Securing NATted IoT Devices Using Ethereum Blockchain and Distributed TURN Servers

Elie Kfoury, David Khoury
Department of Computer Science
American University of Science and Technology
Beirut, Lebanon
{ekfoury,dkhoury}@aust.edu.lb

***Abstract*— Peer-to-Peer (P2P) networking is a decentralized network topology that enables parties to communicate directly without central servers. The main obstacle preventing the heavy deployment of the P2P topology is the Network Address Translation (NAT) which serves as a solution for the exhaustion of IPv4 addresses. Methods proposed by the Internet Engineering Task Force (IETF) to solve the NAT traversal issues include Simple Traversal of UDP through NATs (STUN) and Traversal Using Relay NAT (TURN). STUN is limited by the type of deployed NAT, and TURN is limited by the peers' discovery mechanism which is application dependent. In this paper we propose a Blockchain-based platform that enables TURN servers to act as relays for Internet of Things (IoT) devices behind NAT. It also provides End-to-End (e2e) security for Constrained and Non-Constrained IoT devices. Results showed that the system has minimal impact on the existing network and can be a potential solution for advancing IoT deployment.**

Keywords—Blockchain; Ethereum; PKI; PSK; Public Key Distribution, CA, trust, IoT, Smart Contract

I. INTRODUCTION

In a Peer-to-Peer (P2P) network, no service application infrastructure or central servers are required for exchanging data between devices [1]. Most of today's emerging applications (Internet of Things, Voice over IP, Real Time services, etc…) benefit from this technology [2] to interchange End-to-End (e2e) data without the hassle of going through a middle entity. As a result, this approach eliminates the cost of installing and maintaining a complex network, and enhances security by removing the single point of attacks. The main obstacle preventing the heavy deployment of the P2P topology is the Network Address Translation (NAT) [3] which serves as a solution for minimizing the exhaustion of IPv4 addresses [4]. The methods proposed by IETF to solve the NAT traversal issues are mainly the Simple Traversal of UDP through NATs (STUN) [5] and Traversal Using Relay NAT (TURN) [6]. They are based on the intelligence in the clients together with an external server. STUN server discovers the type of the NAT, client's public IP, port and sends it back to the client. STUN only operates with specific types of non-secure NATs and cannot work with the symmetric type; moreover, the internal client suffers from external attacks like capturing the STUN traffic [7]. TURN on the other hand, allows an end point behind a NAT to receive traffic on either TCP or UDP ports and solve all the problems of NAT mapping (including symmetric NAT). The TURN Server acts as a relay, forwarding any received data, and allowing inbound traffic through a firewall, with only the TURN client in control. TURN solves the problems of NATs, but the main issue of the P2P connection in TURN is the discovery of the peer's destination IP address and port, especially when they are behind symmetric NATs. Different types of devices can be installed behind a NAT. For instance, IoT devices, which consist of a processing unit, sensors/actuators, and communication module, are typically located behind NATs. Beside NAT issues, security and privacy are critical issues facing the development of IoT networks [8]. Most existing security solutions are based on Public Key Infrastructures (PKI) backed by Certificate Authorities (CAs). These solutions require a large infrastructure to manage and provision certificates. Besides authenticating the servers, a device must own a client certificate to authenticate itself. According to Ericsson research, it is impractical and nearly impossible to distribute and provision client certificates, especially in IoT networks [9]. Furthermore, studies showed that the problem of subverting or having a rogue CA exist as the security here depends completely on trust [10]. Blockchain technology on the other hand provides a decentralized network of nodes that follow a consensus algorithm (Proof-of-Work, Proof-of-Stake, …) to agree on the integrity of the shared ledger; the ledger contains all transactions from the first block (Genesis block) till the last mined transaction [11]. Due to its distributed nature, it also ensures availability as there is no single point of failure.

In this paper we propose a platform that facilitate P2P routing for IoT devices using the Blockchain technology. It enables TURN servers to act as relays for IoT devices. It also allows secure Pre-Shared Key (PSK) and public key distribution to establish e2e encrypted sessions between IoT devices. This work is part of an ongoing research project on developing a generic secure communication platform using Blockchain technology [12] [13] [14]. The main contributions of this platform include: (1) Providing a discovery mechanism for TURN allocations mappings (2) Enabling IoT device authentication without PKI/CA, (3) Distributing securely a PSK between two devices, and (4) Identifying IoT devices to facilitate addressing and routing. The rest of this paper is organized as follows: Section II overviews the theoretical background of TURN, Blockchain technology, and IoT device types. Section III introduces the proposed method and its advantages comparing to the existing architectures. Implementation and results are demonstrated in section IV. Section V provides some concluding remarks and the intended future work.

II. BACKGROUND

A. Traversal Using Relays around NAT (TURN)

NAT is proposed as a solution for preserving IPv4 addresses. It enables devices on a local network to connect to outside network using a private IP. While NAT provides several benefits, it also imposes major limitations and drawbacks on applications. TURN, proposed in RFC5766, is a public relay server residing on the Internet that allows a client to obtain a transport address used for sending and receiving packets. TURN mechanism can be summarized as follows: 1) the client contacts the TURN server by sending an "Allocate" request. 2) The server replies back with an "Allocate Successful" response containing the relayed transport address assigned to the client at the TURN server. 3) The client sends a "CreatePermission" request to the server containing the peer's address as attribute. 4) After successfully creating permission, the client can send data using two mechanisms: Send indications or ChannelBind. 5) Once the server receives the data from the client, it forwards it to the peer as UDP datagram. 6) The peer replies with UDP datagram to the server. 7) The server forwards the data to the client. The relayed transport address and the peer's reflexive address should be communicated between TURN clients to enable data communication. Usually this is application dependent: In Voice over IP (VoIP) systems, the client uses the Session Description Protocol (SDP) [15] of the Session Initiation Protocol (SIP) [16] to communicate its relayed transport address.

B. Blockchain Technology

Blockchain is an emerging technology introduced by Satoshi Nakamoto in his Bitcoin paper [17]. It consists of a public ledger containing all transactions from the Genesis block (first block) till the current transaction. There is no single controlling computer in charge of maintaining the database. Consensus algorithms and protocols [18] are used by the validating nodes to agree on the validity of a particular version of the Blockchain. The use of the Blockchain technology deals with confirming the integrity of data associated with the transaction. This feature is considered a key for securing the integrity of networked devices.

C. Ethereum Blockchain and smart contracts

Ethereum is an open source Blockchain platform featuring smart contract scripting functionality [19]. A smart contract is a program containing rules enforcing its execution, distributed on the Blockchain network. It can be deployed by any wallet containing sufficient Ether and has a function-based interface. Each contract has an address which is used to reference it once deployed to the Blockchain network. A user can call a smart contract function by sending a transaction with this address as the destination, and with the data payload of the transaction containing the function signature and input parameters. The miners, which are public nodes with decent computation capability, execute the program on each function call without having to trust them. Since a smart contract has an Ethereum address, it can accept and send Ether, which is considered as the fuel of the Ethereum network. Calling a function that changes the state of the Blockchain requires sufficient Ether by the caller [19].

D. Light Client - LES (Light Ethereum Sub-protocol)

As explained previously, Blockchain technology requires nodes to download the whole ledger to participate in the system. However, this requires a large memory to store the historical transactions (from the Genesis block). The size of the chaindata is approximately 450 GB (March 2018). Therefore, end-user devices (mobile, IoT, computers …) cannot participate due to the memory and storage constraints. Fortunately, an alternative to the Ethereum full node is the Light node [20]. The lightchain data is approximately 0.005GB, which is affordable on lightweight devices. Moreover, this lightchain is downloaded *once* on the device for all Ethereum decentralized apps. Merkle Tree [21] data structure is the fundamental element of the light client. It uses the Merkle proof for efficient and secure verification of the requested Blockchain contents. To query data from the Blockchain, the client initiates a request to a light client server. The server simply finds the object, fetches the Merkle branch and replies back to the light client with the branch. The branch is a list of hashes going up from the object to the tree root.

E. Public or Private Blockchain

Depending on the application, Ethereum could be deployed as a private or public Blockchain. Public Blockchain allows any user with an Ethereum wallet address to query data, send transactions to other addresses, and explore the network using public explorers (Etherscan [22] for example). Therefore, all transactions' statuses (mined by public nodes after the consensus, pending, out of gas …) can be monitored by the user. Transparency and non-obscurity of the contract code and data is the main advantage of this type of Blockchain. Users can explore the functions and make sure that they are protected from the application's owners and developers. Private Blockchain on the other hand is a permissioned Blockchain with restrictions on who can participate. It allows forming a consortium regarding a particular domain (for example, financial institutions need the approval of 10 participating banks for their transactions to be valid). The consensus mechanism here is predefined by the initiators of the network. Private Blockchain is not as trustless and secure as the public as its rules are defined by the originator.

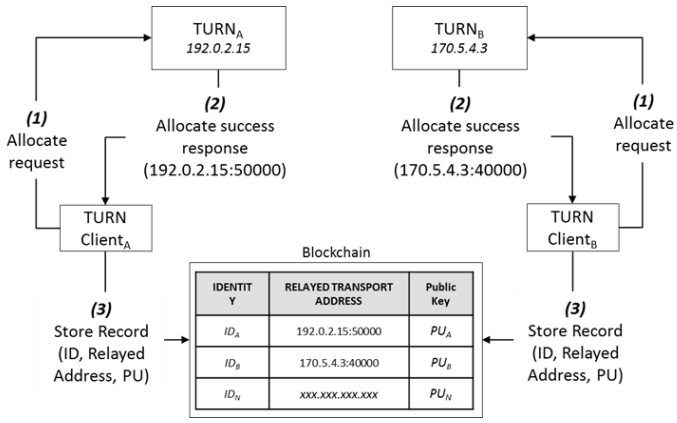
F. IoT Device Types

IoT devices are classified into two main categories: 1) Constrained: These are devices that have limited memory and processing capabilities. They are usually behind a border router (which is also a NAT device) that helps them connect to the Internet. 2) Non-Constrained Devices: These are devices that can connect directly to the Internet without the need of a gateway. For example, an IoT device attached to a vehicle publishing data through the Mobile Network Operator (MNO) [23]. Another example is a wearable device publishing patient's health data to a remote server.

III. PROPOSED SYSTEM

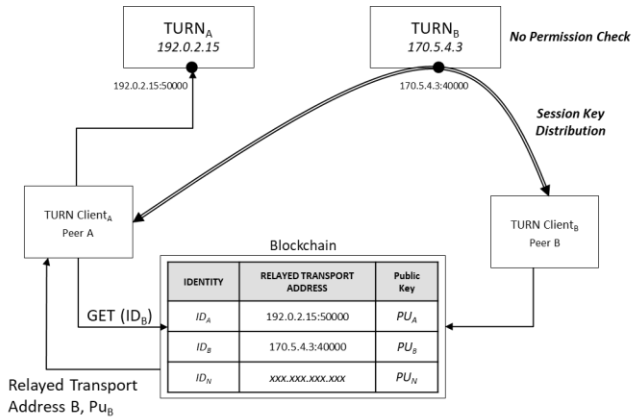
The proposed solution is based on public TURN servers, a modified TURN client and the Ethereum Blockchain network. It consists of deploying many TURN servers in the network as means to relay communication data between two peers. The Blockchain is used by the clients as a distributed database that stores their Identity, Relayed Transport Address provided by TURN, and Public key. The TURN server function is reused without any changes to the standard, but configured to allow communication traffic with no permission checks. Usually peers are not allowed to send data towards the TURN client via the relay before permission is installed for that peers. The high-level proposed allocation mechanism is depicted in Fig. 1. Each client (A and B) acts as a TURN client and peer at the same time. According to the RFC, a TURN client gets its Relayed Transport Address after allocation (Messages in Steps 1 and 2). Each peer is connected continuously to a TURN server in the network through a keep alive message. After the allocation, the TURN clients store in Ethereum Blockchain the aforementioned attributes (Step 3).

Fig. 1. Proposed System's allocation mechanism



When a peer A wants to communicate with a peer B (Fig. 2), it contacts the Blockchain, retrieves the Peer B's Relayed Transport IP and port, and establishes a connection towards the B's relayed IP address. Before Peer A and B start exchanging payload data, a session key is generated and distributed among the peers to secure further payload data.

Fig. 2. Session Establishment Mechanism



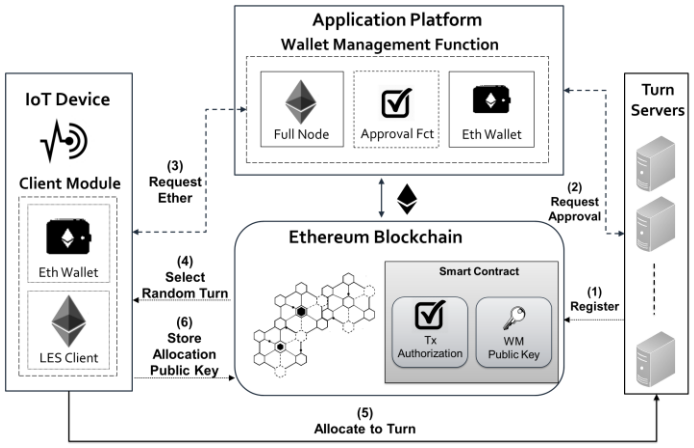
The session key is distributed based on the generic protocol for PSK distribution method described in Section B, where peers can retrieve public keys of each other from the Blockchain. When the key is distributed between peer A and B, the secure payload data is exchanged transparently through the Relayed Transport address in TURN.

A. System Architecture and Components

The software components of this distributed platform consist of the following:

1. Wallet Management Function (WM): Network function in the platform located on the server side that transfers Ether to the client module on the IoT device, and approves the Blockchain storage requests.
2. TURN Servers: Server with a public IP that registers to the Blockchain and acts as a relay between IoT devices. Any TURN server can participate in the system to expose its IP for IoT devices.
3. IoT Client-Module: Plug-in software module installed on the IoT device. It transparently creates an Ethereum wallet and randomly allocates the IoT device to a registered TURN server. It then securely stores its public key and relayed transport address in the Blockchain. Finally, it distributes a PSK between the communicating parties.
4. Smart contract: Software deployed to the Ethereum Blockchain network containing the needed functions to store/retrieve keys and TURN allocations.

Fig. 3. System Architecture



1) Configuration

The configuration mechanism is divided into two main parts: TURN Servers and IoT Devices:

a) TURN Servers: As mentioned earlier, any server with a public IP can be registered to the Blockchain. An Ethereum wallet filled with sufficient Ether is required to call the *registerServer* function in the contract. Ether can be filled through external crypto exchange services [24].
Once the wallet is successfully filled, the TURN server calls the *registerServer* function on the contract by sending a signed

transaction. To validate that the registered server is a TURN server, a validation must be performed by the application platform. Hence, the TURN server sends an approval request to the platform which in turn validates its type (TURN). If the validation is successful, it approves the server storage on the smart contract.

b) IoT Devices: The devices accessing this platform should be registered and configured on the smart contract. Upon initialization, an identity is assigned to the IoT device; if no hardcoded identity is given, a Universally Unique Identifier (UUID) [25] is generated and used as the device's identity. Applications on other peers use this identity to identify and address the device. The client module also contains a token signed by the application platform. This Token is crucial to identify the client module installed on the device, and to ensure that the transaction fees required to store in Ethereum have been paid (in Ether crypto-currency). The confirmation process is discussed next in the Wallet Management Function section.

2) Wallet Management Function (WM)

This function is part of a global platform for any IoT device configuration and authentication. The download or the installation of the user's client-module generates automatically the public/private keys and creates an empty Ethereum wallet. The user does not need to own an Ethereum account or Ether wallet. The main role of the WM is to authenticate the IoT device and the TURN servers. It approves the server's storage by making a temporary test allocation. If the allocation is successful, it approves the server in the contract, making it publicly available for IoT devices. As for the client storage, it sends Ether to the client and approves the storage transaction. This function could contain a full or a light client Ethereum node to interface the Blockchain.

3) IoT Client-Module

When the client module is installed on the IoT device, an empty Ethereum wallet and public/private key-pair are generated. This module could be downloaded or integrated into the firmware of the device. It includes a light client node (LES - Light Ethereum Subprotocol) to interwork with Blockchain network.

To be able to perform the storage of the client's public key and relayed transport address in the Ethereum network, the module requests Ether by contacting the WM. The request contains the following payload: Wallet address, Identity, and $\{Token\}Pr_{WM}$, where $\{Token\}Pr_{WM}$ is the generated Token signed by WM's private key. The data is sent encrypted to WM using TLS-PSK as described later in Section B.

Upon receiving the encrypted payload, the WM verifies its signature on the token and if validated, it transfers Ether to the IoT device.

4) DPK Smart Contract

The smart contract is deployed on the Ethereum Blockchain network (See Appendix A for code listing). It contains the following functions:

1. registerServer(IP)
2. approveServer(IP)
3. addClient (ID, Pu_i, Tok_i, $RelayedAddress_i$)
4. getClientPuk(ID)
5. getClient (ID)
6. approveClient(Tok)

TABLE I. CLIENTS MAPPING TABLE - SMART CONTRACT

ID $_i$	**Public Key $_i$**	**Token i**	**Relayed Transport Address $_i$**
UD_1	Pu_1	Tok_1	192.0.2.15:50000
ID_2	Pu_2	Tok_2	170.5.4.3:40000
….	….	….	….
ID_n	Pu_n	Tok_n	171.1.2.3:45000

The Turn server's registration steps are as follows:

1. The TURN server calls the function *registerServer* in the smart contract by signing a transaction. The supplier parameter is the server's public IP address.
2. The server contacts the WM function and sends to it its public IP.
3. The WM function creates a test allocation to the received IP.
4. If the allocation is successful (valid TURN server), it approves the server storage by calling the *approveServer* function in the smart contract. This function is programmed to be invoked *only* by the WM function. This function allows the allocation to be validated and publicly available for IoT devices.

The client (IoT device) registration and approval sequence steps are as follows:

1. The client calls the function *addClient* in the smart contract by signing a transaction. This function adds any client to the pending list [12]. The input parameters are:
    a. Client Identity (ID): Hardcoded or UUID.
    b. Public Key (Ex: RSA 2048)
    c. Token: a string generated randomly and signed by WM.
    d. Relayed Address: The address returned from the successful allocation to a random TURN server.
2. The client module sends an approval request to the WM containing the signed token.
3. The WM calls the function *approveClient* in the smart contract by signing a transaction. The token is the only parameter used. Only the owner of the contract can approve the client: As shown in the code listing, the *approveClient* function has a condition that allows

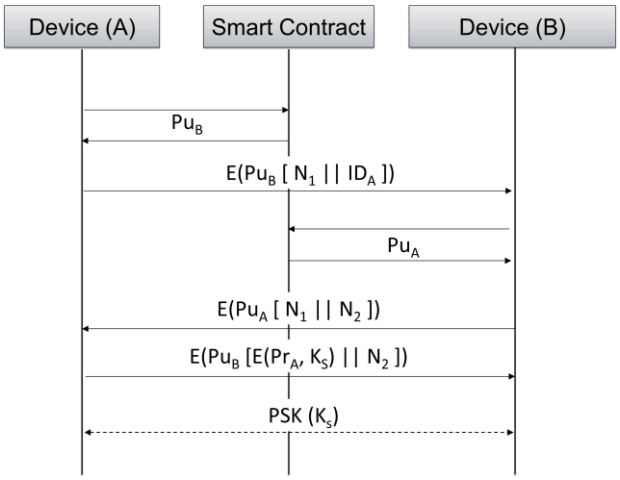
only the contract deployer (WM) to set the *approved* Boolean variable to *True*.

getClientPuk(ID) function is used to fetch the public key of the requested identity and *getClientRelayedAddress(ID)* function is used to fetch the relayed transport address of the ID parameter.

B. Generic Protocol for PSK Distribution

This generic protocol uses the Needham-Schroeder method [26] to distribute the Pre-Shared Key using asymmetric encryption as shown in Fig. 4.

Fig. 4. PSK Key Distribution



It is assumed that both devices (A and B) have stored their public keys Pu_A and Pu_B in the smart contract according to the mechanism described above. The PSK key distribution steps are as follows:

1. $Device_A$ requests the public key of the responder $Device_B$ from the Blockchain.
2. $Device_A$ encrypts with the public key of $Device_B$ Pu_B, a randomly generated nonce (N_1) which is used to identify this transaction uniquely, and the Identifier of $Device_A$.
3. $Device_B$ requests $Device_A$'s public key (Pu_A) from DPK.
4. $Device_B$ sends devices nonce (N_1) and a new nonce generated by $Device_B$ (N_2) encrypted with Pu_A. This nonce (N_1) helps $Device_A$ ensure that the correspondent is $Device_B$ as only $Device_B$ can decrypt the message.
5. $Device_A$ returns (N_2), encrypted using Pu_B.
6. $Device_A$ starts the pre-shared handshake protocol by generating randomly a session key K_S. This session key K_S is used as a pre-shared Key (PSK) for the security protocol adopted to encrypt the communication session between the two devices (TLS-PSK, DTLS-PSK, SRTP etc).
7. $Device_A$ encrypts with Pr_A the generated secret key K_S and then encrypts the whole message along with (N_2) with Pu_B. K_S is sent encrypted with the public key of the $Device_B$.
8. $Device_B$ decrypts with Pu_A and Pr_B to retrieve the secret session key K_S.

C. TURN server Allocation mechanism

To communicate with a peer, the TURN client contacts the smart contract to check which TURN server is allocating the destination peer. Next, it releases its allocation from the previously selected TURN server and then allocate to the TURN server of the destination peer. This ensures that both devices are allocated to the same server, and hence can exchange messages through their relayed transport addresses. TURN allocations are regularly updated in the smart contract when changed from the client side.

D. Secure Session Setup for Non-Constrained Devices

The symmetric key between the IoT devices is also distributed according to the Generic Protocol for PSK Distribution (Section B). As explained in the Background section, there are two mechanisms for sending data in TURN: 1) Send & Data and 2) Channel Binding. In our system we eliminated the permission check on the TURN server; therefore, a device can access its peer directly through its relayed transport address using UDP Datagram without having to create permission check on the server.

Fig. 5. Key Distribution and Secure Data Exchange

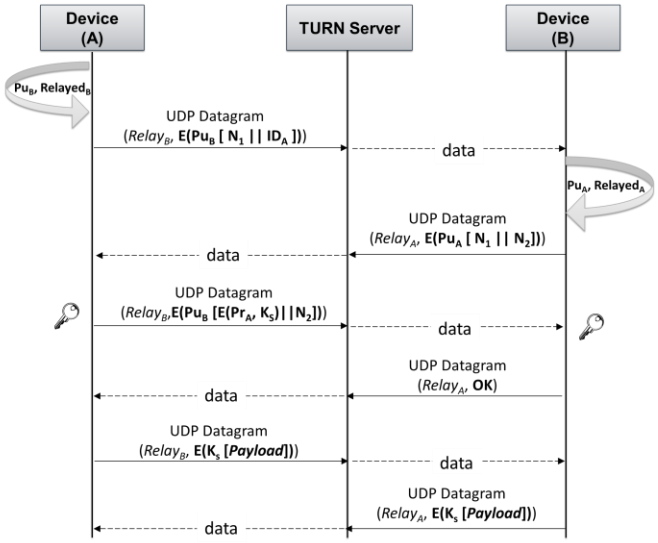


Fig. 5 demonstrates how the "Send & Data" method is used in our proposed system. Peers' public keys are fetched from the smart contract deployed on the Blockchain, and then the exchange of the PSK is achieved through Send and Data indications. Each indication (Send or Data) includes the ID of the peer and the payload to be sent.

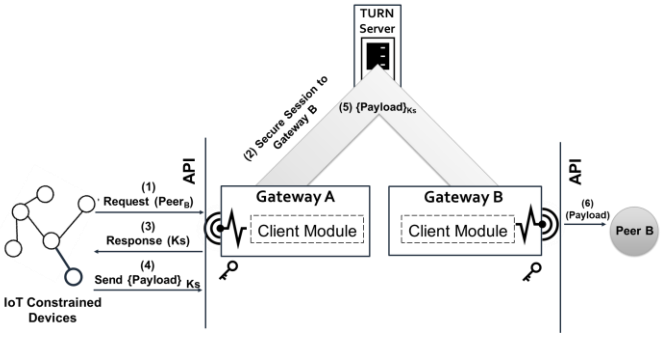
As devices in this mechanism are required to connect to the Blockchain, they require a light node. Moreover, they require a TURN client to send TURN messages to the server. Constrained nodes cannot use this mechanism since they do not possess the minimum requirements for maintaining these

software modules. In section E, we explain how constrained devices might benefit from the proposed system.

E. Secure Session Setup for Constrained Devices

Constrained devices as mentioned in the Background section are note expected to connect directly to the Internet [27]. Instead, they rely on a gateway installed on the border of the network.

Fig. 6. Secure Data Exchange in Capillary Network



When a NATted IoT constrained device wishes to communicate with a peer behind a NAT, it sends a request to the gateway containing the identity of B (Fig. 6). The gateway establishes a secure session to the gateway of B using the Non-Constrained devices method described in Section D. A response containing a session key is returned. The IoT device can now send encrypted payload to the gateway which in turns sends it to the destination peer as describe previously.

IV. Implementation

To validate the proposed system, we implemented the solution using various technologies. Solidity: Contract oriented programming language for writing smart contracts [28], NodeJS: Server side scripting (Wallet management server) [29], Client Module: Java implementation with Web3j [30] to interface the light client. The Ropsten Testnet [31] is used to simulate the Blockchain network.

Appendix A shows the smart contract code listing. Storing the public key and the relayed transport address in the Ethereum Testnet Blockchain took between 11 to 30 seconds to be validated. This time is only spent once upon configuration.

CoTURN, a free open source implementation of TURN and STUN Server [32] is used on TURN servers. When running a TURN server instance, the *--server-relay* flag is activated to enable relaying on endpoints without permission checks.

V. Conclusion and Future Work

In this paper we proposed a novel secure platform based on Ethereum Blockchain and public TURN servers to solve the issue of NAT traversal and e2e secure session setup. This solution works on any type of NAT (including symmetric) as it is based on TURN and any type of IoT device (constrained and non-constrained). This system showed to be feasible as it has minimal impact on the existing network infrastructure. Regarding the future work, we intend to replicate this system using Port Control Protocol (PCP) [33] instead of TURN to solve NATting issues. In this case, no TURN servers are needed to relay data between devices. This future work is dependent on the overall deployment of PCP-enabled operating systems on NAT.

Appendix (Smart Contract Code Listing)

```
pragma solidity ^0.4.2;
contract P2P
{
    uint clientsCount = 0;
    mapping(string => Client) clients;
    mapping(uint => string) clientsIndex;
    mapping (string => Server) servers;
    string wmKey;
    address public owner;
    uint serversCount = 0;

    constructor (string puk) public {
        owner = msg.sender;
        wmKey = puk;
    }

    struct Client{
        string identity;
        string relayedTransportAddress;
        string publicKey;
        address clientWalletAddress;
        bool approved;
    }

    struct Server{
        string transportAddress;
        bool approved;
    }

    function registerServer(string IP)
     public returns (bool success){
        servers[IP] = Server(IP, false);
        return true;
    }

    function approveServer(string IP)
     public returns (bool success){
        if(msg.sender == owner){
            servers[IP].approved = true;
            serversCount++;
            return true;
        }
        return false;
    }

    function addClient(string _identity,
     string _relayedTransportAddress,
     string _publicKey, string _token)
     public returns (bool success) {
        clients[_token] = Client(_identity,
        _relayedTransportAddress, _publicKey,
        msg.sender, false);
        return true;
    }

    function approveClient(string _token)
     public returns (bool success){
        if(msg.sender == owner){
            clients[_token].approved = true;
            clientsIndex[clientsCount] = _token;
            clientsCount++;
            return true;
        }
      return false;
    }
```

```
    function updateAllocation(string _rAddress,
     string _token)
     public returns (bool success) {
        if(msg.sender == clients[_token].clientWalletAddress)
         clients[_token].relayedTransportAddress = _ rAddress;
        return true;
    }

    function getClientPuk(string _identity)
     public constant returns (string publicKey) {
      for(uint i = clientsCount - 1; i >= 0; i--){
       if(keccak256(clients[clientsIndex[i]].identity)
              == keccak256(_identity))
             return clients[clientsIndex[i]].publicKey;
      }
      return "";
    }

    function getClientRelayedAddress(string _identity)
     constant returns (string publicKey) {
      for(uint i = clientsCount - 1; i >= 0; i--){
       if(keccak256(clients[clientsIndex[i]].identity)
              == keccak256(_identity))
             return
clients[clientsIndex[i]].relayedTransportAddress;
      }
      return "";
    }

    function getWMKey() public constant returns (string) {
        return wmKey;
    }
}
```

REFERENCES

[1] Fox G. Peer-to-peer networks. Computing in Science & Engineering. 2001 May;3(3):75-7.

[2] Yeo SS, Chen Y, Wang CL. Cloud, grid, P2P and internet computing: Recent trends and future directions. Peer-to-Peer Networking and Applications. 2015 Sep 1;8(5):835-7.

[3] Tsirtsis G, Srisuresh P. Network address translation-protocol translation (NAT-PT). 2000.

[4] Boucadair M, Grimault JL, Lévis P, Villefranque A, Morand P. Anticipate IPv4 address exhaustion: a critical challenge for internet survival. InEvolving Internet, 2009. INTERNET'09. First International Conference on 2009 Aug 23 (pp. 27-32). IEEE.

[5] Rosenberg, J., Mahy, R., Huitema, C. and Weinberger, J., 2003. STUN-simple traversal of UDP through network address translators.

[6] Matthews, P., Mahy, R. and Rosenberg, J., 2010. Traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun).

[7] Wang, Y., Lu, Z. and Gu, J., 2006, August. Research on symmetric NAT traversal in P2P applications. In Computing in the Global Information Technology, 2006. ICCGI'06. International Multi-Conference on (pp. 59-59). IEEE.

[8] Zhao, K. and Ge, L., 2013, December. A survey on the internet of things security. In Computational Intelligence and Security (CIS), 2013 9th International Conference on (pp. 663-667). IEEE.

[9] Novo, O., Beijar, N., Ocak, M., Kjällman, J., Komu, M. and Kauppinen, T., 2015, December. Capillary networks-bridging the cellular and iot worlds. In Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on (pp. 571-578). IEEE.

[10] Khoury, D. and Kfoury, E., 2017, September. Generic hybrid methods for secure connections based on the integration of GBA and TLS/CA. In Sensors Networks Smart and Emerging Technologies (SENSET), 2017 (pp. 1-4). IEEE.

[11] Pilkington, M., 2016. 11 Blockchain technology: principles and applications. Research handbook on digital transformations, p.225.

[12] E. Kfoury and D. Khoury, "Secure End-to-End VoIP System Based on Ethereum Blockchain", *Journal of Communications*, vol. 13, no. 8, pp. 450-455, 2018.

[13] E. Kfoury and D. Khoury, "Secure End-to-End VoLTE based on Ethereum Blockchain", in *41st International Conference on Telecommunications and Signal Processing*, Athens, Greece, 2018.

[14] E. Kfoury and D. Khoury, "Distributed Public Key Infrastructure and PSK Exchange Based on Blockchain", in *IEEE International Conference on Blockchain*, Canada, 2018.

[15] Handley, M., Perkins, C. and Jacobson, V., 2006. SDP: session description protocol.

[16] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and Schooler, E., 2002. SIP: session initiation protocol (No. RFC 3261).

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

[18] Mingxiao, D., Xiaofeng, M., Zhe, Z., Xiangwei, W. and Qijun, C., 2017, October. A review on consensus algorithm of blockchain. In Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference on (pp. 2567-2572). IEEE.

[19] Wood, G., 2014. Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper, 151, pp.1-32.

[20] "Light client protocol," Sep. 2016. [Online]. Available: https://github.com/ethereum/wiki/wiki/Light-client-protocol

[21] Merkle, R.C., Leland Stanford Junior University, 1982. Method of providing digital signatures. U.S. Patent 4,309,569.

[22] Team, E., 2017. Etherscan: The Ethereum block explorer. [Online]. Available: https://etherscan.io/

[23] Nasr, E., Kfoury, E. and Khoury, D., 2016, November. An IoT approach to vehicle accident detection, reporting, and navigation. In Multidisciplinary Conference on Engineering Technology (IMCET), IEEE International (pp. 231-236). IEEE.

[24] Swan, M., 2015. Blockchain: Blueprint for a new economy. " O'Reilly Media, Inc.".

[25] Leach, P.J., Mealling, M. and Salz, R., 2005. A universally unique identifier (uuid) urn namespace.

[26] G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. In International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, pages 147–166. Springer, 1996.

[27] Bormann, C., Ersue, M. and Keranen, A., 2014. Terminology for constrained-node networks (No. RFC 7228).

[28] Dannen, C., 2017. Introducing Ethereum and Solidity. Apress.

[29] Wilson, J., 2018. Node. js 8 the Right Way: Practical, Server-side Javascript that Scales. Pragmatic Bookshelf.

[30] Web3j library. [Online]. Available: https://github.com/web3j/web3j

[31] Iyer, K. and Dannen, C., 2018. The Ethereum Development Environment. In Building Games with Ethereum Smart Contracts (pp. 19-36). Apress, Berkeley, CA.

[32] CoTURN – [Online]. Available: https://github.com/coturn/coturn

[33] Wing, D., Cheshire, S., Boucadair, M., Penno, R. and Selkirk, P., 2013. Port control protocol (PCP) (No. RFC 6887).