Distributed Public Key Infrastructure and PSK Exchange Based on Blockchain Technology

Elie Kfoury
Department of Computer Science
American University of Science and Technology
Beirut, Lebanon
Email: ekfoury@aust.edu.lb

David Khoury
Department of Computer Science
American University of Science and Technology
Beirut, Lebanon
Email: dkhoury@aust.edu.lb

Abstract—**Public key distribution and device authentication remain the main security challenges in many systems and applications. Existing solutions are based on Public Key Infrastructures (PKI) backed by Certificate Authorities (CA) to validate the authenticity of the devices. However, distributing and provisioning certificates for each client showed to be impractical especially for Internet of Things (IoT) devices. In this paper we propose a distributed PKI (Public Key Infrastructure) platform based on the Ethereum Blockchain. It contains a decentralized key-store that holds the public keys of all devices, and includes a generic protocol for PSK (Pre-Shared Keys) distribution. PSK keys can then be used by PSK-based security protocols (TLS-PSK, DTLS-PSK, SRTP ...) for securing the communication channel between two devices. This platform includes a client-side module, a public key management module configured on the server, and a smart contract software deployed on the Ethereum Blockchain network. This generic platform can be used by many applications for client and server authentication, data integrity, and secure peer to peer communications. Moreover, this promising system may potentially eliminate the trust requirement imposed by the existing PKI/CAs infrastructure on clients.**

1. Introduction

Security and privacy are critical issues facing the development of various technologies and applications. The Internet of Things (IoT) for instance, is still not widely developed due to the lack of proper and standard security measures [1]. Most existing security solutions are based on Public Key Infrastructures (PKI) and Certificate Authorities (CAs). These solutions require a large infrastructure to manage and provision certificates. Moreover, each device must possess a client certificate to authenticate itself. Distribution of certificates for many devices is impractical, expensive, and nearly impossible especially in IoT networks [2]. Furthermore, the current PKI/CA security system holds an inherent risk in the possibility of subverting a CA or having a rogue CA. DigiNotar, for example, a formerly reputable CA which serviced all major browsers as a root CA was successfully hacked by an intruder on June 2011. This resulted in issuing over 500 rogue certificates for major domains such as microsoft.com, cia.gov among others [3]. In another incident, VeriSign got subverted and issued two certificates to a user claiming to be Microsoft [4]. Potential security solutions like hardcoding the credentials on the devices directly, are not the answer either, because the impose the threat of reverse engineering.

Luckily, Blockchain technology provides a decentralized network and immutable database since it leverages consensus algorithms and protocols based on cryptographic functions. It also ensures that there would be no single point of failure. Ethereum, specifically, is an open source Blockchain platform with smart contract scripting functionality that enables the development of decentralized systems on a high level to benefit from the advantages of the Blockchain technology [5]. Several applications can be implemented on Blockchain such as secure payment, asset management, smart property, health-care and others [6].

In this paper we propose a generic platform called DPK (Distributed Public Key-store) to store and distribute the DPK devices public key by using the Ethereum Blockchain technology. This system introduces a novel generic handshake protocol for the secure distribution of the pre-shared keys used in the PSK based security protocols like Transport Layer Security-Pre Shared Keys (TLS-PSK) [7], Datagram TLS (DTLS)-PSK [8], Secure Realtime Transport Protocol (SRTP) [9], etc... This out of band generic protocol is responsible for the distribution of the PSK between two DPK device entities.

The main contributions of this platform include (1) Authenticating clients without the need of a PKI/CA, (2) Ensuring the PSK distribution between communication parties, (3) Protecting against public key alteration due to the immutable property of Blockchains, (4) Enabling security between devices from different organizations, (5) Providing a unique identity for each device to facilitate addressing and management. The rest of this paper is organized as follows: Section II overviews the theoretical background of Blockchain technology. Section III introduces the proposed method and its advantages compared to the existing architectures. Implementation and results are demonstrated in section IV. Section V provides some concluding remarks and the intended future work.

2. Background

A Blockchain is a distributed database where copies are stored on multiple nodes simultaneously. There is no single controlling computer in charge of maintaining the database, or what is referred to as the ledger. The validity of a particular version of a Blockchain is established by consensus amongst several nodes that form a central one. [10] Blockchain is more than just a decentralized digital ledger; It may also contain data and transaction records. The use of the Blockchain technology deals with confirming the integrity of data associated with the transaction. This feature is key for securing the integrity of networked devices. [11]

2.1. Ethereum Blockchain and smart contracts

As mentioned earlier, Ethereum Blockchain architecture includes storing programs and their state as smart contracts. A smart contract has a function-based interface and can be deployed by any Ethereum user. Once deployed the smart contract can be referenced by its address, which is a cryptographic identifier. 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. Calling a contract function will invoke the miners to execute the program with no trust to update the program state. A smart contract can accept and transfer Ether, which is the official token used in Ethereum, and can call functions of other deployed contracts. [12]

2.2. LES (The Light Ethereum Subprotocol)

Blockchain technology by design consumes a considerable amount of memory to store the historical transactions. Obviously, it is not applicable to use a normal Full node on some devices (IoT, mobiles, etc ...) due to the constraints imposed by the device. Fortunately, an alternative to the Ethereum full node is the Light node [13]. The size of the lightchain data by the time of writing (March 2018) is 0.005GB, which is affordable on a mobile device. It is worth noting that this lightchain data will be downloaded once only on the mobile for all Ethereum decentralized apps. The essential element of the light client is the Merkle Tree [14] data structure which is used for efficient and secure verification of the 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.

2.3. Public or Private Ethereum Blockchain

Based on the application's requirements, the Ethereum network could be deployed as private or public. The public Blockchain allows any user with an Ethereum wallet to query data, send transactions to other addresses, and explore the network using public explorers (Etherscan [15] for example). This helps ensure that the transactions are included and mined by the public nodes after the consensus. The main advantage of this type of Blockchains is the openness of the contract code, where some functions cannot be executed even by the developers of the contract. This protects the application's users from its developers.

3. Proposed System

In this section we introduce the proposed system, explain its architecture, and show the different components and their interactions. Fig. 1 illustrates the high level system architecture of the system.

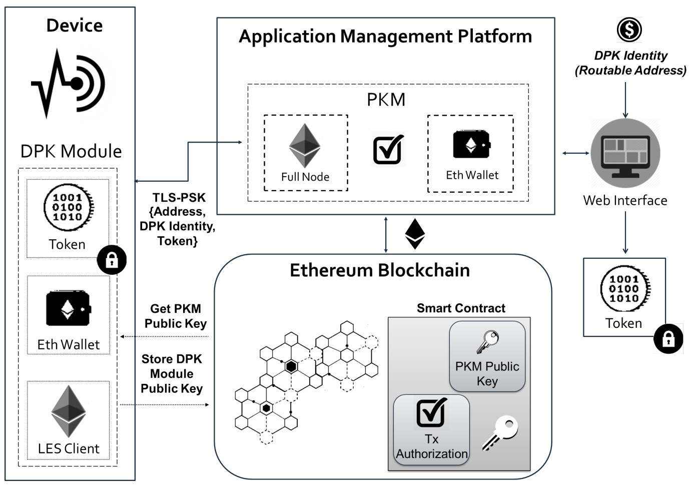


Figure 1. System Architecture

3.1. DPK (Distributed Public Key-Store) System Architecture and Components

The Software components of the DPK platform consist of the following:

1) Public Key Manager (PKM): Network function in the DPK platform located on the server side that authenticates the DPK user and approves the Blockchain storage requests.
2) DPK client module: Plug-in module software installed on the device, that creates transparently an Ethereum address, stores securely a generated public key in the Blockchain, and distributes a PSK between the communicating parties.
3) Smart contracts in the Ethereum Blockchain network containing the major functions needed to store and retrieve the keys.

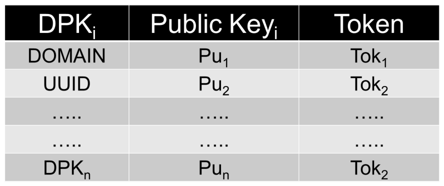
3.1.1. Configuration. The devices accessing the DPK platform should be registered and configured through a web interface (or other means). The user requesting a DPK module should select the type of the DPK identity: Addressable or Non-Addressable. In the addressable type, the user might enter a domain name, an IPv6 address, a public IPv4 or others. The precondition here is to verify that the user requesting the public addressable identity owns it. For example, if the user is requesting a domain name, the Domain Control Validation (DCV) mechanism [16] can be used to prove ownership or control of the claimed domain name. In case the user select a non-addressable identity, a Universally Unique Identifier (UUID) is generated. This type of identity is usually used on devices without a reachable address (behind NATs) like browsers, IoT and mobile devices. For further authentication of the client devices (based on SIM or E-SIM [17]), the Generic Bootstrapping Architecture (GBA) [18] can be used for authentication if the device owns a valid identity on the mobile operator's HSS (Home Subscriber Server).

The configuration consists of assigning a unique Token to the DPK client module. This Token is crucial to identify and authenticate the DPK 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 Wallet management functions in PKM transfers Ether to the DPK client after confirmation of the identity and the transaction fees payment of the device through the associated Token.

3.1.2. Public Key Manager (PKM). This function could be part of a dedicated network or a global platform for any device configuration and authentication. The download or the installation of the user's DPK 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 PKM is to authenticate the DPK client module in the device and send the necessary Ether to the Client to be able to store the user public key in Ethereum. The PKM performs the wallet management functionality by sending Ether to the client and approving the storage transaction. The PKM module could contain a full or a light client Ethereum node to interface the Blockchain.

3.1.3. DPK Client Module. This module can be installed on a variety of clients (mobile application, IoT device, web browser, server ...). When the DPK module is installed, an empty Ethereum wallet and public/private keypair are generated. This module could be downloaded or part of the firmware of a device. There are two types of DPK module: 1) Light with LES (Light Ethereum Subprotocol) which is considered a Light Ethereum Node . 2) Full Node Ethereum, usually installed in Servers or high processing and memory devices. The DPK client module includes an Ethereum light client (LES) to interwork with Blockchain. The module contacts PKM to fill in the client wallet with the necessary Ether to be able to perform the storing of the client public key in the Ethereum network. At the configuration, a Token file is deployed to each DPK client module to identify and authenticate the client. The Token is part of the information sent to the PKM wallet management which are the following: *Wallet address, DPK module identity,* $\{Token\}_{PrPKM}$, where *[Token]*$_{PrPKM}$ is the generated Token signed with PKM's private key. The data is sent encrypted to the PKM using TLS-PSK. The encryption process is described later in Section 3.2.

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



DPK_i	Public Key_i	Token
DOMAIN	Pu_1	Tok_1
UUID	Pu_2	Tok_2
.....	.....	.....
.....	.....	.....
DPK_n	Pu_n	Tok_2

Figure 2. Smart Contract Mapping Table

1) addClient(DPK_{ID}, Pu_k, Tok_i)
2) getClient(DPK_{ID})
3) approveClient(Tok)

The approval sequence steps are the following:

1) The client calls the function *addClient* in the smart contract by signing a transaction. This function adds any client to the pending list. The input parameters are:
   a) DPK client Identity: Can be a domain name or a UUID.
   b) Public Key (Ex: RSA 2048)
   c) Token: a string generated randomly and signed by the application service platform
2) The DPK client module sends an approval request to the PKM containing the signed token.
3) The PKM calls the function *approveClient* in the smart contract by signing a transaction. The token is the only parameter used.
4) Only the owner of the contract (the platform provider) can approve the client: As shown in the code listing, the approveClient function has a condition that allows only the contract deployer (PKM) to move the entry from the pending to the client list. getClient(DPK_{ID}) function is used to fetch the public key of the requested identity

3.2. Generic Protocol for PSK Distribution

This generic protocol uses the Needham-Schroeder method [19] to distribute the Pre-Shared Key using asymmetric encryption as shown in Fig. 3.

1. It is assumed that both DPK devices have stored their public keys Pu_A and Pu_B in DPK platform according to the mechanism described above.

2. $Device_A$ requests the public key of the responder $Device_B$ from the blockchain.

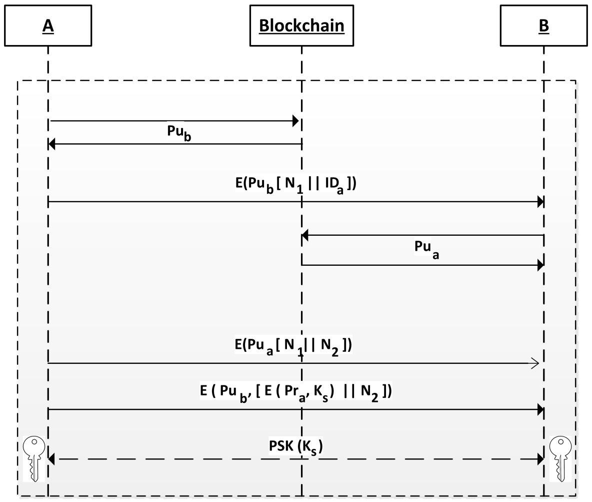


Figure 3. PSK Key Distribution

3. Device$_A$ encrypts with the public key of Device$_B$ Pu$_B$, a randomly generated nonce (*N1*) which is used to identify this transaction uniquely, and the Identifier of Device$_A$.

4. Device$_B$ requests Device$_A$s public key Pu$_A$ from DPK.

5. Device$_B$ sends devices nonce (*N1*) and a new nonce generated by Device$_B$ (*N2*) encrypted with Pu$_A$. This nonce (*N1*) helps Device$_A$ ensure that the correspondent is Device$_B$ as only Device$_B$ can decrypt the message.

6. Device$_A$ returns (*N2*), encrypted using Pu$_B$.

7. 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). Device$_A$ encrypts with Pr$_A$ the generated secret key K_S and then encrypts the whole message along with (*N2*) with Pu$_B$. K_S is sent encrypted with the public key of the device2.

7. Device$_B$ decrypts with Pu$_A$ and Pr$_B$ to retrieve the secret session key K_S.

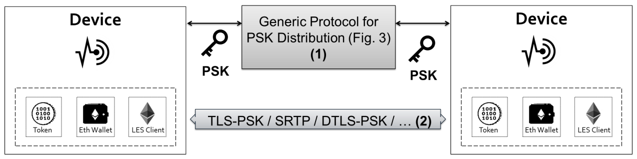

Device
Generic Protocol for PSK Distribution (Fig. 3) (1)
Device
PSK
PSK
Token
Eth Wallet
LES Client
TLS-PSK / SRTP / DTLS-PSK / ... (2)
Token
Eth Wallet
LES Client

Figure 4. Secure Session Establishment

3.3. DPK Module API

Any application running on a DPK-enabled device can access through a well defined Application Programming Interface (API) the destination's PSK or Pu$_K$ as shown in Fig. 5. The same API can be used for remote applications, mainly for constrained IoT devices in capillary networks (Fig. 6).

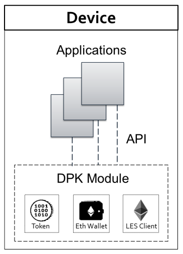


Figure 5. Applications interfacing DPK Module through APIs

The aim in capillary networks is to establish an end to end session key between the end device (which is usually an IoT constrained device) and a server [20]. The constrained devices can be seen as resources of the authenticated Capillary Gateway (CGW). The delegated DPK method requires that the end device requests from the CGW a session key Ks to be used to encrypt the communication between the device and the recipient (Server). The DPK module in the CGW gets the public key of the server Pu$_{K2}$ from the Ethereum Blockchain. An end-to-end session key Ks is generated by the DPK module in CGW. The Ks, and Ks$_{PuK2}$, (the Ks encrypted with public key of the destination server) are sent to the constrained device which in turn transfers Ks$_{PuK2}$ to the server as a Ticket. The session key Ks is shared between the two entities, this key can be used for setting up any type of secure communication such as pre-shared-key based TLS (TLS-PSK) session between the end device and the server.

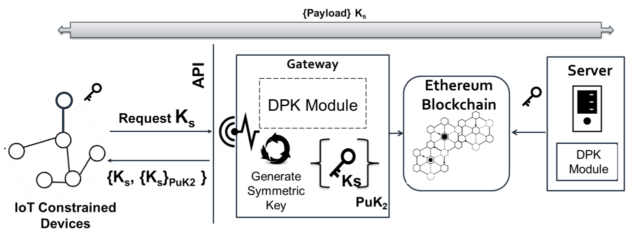


Figure 6. Capillary Network

4. Implementation and Results

To validate the proposed system, we implemented the solution using various technologies. Solidity: Contract-oriented programming language for writing smart contracts [21], NodeJS: Server side scripting (Wallet management server) [22], DPK Module: Java implementation of the DPK module features with Web3j [21] to interface the light client. The Ropsten Testnet [23] is used to simulate the Blockchain network. Storing the public key in the Ethereum Testnet Blockchain took between 11 to 30 seconds to be validated. This time is only spent once upon configuration. Retrieving the public key on the other hand took only 0.5 seconds; this amount of time is spent on each secure session establishment.

5. Conclusion and Future Work

In this paper, we have proposed a distributed public key infrastructure backed by the Blockchain technology to resolve the complexity of certificate distribution and provisioning especially on client side. Our implementation was tested on Ethereum Blockchain Testnet. Results showed that the solution performs well in terms of session setup time. Moreover, this system can replace the PKIs backed by CAs and overcome the trust concerns as the device's data stored in the Blockchain eliminates the transmission of certificates. Regarding the future work, we intend to combine DPK and PKI/CA in order to enhance the trust by double checking the identities of devices (from certificates and Blockchain).

Appendix A. Smart Contract Code Listing

```
pragma solidity ^0.4.2;

contract DPK
{
  uint clientsCount = 0;
  mapping(string => Client) pending;
  mapping(string => Client) clients;
  mapping(uint => string) clientsIndex;
  string pkmKey;
  address public owner;


  function DPK(){
    owner = msg.sender;
  }

  struct Client{
    string identity;
    string publicKey;
    address clientAddress;
  }


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

  function approveClient(string _token) returns (bool success){
     if(msg.sender == owner){
        clients[_token] = Client(pending[_token].identity,

        pending[_token].publicKey, pending[_token].clientAddress);
          clientsIndex[clientsCount] = _token;
          clientsCount++;
          return true;
      }
      return false;
  }


  function getClient(string _identity) constant returns (string publicKey) {
      for( uint i = clientsCount; i >= 0; i--){
        if(sha3(clients[clientsIndex[i]].identity) == sha3(_identity))
          return clients[clientsIndex[i]].publicKey;
        }
    return "";
   }

  function storePKMKey(string puk) returns (bool success){
      if(msg.sender == owner){
          pkmKey = puk;
          return true;
      }
  }

  function getPKMKey() constant returns (string) {
    return pkmKey;
  }

}
```

References

[1] Zhi-Kai Zhang, Michael Cheng Yi Cho, Chia-Wei Wang, Chia-Wei Hsu, Chong-Kuan Chen, and Shiuhpyng Shieh. IoT Security: Ongoing Challenges and Research opportunities. In *Service-Oriented Computing and Applications (SOCA), 2014 IEEE 7th International Conference on*, pages 230–234. IEEE, 2014.

[2] *Bootstrapping Security - The Key to Internet of Things Authentication and Data Integrity*. Ericsson 2016.

[3] Dennis Fisher. Final Report on DigiNotar Hack Shows Total Compromise of CA Servers. *Retrieved September*, 8:2013, 2012.

[4] David Khoury and Elie Kfoury. Generic Hybrid Methods for Secure Connections based on the Integration of GBA and TLS/CA. In *Sensors Networks Smart and Emerging Technologies (SENSET), 2017*, pages 1–4. IEEE, 2017.

[5] Gavin Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger. *Ethereum Project Yellow Paper*, 151:1–32, 2014.

[6] Marc Pilkington. Blockchain Technology: Principles and Applications. *Research Handbook on Digital Transformations*, page 225, 2016.

[7] Pasi Eronen and Hannes Tschofenig. Pre-shared Key Ciphersuites for Transport Layer Security (tls). Technical Report, 2005.

[8] H Tschofenig and T Fossati. Transport Layer Security (TLS)/Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things. Technical Report, 2016.

[9] Mark Baugher, D McGrew, M Naslund, E Carrara, and Karl Norrman. The Secure Real-time Transport Protocol (SRTP). Technical Report, 2004.

[10] Arati Baliga. Understanding Blockchain Consensus Models. Technical Report, 2017.

[11] Guy Zyskind, Oz Nathan, et al. Decentralizing Privacy: Using Blockchain to Protect Personal Data. In *Security and Privacy Workshops (SPW), 2015 IEEE*, pages 180–184. IEEE, 2015.

[12] Vitalik Buterin et al. Ethereum White Paper. *GitHub repository*, 2013.

[13] Light Ethereum Subprotocol (les), 2017.

[14] Ralph C Merkle. Protocols for Public Key Cryptosystems. In *Security and Privacy, 1980 IEEE Symposium on*, pages 122–122. IEEE, 1980.

[15] Etherscan Team. Etherscan: The Ethereum Block Explorer, 2017.

[16] DCV (Domain Control Validation) Methods — CertCentral.

[17] Chul Hyun Park, Kwan Lae Kim, Joo Young Kim, Jin Hyoung Lee, and Hyung Jin Lee. Method for Providing SIM profile in EUICC Environment and Devices, 2015. US Patent 9,204,300.

[18] Muhammad Sher and Thomas Magedanz. Secure Access to IP Multimedia Services using Generic Bootstrapping Architecture (GBA) for 3G & Beyond Mobile Networks. In *Proceedings of the 2nd ACM International Workshop on Quality of Service & Security for Wireless and Mobile Networks*, pages 17–24. ACM, 2006.

[19] Roger M Needham and Michael D Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, 1978.

[20] JOACHIM Sachs, NICKLAS Beijar, P Elmdahl, J Melen, FRANCESCO Militano, and PATRIK Salmela. Capillary Networks–Smart Way to get Things Connected. *Ericsson Review*, 8:1–8, 2014.

[21] Chris Dannen. Dapp deployment. In *Introducing Ethereum and Solidity*, pages 149–157. Springer, 2017.

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

[23] Simon Kim. *Measuring Ethereum's Peer-to-Peer Network*. PhD thesis, 2017.