

KOIN

An encapsulation approach to token issuance.

By Mark Wagner and Afshin Kargarmovakher

In 2017, we saw the beginning of a new form of fundraising; the Token Sale. Be it an ICO where coins are mined, gathered and disbursed or an ITO where a token contract is drawn up and executed, startups now have a way to circumvent traditional venture capital firms and directly raise funds, in some cases even democratizing control of the company via the block chain. By creating a series of tokens tied to an immutable distributed contract that specifies their initial value, volume, usability, and ownership, founders can assign securitized value to equity in their firm or create an efficiently tradable store of value for use with their product.

While the SEC is slowly making progress tightening down restrictions on who can have a token-sale, how and for what purposes the tokens can be redeemed, there will always be a need for efficient software to execute the transaction of tokens for ether.

Right now, Ethereum, a currency-bound block-chain is the most commonly used platform on which tokens are generated. A new standard of included contract functions; ERC20 has been adopted by many exchanges to make their client tokens more fluid and uniform for their software to work with. It also has been interpreted as a means to bring standardization and some form of compliance to what has lately been the wild west.

Currently, most tokens are generated by novices using the fairly powerful MIST/Ethereum Wallet desktop client, which runs a GETH node on your system and issues the first notice to the other ledgers that a new token contract exists.

The problem:

Many token sales currently have no means of distribution. They collect a “whitelist” of email addresses tied to a requested token amount, and an Ethereum address. During the token sale, buyers send ether to the company’s wallet address and someone (either manually with Ethereum Wallet/MIST or with sloppy software) at the firm looks up the address of the sender in the white list and reciprocates an appropriate amount of token at whatever rate was agreed upon.

This has resulted in so many delays, failed sales, thefts, miscalculations, mistyped addresses (which causes tokens to be lost), and exchanges happening at unfair exchange rates that the SEC and the Federal Government have accelerated their clampdown on crypto funding, regardless of the token structure or use.

What Is KOIN?

This may be refreshing to hear in 2017 but “KOIN is NOT a token sale!”

KOIN is a piece of software written with HTML, Java, the web3j library, JavaScript, Apache Tomcat, and (using the client’s) Solidity smart contract, served as a JSP in a web environment.

What it does is automate a swift and secure exchange of Ethereum for a token without room for hackers or miscalculations and free from the negative consequences of currency price volatility. In 2018, we will be seeing token issuances where the users receive their token within 60 seconds of making the purchase, rather than weeks as with major launches of 2017.

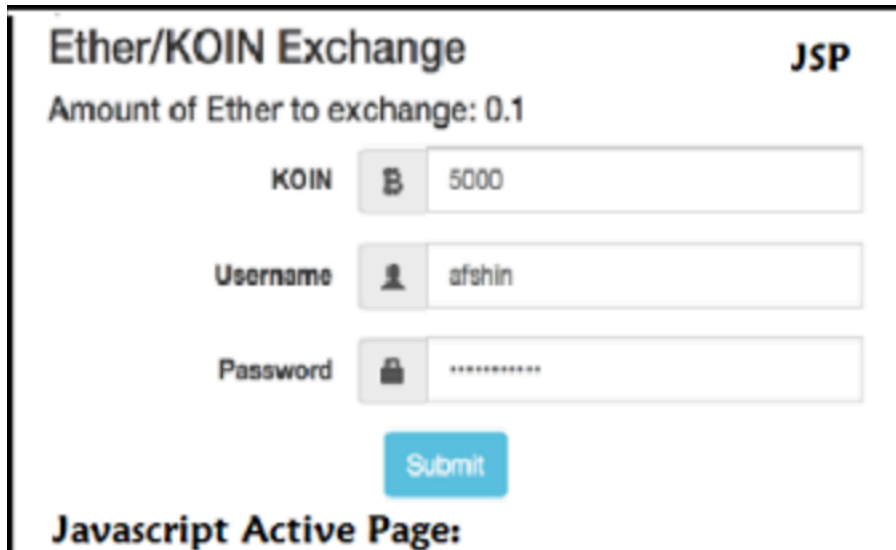
How does it work for a token buyer?

The user logs into the JSP and enters their information as well as uploads their wallet file (a .json they can download from Coinbase, MetaMask, or any Ethereum client in seconds).

Register for KOIN!

First name	<input type="text" value="Mark"/>
Last name	<input type="text" value="Wagner"/>
Username	<input type="text" value="merk"/>
Keystore File	<input type="button" value="Choose File"/> <input type="text" value="UTC--2017...d78f.json"/>
Password	<input type="password" value=""/>
<input type="button" value="Submit"/>	

When they press submit, they are directed to the following page which is running active JavaScript to demonstrate how many ether you will spend to buy the amount of token specified to the servlet. The exchange rate is actually pulled via REST API from CoinDesk's active Ether/USD exchange rate so that if a token should be 1:1 with the USD, the rate of ether:token will vary with the market, updated every 30 seconds.



Ether/KOIN Exchange **JSP**

Amount of Ether to exchange: 0.1

KOIN

Username

Password

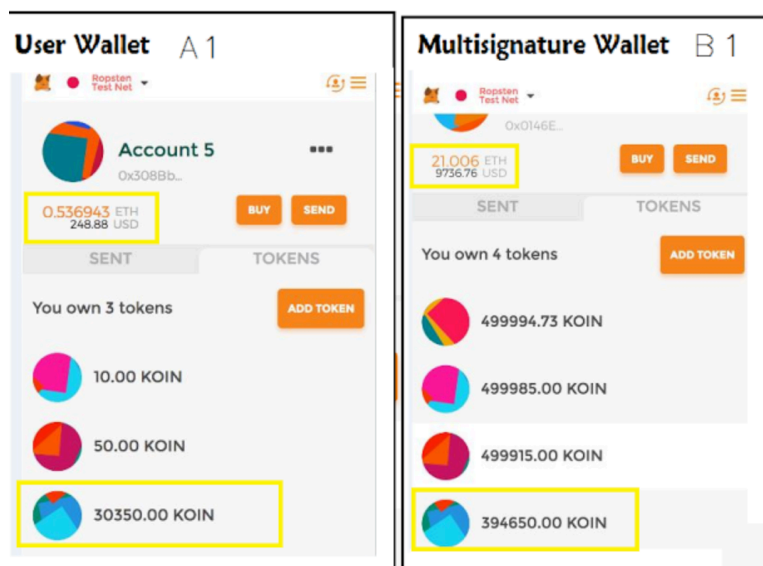
Javascript Active Page:

Once they click submit, web3j Java library provides method calls to the nodes to sign transfer functions (using the .json) and send ether from the customer's wallet to the company's board-controlled multi-signature wallet.

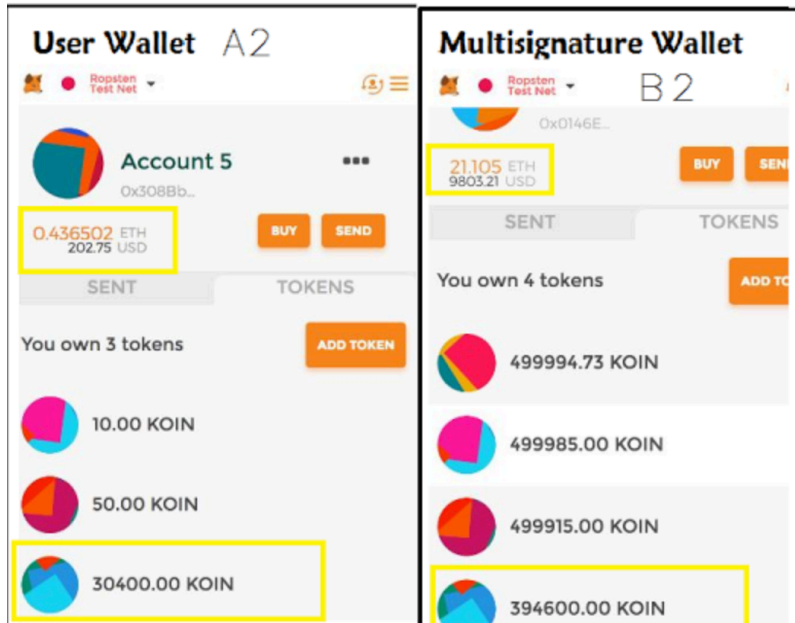
Simultaneously, it uses the address parameter from the .json as an address to which tokens are sent from the company's token wallet (which was filled with tokens when the contract first deployed on the token creation date far prior to the sale)

This ensures that everyone gets what they want within a 60 seconds window (* dependent only on the current Ethereum network speed)

Here are the Ropsten Testnet Accounts on MetaMask before execution:



Here are the accounts after the execution:



And here is evidence of the Ether reception transaction processing in rapid succession after the execution on EtherScan:

The screenshot shows the Etherscan website interface for the ROPSTEN (Revival) TESTNET. It displays the contract address 0x13B507B9554B231eBFdAF7B99A96890D4b58A53. The contract overview shows an ETH balance of 0 Ether and 30 transactions. The transactions table lists the latest 25 transactions from a total of 30 transactions.

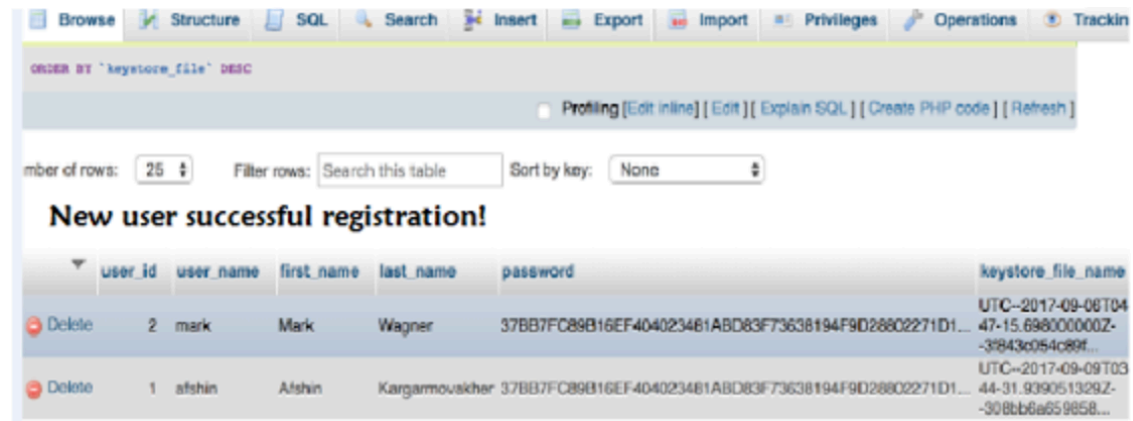
TxHash	Block	Age	From	To	Value	[TxFee]
0x57c319e70a813b...	2160257	1 min ago	0x0145e80a705f6e9...	0x13b507b9554b23...	0 Ether	0.000000000
0x3681571fcd9d776...	2160146	30 mins ago	0x0145e80a705f6e9...	0x13b507b9554b23...	0 Ether	0.000000000
0xfed22e425e30628...	2150673	2 hrs 20 mins ago	0x0145e80a705f6e9...	0x13b507b9554b23...	0 Ether	0.000000000

What about hackers?

Because it uses the robust cryptographic protections of the Ethereum Cryptographic Algorithm, the only considerable security flaw could be seen as the uploading of the wallet file.

Depending on configuration, an upload of the wallet file can be requested every time a token is purchased, but since we utilized the SHA-256 hash algorithm with a salt to encrypt the account password, we consider the wallet file to be safe enough to keep on the server.

The manner in which they are stored was a particularly sticky dilemma as speed was our goal in development. Rather than isolating the fields of each wallet file into a SQL database table (which would be vulnerable to malicious query attacks), we decided to keep the wallet files in a directory on our AWS server and simply save a path/to/wallet string in the table associated with the user account, access to which would require the hashed password mentioned above.



ORDER BY 'keystore_file' DESC

number of rows: 25 Filter rows: Search this table Sort by key: None

New user successful registration!

	user_id	user_name	first_name	last_name	password	keystore_file_name
Delete	2	mark	Mark	Wagner	37BB7FC89B16EF404023481ABD83F73638194F9D28802271D1...	UTC--2017-09-08T04:47-15.698000000Z--3f843e054c89f...
Delete	1	afshin	Afshin	Kargarmovakher	37BB7FC89B16EF404023481ABD83F73638194F9D28802271D1...	UTC--2017-09-09T03:44-31.939051329Z--308bb6a659858...



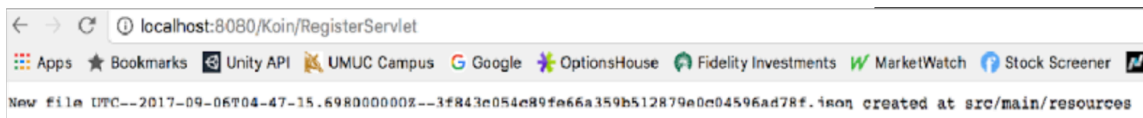
Database mapping of User IDs to Wallet Files

server: localhost Database: koin Table: user

number of rows: 25 Filter rows: Search this table

user_id	user_name	first_name	last_name	password	keystore_file_name
1	afshin	Afshin	Kargarmovakher	37BB7FC89B16EF404023481ABD83F73638194F9D28802271D1...	UTC--2017-09-09T03:44-31.939051329Z--308bb6a659858...

After Successful Registration and addition of wallet file to server (for testing only)



localhost:8080/Koin/RegisterServlet

Apps ★ Bookmarks Unity API UMMC Campus Google OptionsHouse Fidelity Investments MarketWatch Stock Screener

New file UTC--2017-09-08T04-47-15.698000000Z--3f843e054c89f666a359b512879e0e04596ad78f.1e0a created at src/main/resources

Just as well, we never open or reconfigure the wallet file, so there can be no typographical or numerical errors within it's information. The wallet file is read in the same format every time a transaction takes place so if there was a change in the fields, the Ethereum nodes would reject the transaction due to the discrepancy.

```
public void addFileToKeystore(PrintWriter writer, String path, String fileName, Part filePart) {
    OutputStream out = null;
    InputStream fileContent = null;

    try {
        out = new FileOutputStream(new File(path + File.separator
            + fileName));
        fileContent = filePart.getInputStream();

        int read = 0;
        byte[] bytes = new byte[1024];

        while ((read = fileContent.read(bytes)) != -1) {
            out.write(bytes, 0, read);
        }
        writer.println("New file " + fileName + " created at " + path);
        //    LOGGER.log(Level.INFO, "File{0}being uploaded to {1}",
        //        new Object[]{fileName, path});
    } catch (Exception e) {
        writer.println("You either did not specify a file to upload or are "
            + "trying to upload a file to a protected or nonexistent "
            + "location.");
        writer.println("<br/> ERROR: " + e.getMessage());

        //    LOGGER.log(Level.SEVERE, "Problems during file upload. Error: {0}",
        //        new Object[]{fne.getMessage()});
    }

188     public Credentials connectToEthereumWallet(String password, String pathToWalletFile, String walletFileName) {
189         try {
190             this.credentials = WalletUtils.loadCredentials(password, pathToWalletFile + "/" + walletFileName);
191             System.out.println("Credentials loaded...");
192             return credentials;
193         } catch (IOException ex) {
194             Logger.getLogger(RequestService.class.getName()).log(Level.SEVERE, null, ex);
195         } catch (CipherException ex) {
```

What is the pricing model?

This is where our model has options:

For an upfront fee, 100% of the Ethereum raised (minus network costs) will go to a single or multi-signature wallet controlled by the token issuers.

For a percent share in the fundraising, the remaining % of Ether will fill the fundraiser's wallet.

What are the next product development steps?

- Building a more user-friendly UI with Angular
- Hosting KOIN as a SaaS, for easily scalable implementation
- Partnering with an exchange
- Building a launching platform for tokens similar to an exchange but for creator -> buyer sales rather than peer -> peer exchanges.
- Purchasing adequate marketing to target founders hoping to launch ICOs
- Providing an accessory legal/compliance/business structure/technical support service into the product for further client monetization.

Rather than beginning with "words and a manual service", we are using the functional software as our base to launch from and we expect the more sophisticated token market of 2018 to demand the structure we can offer fundraisers in crypto!