Whitepaper

T-REX (Token for Regulated EXchanges)

The token standard allowing ownership transfers of tokenized securities

Version 1.0.1. - December 17th, 2018
The source code is available on Github: https://github.com/TokenySolutions

TABLE OF CONTENT

*****

Contributors:
Luc Falempin,
Fabrice Croiseaux,
Joachim Lebrun,
Dilip Chouhan,
Philippe Van Hecke
Antoine Detante

Executive summary

Following the emergence of Bitcoin and other so-called crypto-currencies, the last two years have seen through a wave of ICOs (Initial Coins Offerings), leveraging on the DLT technology underpinning most cryptocurrencies to support the issuance of other types of instruments. This wave has mainly seen the issuance of utility tokens[1] in a completely unregulated environment. More recently, we have seen a new type of token emerging in the form of security (or investment) tokens which, in essence - and a number of regulators have started to confirm that - should be assimilated to securities i.e. equivalents to traditional securities but which are issued, maintained and transferred on a DLT infrastructure. One of the most important features that security tokens bear is, contrary to utility tokens, the fact that existing securities laws and practices should be considered as applying to them and, among others, all requirements in terms of KYC and AML regulations. These regulations are there to control who holds a security and who transfers it in order to detect and prevent money-laundering, terrorism financing and other illegal or fraudulent activities.

The main goal of the T-REX standard is to create a set of global tools, fully based on blockchain technologies, to allow for the frictionless and compliant issuance and use of tokenized securities on a peer to peer basis or through marketplaces. These tokens are issued in full compliance with regulations and issuers requirements, as control mechanisms are embedded in the tokens themselves. With T-REX, we are implementing a "Compliance by Design" approach where it is simply impossible for an investor to buy a security without being compliant. The regulator itself can verify the compliance of the Issuer through the auditing of smart contracts that support the Security Token life cycle.

[1] In this document, we will refer to:

- **Utility tokens**: are a representation on the blockchain of a specific right to goods or services that the issuer of the token has created or is in the process of creating and. We will generally restrict the usage of the term "ICO" to designating the issuance of utility tokens;
- **Security tokens** (or, more broadly, **investment tokens**): are assets representing an expectation of (and a claim on) future cash flows (other than a simple market price increase) resulting from the activity of the issuer. They can be assimilated to "traditional" securities and represent debt/loans, equity or investment funds for instance (They can further represent securities assets as land, real-estate, planes, ….). We will generally use the term STO - Security Tokens Offering - to designating the issuance of security tokens.

The management of compliant transactions through T-REX backed permission tokens will be based on 3 main pillars creating a **decentralized Validator:**

- A blockchain based **identity management system**, allowing for the creation of a globally accessible identity for every stakeholder.
- A **set of validation certificates** (technically speaking, these certificates are the claims, described in the ERC-725 and ERC-735 standards, that will be described further in the document. For a better understanding, we will name these as certificates in this introductory section as it has more semantical sense.) emitted by trusted third parties and signed on-chain, each of them linked to a single identity.
- A **transfer manager** whose role is to act as a filter of all the transactions of tokenized securities and will check the validation certificates of the stakeholders Essentially, the transfer manager will check that the receiver has the rights to receive the tokens following the specific compliance rules and issuer requirements applicable for this specific asset. The transfer manager will block the transaction if the receiver misses a mandatory certificate and will notify him about the reason of the failure.

These 3 key elements allow issuers to use a **decentralized Validator to control transfers and enforce compliance on the holders of the security token**. The Validator includes rules for the whole offering (e.g. managing the max number of holders allowed in a specific markets, when such rule apply), and rules for each investors (e.g. KYC or issuer-defined eligibility criteria) thanks to the identity management system.

Constraints for tokenized securities

Although, so far, the rules applicable in issuing and holding utility tokens have been largely undefined - or at least very vague - in most countries, an STO consists in the issuance of a security that uses the blockchain technology as its registry, proof of ownership and transfer infrastructure. Such instrument is regulated in every country and, as a consequence, STOs have to comply with the related regulations of the country where the security token is issued in as well as those of the countries where it is distributed (sold).

	Utility Token	**Security Token**
Purpose	Usage	Investment
Regulation	Non existing or vague in most cases	Stringent as existing securities laws should be taken as reference.
Lifecycle	Simple	As complex as a security
Secondary Market	Nearly no constraints	As complex as a security

Figure 1 : comparison between utility and security token

Another significant difference between ICOs and STOs is related to the token lifecycle. ICOs - dealing with utility tokens - result in the issuance of tokens that have a relatively simple life cycle: once the token is shared among a decentralized network, its governance is mostly the results of its token economics. Contrary to security tokens, the situation is quite different, as the issuer - or its appointed agent - remains generally liable for applying a number of controls to its token after issuance and during the entire "life" of its security token. In addition, the issuer might need to apply a number of corporate actions (dividend/interests payments, …) or corporate events (calling for an AGM/EGM, …) to its token which further increase the need for the issuer to keep in touch with (keep some control on) their investors.

One could identify two main types of control requirements related to the issuance, the holding and the transfer of security tokens :

- One relates to regulations applicable to the security considered, that are independent of the security token itself (i.e. general rules). For example, the need to identify the investor, to collect proof of their identity, to check their name against blacklists, i.e. generally speaking, control requirements related to AML/KYC, or other applicable regulatory rules.
- Then some controls might be related specifically to the security that is issued, for example, restrictions about the investor type and location or about the amount of money that can be

invested in a certain period. These might be linked to the regulatory environment under which the issuer has decided to issue their token or simply linked to eligibility criteria defined by the issuer for instance, for commercial reasons (e.g. restricting the access of a certain share class, having specific fees characteristics, to investors of a specific country).

Addressing these different control requirements will require a high level of reusability and flexibility when designing the token.
This is the reason why we have designed the T-REX standard. It provides a set of generic tools helping token issuers apply and manage the necessary controls and permissions to security tokens through a flexible decentralized validation system (the transfer manager), allowing them to add all the rules that need to be addressed to approve holding and transferring their tokens.

Decentralized Validation System

Overview

A transfer of ERC-20 tokens generally happens in the following way on Ethereum blockchain, with the standard ERC-20 implementation:

Figure 2 : illustration of an ERC-20 transaction

Transactions are executed between 2 peers, without any restrictions and without any control, the freedom of transaction is complete and pseudonymous, AML/KYC checks are only performed when crypto is converted into fiat currencies or when fiat currencies are converted into crypto and there are a lot of ways to avoid these through unregulated exchanges, direct exchanges between peers, etc.

In comparison, a transaction in a T-REX compliant ERC-20 permissioned token will be processed as follows:

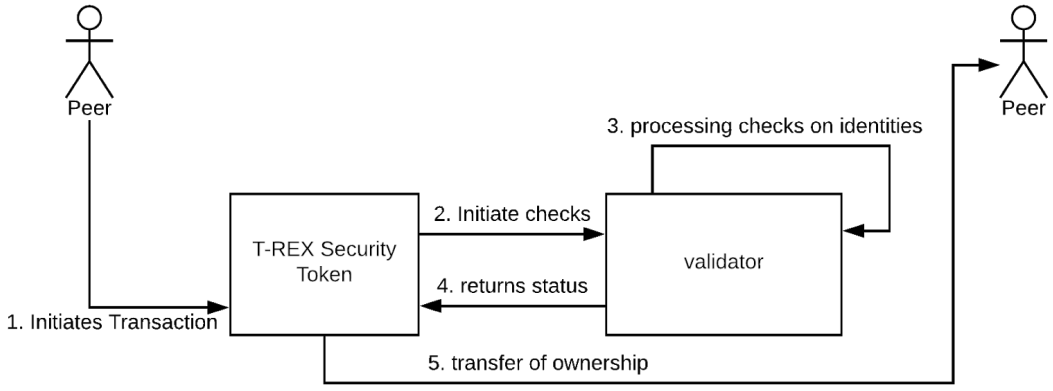

Figure 3 : illustration of a T-REX permissioned token transaction

The T-REX security tokens allow transactions between peers after a complete compliance check. The holder of the tokens initiates the transaction through the security token smart contract (1.). In contrast to a standard ERC-20 token, the transaction function of the smart contract is modified, the transfer function of the smart contract calls the validator (transfer manager + identity registry + trusted claim issuer registry + trusted claim types registry + claim verifier) (2.) to initiate checks on the identity of the receiver to ensure that his identity holds the necessary permissions (claims) to receive the token in question (3.). If the receiver identity holds the required claims (personal data validated by trusted third parties (e.g. KYC, AML, sovereign identity,...), the transfer of the tokens is validated (4.) and executed (5.). If he doesn't have the necessary claims in his identity, the transfer is rejected and an error message is delivered to explain the necessary steps to get the missing claims (4.).

Permissioned tokens

In our opinion, only permissioned tokens are suitable to issue security tokens because there cannot be a total, uncontrolled, freedom in the transaction of such instruments and, investors need to comply with a number of criteria - either by regulation or imposed by the issuer himself in order to be eligible for holding the tokens. The main technical difference between standard ERC-20 tokens and T-REX permissioned tokens resides in the transfer function of T-REX tokens being made conditional, the condition for a transaction to be executed being that the transfer manager approves it in accordance to the governance criteria defined for the token in question. However, despite this modification of the transfer function of the token, it is to be highlighted that, because the token structure is based on the ERC-20 standard, it remains fully compatible with all the available exchanges and tools based on ERC-20 tokens.

Most of the "Security token protocols" promoted in the industry so far are permissioned tokens. The transfer function is modified and requests a transfer approval from an external validator service to control the transfer of tokens. T-REX involves a fully on-chain identity management system allowing issuers to directly control the transfer of ownership.

On-chain identities management

As mentioned before, by essence, a security token being subject to a stringent governance, its distribution has to follow all the applicable regulations and, in particular, those aspects related to KYC rules. In that respect, we believe that identity management is key to implement such compliance on the blockchain.

As the ownership of a security token is registered on the blockchain, we believe it is necessary to have a way to track the token ownership and to prohibit illicit transactions directly on the blockchain. This is why there is a need to make a link between wallet addresses and identities to manage rights through an identity contract directly on the blockchain. In addition, we also need to ensure privacy of those identities

in order to comply with personal data related regulation. For this reason, personal data should not be stored directly on the blockchain but only the validation certificates (claims) issued by trusted third parties (KYC provider, government, lawyer,…) having checked these data. Those certificates (claims), stored in the identities of each party will be used by the transfer manager to validate whether or not those parties can hold or transfer a specific security token.

Linking an investor's wallet to his identity can bring significant added value to stakeholders in the nascent security tokens market. For example, it will allow a token issuer to replace the tokens of an investor if the investor loses access to his wallet (which happens quite often and generally results in the loss of the owner's assets[2]), by verifying that his on-chain identity fits with off-chain data linked to the identity contract corresponding to the lost wallet. After the identity of the investor is confirmed, the issuer can burn the lost tokens and mint new tokens on the new wallet of the investor.

Also, on-chain identities and the certificates (claims) they store can potentially be re-used for passing KYC's for other security tokens than the one for which those claims were originally provided or even in situations other than investments (e.g. account opening at an exchange, identification with compatible web services, …). If Google and Facebook accounts are the identities of most people on the internet of information, on-chain identities can be the ones of the internet of value. They are genuinely owned and controlled by their owner.

[2] http://www.fortune.com/2017/11/25/lost-bitcoins

T-REX infrastructure

Overview

The goal of T-REX is to provide a complete suite of tools for the issuance, management and transfer of security tokens on the Ethereum blockchain. The following section will dive into the technical specificities of T-REX by explaining the different capabilities of the smart contracts that are part of it. In addition, there are other smart contracts to handle corporate actions, taxes, etc. that can easily be added to the suite of tools.

Based on standards

Token standards

ERC-20

ERC-20 tokens[3] are considered as a standard and are widely adopted by all the actors in the blockchain industry. ERC-20 tokens are fungible tokens, usually non-permissioned, that can be transferred easily between peers on the Ethereum blockchain.

The ERC-20 smart contract defines and implements the basic characteristics of the token i.e. its name, symbol, the total supply and the number of decimals allowed. The ERC-20 smart contract implements all the necessary functions needed to create a standard token, as seen in Appendix A.

Identity standards on the Blockchain

Performing KYC and AML checks is a must for any project that intends to follow proper governance and comply with regulations. ICOs (by good practice) and even more STOs (by regulations) are required to integrate strong KYC and AML procedures. In our opinion, performing KYC and AML checks effectively on a blockchain infrastructure requires to link these with a shared identity model. In order to do so, we have decided to rely on ERC-725[4] and ERC-735[5] standards. These standards define a commonly accepted identity and claims management model for creating, maintaining and populating identities on the blockchain. Using these standards, we have built features into our smart contracts to only permit interactions from well identified, reputable persons (i.e. natural or legal persons can have identities). The claims attached to an identity contract facilitate the emergence of a web of trust between

[3] https://github.com/OpenZeppelin/openzeppelin-solidity/tree/master/contracts/token/ERC20
[4] https://github.com/ethereum/EIPs/issues/725
[5] https://github.com/ethereum/EIPs/issues/735

token issuers, 3rd party claim issuers and the party holding the identity contract itself and can make the process of identity management more efficient.

ERC-725: Identity

ERC-725 describes standard functions for a unique identity for humans, groups, objects and machines. This identity can hold keys to sign actions (transactions, documents, logins, access, etc), and claims, which are attested from third parties (claim issuers) and/or self-attested, as well as a proxy function to act directly on the blockchain.

ERC-725 is a standard for publishing and managing an identity via a smart contract on the Ethereum blockchain. One can say it acts as a type of identification card for a particular address.

We can associate several keys with an identity which can have the following functions::

- MANAGEMENT keys, allowing to manage the identity.
- ACTION keys, allowing to take actions using the identity (signing, login-in, initiating transactions, etc).
- CLAIM i.e. a signer key, used by claim issuers to sign claims on other identities. These claims are revocable
- ENCRYPTION keys, used to encrypt data (e.g. to encrypt the data hold in claims).

Keys are keccak256's of public addresses, and as such map to externally owned accounts (wallets) or smart contracts.

Importantly, an ERC-725 identity contract can be used as a proxy, through its execute/approve methods. If a third party contract recognizes the legitimacy of a given identity contract, its owner can "execute" on it.

Multiple keys are particularly useful in the context of privileged access. As an example, an identity represents a legal entity (organization) and members of this legal entity holding less privileged keys can submit transactions on behalf of the legal entity identity. However, only those with access to higher-privilege keys can approve the execution of those transactions.

ERC-735: Claim Manager

A claim is specific information a claim issuer has about the identity holder. Such claims have the following characteristics:

- **Type:** A uint256 number which represents the type of the claim. (e.g. 1 biometric, 2 residence, 3 KYC checked for a specific token, and so on).
- **Scheme:** The scheme with which this claim SHOULD be verified or how it should be processed. It's a uint256 for different schemes. E.g. 3 could mean contract verification, where the data will be call data, and the issuer a contract address to call. Those can also mean different key types e.g. 1 = ECDSA, 2 = RSA, etc.
- **Claim Issuer:** The claim issuer's identity contract address, or the address used to sign the signature described hereunder. If an identity contract, it should hold the key with which the above message was signed, if the key is not present anymore, the claim SHOULD be treated as invalid.
- **Signature:** Signature which is the proof that the claim issuer issued a claim of topic for this identity. It MUST be a signed message of the following structure: keccak256(address identity Contract, uint256 _ type, bytes data).
- **Data:** The hash of the claim data, sitting in another location, a bit-mask, call data, or actual data based on the claim scheme.
- **Uri:** The location of the claim, this can be HTTP links, swarm hashes, IPFS hashes, and such.

ERC-735 describes the standard functions for adding, removing and holding claims. These claims can be attested by third parties (claim issuers having a certain legitimacy to sign the considered claim) or self-attested. A claim issuer who issues claims about an identity, does have itself and is represented by an identity contract that has a claim signer key used to sign the claims in question.

This standardized claim manager interface will allow Dapps and smart contracts to check the claims held by a claim holder through his identity. In that respect, we can consider that trust is transferred to the issuers of claims (he is the one assessing a specific entity/claim holder and, based on this assessment, issuing a claim to his identity). In summary, **ERC 735 deals with the management of claims issued to qualify an ERC 725 identity**.

These **identity claims can be of several types, from biometric data to email account ownership**. Depending on the use case, the appropriate qualifying claims can be defined.

In order for an identity holder to have a claim added to his identity, he first has to request it from a relevant trusted third party (i.e. the relevant trusted claim issuer for this specific type of claims). The claim issuer, upon successful assessment of the eligibility of the requestor for the claim considered, will then sign a message containing: a) the (requestor) identity smart contract address, b) the claim type, and optionally 3) some data to go along with it (e.g.: the hashes to off-chain references of data stored by trusted claim issuers). The identity owner (the claim holder), will then store this claim in his identity contract (alternatively, the claim issuer can also add the claim himself, following the approval by the identity owner to do so).

In T-REX, it is this combination of identities and claims that allows for the on-chain validation and the eligibility of investors to hold (receive) a specific security token i.e. every time an investor is subject to receive a position in a specific security token, T-Rex will check whether his identity contains the claim(s) allowing him to hold this token.

T-REX Components (Smart contracts library)

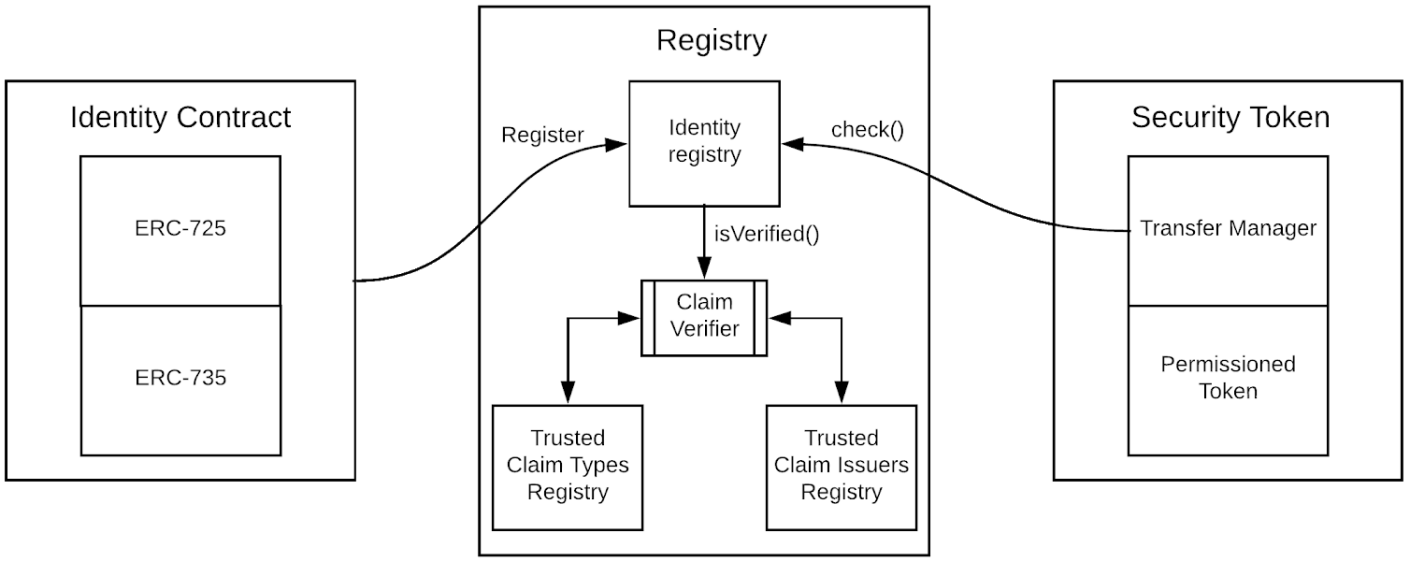

Figure 4 : illustration of T-REX components with global interactions

Identity Contract

This is the smart contract deployed by a user in order to interact with the security token (and potentially for any other further use where the his onchain identity might be relevant) . It holds the keys and claims. The identity contract is based on the ERC-725 and ERC-735 standards and it includes all the necessary functions to manage keys and claims related to the considered identity. The **Identity Contract is not linked to a specific token** and it only needs to be deployed once by each user. It can then be used for whatever purpose where the use of an onchain identity might be relevant. A detailed description of the functions can be found in Appendix A.

Identity Registry

This smart contract stores the identity addresses of all the authorized investors in the issuer's security token i.e. all identities of investors who have been authorized to hold the token after having gone through the appropriate KYC and eligibility checks. It is actually a **dynamic whitelist of identities**. It also contains a function called Verified(), which returns a status based on the validity of claims (as per the security token requirements) in the user's identity contract. The Identity Registry is managed by the issuer (or his agent) i.e. only the issuer (or his agent) can add or remove identities in the registry (note: this is the basic configuration but can be customized depending on the requirements of the token issuer). There is a specific identity registry for each security token. A detailed description of the functions can be found in Appendix A.

Claim Verifier

Is a smart contract that verifies the claims attached to an investor's identity contract for the issued security token. For example, if a security token requires that claims are signed by a specific claim issuer (e.g. the entity responsible for KYC checks for the token considered) and that the investor identity should contain the KYC claim type (user has undergone successful KYC verification for the token considered), then the claim verifier simply checks whether the investor's identity contract has a claim signed by the relevant claim issuer and it contains the KYC claim type needed. Based on the results, the contract returns a true or false value. When the Identity registry makes a call to its Verified() function it makes calls to the claimIsValid() function from the Claim Verifier for each claim that needs to be verified, and for each call, the Claim Verifier returns true or false. The transfer of token ownership is only possible if all the calls to claimIsValid() return a true value.

Trusted Claim Issuers Registry

This smart contract stores the contract addresses (identities) of all the trusted claim issuers for a specific security token. The identity contract of token owners (the investors) must have claims signed by the claim issuers stored in this smart contract in order to be able to hold the token. The ownership of this contract is given to the token issuer allowing them to manage this registry as per their requirements. A detailed description of the functions can be found in Appendix A.

Trusted Claim Types Registry

This smart contract stores all the trusted claim types for the security token. The identity contract of token owners must have claims of the claim types stored in this smart contract. The ownership of this contract is given to the token issuer allowing them to manage this registry as per their requirements. A detailed description of the functions can be found in, please refer to Appendix A.

Permissioned Tokens

T-REX permissioned tokens are based on a standard ERC-20 structure but with some functions being added in order to ensure compliance in the transactions of the security tokens. The functions *transfer* and *transferFrom* are implemented in a conditional way, allowing them to proceed with a transfer only IF the transfer manager approves the transaction. All the functions that can be added to a standard ERC-20 token can also be added to the T-REX permissioned tokens (Mintable, Burnable, Pausable, …). The permissioned tokens are allowed to be transferred only to validated counterparties, in order to avoid tokens being held in wallets/identity contracts of ineligible/unauthorized investors. The T-REX standard also supports the re-issuance of security tokens in case an investor loses his/her wallet private key. A history of re-issued tokens is maintained on the blockchain for transparency reasons.

Transfer Manager

The transfer manager is the last piece to the puzzle. It is the contract that will make the link between all the collectible data and verify the compliance of a transaction. This is the contract that checks for the validity of a transfer. By interacting with the identity registry on the validity of claims in the identity contracts of the seller, the transfer manager may or may not allow the transfer of security tokens (depending on the status returned by the identity registry to the transfer manager in response to the check() initiated). Apart from investor identity eligibility, the transfer manager will also validate more general token (or issuer) restrictions e.g. maintaining a max investor cap or a max tokens cap (as it might be needed for certain securities in certain specific countries of distribution). **The contract is modular to support the addition of multiple general compliance rules** as per the requirement of the token issuer or the regulatory framework under which the token is operated.

Additional smart contracts

Investor rights (voting, dividends, announcements, etc.) can be implemented easily by the issuer or its tokenization platform to complete the range of functions in the security token. The use of those capabilities will depend on the needs of each issuer/security token. All the features of standard securities markets could potentially be reproduced in a tokenized securities ecosystem but benefiting from efficiencies inherent to a blockchain environment (e.g. transparency, no need for several stages of reconciliation, 24/7 processing, global reach, etc). Compliant treatment of taxes will also be handled through smart contracts linked to security tokens.

Stakeholders

Overview

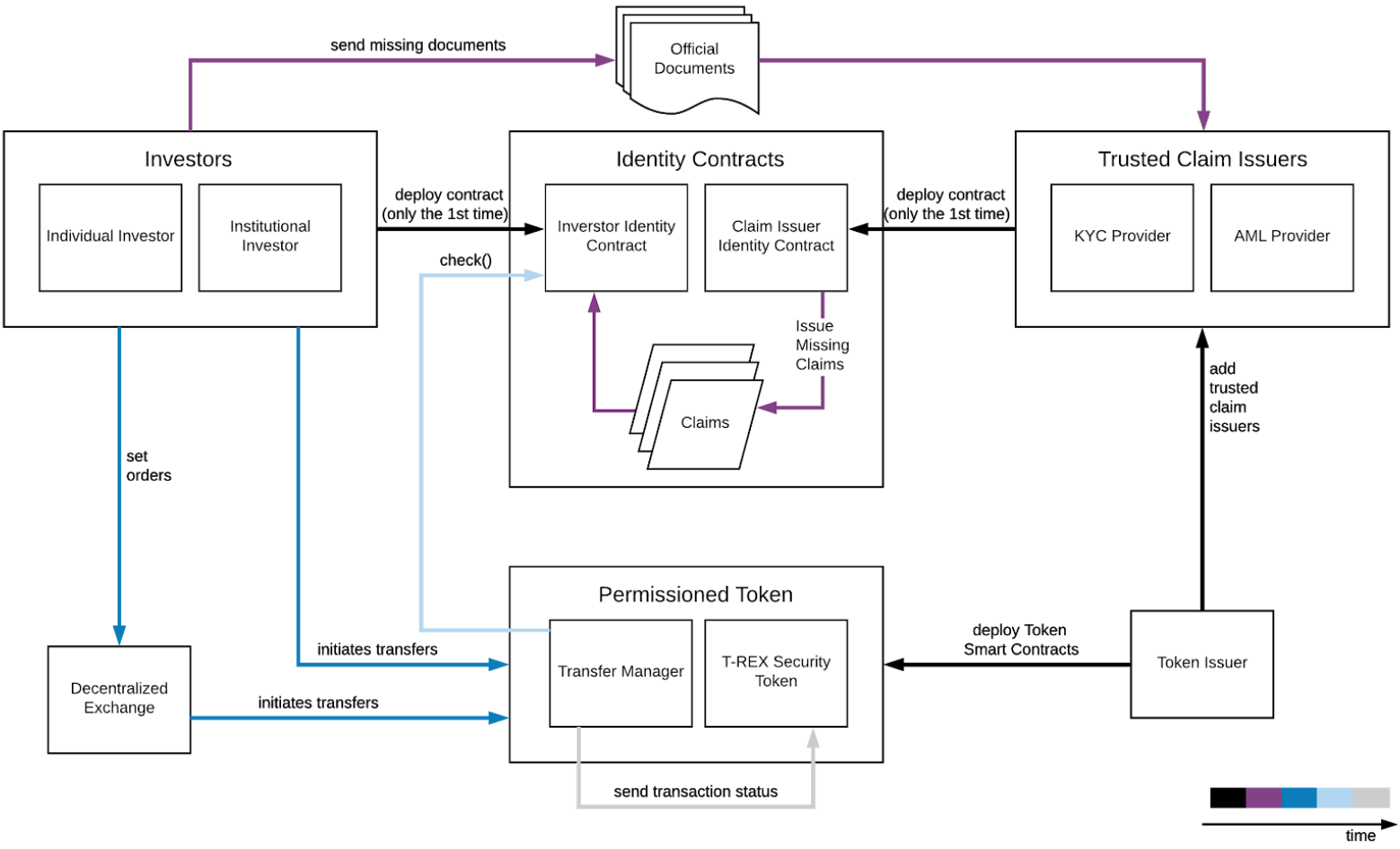

Figure 5 : Stakeholders of the ecosystem and their interactions

Issuers

The company issuing the security tokens under the T-REX framework will need to **determine the claim types needed for their investors.** This depends on the jurisdiction of issuance, the countries where they want to distribute and the characteristics of the security token in general. Issuance platforms will help issuers deploy and manage the set of smart contracts needed to support their token. They can also help investors create their ERC-725 identity or connect with an existing identity if they already have one.

Issuers (and/or their agent(s) involved in the administration of the security token) will have their own identity as well. They will indicate authorized exchanges on how to interact with the T-REX solution in order to support secondary market trading of the tokens.

Investors

They create and control their identity i.e. allowing or preventing access to claim issuers and claim controllers (entities checking if a certain identity has a specific claim or not). The investor will have to deploy the smart contracts supporting his identity (ERC-725) and claim management (ERC-735) onchain.

The sensitive data of the investors, as well as the data of all other stakeholders will not be accessible directly on the blockchain, only assessments from trusted claim issuers will be directly linked to the investor's identity contract and these will be verified by the identity registry to approve or refuse transactions through the transfer manager. This process is intended to be compliant with data protection laws.

With their on-chain identities, **investors will not necessarily have to go through identity verifications for each token issuance as long as the claims stored in their identity satisfy the requirements of the second (or third, …) token issuer with regards to KYC and eligibility checks**. In such case investors can directly give access to the needed claim checks. If the security token Issuer recognizes the claim issuer as a trusted party (for example it recognizes the estonian digital identity as trusted), the validation will be simplified.

Claim issuers

Claim issuers can act in such capacity if they are authorised by the token issuer (e.g. a third party KYC platform appointed by the issuer to perform KYC checks on his behalf) and are registered in the Trusted Claims Issuers Registry. On that basis a Trusted Claim issuer has the ability to **enrich the identity of a specific investor** as long as this investor allows him to add a claim in his identity. For example, KYC/AML providers, third party identity tools like governmental issued digital ID, luxtrust, etc.
Claim issuers can have multiple claim signer keys (and at least one) in their own ERC-725 identity.

This model allows for true interoperability between multiple KYC/AML providers. It shields Issuers from the specificities of each AML/KYC provider by using a standard protocol to verify the claims.

Exchanges and Relayers

Exchanges will also have identities and be claim holders. To integrate identity management with exchanges, the transfer mechanism used internally by an exchange is an important factor. Depending on this, how the identity protocol is integrated will vary:

Direct P2P Trades

Individual investors can trade their tokens directly amongst each other. The compliant service of the token restricts the transfer of tokens to unauthorized addresses. For a transfer to happen successfully, the buyer needs to have a proper identity as per security token standards registered to the identity registry. Also, a token can only be transferred when it is not in the lockup period.

Decentralized Exchanges

There are exchanges that enable direct address to address (wallets or identity contracts) transfers in a distributed manner. Examples of such models are exchanges based on protocols like Swap, used by AirSwap, or the 0x protocol used by relayers like ERC DEX. For Decentralized Security Token Exchanges (STE) to interact with the T-REX identity management and transfers protocol, the following steps on the next page will need to be followed:

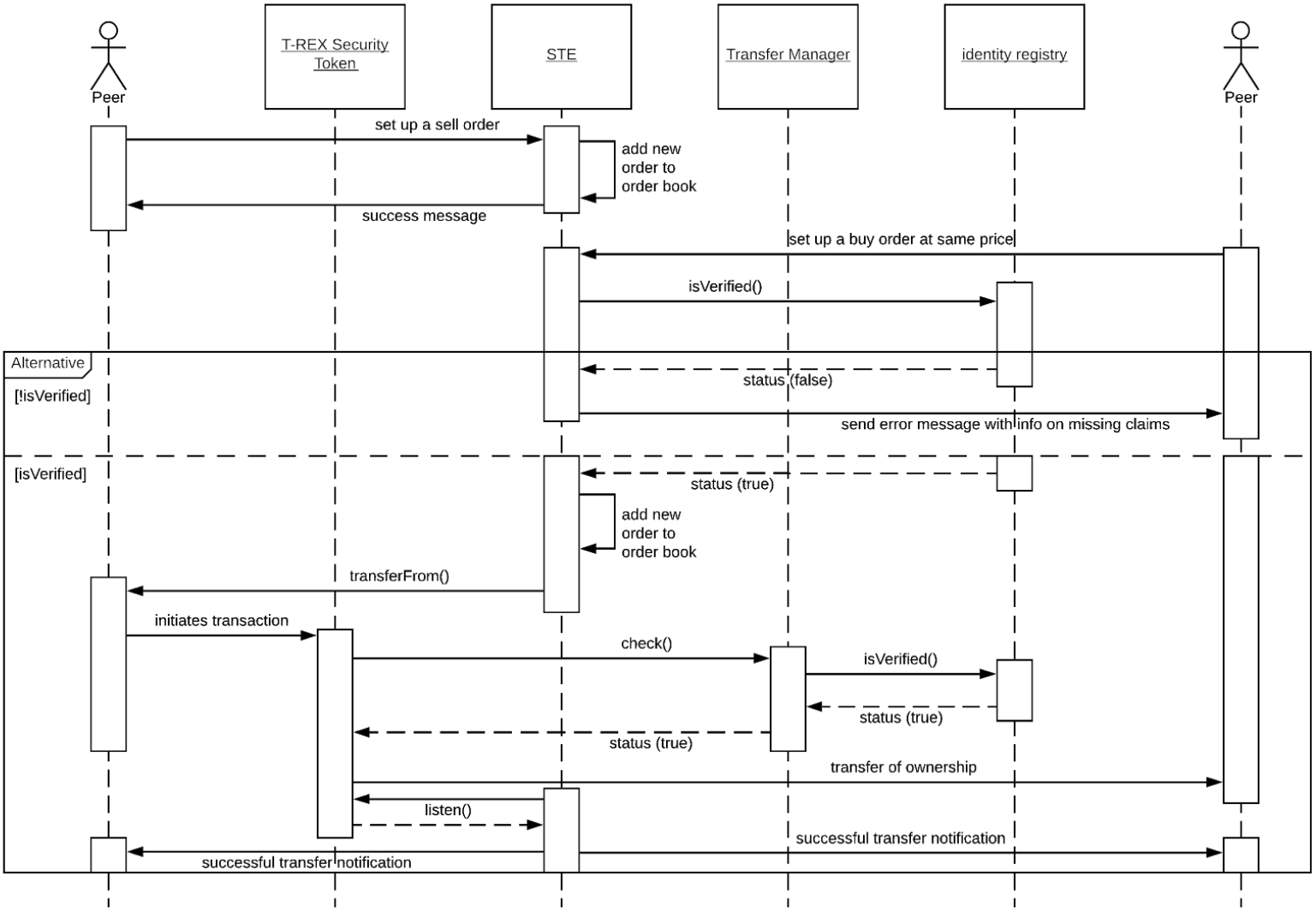

Figure 6 : sequence diagram of a Distributed Security Token Exchange transaction with T-REX tokens

1. A token holder connects to the STE and sets up a sell order.
2. The STE checks the balances of the seller to see if he has enough tokens in his wallet/identity contract to fulfill this order.
3. If the seller has the necessary balance, the order is added to the order book of the STE and a success message is returned to the order issuer. If he doesn't have the necessary balance, the order will be rejected and an error message returned to the order issuer.

4. A buyer sets a buy order on the same token, at a price higher or equal to the one of the sell order that is already in the order book.
5. The STE checks the identity contract of the buyer to validate whether he holds the necessary claims to hold/receive the token being traded, exactly the same way and with the same code that is used to test the compliance before executing the transfer.
6. The identity registry returns the status of the buyer. If he doesn't have the necessary claims, the order will be cancelled and an error message will be sent detailing what's needed for the transaction to be granted.
7. If the peer has the necessary claims, it returns true, the buy order is added to the order book and the STE initiates the transactions through the transferFrom() function. The transferFrom() function calls the transfer manager to check the claims of the receiver as it does on a standard peer to peer T-REX transaction. If everything is in order (which should be the case, as the STE already verified the claims before adding the orders in the orderbook) the transfer is executed. Depending on the DEX protocol used, the process can be a bit different, but this description is the most typical.
8. The STE scans the blockchain and when the transaction is successfully included in a block it returns a success message to the counterparties to the transaction.

In this process, we have seen how the protocol only ensures trades validated by the Compliance Service (Transfer Manager) are permitted on decentralized exchanges. We have also seen how these exchanges have the capacity to increase the reach of the system by interacting with non-compliant peers. These non-compliant peers are then guided through customized error messages including step-by-step instructions on how to resolve their claims.

Centralized Exchange - Investor Owned Wallet

The flow, in this case, is very similar, but it requires the Exchange to create and register (or reusing) specific wallets for new investors, and the investors interacting with them.

The following process would happen for an investor wanting to buy Security Tokens:

1. The Exchange runs its own KYC process (or any claim type) on the Investor and has their identity.
2. The Exchange creates an Ethereum wallet for the investor.
3. The Exchange deploys an identity contract for that Ethereum wallet.
4. The Exchange registers this identity contract in the identity registry through an API solution linked to the issuer's platform (as the registration can only be done by the issuer).
5. The Exchange provides the KYC details to the claim issuer. The claim issuer verifies the details and signs the claim data. The signed claim is then added to the identity contract of the new wallet.

6. The Exchange matches the buy order with one (or several) sell order(s).
7. The Exchange transfers the tokens between wallets, using the transfer() method.
8. This will trigger the execution of the regulatory checks by the Compliance Service. If no regulatory limits are surpassed the trade takes place.
9. The new investor has the Security Tokens in their Exchange wallet, and can do additional trades, or withdraw them to another wallet by registering that wallet in the identity registry through the exchange or issuer.

The following process would happen for an investor wanting to sell Security Tokens:

1. The Exchange runs its own KYC process (or any claim type) on the Investor and has their identity.
2. The Exchange creates an Ethereum wallet for the investor.
3. The Exchange deploys an identity contract for that Ethereum wallet.
4. The Exchange registers this identity contract in the identity registry.
5. The Exchange provides the KYC details to the claim Issuer. The claim issuer verifies the details and signs the claim data. The signed claim is then added to the identity contract of the new wallet.
6. The Exchange presents the wallet address to the investor so that the investor can send the tokens to the Exchange wallet.
7. This should work as both wallets are registered to the investor and investors can move their tokens freely across their own wallets.
8. The exchange matches the sell order with a buy order.
9. The Exchange transfers the tokens between wallets, using the transfer() method.
10. This will trigger the execution of the regulatory checks by the Compliance Service.
11. If no regulatory limits are surpassed the trade takes place.
12. The investor no longer has the tokens in their Exchange wallet and has received the currency/tokens they requested.

The other added value of T-REX for Centralized Exchanges with an Investor owned Wallet is that they can onboard a new customer provided that the AML/KYC claims are already present and linked to the customer's onchain identity. T-REX can greatly simplify the onboarding process for these exchanges.

Centralized Exchange - Pooled Wallets

This may have proved a great success for currency tokens but is not recommended for security tokens. In this case, the Exchange uses shared wallets that receive tokens from multiple investors and is able to execute trades between them off-chain. In this model, the ability for the Issuer to identify investors and validate compliance is very limited, as it has no visibility or control over what happens off-chain, and also has limitations regarding the ability to provide investors access to certain rights like voting events or dividends, since it is unclear who is holding the tokens.

In this case, the T-REX system will only have the ability to act as a validator on IN/OUT transactions but will be blind for all the internal transactions made on the exchange. Therefore, in order to validate the compliance of their users for the transfer of specific securities, centralized exchanges will be provided with API solutions that will help them to make a link between their user database and the on-chain identities of the users. Once the link is made between database and blockchain, the exchange will be able to call the isVerified() function on the identity contract of its user and change his status off-chain, depending on the result of the function.

T-REX processes

We know that the ERC20 token standard has transfer and transferFrom functions which enable token holders to transfer tokens to another address. We have seen that the T-REX standard defines a permissioned token on the Ethereum blockchain, enabling token transfers to occur if and only if they are approved by an on-chain Transfer management Service. Implemented with the correct configurations, the T-REX standard enables compliant transfers, both on exchanges (preferably decentralized) and on a pure peer to peer basis, in STOs and secondary trades, and across jurisdictions. In essence, the T-REX standard enables ERC-20 tokens to support the issuance management and transfers of compliant security tokens.

T-REX implements ERC-20 methods transfer() and transferFrom() with an additional check to determine whether or not a transfer should be allowed to proceed. The check is done over the identity contract of the receiver of tokens and it verifies whether this identity contract has the required claims needed to hold/receive the security token.

There will be four major entities:

1. **Token Deployer** : The entity responsible for deploying the smart contracts of the security token;
2. **ABC Corporation**: The company willing to issue security tokens;
3. **Trusted Claim Issuer**: An entity handling the issuance of claims to identity contracts (e.g. a third-party KYC provider). This claim Issuer is trusted by the issuer of the token. All the token owners must have claims signed by this claim issuer.
4. **Investors**: are the owners of security tokens having permissions for the trading of tokens among each other.

Security token deployment

Role: Token deployer

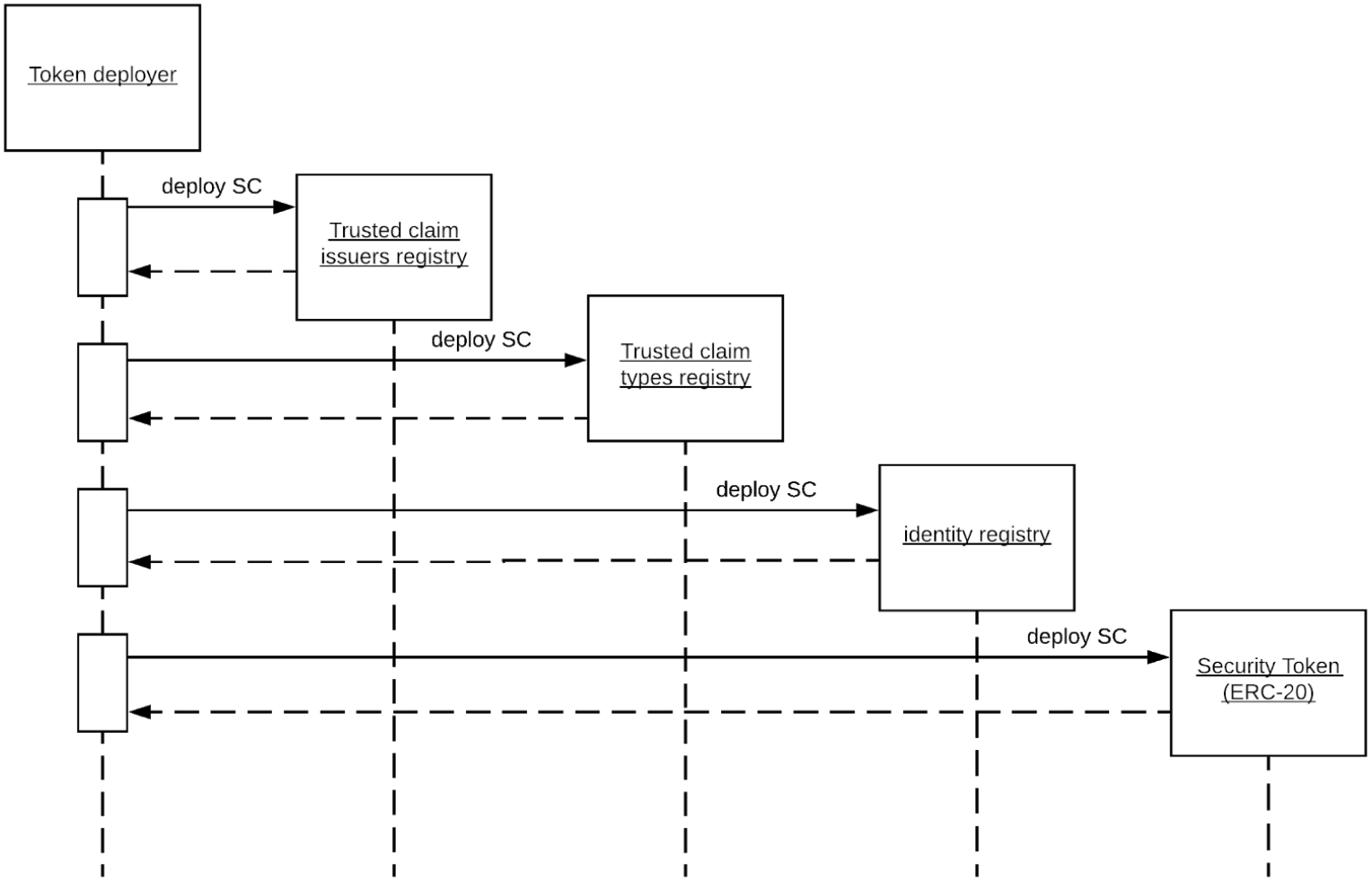

Figure 7 : sequence diagram of security token deployment

1. Token deployer first intiaties the trusted claim issuers registry smart contract (SC). This contract stores addresses of all the identity contracts of the claim issuers it trusts with regards to this specific token.

2. Subsequently, the trusted claim type registry contract is deployed. This contract stores all the trusted claim types related to the security token.
3. The identity registry contract follows, which stores the identity contract addresses of all the users allowed (i.e. eligible) to hold the token. It is also in charge of claims verification by interacting with the above two previously mentioned contracts.
4. Finally, the security token contract is deployed. The token contract interacts with the identity registry to check the eligibility status of investors and hence, making holding/transacting possible.

Now that the token is deployed, it can interact with validated investors.

Updating trusted claim issuers and trusted claim types

Role: ABC corporation

ABC corporation will populate the Trusted Claim Issuers and Trusted Claim Types in the T-REX ecosystem depending on their requirements.

- **Trusted Claims Issuers Registry:** ABC corporation has ownership of the Trusted Claims Issuers Registry where it manages the claim issuers and their activities. For example, if ABC corporation trusts Luxtrust as a claim issuer, it adds Luxtrust's identity contract in the Trusted Claim Issuers Registry. It will also have the possibility to manage, i.e. delete, update, and add Trusted Claim Issuers in the registry.

- **Trusted Claim Types Registry:** ABC corporation has ownership of the Trusted Claim Types Registry where it manages the claim types it wants for an investor to be an eligible token holder. ABC corporation has the right to delete and add multiple claim types to this registry. For example, if ABC corporation wants its investors to have gone through a KYC process, it will add the index of this claim type (for example, KYC = 4) in this registry. Other claim types can be verified via a verified address proof, a valid accreditation certificate etc.

So, after ABC corporation has added Trusted Claim Issuers and Trusted Claim Types in the T-REX ecosystem of its token, an investor's identity contract needs to have claims of trusted claim types issued by the trusted claim issuers to become an eligible token holder.

Creation of identities on the blockchain

There are two roles that are needed to deploy identity contracts on the blockchain in the T-REX ecosystem.

- Claim Issuers.
- Investors/Token Holders.

An identity can be created by different processes:

- An entity can self-deploy an ERC725 compliant Smart Contract, by directly on the Ethereum blockchain or by using a decentralized identity management solution like EDDITS.io or Origin;
- An ERC725 contract can be deployed on behalf of an entity by a third party (for example an exchange platform or a token issuance platform creating identities for their users in order to have them subsequently populated with claims related to token eligibility).

Role: Claim issuers

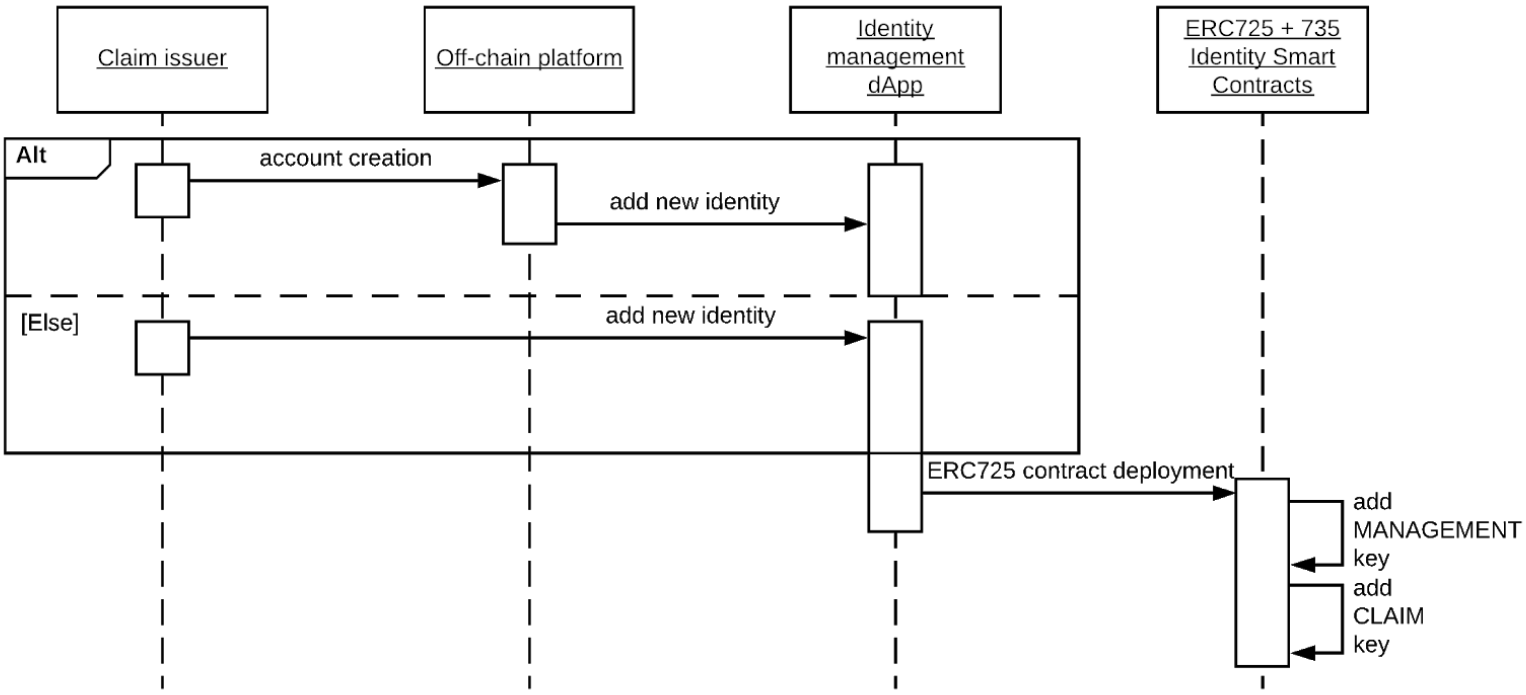

Figure 8 : claim issuer's identity creation

1. Claim issuer deploys an identity contract by any of the methods mentioned above.

2. It then adds a claim signer key to the identity contract deployed. This claim signer key can now be used to sign claims requested by other identities. If this key is deleted, all the claims signed by this key will become invalid.
3. The claim issuer then provides the identity contract address to ABC corporation who in turn adds this address to their trusted issuers registry. (If ABC corporation wants its investors to have claim issued by this claim issuer).

Role: Investors

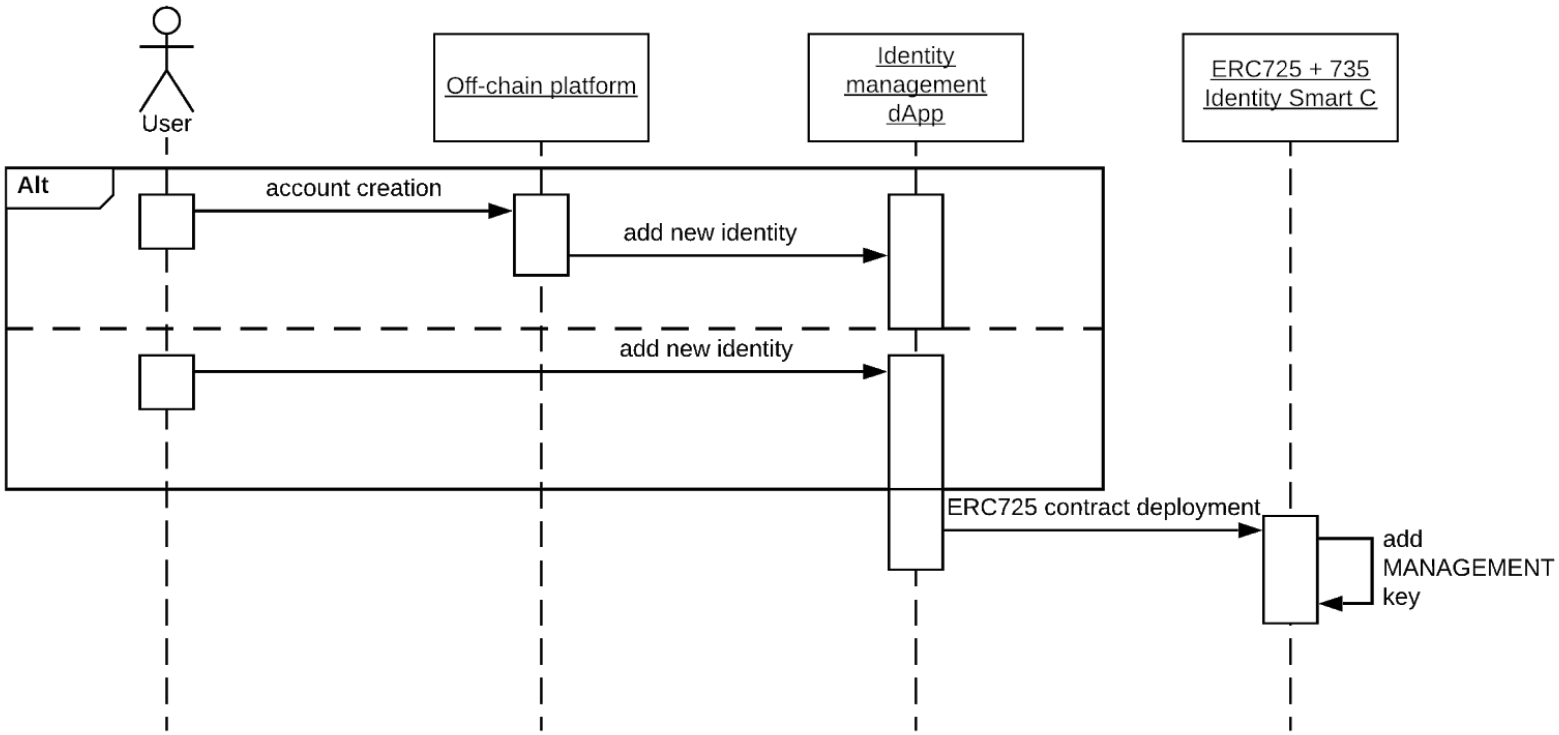

Figure 9 : User's identity creation

A user can deploy his/her own identity contract using our T-REX dapp (or other identity providing services) or can be deployed on behalf of the user by 3rd party entities such as exchanges. On deployment of the identity contract, a management key is added by default which is crucial in the management of keys and claims in the contract. Also, for security purposes, the ownership of an identity contract is not transferable as it is the sole proof of identity on the blockchain.

Claim addition

This is probably the most crucial aspect of the T-REX ecosystem. The process is similar to getting your driving license signed by an issuing authority. Once verified and signed, this card becomes proof of your driving abilities and you are then permitted to drive a vehicle anywhere in the country. Analogous to this is the claim issuance protocol of T-REX. The identity contract you own is the driving license card. Claim issuers are the issuing authorities that sign your identity claims after proper verification of your identity. Once verified and signed, you now have the right to own the security tokens for which your eligibility has been checked, just like your right to drive anywhere in the country. Also, it is important to note that depending on the requirement of the token, claims can be self-attested too. For example, a user can self-sign a claim regarding the correctness of his full name. The token issuer, in this case, is willing to trust the user's "self-claim".

Role: Investors + claim issuers

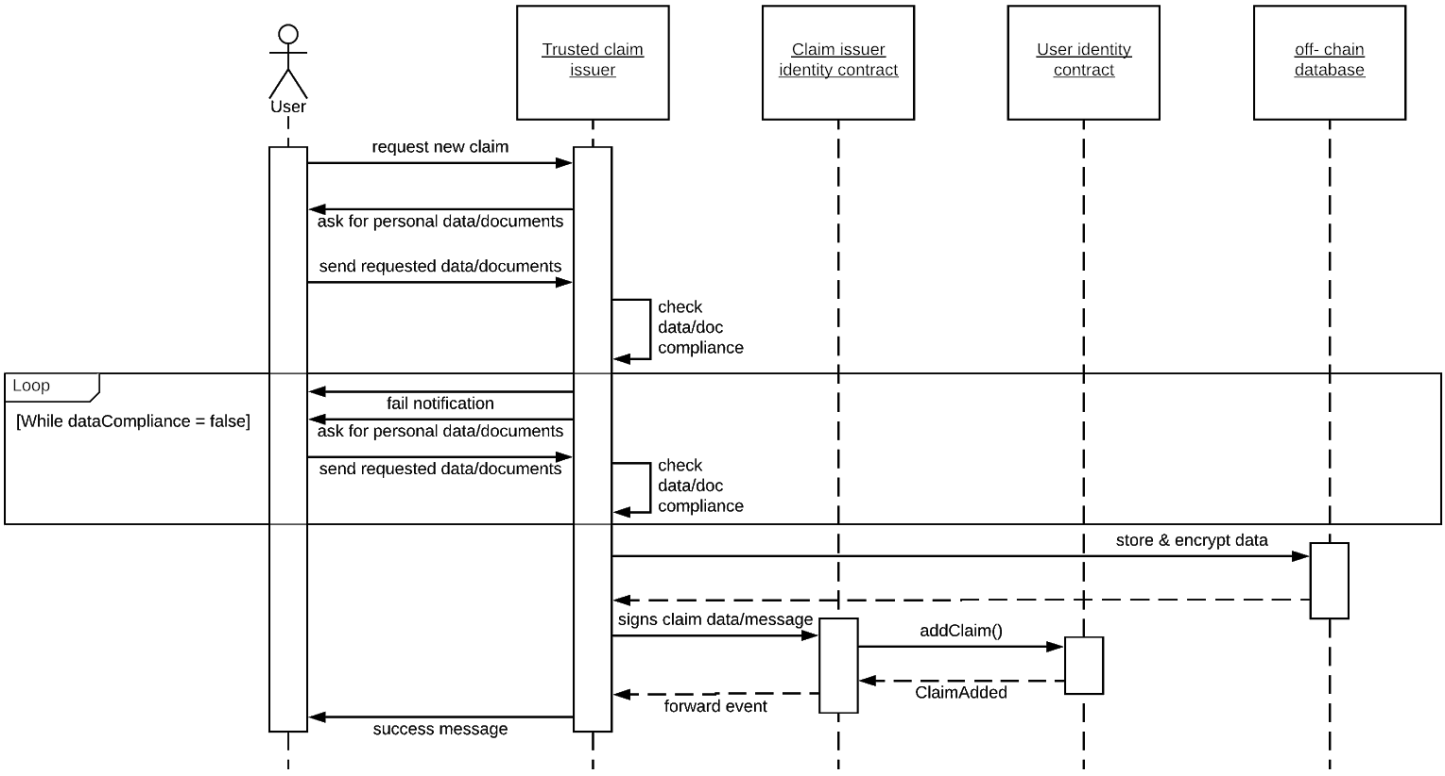

Figure 10 : claim addition to user's identity contract

1. The user requests a claim from the trusted claim issuer. A claim issuer has an identity contract deployed having a claim signer key.
2. The claim issuer does some user off-chain verification based on the claim type requested. For example, if the claim type requested is KYC check, then the claim issuer does the KYC check for that user off-chain.
3. On successful verification, the claim issuer stores user data in an encrypted format in an off-chain database.
4. The claim issuer then signs a message that includes the claim type, user's identity contract address and the hash of the encrypted data stored using the claim signer key.
5. A claim is finally added to the user's identity contract. This claim majorly consists of
    - The claim type
    - The signing scheme
    - Claim issuer's signature
    - Hash of the encrypted data

This now becomes the proof of identity/eligibility for the user and can be used by any service that trusts the issuer undertaking the issuance of claims.

User registration in the identity registry

Role: ABC corporation

Once the investor has an identity contract deployed, and has passed KYC and eligibility checks requested for the token considered, it must be added in the identity registry in order to enable the user to hold/transfer the token. The purpose of this registry is to store the identity contract address corresponding to the deployer address. This contract has the all-important function - isVerified() which is responsible for the verification of claims in the identity contract corresponding to a user's address. As discussed earlier, the identity registry interacts with the issuers registry to verify claims based on the initial security token requirements.

For security purposes, only the token issuer or his mandate can add an identity contract to the registry and only if the owner of the identity contract initiated the request to be added in the identity registry. Also, one address can only have one identity contract stored in the registry.

The token issuer or ABC corporation has full ownership of this contract and is responsible for the management of user identities. To register a new identity, the Issuer has to call the registerIdentity() function in the identity registry contract. The user's ethereum address and the identity contract address are used as parameters.

Compliant transfer of ownership

The T-REX standard makes sure that the transfer of security tokens is compliant

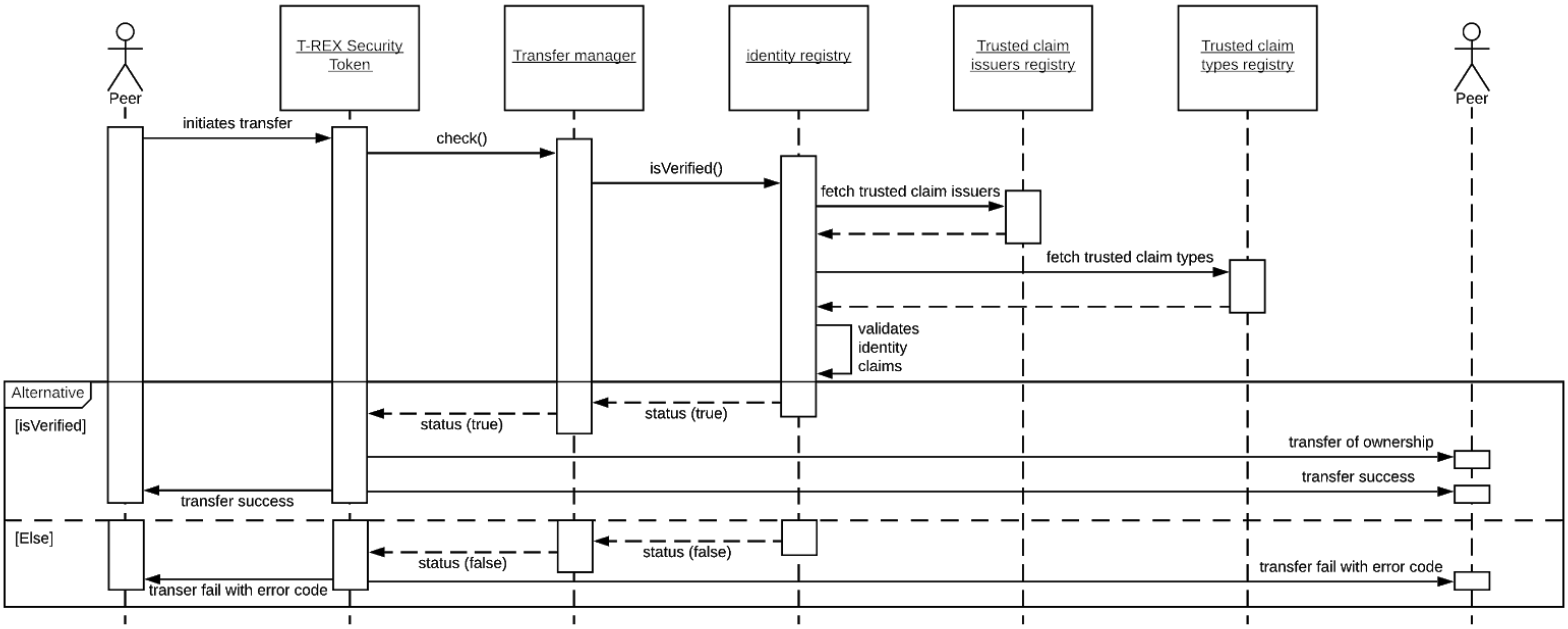

Figure 11 : compliant transfer of ownership of T-REX token

1. User calls the transfer/transferFrom function.
2. The check function is called from within the transfer/transferFrom functions to instruct the validator to check for the validity of the transfer.
3. The transfer manager calls the isVerified() function on the identity registry to check if the receiver is a valid investor (i.e. if his address is in the identity registry of the token - meaning he holds the necessary claims needed to be eligible for this token). If no identity contract is found corresponding to the address of the receiver, an error status is returned.
4. The identity registry fetches the claims in the receiver identity contract and decodes the claims to extract the claim types and the signatures. It then compares the found claim types with the claim types in the trusted claim type registry. If no matches are found, a failed status is returned and the transfer is not allowed to proceed.

5. If a match is found, then the signature is checked. First, the trusted claim issuers are fetched from the trusted claim issuers registry. Then for each trusted issuer, it is verified whether the signature was done by a claim signer key present in the trusted claim issuer's identity contract. If no matches are found, then a failed status is returned and the transfer is not allowed to proceed.
6. If a match is found, then a success status is returned to the security token. If the success status is received, the transfer is allowed to proceed and ownership of the token is successfully transferred. When the transaction is successfully included in a block, both stakeholders can acknowledge the success of the transfer from the Tx Hash, the interface linked to an ethereum block explorer returns a "transfer success" message.

Token ownership on the blockchain

As described in the Ethereum Whitepaper[6], there are two types of accounts on Ethereum: externally owned accounts (EOA), controlled by private keys, and contract accounts, controlled by their contract code. Externally owned accounts are managed by wallets like Metamask or MyEtherWallet. Wallet applications have been enhanced to display ERC-20 Tokens owned by EOA and to perform transfer of tokens from an EOA to an address.

For compatibility purpose with these wallets, the model described here assume that the tokens are owned on the blockchain by an EAO whom address is linked to a ERC-725 contract account. This add a small layer of complexity compared to a model where the tokens are owned by an ERC-725 contract account, which is the philosophy behind ERC-725 and 735 creation. In the near future, common wallets will evolve to support ERC-725 Identity linked to an EAO, just as they have evolved to support ERC-20 and ENS for example.

T-REX supports both models from the beginning. If the address of the token receiver is an account address that is ERC-725 compliant, then we perform the KYC checks as we have described in the document. If it is an EOA, we lookup in the Identity Registry to find if an Identity is linked to this EAO and then perform the checks.

[6] https://github.com/ethereum/wiki/wiki/White-Paper

Appendix A

ERC-20 Functions

Function Name	Parameters	Description
totalSupply	none	Returns total number of tokens in existence.
balanceOf	address owner	Gets the balance of the specified address.
allowance	address owner, address spender	Function to check the amount of tokens that an owner allowed to a spender.
tranfer	address to, uint256 value	Transfer token for a specified address.
approve	address spender, uint256 value	Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
transferFrom	address from, address to, uint256 value	Transfer tokens from one address to another.
increaseAllowance	address spender, uint256 addedValue	Increase the amount of tokens that an owner allowed to a spender.
decreaseAllowance	address spender, uint256 subtractedValue	Decrease the amount of tokens that an owner allowed to a spender.
_transfer	address from, address to, uint256 value	Transfer token for a specified addresses.
_mint	address account, uint256 value	Internal function that mints an amount of the token and assigns it to an account. This encapsulates the modification of balances such that the proper events are emitted.
_burn	address account, uint256 value	Internal function that burns an amount of the token of a given account.
_burnFrom	address account, uint256 value	Internal function that burns an amount of the token of a given account, deducting from the sender's allowance for said account. Uses the internal burn function.

burn*	Uint256, value	Burns a specific amount of tokens.
burnFrom*	addressFrom, uint256 value	Burns a specific amount of tokens from the target address and decrements allowance.
cap**	none	Return the cap for the token minting.
mint***	Address to, uint256 value	Function to mint tokens.

* if the token is ERC20Burnable
**if the token is ERC20Mintable and ERC20Capped
***if the token is ERC20Mintable

Identity Contract Functions

Function Name	SC concerned	Parameters	Description
addClaim	ERC-735	uint256 claimType, uint256 scheme, address issuer, bytes signature, bytes data, string uri	This function is used to add claims to an identity contract. A claim consists of claim type, scheme, a signature that a claim issuer has signed, some verification data and a URI. It returns a claimId of the added claim. Triggers ClaimAdded event.
removeClaim	ERC-735	Bytes32 claimId	Removes the claim corresponding to the claimId provided.Triggers ClaimRemoved event.
addKey	ERC-725	bytes32 key, uint256 purpose, uint256 type	Adds a key to the identity contract with the specific purpose provided. Triggers KeyAdded event.
removeKey	ERC-725	bytes32 key	Removes the key from the identity contract. Triggers KeyRemoved event.
approve	ERC-725	uint256 id, bool approve	A proxy function that approves or rejects an execution. Triggers Approved event.
execute	ERC-725	address _to, uint256 _value,	Executes an action on other contracts, or itself, or a transfer of

		bytes _data	ether. Triggers. ExecutionRequested event. Also triggers Executed event when corresponding Approved event is triggered.
getKey	ERC-725	bytes32 _key	Returns details of the key provided.
getKeysByPurpose	ERC-725	uint _purpose	Returns all the keys of the purpose provided.
keyHasPurpose	ERC-725	bytes32 _key, uint _purpose	Checks whether the provided key has the provided purpose. Returns either true or false.
getKeyPurpose	ERC-725	bytes32 _key	Returns the purpose of the key provided.
getClaim	ERC-735	Bytes32 _claimId	Returns claim details of the given Claim Id.
getClaimsByType	ERC-735	uint claimType	Returns all the claims corresponding to the provided claimType.

Identity Registry Contract Functions

Function Name	**Parameters**	**Description**
Register Identity	address user, address identity	Adds an identity contract address to the registry corresponding to the user.
Update Identity	address user, address identity	Updates the existing identity contract address with the new identity contract address.
Delete Identity	address user	Deletes the identity contract saved corresponding to the user.
View Identity	address user	Displays the identity contract stored in the registry corresponding to the user.
Is Verified	address user	Returns a bool value based on the validity of claims in the user's identity contract.

Trusted Issuers Registry Contract Functions

Function Name	Parameters	Description
Add trusted issuer	address trustedIssuer, uint index	Adds a trusted issuer contract address for the security token in the registry corresponding to the index provided. For example, index 1 could be Tokeny, index 2 could be Leaseum. The index has to be unique.
Remove trusted issuer	uint index	Removes a trusted issuer corresponding to the index provided.
Update Issuer contract	uint index, address newTrustedIssuer	Updates the trusted issuer contract corresponding to the index provided.
Get trusted Issuer	uint index	Returns the trusted issuer contract address corresponding to the index provided.
Get trusted Issuers		Returns the indexes of all the trusted issuers in the registry.

Trusted Claim Types Contract Functions

Function Name	Parameters	Description
Add claim type	uint claimType	Adds a trusted claim type to the registry. This is stored in uint format like KYC could be 7, AML could be 8 etc.
Remove claim type	uint claimType	Removes the corresponding claim type from the registry.
Get claim types		Returns all the claim types stored in the registry.