Trustless Two-Factor Authentication using Smart Contracts in Blockchains

Varun Amrutiya, Siddhant Jhamb, Pranjal Priyadarshi, Ashutosh Bhatia
Dept. of Computer Science
Birla Institute of Technology and Science, Pilani
{h20170244, h20170243, h20170242, ashutosh.bhatia}@pilani.bits-pilani.ac.in

***Abstract*—Two-factor authentication (2FA) is widely prevalent in banking, emails and virtual private networks (VPN) connections or in accessing any secure web service. In 2FA, to get authenticated the users are expected to provide additional secret information along with the password. Typically, this secret information (tokens) is generated by a centralized trusted third party upon receiving an authentication request from users. Thus, this additional layer of security comes at the cost of inherently trusting the third party for their services. The security of such authentication systems is always under the threat of the trusted party is being compromised. In this paper, we propose a novel approach to make server authentication even more secure by building 2FA over the blockchain platform which is distributed in nature. The proposed solution does not require any trusted third party between claimant (user) and the verifier (server) for the authentication purpose. To demonstrate the idea of using blockchain technology for 2FA, we have added an extra layer of security component to the OpenSSH server a widely used application for Secure Shell (SSH) protocol.**

I. Introduction

Nowadays the security of mobile devices and web service can no longer be guaranteed with the password alone. The people tend to use passwords which are easier to remember and also use the same password across multiple platforms. Such passwords are easier to crack using a dictionary and other types of guessing attacks. Hence, there is a strong need for enterprises to switch to a stronger authentication system which provides tighter security than the security provided by passwords. Two-factor authentication (2FA) uses an additional secret along with the password that is known to the claimant only [1]. This enhances the security of the system to a certain extent especially for the cases when there is a risk of the password being stolen or compromised.

Typically, in 2FA based systems a centralized entity provides a one-time password or hardware tokens to the users through SMS messages at the time when the user tries to access the system. However, the actual method to distribute the secret codes depends upon the centralized entity or a corporate infrastructure to provide this service to its mass users. Note that security requirements of these secret codes are not same as passwords as these codes can only be used one time and the attacker cannot crack the system even if he is able to guess the secret successfully. This extra layer of security provided by 2FA comes at the cost of inherently trusting the third party for their services. Any security breach for such organization could weaken the security of the application by many folds. The paper [2], provides a comprehensive study on multi-factor authentication schemes.

In this work, we propose the design of a 2FA infrastructure on a distributed trustless scalable platform provided by the blockchain at its core. Rather than relying upon the centralized design, the proposed solution uses a distributed and trustless design to make the process of authentication secure. Another advantage of the proposed design over the traditional OTP based 2FA mechanisms is that it is more secure during the scenarios when the application that is trying to authenticate the user, itself has been compromised. In most of the existing systems, the OTP or token is generated by the applications running on the server. So, inherently the application know the token, and therefore, it can gain the access of the system themselves if the user password has already been compromised. Hence, in this case, the OTP does not provide additional security over the password only based authentication. On the other hand, in our design, the user generates OTP and the server application only has the hash of this OTP. Thus they cannot recover the OTP themselves unless the user gives them.

The proposed solution can enhance the security of the server deployed in a data center for any organization. For example, in an enterprise a data center with a large number of servers resources, handling authenticated access to the server by remote users is mainly handled by OpenSSH server which is a secure and industry trusted tool. The security of the OpenSSH protocol comes from the use of public-key pairs [3]. This poses the requirement of having a key-management software for enterprises. One security aspect related to the integration of these key-management software to the OpenSSH is that these solutions are just wrappers around OpenSSH and their security of the OpenSSH authentication system depends on the security of the tool which is injecting keys in the remote OpenSSH server. In case the tool is compromised the security of the server can be broken easily. Such a key-management solution provides the ease of handling resources but comes at a price of security. The proposed 2FA, instead of solving the challenges of SSH key-management, would enhance the security of such systems by adding an extra trustless two-factor authentication layer to the OpenSSH server. Although the idea of using blockchain technology for 2FA is applicable for any kind of authentication system, in this work, we have

incorporated it with OpenSSH to enhance its security.

II. BACKGROUND

Blockchain is a relatively new technology which is being considered as the key enabler for many systems where having a centralized trusted third party among multiple parties belonging to the system is an issue, either due to lack of trust or being compromised. The idea of blockchain was first conceptualized by Satoshi Nakamoto in 2008 and subsequently he used this idea to implement first cryptocurrency - the Bitcoin [4].

It is basically a decentralized database that can store the registry of assets and transactions across a peer-to-peer network. These transactions are secured through cryptography and over time the transactions are locked in the form of blocks of data. This creates an immutable and unforgeable record of all the transactions across the network and replicated on every computer that is participating in the network. This replication of the database over the entire network makes the blockchain hard to tamper with. From the implementation perspective, the blockchain can be seen as a sequence of hash-chained records in which only new records can be added, but old records cannot be deleted. These records are added to the blockchain only after being validated. These validity conditions depend upon the type of application running on the blockchain and are enforced by the confesses among the participating parties. In other words, the chain with invalid blocks cannot survive. However, a mechanism is required to decide between alternative blockchains. A quite complete explanation of the operation and technical details of the blockchain technology is available here [5].

Smart contract [6] is the way of programming blockchain. Smart contract allows users to create their own operations of any complexity they wish. In this way, it serves as a platform for many different types of decentralized blockchain applications, including but not limited to cryptocurrencies.

Ethereum [7] is a blockchain which supports smart contracts. At the heart of smart contract is Ethereum Virtual Machine (EVM) which executes the code of arbitrary algorithmic complexity. Like any blockchain, Ethereum also includes a peer-to-peer network protocol. The Ethereum blockchain database is maintained and updated by many nodes connected to the network. Every node of the network runs the EVM and executes the same instructions. For this reason, Ethereum is sometimes described evocatively as a “world computer. This massive parallelization of computing across the entire Ethereum network is not done to make computation more efficient. In fact, this process makes computation on Ethereum far slower and more expensive than on a traditional “computer. Rather, every Ethereum node runs the EVM in order to maintain consensus across the blockchain. Decentralized consensus gives Ethereum extreme levels of fault tolerance, ensures zero downtime and makes data stored on the blockchain forever unchangeable and censorship-resistant and trustless.

III. RELATED WORK

To the best of our knowledge, there are very few works that explore the use of blockchain for designing better 2FA platform. Here, we discuss a couple of them, and also the works which in general have tried solving authentication issues using the blockchain technology.

In [8], the authors propose a 2FA mechanism to enhance the security of Bitcoin protocol. The idea is to secure the private key of the Bitcoin wallet, by splitting the private key into two separate devices so that both the devices are required to be present to perform a transaction. They implemented this concept, using the two-party Elliptic Curve Digital Signature Algorithm (ECDSA) signature protocol [9]. In ECDSA signature protocol, a transaction can be signed only with the collaboration of both devices without recombining the private key. This mandates that every transaction made from the wallet require the presence of both devices, which enhances the security of the Bitcoin wallet.

Ben Cresitello et al., in [10], explains another way of implementing the 2FA platform. They suggested that instead of using SMS (Short Message Service) of signed tokens for 2FA, we can use a common blockchain that acts as an identity store and has all the required information related to user identity and authorization. The idea is, for any application that requires 2FA, we can interface the application to the blockchain so that, every time a user tries to login using normal credentials, he would be required to submit a signed piece of data to the application. The signature for this piece of data can easily be verified via the blockchain as all public keys are already stored in the blockchain.

Bamert et al., in [11], introduced the idea of a hardware token for Bitcoin. The device having the token communicates using Bluetooth and can perform secure Bitcoin transactions by signing it. Similarly, to enhance the trustworthiness among the Bitcoin users, into the system, Ateniese et al. [12], propose a certification system for Bitcoin that allows only certified users to send and receive Bitcoins.

IV. PROPOSED BLOCKCHAIN-BASED FRAMEWORK FOR TWO-FACTOR AUTHENTICATION

In this paper, we propose a solution for 2FA without the need of having a centralized trusted third party required for the purpose of generating and distributing tokens to the users whenever they are being authenticated. In order to implement the proposed solution, we use Ethereum blockchain which gives the ability to program the logic using smart contracts for 2FA token generation and distribution in a trustless decentralized manner. The proposed solution is deployed on the Ethereum blockchain and subsequently integrated with the OpenSSH server to enable 2FA for OpenSSH clients. Fig. 1, depicts the major modules of proposed blockchain based 2FA architecture in OpenSSH. In the following, we describe the various components involved in the proposed framework for user authentication. The process of authentication using this framework is described in the next section.

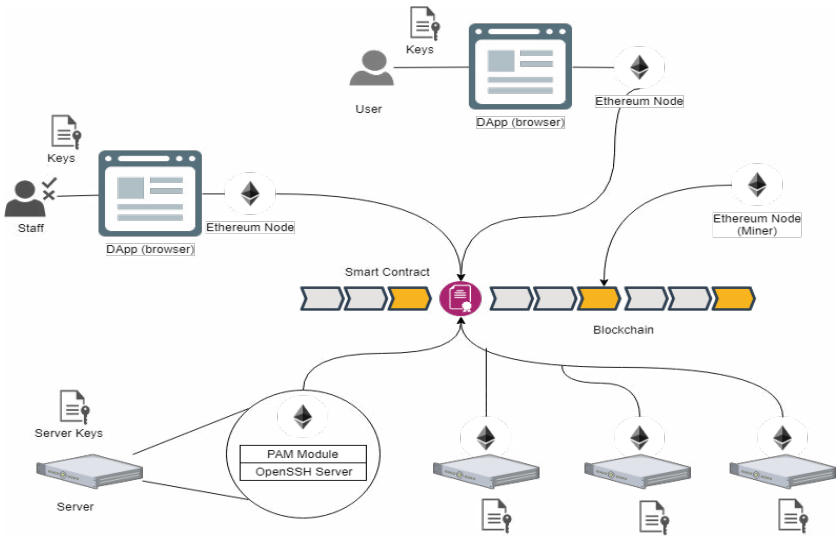


Fig. 1: Blockchain based Two-Factor authentication in OpenSSH

A. Ethereum Blockchain

Ethereum is a secure decentralized ledger which offers scalable and user-friendly ecosystem for building trustless decentralized applications. It is a proof-of-work based blockchain and barebone for cryptocurrency Ether. Ethereum can be considered as distributed computing platform where smart contracts are executed by anyone paying necessary charges. The integrity of the execution of the smart contract relies upon the miners and underlying consensus protocol that ensures that only honest miners are incentivized.

B. Smart Contract

Ethereum provides smart contracts that can be programmed to execute application logic based on various input parameters or events along with utilizing the non-volatile storage provided by the distributed ledger. Smart contracts are mainly programmed in a language called Solidity [13] which upon compilation is converted into a bytecode which can be executed on the Ethereum Virtual Machine. This bytecode is stored in the blockchain and has a unique identifier associated with it known as contract address. With this contract address, we can execute various functions of the smart contract along with passing required arguments as input. To design a 2FA system, we have the following functionalities programmed in the smart contract. The details of these functionalities are discussed in section V.

- User administration.
- System administration.
- Storing secure access tokens (storing OTP in SHA3 form)
- Providing access to the system
- Revoking access of system in case of multiple incorrect attempts
- Provides workflow administration via approving/disapproving requests based on staff requirements.

C. DApp (Decentralized Application)

A decentralized application (DApp), is a service that provides user-friendly interaction with smart contracts [14]. Ethereum DApps typically interface users via an HTML/JavaScript web application using a JavaScript API to communicate with the Ethereum blockchain. DApp connects to local blockchain node via JSON-RPC API [15] through which it can directly execute various functions on blockchain via JavaScript calls. All these JavaScript function calls are a part of Ethereum's Web3 Stack API, which allows communication with local Ethereum client. Typically, DApps have their own suite of associated contracts on the blockchain which they use to encode business logic and allow persistent storage of their consensus-critical state. Our DApp provides the following functionalities to the users.

- Provide web-based UI design for smart contract
- Take inputs from a user and store them in the blockchain.
- Retrieve the user data stored in the Smart Contract and display it to the user.

D. PAM (Pluggable Authentication Modules)

Pluggable authentication module [16] is a mechanism provided Linux operating system to integrate multiple low-level authentication schemes into a high-level API. This allows users to write their authentication schemes and integrate it with OS along with default methods. By default, OpenSSH server has its own method for authentication of users, but the software has a feature, where users can extend the authentication mechanism by integrating custom PAM modules. For the scope of this project, we have developed our own PAM module which is interfaced with blockchain using the JSON RPC API provided by the Ethereum. Based on the token input during the login screen, PAM queries the authenticity of token via blockchain, and based upon the response it informs the OpenSSH server regarding the decision of granting access.

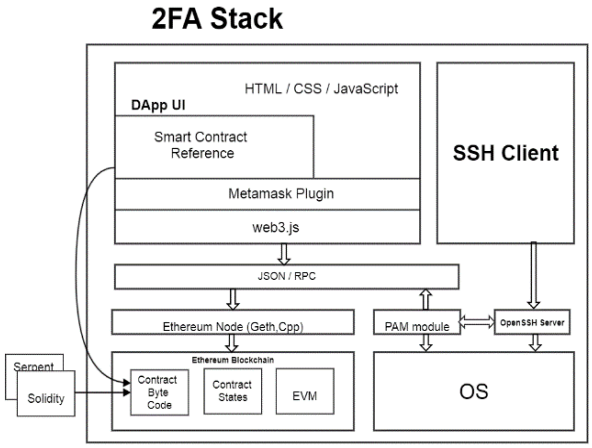


Fig. 2: Software stack for Blockchain based two factor authentication

All the servers as a part of this ecosystem are required to have their own configured Ethereum and PAM module loaded for its proper functioning.

Integrating the PAM module with the OpenSSH provides a more dynamic and flexible solution as the responsibility of authentication rests on the PAM module rather than the OpenSSH server. In contrast to our approach, the existing tools for 2FA are typically based on configuration modification at the server side, and therefore are less flexible.

E. Software Stack

Fig. 2, outlines the architecture of proposed blockchain based 2FA authentication in the form of a software stack comprising of operating system, blockchain, plugin, user interface, SSH-client, and SSH-server. The diagram mainly shows interaction and coupling between various software modules. It is moreover a layered design where the APIs, exposed by the lowers layers are used by the higher layers. The Dapp UI uses the web3.js library provided by the Metamask plugin to communicate with the blockchain. Both the Metamask plugin and the PAM module use the JSON/RPC API provided by Ethereum to interface with the blockchain. The OpenSSH server relies upon the PAM module for authentication functionalities that are part of our 2FA scheme.

V. Authentication Process According to Proposed Framework

All the entities involved in the proposed framework for 2FA can communicate with the smart contract only with the interface provided to them. Role-based authorization is imposed so that they cannot impersonate one another. As per our architectural design, there are mainly two type of users who interact with the smart contract.

User: Employee of the company who wants access to the server resources.

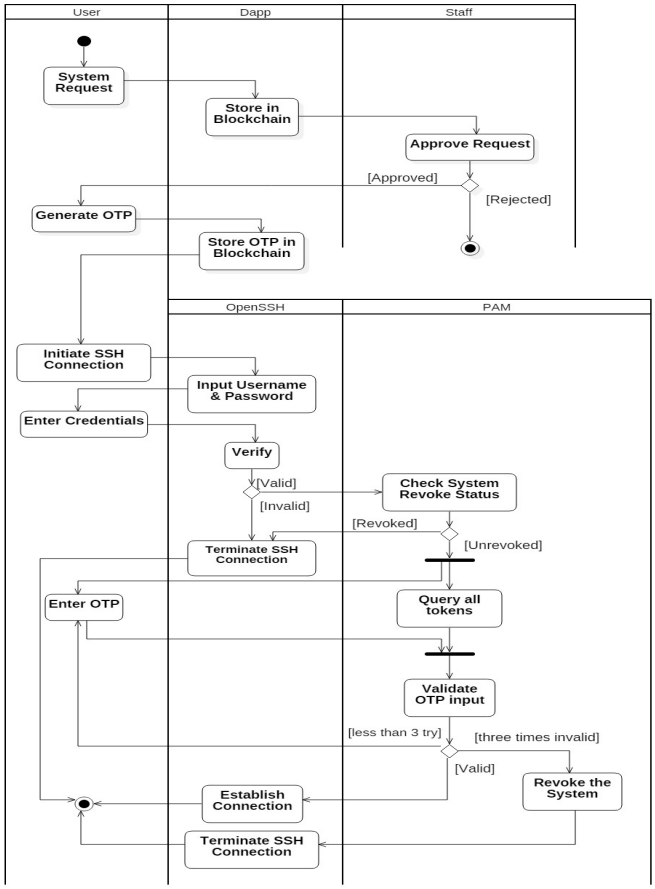


Fig. 3: Sequence of interactions between the various components (user, Dapp Staff, OpenSSH server and the PAM module) in the proposed blockchain based 2FA

Staff: An administrator level entity of the company who would be responsible for provisioning of servers to various users based on the requirement.

All entities (Users, Staff, and Servers) possess a public-private key pair, and each of these key-pairs have a corresponding unique Ethereum Address associated with them to serve as the unique identifier for these entities.

Initially, the staff registers all the servers and users based on their Ethereum address in the smart contract. Once a user is registered, he/she can generate a request to allow him to access any of the servers registered in the smart contract. These requests can be accepted or rejected based on the staff decision. If a user is allowed for access a particular server, he/she can generate a random six-digit numeric OTP which is then hashed with a SHA3 algorithm and be eventually stored in the blockchain. As blockchains are completely synced across all the nodes, this data would be available at every node. At a later time, when a user tries to SSH to a particular server and enters the same OTP, the PAM module of the server will query all the hashed token corresponding to itself from the blockchain and will try to find the hash of the entered OTP. If

it is found it will allow access to the user otherwise connection would be terminated. In this manner, we used blockchain to serve as distributed computing and storage platform to carry out various functionalities related to 2FA.

Now, we illustrate the working of the proposed blockchain based 2FA mechanism in term of the actions taken by different entities of the system.

A. Workflow (User/Staff perspective)

- Staff register a system which is part of this infrastructure.
- Staff also, register the set of users who can communicate with the smart contract.
- User sees all the system available and places the request for the system which he wishes to use.
- Staff sees all the requests made by the user and accepts them.
- User can now generate an OTP and save it in blockchain for the approved system.

B. Workflow (PAM perspective)

- User initiates an SSH connection to the system which he wishes to use.
- Initially, the SSH will authenticate the user using a normal username and password as a part of normal SSH authentication mechanism.
- PAM module will check in the blockchain via JSON RPC API calls, whether the system is revoked or not. If the system is already marked as revoked, it will not allow access to the user.
- If the System is not revoked, it will provide a prompt to the user to enter his OTP for this access. At this point, the module will retrieve all the tokens which are stored in the form of SHA3 hash from the blockchain.
- If the SHA3 hash of OTP entered by the user matches any of the tokens, then a request would be generated to delete that particular token, to prevent its future reuse and at the same time system access would be provided to the user.
- If OTP is incorrect, an appropriate message will be shown, and on three incorrect OTP entries the system would be marked as revoked in the blockchain and would not allow any further ssh access until staff de-revokes it. (PAM module would be pre-configured with smart contract address and System Ethereum address at compile time.)

Fig. 3, demonstrates the end-to-end process of proposed blockchain based 2FA scheme by showing the sequence of interactions between the user, OpenSSH server and the PAM module, that take place whenever a user wants to establish a secure SSH connection with the server.

VI. Implementation

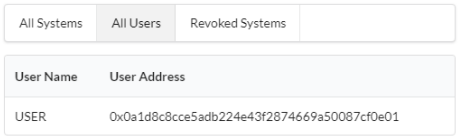
We implemented the proposed framework over the Ethereum blockchain. During the process of development of the project, the testing was conducted on the Ganache which is a well known Ethereum testing platform. Ethereum is a public blockchain platform, but it also provides features to deploy a private blockchain. There is no point to deploy such a solution on a public chain as it would incur the cost and would be significantly slower in execution than a private chain. In terms of an organization perspective, a private chain would be the most optimal solution as they can have a specific set of nodes for mining and the chain size would significantly smaller that would allow scalable storage of tokens across the server machines. As in terms of practical deployment, the organization would have miner nodes deployed under a secure infrastructure which would ensure that the smart contract execution is done with honest consensus at the same time providing fault tolerance.

To achieve the required functionalities, we programmed our own smart contract with specific functions that can be called via DApp. Smart contract once deployed generates a unique contract address that is configured in the DApp and all the PAM module located on every server. Each smart contract function has been allotted specific visibility and modifiers that prevent unauthorized/unwanted access to these functions and widely recommended by Ethereum community. The account which deploys the smart contract is by default allotted as staff account, and it is utmost necessary that the keys of staff accounts be kept private. More complex authorization logic could be easily developed based on organization requirements.

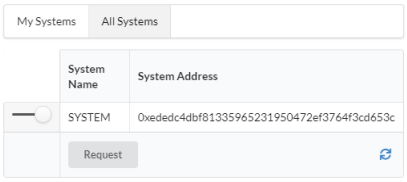
As part of the implementation, all the servers we loaded with the PAM module developed by us. The Linux operating system was then configured to execute our PAM module along with default username/password based authentication provided by the operating system. Also, the OpenSSH server is required to be configured to use PAM based authentication instead of its default key-pair based mechanism. And finally, the PAM module is configured with the Ethereum node details and contract address.

We developed DApp using HTML and JavaScript along with few JavaScript libraries to provide a user-friendly interface to the users. To interact with the Ethereum node Metamask browser plugin [17] is used that allows DApp to communicate with remote Ethereum node and also allows users to easily load/unload their Ethereum keys into the DApp. With this plugin, the users are no longer required to set up their own personal Ethereum node to interact with the smart contract. This plugin can be configured to digitally sign all the Web3 transactions and send it to a common gateway, where all the transactions are added to the pool and processed by the miners. As part of a real-world deployment, we need only one single Ethereum endpoint, where all the users can configure this endpoint in the plugin and start making their transactions/DApp interaction.

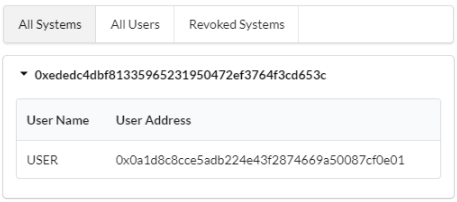
Fig. 4, it shows a couple of screenshots that we have taken during the test run of our implementation corresponding to the proposed 2FA using blockchain. Fig. 4(a) shows the information about the user that is registered by the staff. Fig. 4(b) shows the information of the system that is registered in



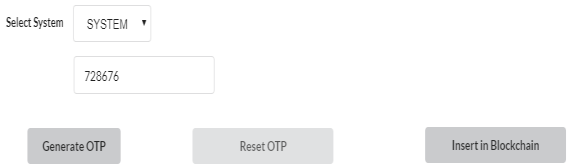
(a) Staff registers a user



(b) Staff registers a system



(c) Staff accepts the request of an user to access a system



(d) User generates the OTP and inserts into blockchain to be used at the time of authentication

Fig. 4: A sequence of snapshots taken during a test run of the proposed 2FA using blockchain

the blockchain by the staff. Fig. 4(c) shows the user request being accepted by the staff and as a result of this the user is allocated to that system. Fig. 4(d) shows the user selecting a particular system and generating a random OTP, and on clicking 'Insert in Blockchain' the OTP would be stored in the blockchain.

VII. Conclusions

In this work, we proposed a mechanism for two-factor authentication (2FA) using smart contracts in blockchains. The major advantages of the proposed design are that it is decentralized and does not require the presence of any trusted third party. Additionally, the proposed mechanism does not require to be integrated with SMS or mailing systems, as the user himself is generating the random token which is contrary to the traditional design. Finally, it is completely free and can easily be deployed on a private infrastructure.

One of the drawbacks of this system is that any changes in the smart contract would require modification of all the PAM modules deployed on the servers. Also, network downtime can hamper ssh access as nodes rely on each other to pull the latest changes in the smart contract. To counter these issues, rather than installing Ethereum client on all the server, we can have a common gateway to which all PAM module can connect via API. For such a common gateway design, all the PAM modules would require access of Ethereum private keys to be able to digitally sign JSON RPC API calls, such a library is not available in C language as of now and neither is this design in the scope of this paper.

References

[1] E. D. Cristofaro, H. Du, J. Freudiger, and G. Norcie, "Two-factor or not two-factor? A comparative usability study of two-factor authentication," *CoRR*, vol. abs/1309.5344, 2013. [Online]. Available: http://arxiv.org/abs/1309.5344

[2] K. Abhishek, S. Roshan, P. Kumar, and R. Ranjan, "A comprehensive study on multifactor authentication schemes," in *Advances in Computing and Information Technology*, N. Meghanathan, D. Nagamalai, and N. Chaki, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 561–568.

[3] D. Miller, "Security measures in openssh," *Proceedings of the AsiaBSDCon 2007, Usenix*, 2007.

[4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[5] J. Kehrli, "Blockchain explained," https://www.niceideas.ch/roller2/badtrash/entry/blockchain-explained-beta, accessed: 2018-06-28.

[6] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, 2014.

[7] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[8] C. Mann and D. Loebenberger, "Two-factor authentication for the bitcoin protocol," *International Journal of Information Security*, vol. 16, no. 2, pp. 213–226, 2017.

[9] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, Aug 2001. [Online]. Available: https://doi.org/10.1007/s102070100002

[10] B. Cresitello-Dittmar, "Application of the blockchain for authentication and verification of identity," www.cs.tufts.edu/comp/116/archive/fall2016/bcresitellodittmar.pdf, 2016, accessed: 2018-07-13.

[11] T. Bamert, C. Decker, R. Wattenhofer, and S. Welten, "Bluewallet: The secure bitcoin wallet," in *International Workshop on Security and Trust Management*. Springer, 2014, pp. 65–80.

[12] G. Ateniese, A. Faonio, B. Magri, and B. De Medeiros, "Certified bitcoins," in *International Conference on Applied Cryptography and Network Security*. Springer, 2014, pp. 80–96.

[13] "Solidity documentation," http://solidity.readthedocs.io/en/v0.4.24/, accessed: 2018-06-28.

[14] J. Ray, "Decentralized apps (dapps)," https://github.com/ethereum/wiki/wiki/Decentralized-apps-(dapps), accessed: 2018-06-28.

[15] "JSON-RPC:A light weight remote procedure call protocol," https://www.jsonrpc.org/, accessed: 2018-06-25.

[16] V. Samar, "Unified login with pluggable authentication modules (pam)," in *Proceedings of the 3rd ACM conference on Computer and communications security*. ACM, 1996, pp. 1–10.

[17] "Metamask browser plugin," https://metamask.io/, accessed: 2018-06-28.