order to do that we need to introduce events.

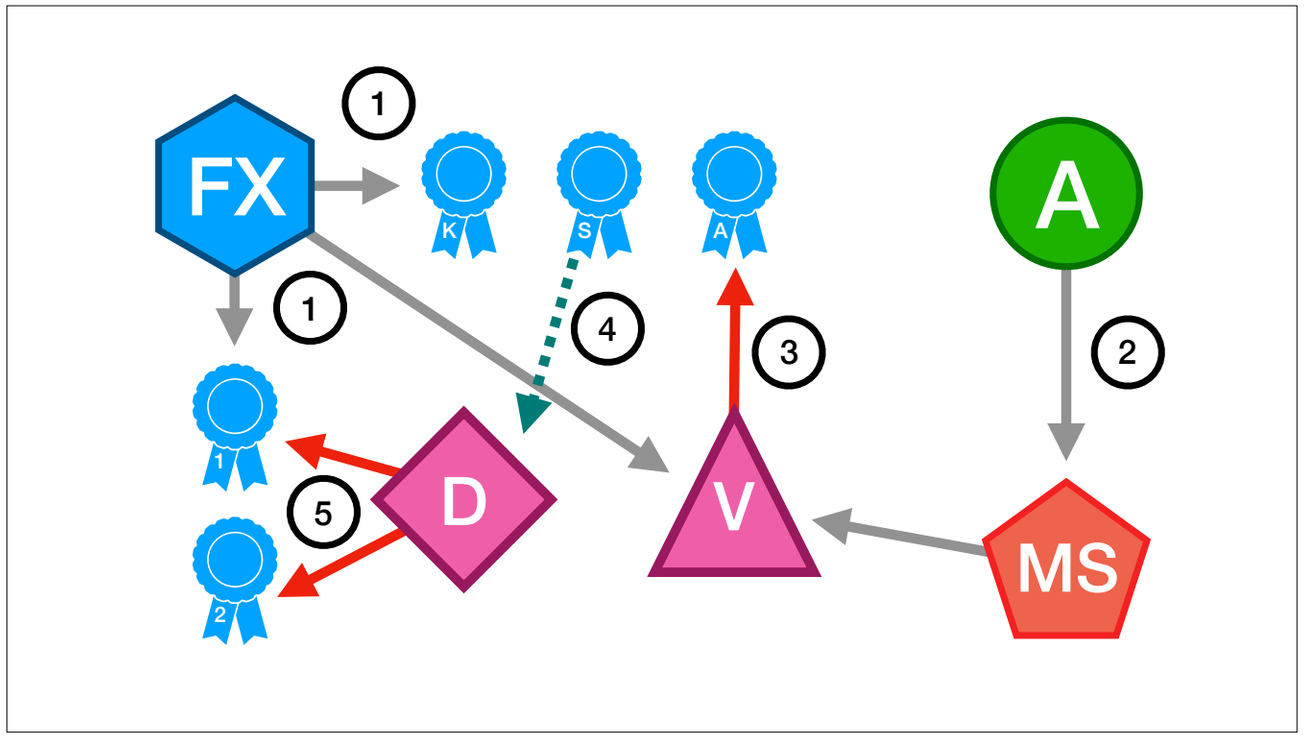
```
pragma solidity ^0.5.0;

interface Endorsement {
    enum State{None, Endorsed, Expired, Revoked}
    enum RevocationReason{Preemptive, VerifierError, Cheating, Disassociation}

    function by() public pure returns(address);
    function description() public pure returns(string);
    function has(address who) public view returns(State,uint256);
    function grant(address who) public;
    function revoke(address who, RevocationReason why) public;
}

// Events (these would normally go in a contract)
event Granted(address who);
event Revoked(address who, RevocationReason why);
```

Recall the grant and revoke functions in the interface. Whenever each occurs we should fire an event. In our case we are going to provide globally available and consistent events but they can be custom to each implementing contract.

The first event is fired whenever a grant call grants the endorsement. It contains who was granted the endorsement.

The second event is fired whenever an endorsement is revoked. It contains the reason for revocation.

Note that these events are not fired on every call of grant or revoke but when the contract logic indicates that an Endorsement was granted or revoked.



Ok, so what does this look like now? Well, perhaps a bit different as we want this to be scalable.

Rather than having a single endorsement or even a couple we need to be focused on what the verifier is looking at.

1) FX creates the higher level endorsements but also creates several endorsements for each KSA (Knowledge, Skills, and Ability) and delegates V with the ability to grant them.

2) A takes a course on MS.

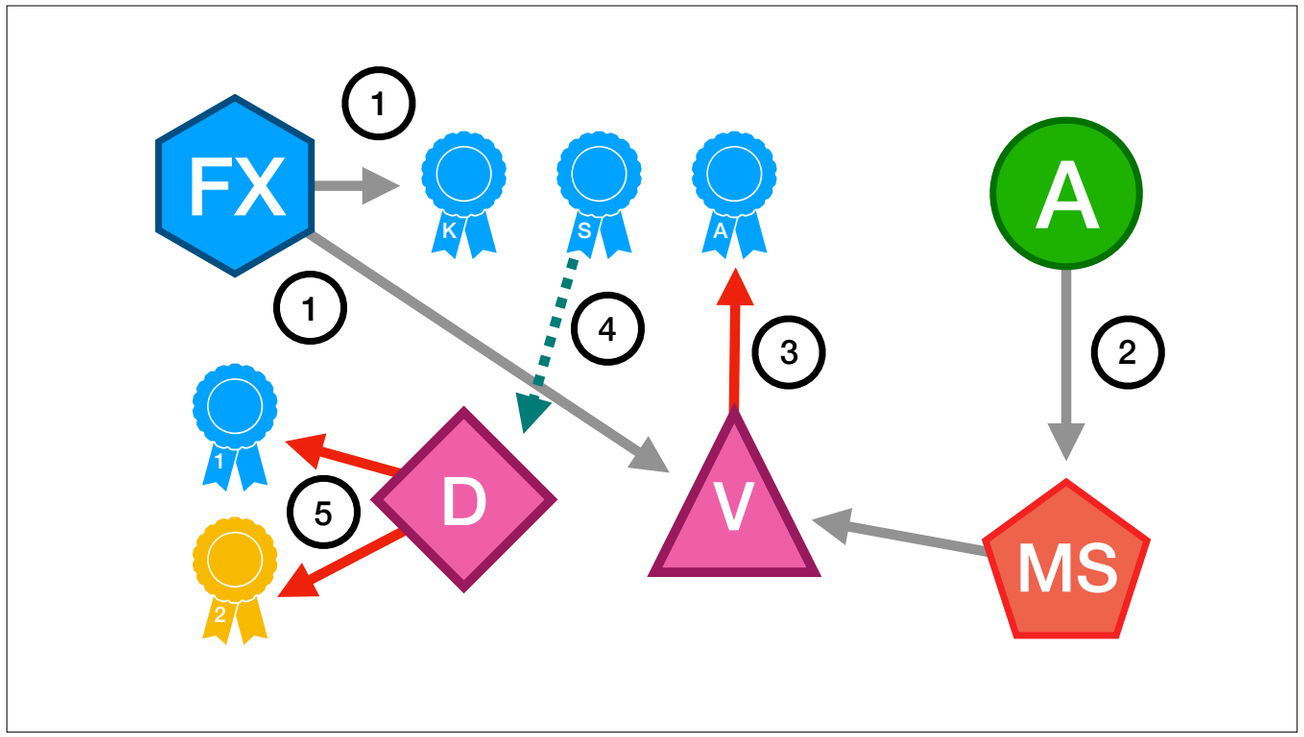
3) V now grants the KSA endorsements based upon their evaluation of A's activity on MS.

4) What happens at this point is that each of the KSA endorsements will each emit a Granted event.

Those events can be picked up by a system monitoring events, in this case D.

D has also been delegated by FX to grant endorsements but D is entirely virtual. A service that subscribes to web3 events for particular endorsements and calls grant on other endorsements.

5) Now, D can evaluate the events and manage logic on what particular endorsement to grant when a particular combination of KSA are granted. But that's not transparent. Instead, let's put the logic into 1 and 2 themselves.



Also, a more typical architecture would allow MS to assess knowledge directly and it could act as a verifier as well. It is up to FX and any other delegating entities to determine whom to delegate endorsement to.

What’s next?

- Source Code
- Documentation
- Reference Code
- LiveNet Deployment
- Data Privacy

With regards to this system, what’s next? There are a number of initiatives underway with regards to credentials. We are going to participate in the ecosystem with a release of interfaces and implementing smart contracts.

Up next we will publish the source code, update documentation, including scoring when granting, provide reference contracts, factories, and listeners. We will also start issuing credentials through this system with our partners and encourage others to do so as well.

Finally, we will provide contract addresses so that others can interact and utilize these contracts without needing to setup their own full scale infrastructure.

Thank you

Twitter: @tusharsingh

Thank you for your time, please feel free to connect with me on this if you’d like more details.