Blockchain-based Decentralized Public Key Management for Named Data Networking (Invited Paper)

Kan Yang*, Jobin J. Sunny†, Lan Wang*
*Department of Computer Science, The University of Memphis, Memphis, TN, USA
†St. Jude Cloud, St. Jude Children's Research Hospital, Memphis, TN, USA
Email: {kan.yang, jjsunny, lanwang}@memphis.edu

***Abstract*—Named Data Networking (NDN) uses public-key based identities and trust models to achieve *data-centric security*. Each NDN data packet is signed by its producer, and any data consumer can check the data integrity and authenticity by following a chain of trust to verify that the data is signed by a public key associated with the data producer. Such trust chains typically end at an application-specific trust anchor whose public key is either preconfigured into the software package or can be verified through some means outside the application. As these trust anchors play a critical role in ensuring the security of NDN applications, it is highly desirable to develop a public key management system to register, query, update, validate, and revoke their public keys. However, traditional public key management system such as Public Key Infrastructure (PKI) and Web-of-Trust (WoT) suffer from various problems. In this paper, we propose BC-PKM, a public key management system for NDN that takes advantage of the decentralized and tamper-proof design features of Blockchains. We further prove that BC-PKM can resist a variety of attacks from adversaries that compromise less than half of the public key miners. Moreover, we demonstrate a prototype that implements the proposed API of BC-PKM.**

***Index Terms*—Blockchain, BC-PKM, NDN, PKI**

I. Introduction

Named Data Networking (NDN [1]) is a new Internet architecture in which data is retrieve by its name rather than from a specific address. In NDN, each piece of data has a unique name and a consumer interested in the data sends a request for that name to the network. The network forwards the request to a suitable next hop until the named data is found. The data will then be delivered back to the consumer along the reverse path of the request. In contrast to today's host-centric data delivery model, NDN data is independent from location, storage, and means of transportation, which enables efficient data distribution, multi-party communication, and robust communication in challenging networking scenarios [2].

NDN's data-centric retrieval model requires securing the data directly, instead of securing the channel between the sender and the receiver as in traditional IP networks. Specifically, *data-centric security* in NDN needs to achieve three fundamental goals: *1) Data Integrity:* data cannot be modified after it is produced without being detected; *2) Data Authenticity:* data is published by the claimed producer; and *3) Access Control:* only consumers (producers) with appropriate authorizations can access (publish) the data. Public Key Cryptography has been used extensively in NDN to achieve these goals. Each piece of data and its associated name are bound by a *cryptographic signature* using the producer's private key [1]. A data consumer checks data integrity and authenticity by verifying the signature using the producer's public key. Name-based Access Control (NAC) scheme [3] controls the access of named data by distributing data access keys securely using the public key of each authorized consumer.

The above schemes require validating that a named public key indeed belongs to an entity authorized to produce the data in a given name space. Such validation can be based on application-specific *trust models* derived from the relationships among the entities that participate in the applications and who/what the entities ultimately trust [4]. A *trust schema* containing a set of *trust rules* can be developed for the application to automatically infer the correct signing key for each received data (or key) [4]. When a data packet is received, a verification process uses the trust schema to recursively retrieve and validate each signing key until it reaches a *trust anchor* whose public key is pre-configured in the software or can be verified through some means outside the application. Note that the trust anchor here is typically *not a public third-party Certificate Authority (CA)*, but rather a trusted entity within a specific application. For example, a university's student information system may use the university's public key as its trust anchor and a company's news releases may use the company's public key as its trust anchor.

As the application-specific trust anchors play a critical role in ensuring the security of NDN applications, it is highly desirable to develop a public key management system to store and retrieve their up-to-date validated public keys. The traditional *Public Key Infrastructure (PKI)* model based on Certificate Authorities (CA), however, suffers from a serious problem, i.e., *an attacker can compromise a CA to bind a key name to an unauthorized public key and produce false data using the fraudulent certificate*. Even if there are multiple CAs, the compromise of one CA can inflict great damage. In fact, companies have spent significant resources fixing the security breaches caused by misbehaving CAs [5, 6]. An alternative model is *Web of Trust (WoT)* [7, 8] which enables any user

*Jobin J. Sunny participated in this work when he was a graduate student at The University of Memphis.

to issue a certificate (i.e., an endorsement) to anyone he or she trusts. The certificate is then trusted by whoever trusts this endorser. WoT is a notable step towards decentralizing trust, i.e., trust is not derived from a single CA. However, the compromise of a widely trusted endorser in WoT can still impact many users who trust this endorser. In addition, there are many practical obstacles to deploying WoT, e.g., the need to physically attend key signing parties and the security risks of using third party keyservers (see Section VI).

In this paper, we propose a *decentralized public-key management system*, BC-PKM, for managing application-specific trust anchors and other important keys in NDN. This system allows geographically and politically disparate entities to reach consensus on the state of a shared public-key database. To achieve this goal, we employ Blockchain [9], a decentralized ledger technology that enables the recording of transactions or any digital interactions in a secure, transparent, and auditable way. The main idea is to build a public key blockchain for each namespace in NDN (e.g., /com, /edu, /edu/memphis, and /edu/memphis/cs), which is maintained by multiple miners that validate the principals behind public keys and create blocks containing the validated keys. Due to blockchain's properties and our validation mechanism [9]), our public key management system is tamper-proof and can tolerate $t (t < (n+1)/2)$ compromised miners out of n total miners. The contributions of this paper are summarized below:

1) We design a blockchain-based decentralized public key management system (BC-PKM) for NDN which can be used to register, query, update, validate, and revoke public keys of important principals.
2) We prove that, by employing the majority rule of principal validation and the chaining structure and consensus mechanism of blockchain, our design can resist a variety of attacks from adversaries that compromise less than half of the public key miners.
3) We further implement a prototype of BC-PKM to demonstrate its functionality and feasibility in practice.

We note that the objective for this work is to sketch out a high level design for BC-PKM. While we use specific blockchain mechanisms, e.g., proof-of-work [9], to explain the functionality of the miners, the BC-PKM design is not tied to a specific blockchain implementation. In fact, one of the open research issues is to design a blockchain that is efficient and suitable for public key management.

The remainder of this paper is organized as follows. We introduce NDN's public key management problem in Section II. We present BC-PKM In Section III and analyze its security in Section IV. In Section V, we show a prototype implementation of BC-PKM. Finally, we present related work in Section VI and conclude the paper in Section VII.

II. Problem Description

A. Public Key Management in NDN

The nature of data-centric communication in NDN requires its security to be data-centric as well, because traditional session-based security solutions (e.g., TLS and IPSec) are no longer applicable as there is no concept of session in NDN. To achieve both data integrity and data authenticity, NDN requires that every data producer create a cryptographic signature for each piece of data with its private key to bind the data name and content together. Any data consumer can verify the signature using the public key of the producer. If the verification is successful, the consumer can make sure that the received data has not been changed (i.e., data integrity) and is from the producer (i.e., data authenticity). NDN also provides a name-based access control scheme [3] to control the access of named data in NDN by distributing data access keys securely using the public key of each authorized consumer.

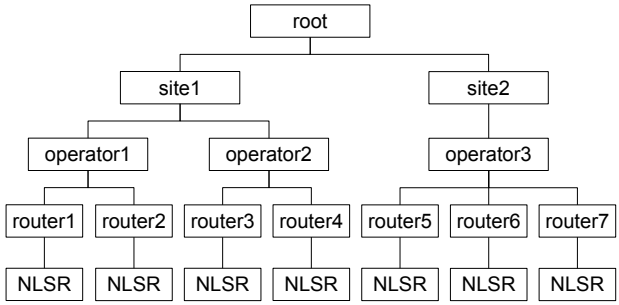

Fig. 1. Trust Model in Named-Data Link State Routing Protocol [10]

In order to authorize a producer to produce data in a given namespace, each application defines a trust model and the corresponding trust schema [4] that allows a consumer to check the following: 1) the key locator field in the data contains a key that is expected by the schema; 2) the key can be retrieved using information contained in the key locator; and 3) the signature matches the data name and content based on the retrieved public key. The verification process repeats these steps until it reaches an application-specific trust anchor. Figure 1 shows the trust model used in the Named-Data Link State Routing Protocol (NLSR), an intra-domain routing protocol for NDN [10, 11], where the trust anchor is the root key of an entire autonomous system, e.g. the NDN testbed or AT&T's network. The trust anchor's key may be preconfigured and distributed in the software package or manually configured by the administrator. However, it is difficult to update such keys when they expire or are revoked, and they must be protected against malicious attacks that modify them in transit or in storage. It is desirable to have a public key management (PKM) system that provides validation and lookup service for these important keys. As shown in Fig. 2, this system needs to provide the following five basic functions:

- *Register*: it allows a principal to register its public key by submitting a pair of (name, public key).
- *Query*: it returns the corresponding public key (validated and latest) for a name.
- *Validate*: it verifies whether a pair of (name, public key) is valid or not.
- *Update*: it enables a principal to update its public key.

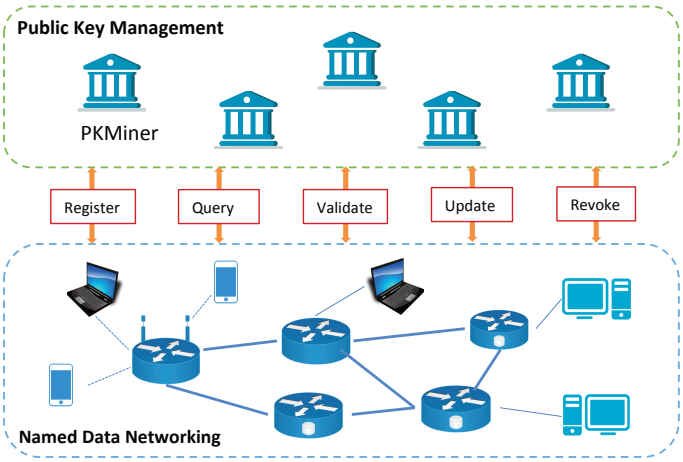

Fig. 2. Public Key Management and NDN

- *Revoke*: it revokes a principal's public key from the system.

Upon receiving a data packet, a consumer first verifies whether the data's signing public key is expected based on the application's trust schema and whether the signature in the data can be verified using the associated public key. It then queries the PKM for the trust anchor's public key and uses the trust schema to verify whether the producer's public key can be verified all the way to the trust anchor. The trust anchor's key can be cached for some time so that it does not have to be repeatedly retrieved for other data in the same application. One may ask how the consumer can obtain the trust schema. If the consumer is part of the application, then the trust schema is embedded in the application, e.g, specified in the configuration. The application developer may also publish the schema using a well-defined name based on the application's namespace and sign it using the trust anchor's public key. In this way, if the consumer is not part of the application, e.g., a router in the network, it can retrieve the trust schema and verify it using the trust anchor's public key.

In principle, a PKM can be used to manage public keys generated by any entity, not just trust anchors. For example, all the public keys of universities, students, their devices, and software processes running on those devices can be stored in the PKM. The question is whether the PKM, rather than a simple trust chain, is needed for managing specific keys deep in an application's namespace as there is overhead associated with running such a system. This is an open question that requires further study. In the rest of this paper, we simply use the word "principal" to represent an entity that desires to use the PKM to manage its key.

B. Compromised CA Problem in Public Key Management

Traditional public key management systems (e.g., PKI) suffers from the *Compromised CA Problem*, because each CA has unconstrained privilege to certify new keys. Specifically, a compromised CA can launch the following attacks:

- *Register Public Keys for Ilegitimate Principals*: the compromised CA can issue a certificate to a principal that cannot be authenticated correctly in practice. For example, a web server that does not belong to The University of Memphis may obtain a certificate in the university's namespace from this CA.
- *Update Public Keys for Existing Principals*: the compromised CA may arbitrarily change a public key for a principal that registered its public key under this CA.
- *Revoke Public Keys for Legitimate Principals*: the compromised CA may maliciously revoke a public key, which is still valid and up to date, for a legitimate principal.

The *Compromised CA Problem* is one of the most challenging security problems in public key management. In this paper, we aim to solve this problem by splitting the privilege of public key validation among multiple validators.

III. BC-PKM: Blockchain-based Decentralized Public Key Management

In this section, we propose a blockchain-based decentralized public key management system, called BC-PKM, for NDN. For each namespace, a number of designated PKMiners validate the principals that submit their public keys in that namespace and maintain a blockchain containing validated public keys. To counter attacks from compromised PKMiners, we employ a *majority rule* for the public key validation. As long as at least half of the PKMiners are honest, the longest blockchain will not contain an invalid public key record. The chaining structure and consensus model of blockchain further ensure that the blocks containing public key records are tamperproof and consistent among the PKMiners, which is different from a traditional distributed database. To support NDN, we use NDN data names for the blockchains in BC-PKM.

A. Design

As shown in Fig. 3, instead of relying on individual CAs to issue certificates, we create a blockchain to manage the public keys. This blockchain can be updated only by the PKMiners, but it can be retrieved by anyone and the PKMiners provide services for public key management: Register, Query, Validate, Update, and Revoke.

- $\mathsf{Register}(\mathsf{name}, \mathsf{pk}) \rightarrow \mathsf{Success}/\mathsf{Fail}$. Any principal can register a name and bind a corresponding public key to this name. The principal is the entity behind the name. To register a pair of name and public key in the system, the principal will first generate a public-private key pair, then sign the record (name, pk) using the private key to bind the public key to the name. As shown in Fig. 4, the registration consists of the following steps:
  1) The record $(name, pk)$ is sent to the multicast name prefix shared by all the PKMiners that maintain the blockchain.
  2) Upon receiving the record $(name, pk)$, each PKMiner validates the principal using channels specific to the namespace, e.g., verify the university issued ID and email of the principal if the namespace

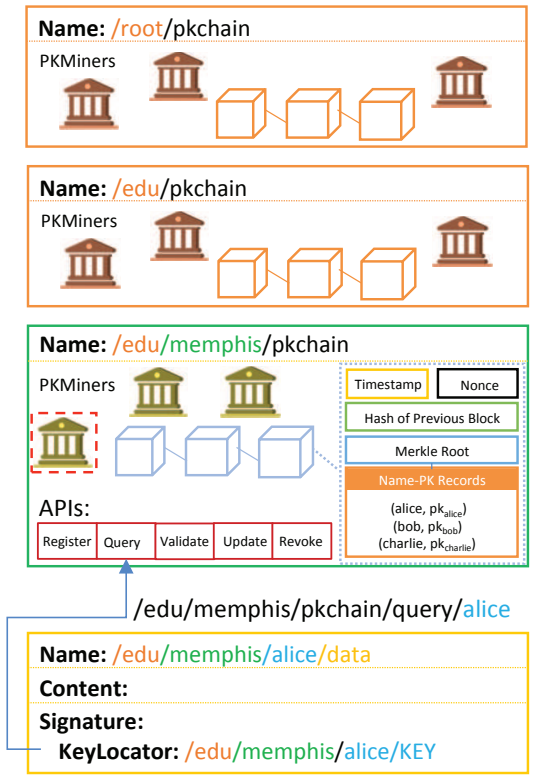

Fig. 3. BC-PKM Framework

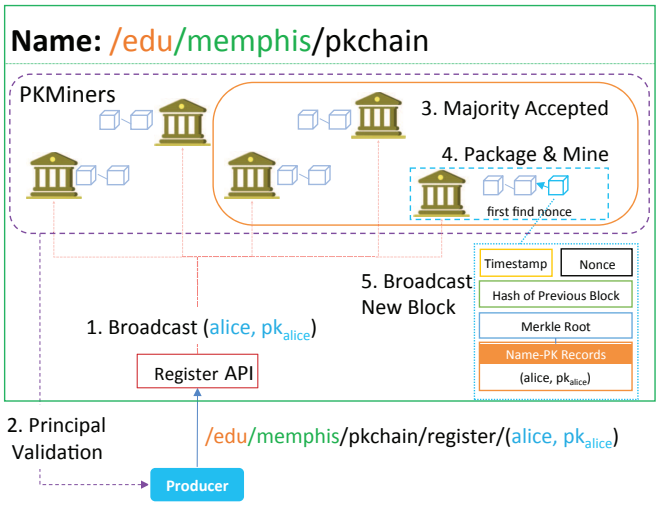

Fig. 4. An Example of BC-PKM Registration

belongs to a university. Then each PKMiner will notify all the other PKMiners its validation result.

3) If more than half of the PKMiners have positive results, the principal is considered legitimate.
4) The PKMiners then put the record $(name, pk)$ into a block and start to mine it. If the blockchain uses hash-based proof-of-work for chaining and consensus, the PKMiners will compute the nonce such that the hash of the block is less than a threshold.
5) The PKMiner that first completes the mining can append this block to the blockchain and notify all the other PKMiners of the new block.

When the record is added into the blockchain, the principal receives the result Success, otherwise it receives Fail.

- $\mathsf{Query}(\mathsf{name}) \rightarrow \mathsf{pk}_{\mathsf{name}}/\mathsf{NotFound}$. The query function returns the latest and valid public key corresponding to a given name. A node can retrieve the whole chain and store the up-to-date validated name/key pairs in a hash table. For a less capable node, it can send a request to the PKMiners' name prefix to fetch the key.
- $\mathsf{Validate}(\mathsf{name}, \mathsf{pk}) \rightarrow \mathsf{Valid}/\mathsf{Invalid}$. Given a pair of name and public key $(name, pk)$, the verification function returns whether this pair is valid or invalid. It calls the Query funtion with the *name*, then compares the output with the given public key *pk*. If they are the same, it returns Valid. Otherwise, it returns Invalid.
- $\mathsf{Update}(\mathsf{name}, \mathsf{pk}^*) \rightarrow \mathsf{Success}/\mathsf{Fail}$. This update function enables any principal to update the corresponding public key for a registered name. Because the blockchain is tamperproof, we cannot modify the content in previous blocks. Instead, we attach a new block to the blockchain to indicate the update of the public key. When updating a public key for a given name, the $(name, pk^*)$ is sent to all the PKMiners, which will redo the principal validation. With this validation, if the public key update is requested by a compromised PKMiner instead of the principal, the principal will be aware of this unexpected public key update. When the updated public key is appended to the blockchain, the Update function returns Success, otherwise it returns Fail.
- $\mathsf{Revoke}(\mathsf{name}, \mathsf{pk}) \rightarrow \mathsf{Success}/\mathsf{Fail}$. If a public key is compromised, it needs to be revoked. However, one cannot delete the records for that public key, as the blockchain is tamper-proof. Similar to the update operation, we add a revoking record to indicate the pair of $(name, pk)$ is revoked. Before competing to add the record to the blockchain, more than half of the PKMiners must agree that the public key should be revoked. If the record is successfully added into the blockchain, the function returns Success. Otherwise, it returns Fail.

Equipped with the above API, the blockchain can provide all the public key management functions. As shown in Fig. 3, the namespace /edu/memphis has the public key blockchain named /edu/memphis/pkchain, and /edu has the blockchain /edu/pkchain. To obtain a key for the principal named /edu/memphis/alice, one can issue an interest with the name /edu/memphis/pkchain/query/alice. Other APIs can use similar names. For example, an interest /edu/memphis/pkchain/update/(alice, pk) can be used to update the key for Alice.

Note that the construction of blockchain also needs the public keys of all the PKMiners, which are used for securing the communication among them. All the public keys of PKMiners who maintain /edu/memphis/pkchain are registered

in the blockchain /edu/pkchain. Likewise, the public keys of PKMiners who maintain /edu/pkchain are registered in the blockchain /root/pkchain. The public keys of the root PKMiners are well-known. Since they also use the majority rule to validate public key records and the blocks they create are public and tamper-proof, a misbehaving root PKMiner has limited capabilitiy to cause damage.

B. Discussion

We now discuss a few design questions. First, who can be the miners/validators? In some public ledgers such as Bitcoin, anyone in the system can be a miner. However, in a PKM system, it may not be desirable to enable everyone in the system to be a validator for public key registration [12]. In our future work, we will explore this question together with the design of the consensus mechanism.

Second, how to validate a public key? In Bitcoin, the verification of a transaction can be done by verifying all previous transactions related to this transaction. In our design, the validation needs to use an out-of-band mechanism specific to a namespace which may require the identity, address, email, or picture of a principal. An automated validation mechanism such as Let's Encrypt [13] can be incorporated into the PKM.

Third, which consensus mechanism should we use? The consensus mechanism enables all the users in the system to agree on the same accepted blockchain, which is the core technique of any blockchain-based system. Bitcoin uses proof-of-work and longest chain as the consensus mechanism. However, proof-of-work wastes a huge amount of resources. We plan to design a more efficient consensus mechanism based on other consensus mechanisms (e.g., proof-of-stake [14] and Algorand [15]), for BC-PKM.

IV. Security Analysis

In this section, we show how BC-PKM can counter attacks from compromised PKMiners, which may register a public key for an illegitimate principal, and modify or revoke the public keys of existing principals.

Theorem 1. *BC-PKM can resist t out of n $(n > 2t-1)$ compromised PKMiners against registering public keys for fake principals.*

Proof. A compromised PKMiner may register a public key for a name that does not match the principal behind the name. For example, the PKMiner may register a public key for Eve but with a name called Bob, i.e., (Bob, pk_{Eve}). If the registration succeeds, the adversary Eve can impersonate Bob to do anything, e.g., sign a document, or receive messages sent to Bob. In BC-PKM, any registration request will be broadcast to all the PKMiners for validation. Each PKMiner then validates the name and principal using some other channels, and broadcasts the result to all the other PKMiners. If and only if more than half PKMiners say that the name corresponds to the principal, the PKMiners can compete to add the block. However, we assume that the adversary can only compromise less than half of the PKMiners. Thus, even if all the compromised PKMiners broadcast a fake "validation succeed" message, the other honest PKMiners will still reject this registration request.

Suppose the compromised PKMiner does not follow the protocol, and insists on adding the record to the blockchain. The blockchain will finally be synchronized among all the PKMiners by broadcasting the new block to all the other PKMiners. Then, the honest PKMiners will find that the newly added block is not correct, as it does not pass the name-principal validation. They will not add this block to their chain. Even if the compromised PKMiners continue to fork the chain, their computing power will be less than the honest PKMiners, so their chain will eventually be discarded. Similarly, BC-PKM can also resist DoS attacks in which the compromised PKMiners deny the validation for legal principals. □

Theorem 2. *BC-PKM can resist t out of n $(n > 2t-1)$ compromised PKMiners against illegally updating public keys for existing principals.*

Proof. In BC-PKM, the method to update the public key of an existing principal is to add a new block with a record (name, updated public key). This procedure is the same as registering a new public key, which requires all the PKMiners to redo the name-principal validation. During the name-principal validation, the PKMiner will contact the corresponding principal. If this update request is launched by a compromised PKMiner, then the principal will refuse the validation and notify all the PKMiners of this error. Upon receiving many reports from different existing principals, the PKMiners will send a notice to the upper-level blockchain, and the upper-level blockchain may revalidate the PKMiners. □

Theorem 3. *BC-PKM can resist t out of n $(n > 2t-1)$ compromised PKMiners against illegally revoking public keys for existing principals.*

Proof. To revoke a public key, a revoking record will be added to the blockchain to indicate that the pair of (name, pk) is revoked. Unlike the Register and Update, there is no need to do the name-principal validation. Instead, more than half PKMiners should agree that the public key or the name has been revoked before competing to add the revoking block to the blockchain. Because the adversary cannot compromise more than half PKMiners, the adversary cannot maliciously revoke those still valid public key for any existing users. Moreover, it also guarantees that the adversary cannot deny legal revocation by sending disagree messages. □

V. BC-PKM Prototype Implementation

We implement a BC-PKM prototype to further demonstrate the functions of public key management system. To implement this prototype, we choose to use the Node.js framework due to its asynchronous capabilities and ability to handle peer-to-peer communications well. Note that, the main purpose of this prototype is to validate the functions of our BC-PKM system, so for simplicity, the communication is not

Fig. 5. BC-PKM Menu

implemented via NDN. The framework's event-driven, non-blocking I/O model makes it a good fit for our implementation. To handle the p2p communication, we used a Node.js library called peer-exchange (https://www.npmjs.com/package/peer-exchange), which allows for completely decentralized peer discovery and signaling. The library uses WebRTC for peer connections. It also provides some security guards and defenses against attacks. The command line interface was created with the help of a library called Vorpal (https://www.npmjs.com/package/vorpal). This framework helps build interactive CLI applications for Node. We used the crypto-js library (https://www.npmjs.com/package/crypto-js) for its SHA256 hash implementation. We chose to use this popular, well-maintained package to make sure that the hashing function is efficient and secure.

Fig. 5 shows the menu of our prototype, which consists of all the functions we defined in Section III. Then, we briefly demonstrate each function by showing some screenshots of a demo with three PKMiners: PKMiner1, PKMiner2 and PKMiner3. In Fig. 5, we successfully register a name-key pair (John, pk = 1xui45nal) to the blockchain. The added block in the blockchain is displayed as Block #1 in PKMiner1's Window in Fig. 6. We can see that Block #1 consists of Previous Hash, Timestamp, Name, Public Key, Hash of the Block, Nonce, Miner ID (indicating which PKMiner adds this block), and Revoked Status. Block #1 is synchronized among all the three PKMiners, which can also be seen in PKMiner2's Window.

Then, we send a new registration request for Mary (Mary, pk = bqnas83p2a3iz) from PKMiner2. After successful registration, Mary's public key is stored in Block #2, as we can see in PKMiner1's Window in Fig. 7. PKMiner3's Window in Fig. 7 shows the query functions including query by name and query by public key.

PKMiner1 in Fig. 7 sends an update request to update John's public key. The results are shown in Fig. 8: a new block (Block #3) with the name and its updated public key is appended to the blockchain in PKMiner1's Window. In PKMiner3 Window, it shows that after John's public key is updated, we cannot query the name by the revoked key. It also shows that the updated public key is returned when querying the name John.

PKMiner1 in Fig. 8 revokes the John's public key, and the revoked block (i.e., Block #4) is shown in Fig. 9. As we can see in Block #4, the revoke status becomes true. Now, when we query the name John or the public key of John, it returns that the name of the public key is revoked, as shown in PKMiner3's Window. PKMiner2's Window also shows the validate function of our BC-PKM.

VI. Related Work

Public Key Infrastructures such as X.509 PKIX [16] and Web of Trust [7, 8] are designed to certify that a public key indeed belongs to a name and the principal behind the name.

In X.509 PKIX[16], trusted third-parties (TTPs) called Certificate Authorities (CAs) are employed to verify the names/principals and bind the name together with the corresponding public key. Due to the unconstrained privileges of CAs, they become central points of failure of the entire network. If a CA is compromised, the attacker can bind names to arbitrary public keys, which may result in severe security problems.

Another PKI used in practice is PGP Web of Trust [7, 8], which enables any user to issue a certificate to anyone he or she trusts. The certificate is then trusted by a party if he/she can verify that the certificate contains the signature of someone who is trusted. The PGP Web of Trust is not itself a PKM, as it does not provide a mechanism for retrieving public keys or certificate chains. In practice, PGP is implemented by centralized *keyservers* that store and answer queries about certificates. However, the *keyservers* remain central points of failure and become bottlenecks when the system grows large. Due to the complexity of key generation and managements, some web hosting companies even choose to manage the creation of these keys by themselves, rather than letting their clients create their own pairs of secret key and public key. This leads to major security issues, as it may leak out the secret keys for all the users if the web hosting company is compromised. Finally, it is possible that one user may not be able to find a chain of trust to verify another user's key, i.e., the other user's key is not endorsed by anyone this user trusts.

To cope with the problem of faulty CAs, a straightforward solution is to reduce the power of each CA by splitting its power among multiple distributed CAs and use a majority rule during the name-principal validation. This method alone cannot prevent a compromised CA from issuing a certificate to an invalid principal, because the public key database cannot be guaranteed to be consistent among all the CAs without a blockchain.

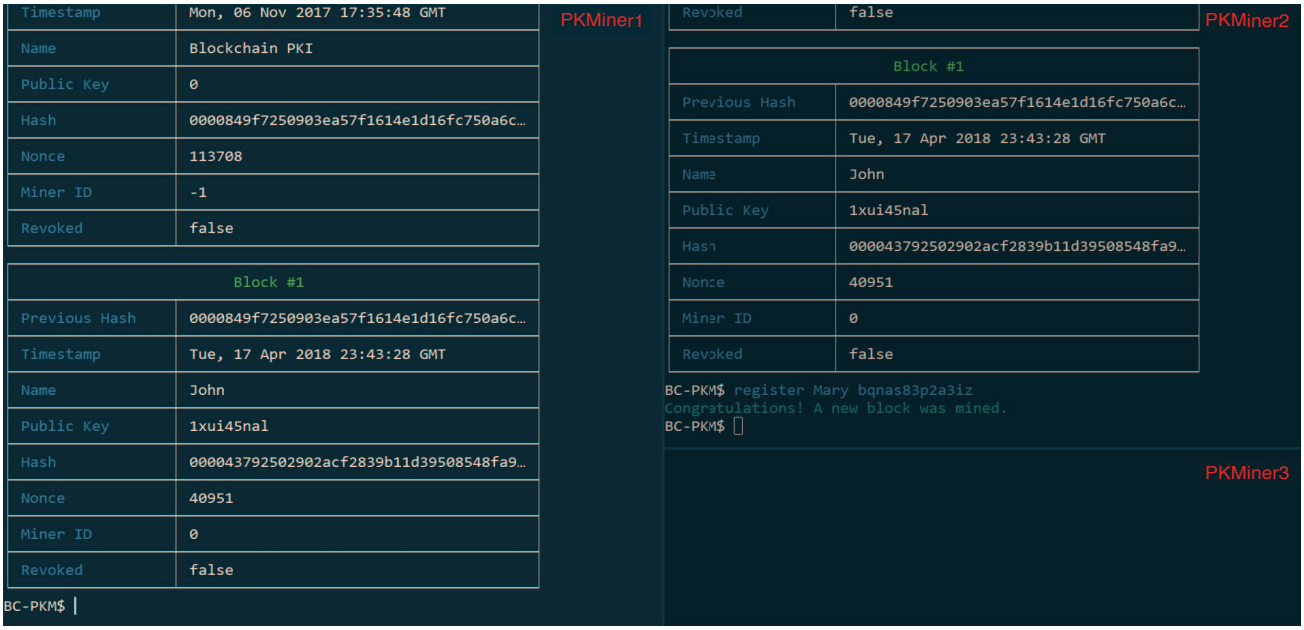

Fig. 6. View of Block #1 and Register Mary

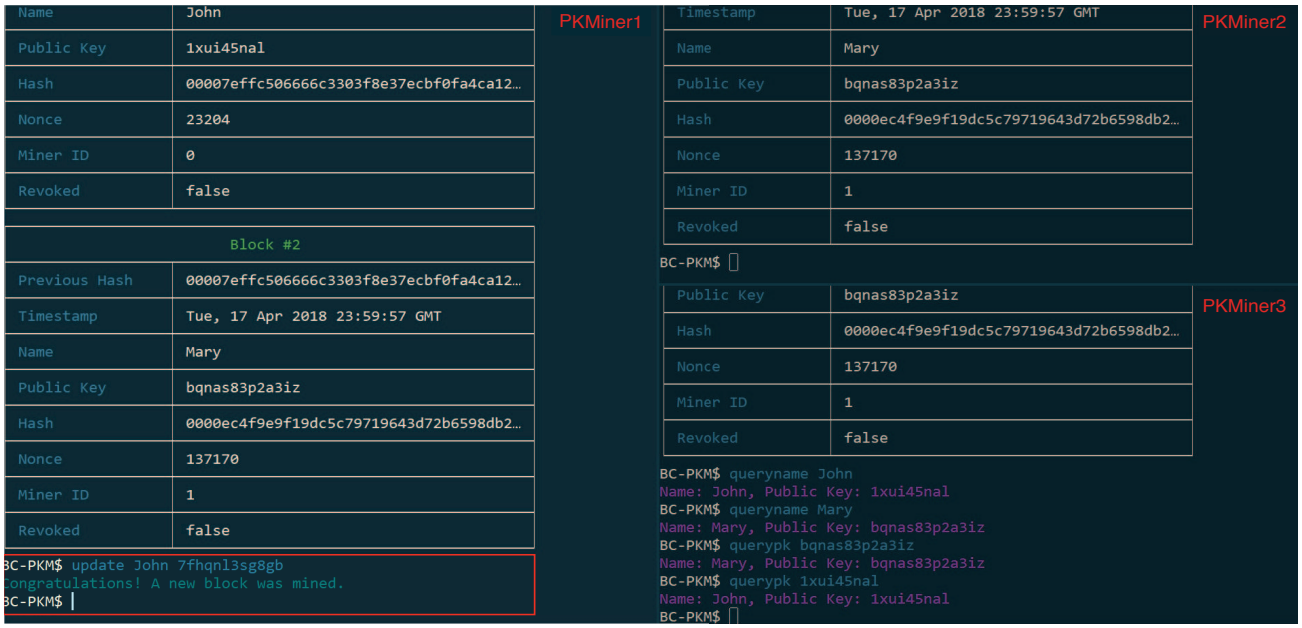

Fig. 7. Query and Update

A blockchain contains a continuously growing set of data records, which is decentralized in nature as there is no central certifying authority holding the entire chain. Its chaining structure and the consensus mechanism (e.g., proof-of-work [9] and proof-of-stake [14]) ensure that the blockchain is tamperproof and consistent. As such, it has been applied in cryptocurrencies (e.g., Bitcoin [9]) and smart contract systems (e.g., Ethereum [17]), with promising applications in Internet of Things [18].

In [19], the authors implement a bitcoin blockchain based on NDN. The blockchain is also used in [20] to disseminate the system parameters when a Hierarchical Identity Based Encryption (HIBE) is employed to build the data access control scheme. Due to the advantages of blockchain, it has been used to implement alternative PKIs [21–27]. In [21], Fromknecht *et al.* proposed to utilize the Namecoin [28] to implement a PKI system. After that, they proposed Certcoin [22], an improved PKI implementation which can prevent identity retention. In [26], the authors protect user privacy in Certcoin by employing a hidden layer that records the real match between the user

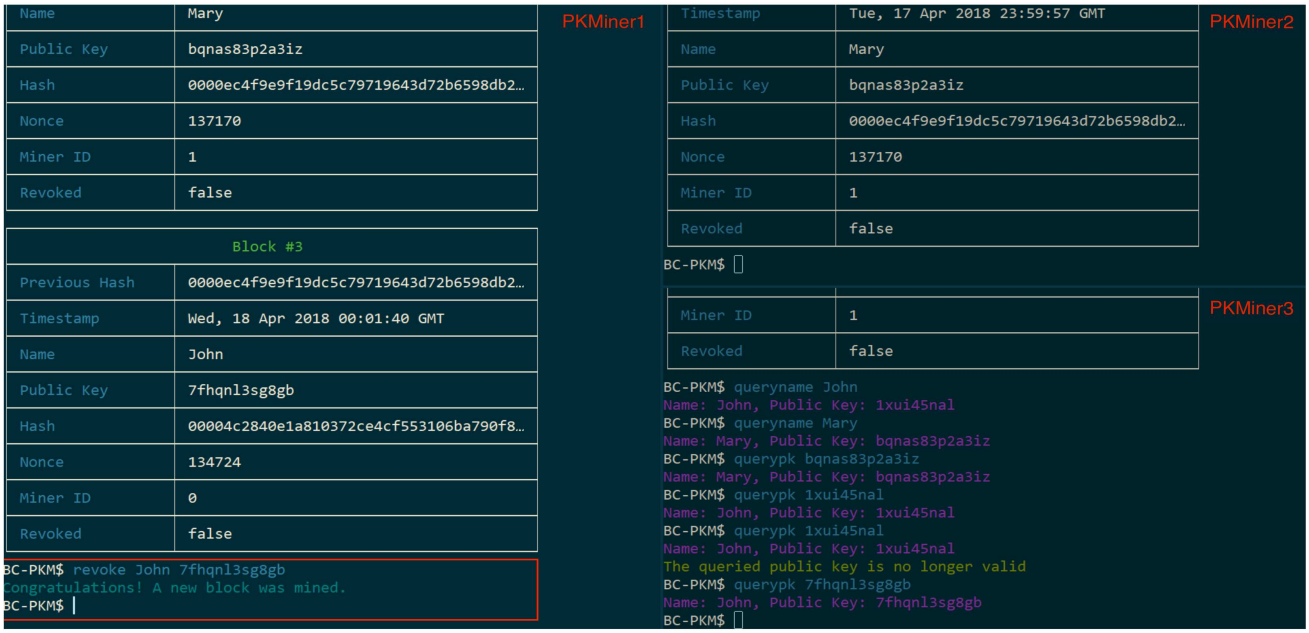

Fig. 8. Updated View of Blockchain and Revoke

identity and an offline key pair. In recent work [27], the authors proposed a decentralized public key management system based on blockchain, where smart contract is utilized to do revocation. In [25], the authors propose a challenge-response approach of validation and authentication for WoT. However, these existing systems did not consider the compromised CA problem and were not designed for NDN.

VII. Conclusion

In this paper, we proposed a decentralized public key management system (BC-PKM) for NDN based on blockchain, which can solve the *Compromised CA Problem* in traditional public key management systems. The basic idea in BC-PKM is to split the power of an individual CA among multiple PKMiners that maintain the public key blockchains. The majority rule in name-principal validation allows BC-PKM to resist attacks from compromised PKMiners. Our security analysis and prototype implementation demonstrate that BC-PKM is secure and practical. In our future work, we will implement the prototype on the NDN platform and investigate efficient blockchain design for PKM. We will also apply BC-PKM to secure NDNS [29].

References

[1] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named Data Networking," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 44, no. 3, pp. 66–73, Jul 2014.

[2] D. Kutscher, S. Eum, K. Pentikousis, I. Psaras, D. Corujo, D. Saucez, T. Schmidt, and M. Waehlisch, "Information-centric networking (icn) research challenges," RFC 7927, 2016.

[3] Y. Yu, A. Afanasyev, and L. Zhang, "Name-based access control," *NDN TR NDN-0034, University of California, Los Angeles, Los Angeles*, 2015.

[4] Y. Yu, A. Afanasyev, D. Clark, V. Jacobson, L. Zhang *et al.*, "Schematizing Trust in Named Data Networking," in *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM, 2015, pp. 177–186.

[5] L. Whitney, "Comodohacker returns in diginotar incident," *CNET: News: Security & Privacy. CNET*, vol. 6, 2011.

[6] R. Gill, "Trust in the era of hackable certificate authorities," Jun. 2016, akami Blog, https://enterprise-access.akamai.com/blog/trust-in-the-era-of-hackable-certificate-authorities/, last accessed 4/21/17.

[7] J. Callas, L. Donnerhacke, H. Finney, , and R. Thaye, "Openpgp message format," RFC 1951, 1996.

[8] S. Carfinkel, "Pgp: Pretty good privacy," OReilly & Associates, 1994.

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

[10] L. Wang, V. Lehman, A. M. Hoque, B. Zhang, Y. Yu, and L. Zhang, "A secure link state routing protocol for ndn," *IEEE Access*, vol. 6, pp. 10 470–10 482, 2018.

[11] A. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang, "Nlsr: named-data link state routing protocol," in *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*. ACM, 2013, pp. 15–20.

[12] C. Allen et al., "Decentralized public key infrastructure - a white paper from rebooting the web of trust," www.weboftrust.info/downloads/dpki.pdf, Dec. 2015.

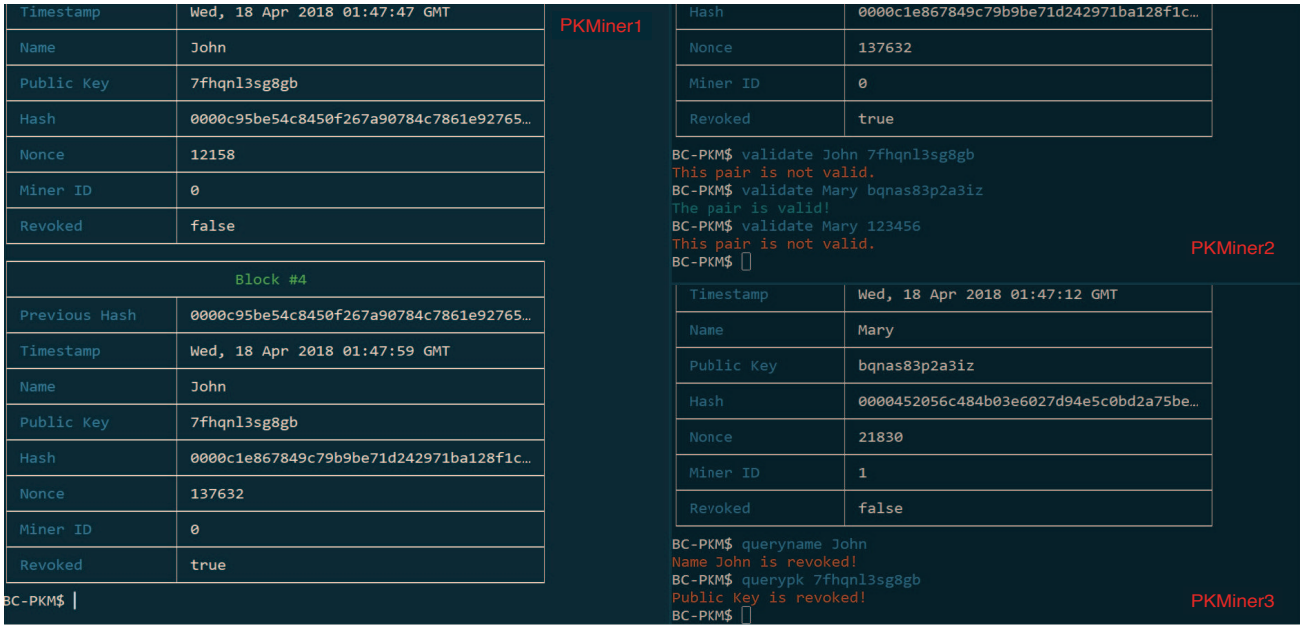

Fig. 9. Revoked View of Blockchain and Validate

[13] J. Aas, "Let's encrypt: Delivering ssl/tls everywhere," *Let's Encrypt*, vol. 18, 2014.

[14] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, "Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract]," *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 3, pp. 34–37, 2014.

[15] S. Micali, "Algorand: The efficient and democratic ledger," *arXiv preprint arXiv:1607.01341*, 2016.

[16] S. Santesson et al., "X. 509 internet public key infrastructure online certificate status protocol - ocsp," RFC 6960, 2013.

[17] Ethereum, https://www.ethereum.org/, 2017.

[18] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[19] T. Jin, X. Zhang, Y. Liu, and K. Lei, "Blockndn: A bitcoin blockchain decentralized system over named data networking," in *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, 2017, pp. 75–80.

[20] N. Fotiou and G. C. Polyzos, "Decentralized name-based security for content distribution using blockchains," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2016, pp. 415–420.

[21] C. Fromknecht, D. Velicanu, and S. Yakoubov, "Certcoin: A namecoin based decentralized authentication system 6.857 class project," *Unpublished class project*, 2014.

[22] ——, "A decentralized public key infrastructure with identity retention," 2014.

[23] H. A. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan, "An empirical study of namecoin and lessons for decentralized namespace design," in *14th Annual Workshop on the Economics of Information Security*, 2015.

[24] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A global naming and storage system secured by blockchains," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, 2016, pp. 181–194.

[25] B. Leiding, C. H. Cap, T. Mundt, and S. Rashidibajgan, "Authcoin: validation and authentication in decentralized networks," in *Tenth Mediterranean Conference on Information Systems (MCIS)*, 2016.

[26] L. Axon and M. Goldsmith, "Pb-pki: A privacy-aware blockchainbased pki," in *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017)*, Madrid, Spain, Jul. 2017, p. 311318.

[27] A. Yakubov, W. Shbair, A. Wallbom, D. Sanda *et al.*, "A blockchain-based pki management framework," in *The First IEEE/IFIP International Workshop on Managing and Managed by Blockchain (Man2Block) colocated with IEEE/IFIP NOMS 2018, Tapei, Tawain 23-27 April 2018*, 2018.

[28] A. Loibl, "Namecoin," in *Seminars FI / IITM SS 2014 Network Architectures and Services*, 2014.

[29] A. Afanasyev, X. Jiang, Y. Yu, J. Tan, Y. Xia, A. Mankin, and L. Zhang, "Ndns: A dns-like name service for ndn," in *26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2017, pp. 1–9.