Decentralized Public Key Infrastructure for Internet-of-Things

Jongho Won*, Ankush Singla*, Elisa Bertino* and Greg Bollella†
*Department of Computer Science, Purdue University, West Lafayette, IN, USA
†VMWare, Palo Alto, CA, USA
Email: *{won12, asingla, bertino}@purdue.edu, †gbollella@vmware.com

***Abstract*—In many envisioned IoT applications, security is crucial. However, designing and/or deploying existing security techniques in IoT systems is not straightforward due to the inherent heterogeneity of IoT devices as well as their huge number. A critical security building block is represented by the Public Key Infrastructure (PKI) relying on Certificate Authorities (CAs). However, even a single-point-of-failure in a PKI may affect entire IoT systems due to its centralized nature. Failures have far reaching effects as the number of IoT devices increases. Furthermore, it is difficult for the owners of IoT devices to manage the certificates for their IoT devices since there are no standard protocols for retrieving, installing and updating the certificates. As a result, IoT device manufacturers often install certificates on the devices on behalf of the owners of the devices, which introduces the risk that the private keys of the devices are leaked by the manufacturers. In this paper, we propose a decentralized PKI for IoT, called IoT-PKI, which utilizes distributed nodes in a blockchain network instead of CAs, and thus addresses scalability. IoT-PKI protects against key leakages at device manufacturers since it allows the owners of IoT devices to manage the certificates of their IoT devices. Finally, we show the feasibility and efficiency of IoT-PKI through our prototype implementation and experiments.**

I. Introduction

The rise of low-cost general purpose computers and the availability of cheap sensors, actuators and radio transceivers is promoting the world of interconnected physical objects, Internet-of-Things (IoT). Such physical objects, like sensors in infrastructure, cars, smart meters, appliances, and health care devices, will soon collect and exchange huge volumes of data. Also, advances in cloud/edge computing enable the vast amounts of data generated by those physical objects (referred to as IoT devices in what follows) to be effectively stored and analyzed. As a result, IoT is making homes, factories and cities more efficient and intelligent. Indeed, IoT-related industries are growing fast. IBM [1] forecasts that the number of connected devices will exceed 25 billion in 2020 and 100 billion in 2050. Not only in the civilian sector, but also in the military sector, IoT is being realized [2]. In future battlefields, human warfighters will collaborate with various things performing sensing and acting, such as sensors, robots, and wearable devices, by communicating with each other. In the case of megacitys military operations, the number of things per square kilometer is considered to be several million when making use of existing local civilian IoT devices.

However, the proliferation of IoT implies that IoT may quickly become a breeding ground for cyber attacks. Privacy-sensitive information, such as personal health data and electricity usage patterns, will be exchanged in IoT applications and an adversary may want to steal such data for financial gains. In addition, since many IoT devices with actuators control physical objects, unauthorized and unauthenticated controls of such devices would cause serious accidents and disasters [3].

Therefore, security and privacy are critical for IoT. In particular, it is critical that security functions, such as authentication, confidentiality, integrity and non-repudiation, are supported. Privacy-sensitive messages exchanged in IoT applications must be encrypted and must not be altered in an unauthorized manner. All entities in an IoT system must be able to verify the identities of other parties. For some applications, accountability is also a crucial requirement, and thus IoT devices must not able to deny having sent messages. In a military operation, it is also important quickly to identify compromised devices and isolate them. However, due to the massive scale of IoT systems, the limited computing/network resources of IoT devices and inherent heterogeneity, designing a security framework for IoT applications is not straightforward.

The current Public Key Infrastructure (PKI) relying on Certificate Authorities (CAs), referred to as C-PKI in what follows, provides such critical security functions when IoT devices have CA-signed certificates. X.509 certificates for IoT device authentication have attracted industry interest [4]–[6].

However, the use of the certificates issued by CAs is not well-suited for IoT devices due to the following five reasons. First, it is difficult for the owners of IoT devices to manage the certificates for their IoT devices since there are no standard protocols for retrieving, installing and updating the certificates on these devices. Second, because of the difficulty of installing certificates, IoT device manufacturers typically generate secret keys for IoT devices and install the keys onto the devices on behalf of the actual owners of the IoT devices. Thus, the manufacturers can leak the secret keys or be the targets of adversaries who want the secret keys. Third, the centralized nature of C-PKI can cause a single-point-of-failure. Once even a single root CA private key is stolen by an attacker, all entities must stop authenticating others and remove the corresponding certificate of the root CA from their Trusted Root Certification Authority lists before restarting. Also, all the certificates signed by the root CA private key are no longer usable since the attacker may have created some of them. Fourth, the root certificate lists in IoT devices must be updated periodically

or on demand. Otherwise, the IoT devices may authenticate bogus devices with certificates signed by a compromised root private key. Without an automatic root certificte update mechanism, it is highly likely that stale root certificates remain in the IoT devices. Finally, Online Certificate Status Protocol (OCSP) responders are inherently vulnerable to distributed denial-of-service attacks since OCSP responders can be easily overloaded due to generating OCSP responses requiring a large computational overhead for signature generations. It is thus clear that a decentralized scalable PKI tailored to IoT systems is needed that addresses the shortcomings of C-PKI.

One possible PKI for IoT can be designed based on the *blockchain* technique initially introduced in the context of the Bitcoin cryptocurrency [7]. Blockchain is a distributed ledger that maintains a continuously-growing list of ordered records called blocks. Each block contains a link to the previous block, a proof-of-work and all the transaction records collected at a time period. A proof-of-work is a piece of data which is difficult to produce but easy to verify. Every block contains a hash of the previous block. Thus, it is hard for the data in a block to be altered retrospectively.

Researchers have already proposed blockchain-based PKIs using *Name-Value Storage (NVS)* [8]–[10] which stores *name/value* pairs in its blockchain, where *name* is a search key for a stored record and *value* is the record. However, none of them can be directly utilized to address the IoT authentication problem since they do not take into account the limited resources and connectivity of IoT devices.

In this paper, we propose a distributed and secure public key infrastructure for IoT, referred to as IoT-PKI, which is assisted by distributed edge devices connected through a blockchain network. We show how it can effectively address all the shortcomings of the conventional C-PKI. In IoT-PKI, distributed blockchain nodes take the role of CAs. In C-PKI, a digital signature algorithm binds the identity of an IoT device to its public key using a root private key, while in IoT-PKI, proof-of-works of the blockchain bind the identity of an IoT device to its public key. IoT-PKI has relevant properties as follows: 1) Since IoT-PKI operates in a distributed manner, it does not suffer from single-point-of-failures. 2) Since each IoT device generates its private/public key pair by itself, there is no risk that the private key is leaked by others. 3) The actual owner of the device can register a record consisting of the device's identity (name), the device public key, the status and the expiration time of the record into the blockchain NVS using a device management tool which can be the owner's smartphone or laptop. Also, the owner can update the public key and revoke the device record in NVS if necessary. 4) Our experiments using the prototype show that, in IoT-PKI, the status of a certificate can be verified approximately up to 25.7 times faster than when a certificate is verified through Online Certificate Status Protocol (OCSP) in C-PKI.

II. Background

A blockchain with Name/Value Storage (NVS) is an extension of Bitcoin. NVS is a distributed database storing

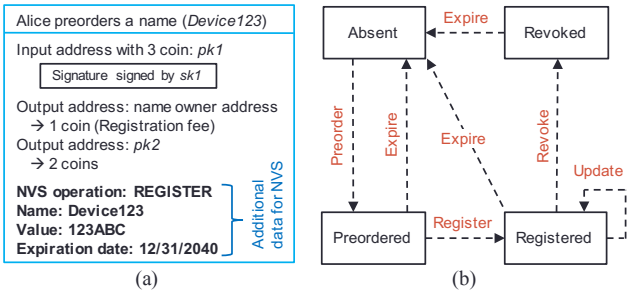

Fig. 1. (a) Transaction with NVS data (b) Name status transition

name/value pairs with status and expiration date, where names are unique and values are arbitrary data. Some blockchains, such as Namecoin [8] and Emercoin [9], provide NVS. Ali et al. [10] have introduced Blockstack [10], which is an upper layer system that can be added on top of existing blockchains like Bitcoin to provide NVS. In such blockchain networks, every node has a copy of NVS as well as all block copies. Blockchains with NVS require a consensus that ensures the uniqueness of names.

As shown in Fig. 1 (a), Alice can register a name/value pair with an expiration date by creating a transaction which contains additional data related to NVS. Only when the name has not been registered yet, the name/value pair can be registered in NVS. Registering a new name requires a name registration fee, which discourages people from registering a lot of names that they do not actually intend to use. Name registration fees vary by blockchain, but typically become higher as the validity period of the name increases [9]. Once a name is registered, the registration fee is destroyed, i.e., becomes permanently unusable. As shown in Fig. 1 (b), a name registration is executed by a two-step commit mechanism [8]. Like the double-spending problem [7], only one user can successfully register a given name.

If the user succeeds in the pre-order and the registration, he/she has the ownership of the name. Then, the user can update the value, revoke the name and transfer the ownership of the name to another user. Like the possession of coins, the ownership of a name means control over the private key (sk_{owner}) of the address (pk_{owner}) used to register the name. To transfer the name, the owner generates a transaction with a valid signature using sk_{owner} and writes the address of a new owner in the output address field of the transaction.

The registered name expires after the expiration date ex. Once the name is revoked, the fields associated with the name, such as value, status and ex, cannot be updated. Since the name resides in the NVS until ex, nobody can abuse the name by registering the same name until ex.

III. Decentralized PKI for IoT

A. Assumption

Although Bitcoin has demonstrated its feasibility and resilience against a variety of different attacks and malicious actors, there still are some issues that need to be addressed. For instance, blockchains with insufficient miners are vulnerable to 51% attacks [10]. Also, the proof-of-work mechanism used in

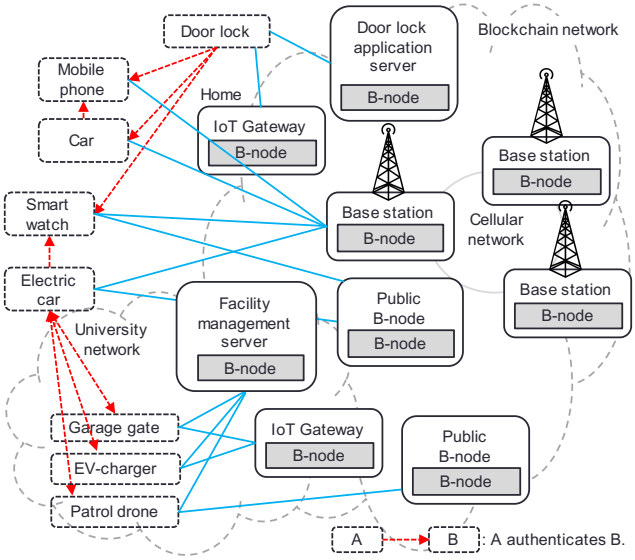

Fig. 2. IoT-PKI architecture. Dotted line rectangles indicate NVS-clients. A blue solid line indicates a trust relationship link between an NVS-client and an NVS-server, i.e., B-node.

Bitcoin inherently requires high computing power and energy[1]. In addition, the network capacity limits the block size, i.e., the maximum number of transactions in a block. However, such issues are out of the scope in this paper.

Although the 51% attack is theoretically shown as an attack against blockchain, we assume that all the underlying cryptographic algorithms and the consensus mechanism work correctly due to two reasons. First, IoT-PKI can be built on a mature blockchain resistant to 51% attacks like Blockstack [10]. Second, many defense/mitigation solutions, such as Proof-of-Stake [11], DDoS counter-attacking and increasing the number of confirmations, can be utilized.

B. IoT-PKI Architecture

Fig. 2 shows our IoT-PKI architecture, in which blockchain nodes, called *B-node*s, are deployed in the Internet. They are full nodes or lightweight nodes[2] which generate/validate transactions. They form an NVS-enabled blockchain network, such as Namecoin [8] or Emercoin [9]. Any entity can be a B-node if its network connection to the Internet is persistent and it has enough storage to save blocks and all the name-value pairs. B-nodes can be located in edge devices[3], data centers, application servers and PCs. Manufacturers, internet service providers (ISPs), telecommunications companies, universities, governments and even individual users can operate B-nodes.

B-nodes provide authentication services for IoT devices. An IoT device or an IoT application program (referred to as an NVS-client) sends an NVS-query with a name to a B-node. Then, the B-node (referred to as an NVS-server) sends a response with all the values associated with the name.

We assume that each NVS client has $\mathcal{N}$ (≥ 1) trusted B-nodes. An IoT device can trust B-nodes operated by the same owner (or administrator). For instance, in home IoT applications, a home IoT gateway (B-node) can be trusted by home devices since they are inside the same local network managed by the home owner. B-nodes operated by an organization to which a user belongs can be trusted. In addition, B-nodes operated by ISPs, IoT platform companies, manufacturers or governments can be trusted depending on the user's choice.

C. Device Setup

In IoT-PKI, the owner of an IoT device controls the certificate of the IoT device. The device's name (i.e., device ID) and value (i.e., the hash value of a self-signed certificate) are registered in NVS using a device manager (DM). DM is a B-node operated by the owner. For example, a laptop can be a DM if the laptop and the IoT device can directly communicate through a network interface, e.g., Ethernet, Wi-Fi, Bluetooth or ZigBee. The device setup procedure is as follows:

- **Step 1:** The owner prepares coins (i.e., name registration fee) for a name-value registration. We assume that the coins are received through an output address pk_o. The private key corresponding to pk_o is sk_o that proves the ownership of the coins.
- **Step 2:** DM generates a private/public key pair sk_{DM}/pk_{DM}. The owner of DM copies sk_{DM} to another secure place just in case DM is lost or broken.
- **Step 3:** DM sends a message M_{init} consisting of C_{init}, pk_{DM}, the list of trusted B-nodes L, the expiration date EX_A and σ to D_A. Here, C_{init} is the initialization command. σ is the signature of $\{C_{init}||pk_{DM}||L||EX_A\}$ signed by sk_{DM} and $||$ is the string concatenation function.
- **Step 4:** When D_A is first unboxed, it is in an initial state and waits for an initialization command from a device manager. After receiving M_{init}, D_A verifies σ using pk_{DM}. If the verification succeeds, D_A generates
  - its identity ID_A,
  - a private/public key pair sk_A/pk_A and
  - a self-signed X.509 certificate $Cert_A$.

  Although ID_A can be any string, we recommend using a Universally Unique Identifier (UUID) in order to reduce the possibility of device ID duplication in NVS. Notice that the private key sk_A is created by the device itself and no one knows sk_A even the owner. D_A creates an X.509 certificate ($Cert_A$) instead of just using its public key (pk_A) for interoperability with C-PKI. In our implementation work, we utilized a well-established open-source SSL/TLS library, i.e., OpenSSL [12], so that an IoT device can establish SSL/TLS channels with other parties by authenticating them via C-PKI.
- **Step 5:** D_A sends C and its signature signed by sk_A to DM, where $C = Enc(pk_{DM}, Cert_A)$. $Enc(pk, m)$ is

[1]Some other cryptocurrencies, such as Peercoin and Emercoin, adopt Proof-of-Stake (PoS) [11]. Their consensus mechanism depends on the wealth (i.e. the stake) of participants rather than miners' computing power. PoS also mitigates the 51% attack vulnerability.

[2]Full nodes fully enforce all the rules of a crypto-currency storing a full copy of the blockchain, while lightweight nodes do not store the entire chain by deleting unnecessary data about transactions that are fully spent.

[3]Edge device is an entry point into core networks. Edge devices include various types of access points and gateways.

a public key encryption scheme, such as RSA or Diffie-Hellman encryption, that encrypts a message m using a receiver's public key pk. $Cert_A$ is encrypted since it contains ID_A. If $Cert_A$ is not encrypted, an eavesdropping attacker may be able to register ID_A in the NVS before DM registers ID_A and cause the next step to fail.

- **Step 6:** After receiving C, DM decrypts C, obtains $Cert_A$ and verifies the signature. If the signature is valid, DM computes $V_A = H(Cert_A)$ and generates a transaction to register the name and its associated values as follows:
  - NVS operation: $REGISTER$
  - Name: ID_A
  - Value: $V_A (= H(Cert_A)) \rightarrow H()$ is a hash function like SHA-256.
  - Expiration date: EX_A
  - Input address: pk_o and a signature signed by $sk_o \rightarrow$ unlock the coins.
  - Output address: pk_{DM}
- **Step 7:** The name registration can fail if ID_A is already registered in the NVS. If it fails, DM starts from Step 3. If the name is successfully registered in the NVS, i.e, the transaction is confirmed by the blockchain consensus mechanism, DM sends C_{reg} with its signature σ' signed by sk_{DM} to D_A.
- **Step 8:** After receiving C_{reg} and σ', D_A verifies σ'. If σ' is not correct, D_A aborts the procedure and goes into the initial state. If σ' is correct, D_A goes into a working state. D_A stores L, EX_A and pk_{DM} in non-volatile storage like flash memory and uses pk_{DM} as the public key of a master device manager. If D_A is in the working state, D_A does not accept any command from other device managers.

Notice that the device setup procedure allows an IoT device to have a globally unique ID without the help of a CA or its manufacturer and does not require a user interface for configuration, like a web interface, which may introduce security vulnerabilities.

D. Device and Device Name Ownership Transfer

If the ownership of D_A is transferred to a new user U_{new}, the control over the name must also be transferred. We assume that the original owner U_{old} of D_A has sk_{DM} that proves that U_{old} has control the over the name ID_A. U_{old} can transfer the ownership of ID_A to U_{new} as follows:

- **Step 1:** U_{old} prepares coins for a name transfer. We assume that the coins are in $\hat{pk}_o$ and the private key corresponding to $\hat{pk}_o$ is $\hat{sk}_o$.
- **Step 2:** U_{new} (or DM of U_{new}) generates a private/public key pair $\hat{sk_{DM}}/\hat{pk_{DM}}$. U_{new} copies $\hat{sk_{DM}}$ to another secure place.
- **Step 3:** U_{old} (or DM of U_{old}) generates a transaction to transfer the name as follows:
  - NVS operation: $UPDATE$
  - Name: ID_A
  - Input address: $\hat{pk}_o$ and a signature signed by $\hat{sk}_o$
  - Input address: pk_{DM} and a signature signed by sk_{DM}
  - Output address: $\hat{pk_{DM}}$
- **Step 4:** U_{old} sends $\hat{pk_{DM}}$ and a signature of $\hat{pk_{DM}}$ signed by sk_{DM} to D_A. D_A switches its master device public key from pk_{DM} to $\hat{pk_{DM}}$ after verifying the signature.

The signature signed by sk_{DM} proves that U_{old} has owned D_A and ID_A, and the right to transfer the ownership to another. After this transaction is confirmed, U_{new} with $\hat{sk_{DM}}$ has the ownership of ID_A. If DM's private key is stolen by an attacker, the ownership of ID_A in the NVS is transferred to the attacker. To address this problem, the owner can store the private key of the master device manager in offline storage.

E. Device Key Update

sk_A can be leaked by cyber/physical attacks. The owner of D_A can update the sk_A/pk_A as follows:

- **Step 1:** The owner prepares coins for a key update. We assume that the coins are in pk'_o and the private key corresponding to pk'_o is sk'_o.
- **Step 2:** DM generates a private/public key pair sk'_{DM}/pk'_{DM}. The owner of DM copies sk'_{DM} to another secure place.
- **Step 3:** DM sends a message M_{update} consisting of C_{update}, a nonce η and σ to D_A. Here, C_{update} is a command for update. η is an increasing number used to prevent replay attacks. σ is the signature of $\{C_{update}||\eta\}$ signed by sk_{DM}.
- **Step 4:** After receiving M_{update}, D_A verifies if σ is correct and η is greater than the nonce in the last update. If the verification fails, D_A aborts the procedure. Otherwise, D_A generates a new private/public key pair sk'_A/pk'_A and a self-signed X.509 certificate $Cert'_A$. D_A updates the public key of the master DM from pk_{DM} to pk'_{DM}.
- **Step 5:** D_A sends $C = Enc(pk_{DM}, Cert'_A)$ to DM.
- **Step 6:** After receiving C, DM decrypts C and obtains $Cert'_A$. Then DM computes $V_A = H(Cert'_A)$ generates a transaction to update the value as follows:
  - NVS operation: $UPDATE$
  - Name: ID_A
  - Value: V_A
  - Input address: pk'_o and a signature signed by sk_o'
  - Input address: pk_{DM} and a signature signed by sk_{DM}
  - Output address: pk'_{DM}

The signature signed by sk_{DM} proves that the owner had control over the name.

F. Device Configuration

The owner of D_A can update D_A's configuration, such as the list of trusted B-nodes, the encryption algorithm and the signature algorithm in the certificate using the master DM of D_A.

- **Step 1:** DM sends a message M_{conf} consisting of C_{conf}, a set of configuration changes S_{conf}, a nonce η and σ to D_A. Here, C_{conf} is a command for configuration and η is

an increasing number used to prevent replay attacks. σ is the signature of $\{C_{conf}||S_{conf}||\eta\}$ signed by sk_{DM}.

- **Step 2:** After receiving M_{conf}, D_A verifies if σ is correct and η is greater than the nonce in the last one. If the verification fails, D_A aborts the procedure. Otherwise, D_A changes its configuration according to S_{conf}.

Notice that, optionally, DM can send $Enc(pk_A, M_{conf})$ instead of M_{conf} to D_A in Step 1 if S_{conf} contains privacy-sensitive information. In Step 2, D_A can decrypt $Enc(pk_A, M_{conf})$ using sk_A.

G. Device Name Revocation

The owner of D_A must revoke the name if D_A is stolen/lost or the owner wants to transfer the ownership of D_A before its name expires. Otherwise, for instance, an unauthorized user with a stolen smart watch can unlock the door of the owner's house. To prevent unauthorized use of D_A, the owner of D_A can revoke the name ID_A as follows:

- **Step 1:** The owner prepares coins for a name revocation. We assume that the coins are in $\hat{pk}_o$ and the private key corresponding to $\hat{pk}_o$ is $\hat{sk}_o$.
- **Step 2:** The owner (or DM of the owner) generates a transaction to revoke the name as follows:
  - NVS operation: $REVOKE$
  - Name: ID_A
  - Input address: $\hat{pk}_o$ and a signature signed by $\hat{sk}_o$
  - Input address: pk_{DM} and a signature signed by sk_{DM}

After the revocation transaction is confirmed by the blockchain network, the status of the name becomes *revoked* and no other party can own the name until its expiration date.

IV. Authenticated Key Exchange

In this section, we present an IoT-PKI-based Authenticated Key Exchange (IoTAKE) protocol. Unlike the TLS handshake protocol that uses CA-signed certificates, the IoTAKE protocol utilizes B-nodes for authentication.

A. System Set-up

For the sake of simplicity, we consider an IoT system consisting of two IoT devices, i.e., D_A and D_B, owned by different users. In this system, the two IoT devices D_A and D_B must perform a one-time procedure, i.e., a set-up procedure, before they communicate with each other. The system set-up procedure consists of the following steps.

- **Step 1:** The owner of D_A registers the name of D_A, i.e., ID_A, in the NVS and inserts the list of trusted B-nodes into D_A using the procedure in Sec. III-C. Also, the owner of D_B registers the name of D_B, i.e., ID_B, in NVS and inserts the list of trusted B-nodes into D_B.
- **Step 2:** The owner of D_A sends its name (ID_A) and $Cert_A$ to D_B in order to register them in the membership list of D_B. After receiving ID_A and $Cert_A$, D_B sends a query for ID_A to one of its B-nodes and waits for responses.

  If D_B receives an NVS response consisting of the value V_A, the expiration date EX_A, and the status S_A, then D_B verifies whether S_A is *registered*, EX_A is greater than the current time and $H(Cert_A)$ is equal to V_A. If the verifications fail, D_B aborts the procedure and outputs the "*verification failure*" error. Otherwise, D_B inserts (ID_A, $Cert_A$, ex_A) into its membership list. ex_A is a membership expiration time for D_A. If it expires, D_B removes ID_A from its membership list.

  The owner of D_B also sends ID_B and $Cert_B$ to D_A. After D_A receives them, D_A sends a query for ID_B to one of its B-nodes. After receiving a response, D_A performs the same verification procedure as D_B. If the verification is successful, D_A inserts $(ID_B, Cert_B, ex_B)$ into its membership list.

ID_A (or ID_B) in NVS can expire before ex_A (or ex_B). In such a case, the owner of ID_A (or ID_B) is required to create a new ID for D_A (or D_B), register it in NVS with a new expiration date and perform the step 2 again.

B. Authenticated Key Exchange Protocol

Whenever D_A and D_B want to establish a secure channel, they perform the following protocol:

- **Step 1:** D_A sends ID_A and $Cert_A$ to D_B. D_B sends ID_B and $Cert_B$ to D_A.
- **Step 2:** D_A checks whether ID_B is in its membership list. If not, it aborts the procedure and outputs the "*not in membership list*" error. D_B checks whether ID_A is in its membership list. If not, it aborts the procedure and outputs the "*not in membership list*" error.
- **Step 3-1:**
  - D_A sends a query for ID_B to n $(1 \leq n \leq \mathcal{N})$ B-nodes and receives query responses with V_{Bi}, EX_{Bi} and S_{Bi} $(1 \leq i \leq n)$, For all V_{Bi}, EX_{Bi} and S_{Bi}, D_A verifies whether S_{Bi} is *registered*, EX_{Bi} is greater than the current time and V_{Bi} is equal to $H(Cert_B)$.
  - If the verifications fail, D_A aborts the procedure and outputs the "*verification failure*" error.
  - If the owner of D_B performs the Device Key Update in Sec. III-E, $Cert_B$ is not equal to the ID_B's certificate stored in the D_A's membership list. Then, D_A changes the ID_B's certificate in the membership list into $Cert_B$.

  D_B performs the same procedure for D_A.
- **Step 3-2:** D_A extracts D_B's public key from $Cert_B$ and D_B extracts D_A's public key from $Cert_A$. They perform the same protocol as the TLS handshake protocol in order to share a master secret key msk. If their private keys are valid, they can successfully share msk. Otherwise, they output the "*msk generation failure*" error and abort the procedure.
- **Step 4:** If Step 3-1 and Step 3-2 are completed without an error, D_A and D_B are mutually authenticated and use msk to derive session keys for subsequent encryption and decryption of messages exchanged between them. Notice that Step 3-1 and Step 3-2 can be performed concurrently to reduce the time required for authentication.

V. Security Analysis

We now discuss two possible attacks on IoT-PKI and how IoT-PKI withstands these attacks or mitigates them. Due to the page limit, the full security analysis and comparison with C-PKI will be presented in the final version of the paper.

1) Device private key compromise: Even if the private key sk_A^{old} of an IoT device D_A is stolen by an attacker, the malicious use of sk_A^{old} is not allowed after the owner of the device makes D_A generate a new private/public key pair (sk_A^{new}/pk_A^{new}) of the device and update the value, i.e, $H(Cert_A^{new})$, in the NVS using the procedure in Sec. III-E. When the attacker wants to be authenticated by another device D_B, he/she needs to send $Cert_A^{new}$ or $Cert_A^{old}$ with ID_A to D_B as described in Sec. IV-B. In either case, the attacker cannot be authenticated by D_B. If the attacker sends $Cert_A^{new}$, D_B will output the "*msk generation failure*" error in Step 3-2 since the attacker does not have sk_A^{new}. If the attacker sends $Cert_A^{old}$, D_B will output the "*verification failure*" error in Step 3-1 since the value in the NVS (i.e., $H(Cert_A^{new})$) is not equal to $H(Cert_A^{old})$.

2) Stolen IoT Device: Even if D_A is stolen by an attacker, the malicious use of the device is not allowed after the owner of the device revokes the device's name in the NVS as described in Sec. III-G. In the authenticated key exchange protocol in Sec. IV-B, the other party will output "*verification failure*" error in Step 3-1 since the status of D_A's name in the NVS is *revoked*.

3) Weak random number generator: Embedded devices are typically cheap and without security hardware, such as a Trusted Platform Module [13] and a Physical Unclonable Function [14]. Furthermore, they often cannot gather enough entropy due to short boot times. Therefore, their random number generators are not cryptographically secure. Given the public key of an IoT device, an attacker might be able to find the private key. The lack of entropy can be mitigated by the device configuration procedure in Sec. III-F. The master device manager of an IoT device D_A can include a random number $\mathcal{R}$ in S_{conf} and send $Enc(pk_A, M_{conf})$ to D_A. Then, D_A can use $\mathcal{R}$ as a seed for random number generations.

4) No Available B-nodes: Even if an IoT device has multiple trusted B-nodes, it may happen that the IoT device cannot receive NVS responses from all the B-nodes. This situation is similar to the situation where web browsers cannot connect to OCSP responders in C-PKI and must choose between *soft-fail* and *hard-fail*. For example, consider a university facility management scenario in Fig. 2; if a garage gate adopts hard-fail, it is always closed. However, the gate may adopt soft-fail. When a car D_A approaches to the gate, it sends ID_A and $Cert_A$ to the gate. If ID_A is in the gate's membership list and $Cert_A$ is equal to the certificate stored in the gate's membership list, the gate may open after performing the key exchange protocol. However, this soft-fail mechanism allows an attacker who steals the car to enter the gate even if the owner of the car revokes the car's name in NVS. To mitigate this problem, the gate may be able to proactively update its membership list by requesting NVS responses when its B-nodes are available. For instance, the gate can update its membership list every hour. Therefore, the gate can keep the list up to date and minimize the possibility that a stolen car enters the gate even if all the B-nodes are unavailable.

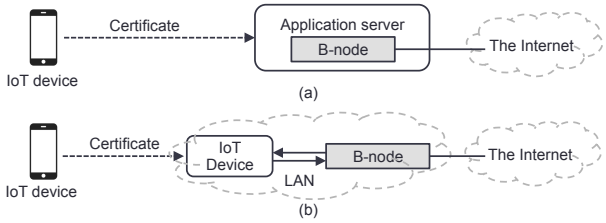

Fig. 3. Two experiment scenarios

VI. Evaluation

In this section, we first present the prototype implementation details of IoT-PKI, and then experimentally compare the performance of IoT-PKI and C-PKI.

A. Experimental Setup

- **IoT device:** We implemented an IoT device using a Raspberry Pi 2 [15]. We modified the OpenSSL [12] library (version 1.1.0) in order to support certificate verification via IoT-PKI in addition to C-PKI. The modified OpenSSL [16] is able to execute Step 3-2 in Sec. IV-B.

  To obtain an NVS response from a B-node, D_A creates a POST request using cURL commands [17] and sends it to the B-node. The POST request includes the URL of the B-node and the name of D_B. If the name of D_B is registered in the blockchain NVS, the B-node returns the stored hash of the certificate V_B to D_A. After receiving V_B, D_A compares it with the calculated hash of the certificate. If they match, the certificate is verified, otherwise the verification fails.
- **B-node:** We implemented a B-node using a PC with the Intel-Core i5 6300 HQ CPU (2.30 GHz) and 8 GB RAM. We installed the Emercoin wallet [9] on the PC. The Emercoin wallet is a full node that maintains an updated copy of the Emercoin blockchain. For our comparisons we place the B-node and the IoT device in the same local area network, which represents the most common scenario as the B-nodes maintained by the respective network administrators will mostly be one or two hops away from the actual device.
- **Device Manager:** We implemented a device manager and installed it in the same machine as the B-node. The device manager initiates the device set-up procedure in Sec. III-C. The device manager randomly generates a 16 byte ID for an IoT device and sends it to the IoT device after checking the ID is unique in the NVS. After receiving the self-signed certificate ($Cert$) from an IoT device, the device manager computes the hash (SHA-256) value of $Cert$ and registers the name/value pair in the Emercoin NVS.

B. Certificate Verification Time

We measured the average time required to verify a certificate in two different scenarios: 1) an application server running on the same machine where the B-node is running verifies the

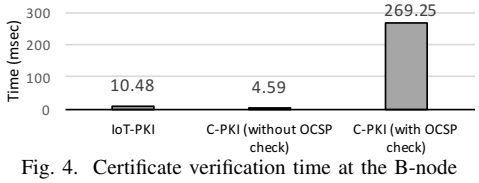

Fig. 4. Certificate verification time at the B-node

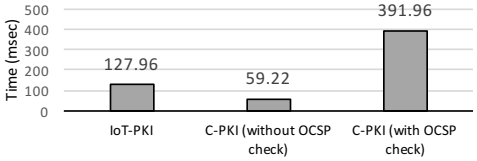

Fig. 5. Certificate verification time at the IoT device

certificates, as shown in Fig. 3 (a); 2) the IoT device verifies the certificates by requesting NVS responses from the B-node in the same LAN, as shown in Fig. 3 (b),

The first scenario mimics an IoT data collection application. For example, a traffic monitoring server collects data from cars. A B-node can run in the same computer where the monitoring server is running if the computer has a persistent connection to the Internet and storage enough to run a B-node. The second scenario mimics the situation where an IoT device communicates with another IoT device. For instance, a university patrol drone that communicates with a car. Since the drone does not have enough storage to run a B-node, the drone needs to connect to a B-node in the university's internal network to verify the status of the car's certificate.

We also measured the time required to verify certificates signed by CAs to compare IoT-PKI with C-PKI. For the purposes of our comparison, we used the certificate of *www.google.com* which has a certificate chain length of 2 to represent C-PKI.

Fig. 4 and Fig. 5 show the average certificate verification time in the first scenario and the second scenario, respectively. For a fair comparison, we only consider the time to verify a certificate with a revocation check through an OCSP responder although we also report the time to verify a certificate without a revocation check. In the first scenario, to check whether a certificate is revoked or not, IoT-PKI takes only 10.48ms, while C-PKI takes 269.25ms. The time required to check a certificate via an OCSP responder includes 1) the time to find the IP address of the OCSP responder via a DNS query, 2) the time to send an OCSP query to the IP address after establishing a TCP connection, 3) the time to generate a response with its signature and 4) the time to receive the OCSP response and verify the signature. This has to be done for every certificate in the certificate chain[4]. However, in IoT-PKI, such overhead can be minimized if revocation checks can be performed over a B-node in the same machine like the first scenario or through a B-node in the same local network like the second scenario.

In the first scenario, IoT-PKI is 25.7 times faster than C-PKI. However, in the second scenario, IoT-PKI is only 3 times faster than C-PKI since the low computing power of the Raspberry Pi 2 increases signature verification time, and thus network latency becomes less dominant than in the first scenario.

The experimental results show that IoT-PKI is more appropriate than C-PKI for IoT applications in which short response time is required and certificate revocation checks must be frequently performed.

[4]According to the OCSP performance measurements from Netcraft [18], the average OCSP response time of 97 OCSP responders is 362ms. The average DNS look-up time is 259ms which is 71% of the total time, i.e., 362ms.

VII. Conclusion

In this paper, we propose a decentralized PKI for IoT, called IoT-PKI, based on a blockchain network. IoT-PKI addresses scalability by utilizing distributed B-nodes. IoT-PKI enables user-controlled certificates and does not suffer from single-point-of-failures. We have shown the feasibility of IoT-PKI by implementing a prototype test-bed. Our experimental results show that the revocation status of a certificate can be verified in IoT-PKI approximately from 3 times to 25.7 times faster than when a certificate is verified though OCSP in C-PKI.

Acknowledgments

This work has been supported by NSF under grant CNS-1719369, and the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] IBM, "Device democracy," 2015. [Online]. Available: https://www.ibm.com/services/us/gbs/thoughtleadership/internetofthings/
[2] A. Kott, A. Swami, and B. J. West, "The internet of battle things," *Computer*, vol. 49, no. 12, pp. 70–75, Dec. 2016.
[3] G. Ho *et al.*, "Smart locks: Lessons for securing commodity internet of things devices," in *ACM ASIA CCS*, 2016.
[4] OCF, "Open connectivity foundation," 2017. [Online]. Available: https://openconnectivity.org
[5] Microsoft, "Azure iot suite," 2017.
[6] Amazon, "Amazon web services iot," 2017.
[7] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin. org/bitcoin.pdf
[8] namecoin, "Namecoin," 2017. [Online]. Available: https://namecoin.org
[9] emercoin, "emercoin," 2017. [Online]. Available: http://emercoin.com
[10] M. Ali *et al.*, "Blockstack: A global naming and storage system secured by blockchains," in *USENIX ATC*, 2016.
[11] I. Bentov, A. Gabizon, and A. Mizrahi, *Cryptocurrencies Without Proof of Work*. Springer, 2016, pp. 142–157.
[12] OpenSSL Software Foundation, "Openssl," 2017. [Online]. Available: https://www.openssl.org
[13] Trusted Computing Group, "Trusted platform module," 2017. [Online]. Available: https://trustedcomputinggroup.org
[14] R. Maes, A. Van Herrewege, and I. Verbauwhede, "Pufky: A fully functional puf-based cryptographic key generator," in *CHES'12*. Springer.
[15] Raspberry Pi 2, "https://www.raspberrypi.org/," 2015.
[16] J. Won, "Openssl fork for iot-pki," 2017. [Online]. Available: https://github.com/JonghoWon/openssl
[17] D. Stenberg, "https://curl.haxx.se," 2017.
[18] Netcraft, "Ocsp response times," 2017. [Online]. Available: http://uptime.netcraft.com/perf/reports/OCSP