Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 3314051, 13 pages
https://doi.org/10.1155/2022/3314051

Research Article

DAKEs: Decentralized Authenticated Key Exchange Protocols via Blockchain for Smart City

Qiong Wu,[1] Yi Luo,[2] Ying Zhao,[1] Bin Qian,[2] and Bin Guo[1]

[1]*Guangzhou Power Supply Bureau, Guangdong Power Grid Co., Ltd., Guangzhou 510730, China*
[2]*Electric Power Research Institute, CSG and Guangdong Provincial Key Laboratory of Intelligent Measurement and Advanced Metering of Power Grid, Guangzhou 510663, China*

Correspondence should be addressed to Yi Luo; luoyi_csg@outlook.com

Received 25 June 2022; Revised 27 July 2022; Accepted 1 August 2022; Published 24 August 2022

Academic Editor: Mu En Wu

Authenticated key exchange (AKE) is a classic problem in cryptography, where two participants want to exchange their secret keys without being guessed feasibly. Recently, there has been renewed interest in this problem in the smart city since millions of devices and servers in this environment may involve the problem. New challenges are raised at the same time. One of the greatest challenges is how to facilitate communication between participants. Traditionally, a trusted third party (TTP) is needed to provide a trusted way to exchange keys. However, devices in the smart city environment are usually distributed and trustless. A central trusted mechanism is not suitable for many applications in it. The second challenge is that the requirements in the applications of the smart city are diverse. Finally, a practical AKE protocol should be efficient and easy to integrate. To address these challenges, we provide a fully decentralized AKE protocol framework called DAKEs. To the best of our knowledge, DAKEs enjoy the most comprehensive security properties to fulfil diverse requirements. The decentralization of DAKEs is captured by using the blockchain while avoiding the availability problem of other similar blockchain-based schemes. Our test is conducted in a real-world test network of Ethereum. The result shows that DAKEs are efficient and at a low cost.

1. Introduction

As a classic cryptography problem, authenticated key exchange (AKE) is prescribed for sharing secret keys over insecure channel [1] and providing a way to authenticate the identity of the participants. Recently, there has been renewed interest in this topic as the concept of "Smart City" goes viral [2]. "Smart City" introduces a large number of participants (devices and servers, etc.) sharing data with each other. AKE protocol as a basic cryptography protocol can be used to secure the system for authentication, access control and data confidentiality, etc. Moreover, participants in a smart city are usually distributed and mutual untrusted. The trust and consensus for a digital economy activity in a smart city need to be more transparent. Thus, new challenges are raised in designing an AKE protocol fitting for a smart city.

One of the greatest challenges is how to facilitate communication between participants in a trustless environment. Traditionally, the two participants in an AKE protocol should build their communication under the assistance of a trusted third party (TTP). The TTP plays a role to manage the digital identities of the participants who have no prior relationship [3]. The trust and consensus are based on a central party. However, in a trustless and transparent environment like a smart city, a distributed trust paradigm is essential for a wide range of applications. In 2015, Patrick McCorry et al. [4] presented an AKE protocol via Bitcoin [5]. As an early study to realize AKE without any TTP, the paper showed us the potential power of blockchain technology. Thereafter, many researchers tried to explore blockchain as a facility of decentralized trust in a smart city. In 2020, Yavari et al. [6] proposed a blockchain-based authentication protocol for secure communication between IoT equipment. In 2022, privacy-preserving problem has become an important component in secure communication [7] [8].

Meanwhile, the capacity and availability limitation in blockchain technology becomes a primary concern, especially in Bitcoin-alike systems [9]. A long time might be taken to

wait for accepting a transaction in these systems. Yifei Hu et al. [10] provided a way to construct a specialized blockchain for building AKE protocols. Apparently, this is too expensive for constructing a basic protocol. Perhaps, a more practical way to design an authenticated key exchange protocol is to program on the blockchain instead of constructing transactions directly. Ethereum smart contract [11] provides such an approach. Indeed, there have been thousands of decentralized applications built via the Ethereum smart contracts. Recently, Robert Muth et al. [12] proposed a Diffie-Hellman Key Exchange with Smart Contracts. But the security requirements in the applications of the smart city can be diverse. And the demands for AKE protocols in different scenarios may be distinct. A considerate AKE protocol should take all the security properties into account. The smartDHX in [12] only fulfills a static security property. Chen et al. [13] provided the first post-quantum blockchain construction for securing a smart city. However, they only design the theoretical protocol but supply no real deployment.

To address these essential problems, we present DAKEs, a decentralized authenticated key exchange protocol framework via Ethereum smart contract. Specifically, our contributions are summarized as follows:

(i) We have proposed a blockchain-based decentralized authenticated key exchange protocol design framework and implemented five protocols with different security properties. The proposed framework called DAKEs provides a practical method to design fully decentralized authenticated key exchange protocols which avoid the problem of availability when using the Bitcoin-alike blockchains

(ii) To the best of our knowledge, the implemented protocols are the first work to take most key security properties in AKE into account. Not only DAKEs is fully decentralized, but also can be used in various applications which may have different security requirements

(iii) We conduct experiments in a live testnet of the Ethereum blockchain. The experiment shows that the protocols in DAKEs are effective and low-cost. We also gave a comprehensive comparison of the protocols, and anyone who wants to use a decentralized authenticated key exchange protocol can choose a suitable protocol in DAKEs according to his/her requirements

The rest of this paper is arranged as follows. In Section 2, we introduce the necessary preliminary knowledge. In Section 3, we give system model and the detail design of our framework. In Section 5, we provide the implementation of our protocol based on the testnet of the Ethereum blockchain. Finally, in Section 6, we draw a brief conclusion.

2. Preliminaries

2.1. Authenticated Key Exchange (AKE). Key exchange (KE) may happen when two participants want to build a secure channel to communicate with each other. To build a secure channel, they could use a common secret key and encrypt all the messages. Here comes the problem: How to exchange the secret key over an insecure channel? Moreover, the two participants should know the identity of the counterparty at the end of a key exchange protocol. We call this kind of protocol as authenticated key exchange (AKE) protocols [14].

Formally, suppose there are two users O and C who want to exchange a secret key called a session key, a secure AKE protocol should at least fulfil two requirements: Firstly, providing a way for the originator O to generate a random session key effectively and the key can only be known to the counterparty C eventually. Secondly, following the protocol, both O and C will know which person they are exchanging the session key with at the end of the protocol.

2.2. Threat Model. In our protocols, the certificates of the users and public security parameters are written on the blockchain. Firstly, we assume that there is an adversary who can eavesdrop on the data transferred between the two participants since the data on a public blockchain can be analyzed by anyone. The data on the blockchain is tamper-proof and traceable due to the feature of blockchain technology. However, we shall assume the adversary can delay or block the transactions since the adversary may be a powerful node of the blockchain and participate in the consensus. Under these assumptions, we then give the potential threats as follows:

(1) Man-in-the-Middle Attack [15]. In this type of attack, the adversary is able to recover a session key by blocking the message transferred between the two participants. To make this clear, we illustrate an insecure protocol as an example. Suppose a user O as the originator of the protocol want to exchange a session key with a user C as the counterparty of the protocol. In step one, O sends a random number r and his/her certificate $Cert_O$ to C. In step two, C query the certificate of O and verify its validity. Then, C sign on the data $\delta \coloneqq Sig_C(r, id_O)$ and encrypt the session key data $c \coloneqq Enc_C(k, id_C)$ by C's public key. Eventually, C sends $(\delta, c, Cert_C)$ to O. To attack this, the adversary R as a middle man can block the message which C send to O in step two and replace c with $c' \coloneqq Enc_C(k', id_R)$. At the end of the protocol, O will get a session key k', which is generated by the adversary

(2) Replay Attack [16]. In this type of attack, the adversary attacks the protocol by re-using the old information. Similarly, we give an insecure protocol example here. In step one, O only sends his/her certificate $Cert_O$ to C without the random number r. In step two, C sends $(c \coloneqq Enc_C(k, id_C), \delta \coloneqq Sig_C(c, id_O), Cert_C)$ to O. Then, the adversary can use a replay attack like this. First, the adversary collect the message between O and C, and record $(c, \delta, Cert_C)$. Later, the adversary starts a new exchange with O and sends $(c, \delta, Cert_C)$ recorded before. Suppose a stream cypher is used later to encrypt the message

with the shared key k. The adversary then gets two ciphertexts from O which encrypts two different messages m_1 and m_2. Let O encrypt $c_1 \coloneqq m_1 \oplus G(k)$ and $c_2 \coloneqq m_2 \oplus G(k)$ in the stream cipher. The adversary who gets c_1 and c_2 can compute $\Delta \coloneqq c_1 \oplus c_2 = m_1 \oplus m_2$. Redundancy in English text makes the adversary able to recover the two messages

(3) Identity misbinding attack [17]. In this type of attack, the adversary tries to mislead the participants into thinking they are talking to the expected counterparty. However, they are talking to the adversary indeed. Again, we give an insecure protocol example. In step one, O sends $r, Cert_O$ to C. In step two, C sends $c \coloneqq Enc_C(k, id_C), \delta \coloneqq Sig_C(r, c), Cert_C$ to O. The adversary can attack this protocol like this. First, the adversary blocks the message which O sends to C in step one and sends $r, Cert_R$ to C. Then, the adversary will get $c, \delta, Cert_C$ from C. And he then delivers $c, \delta, Cert_C$ to O. Finally, O will deem that C is his/her counterpart while C is talking to another originator R

(4) Chosen ciphertext attack [18]. An encryption scheme will usually be used in an AKE protocol. There is a kind of attack on encryption schemes called chosen ciphertext attack (CCA). In a CCA, the adversary is supposed to be able to choose some ciphertexts to ask for the corresponding decryptions. But there is a restriction that the adversary cannot ask for the plaintexts of any ciphertexts. If an encryption scheme used in an AKE protocol cannot resist CCA, it may suffer a number of potential threats. For instance, suppose a stream cipher is used in an AKE protocol. Firstly, the adversary gets a ciphertext c which is encrypted by a unknown string m and a secret key k, $c = m \oplus G(k)$. Then, the adversary can construct a ciphertext $c' = c \oplus \Delta$, where Δ can be an arbitrary string. c' can be taken as a legal ciphertext because the decryptor will decrypt it as $c' \oplus G(k)$ and $c' \oplus G(k) = m \oplus G(k) \oplus \Delta \oplus G(k) = m \oplus \Delta$. Now assume the length of k is n-bit, $m = k \| id_C$ and $\Delta \coloneqq 0^n \| (id_C \oplus id_R)$. The adversary can easily construct an encryption $c' = k \| id_R$ from $c = k \| id_C$

2.3. Security Properties. In the part of the threat model, we have already hinted at some of the properties we want for a secure AKE protocol. Let us try to make these just a bit more precise in this part. First of all, we suppose in an AKE protocol; the keys for authenticating an identity of a participant may be used for a long time. For example, a public key for a certificate *Cert* and its corresponding secret key may be used quite the long term; we call this kind of key a long-term key. Contrarily, the session key exchanged by the participants may be used for just a short time. And we call this kind of key the short-term key. The security properties of an AKE protocol can be distinct. And we give a comprehensive consideration of these security properties in our protocols. One can choose a proper security level for his own application. In general, there are six security properties [19] overall as follows:

(i) Static secrecy. This is the weakest security property where we assume that an honest participant's long-term secret key can never be compromised by the adversary. However, we assume that the adversary can query a session key from an instance of an AKE protocol except the one he/she tries to compromise. For a new instance, the session key k of an AKE protocol should be indistinguishable from a random key in the view of the adversary. Then, we say the protocol has a static secrecy property

(ii) Perfect forward secrecy. This is a stronger security notation than static secrecy where we assume an honest participant's long-term secret key can be compromised by the adversary. However, before the participant's long-term key is compromised, the session keys generated in the previous instances remain secret and cannot be distinguished from a random key. Although the protocol is no longer safe after the adversary gets a long-term key, this security property can limit the damage to the time after that

(iii) Explicit authenticity. This is a security property where the two participants O and C should be sure that they are talking to the other one when sharing the session key k. It means that they can know the identity of the counterparty explicitly

(iv) Implicit authenticity. It is a weaker security notation than explicit authenticity and sometimes may be enough for some applications. Suppose there are two participants O and C. At the end of the protocol, the originator O can make sure that the counterparty C was online following the protocol and hold the same session key with him/her, whereas C cannot get the same guarantee. He/she cannot be sure about whether there is a counterparty holding the same session key. Moreover, he/she also has no idea whether the counterpart was online. But if he/she can make sure that if someone gets the same session key, then that must be O

(v) Identity protection. This is a security property where the privacy of the participants can be protected under the AKE protocol. The adversary cannot get any useful information about the identities

(vi) Deniability. This security property means both the two participants cannot provide valid evidence to a third party to prove that the other one has exchanged the key with him/her in the protocol. This could be useful in some applications. For example, when a mobile device O gets a shared session key from a base station C with an AKE protocol, O might not want to be known that he/she was nearby the station at that time. Deniability of an AKE protocol makes sure that no matter what evidence gathered by C cannot be convincing. For example, with the protocol, the evidence can be generated on $C's$ own without exchanging with O

2.4. ElGamal Encryption. In this part, we introduce the encryption scheme used in our protocols. ElGamal encryption [20] is a probabilistic encryption scheme first proposed by ElGamal et al. ElGamal encryption consists of several components: a cyclic group $\mathbb{G}$ of prime order q with generator $g \in \mathbb{G}$, the key generator, the encryption algorithm, and the decryption algorithm.

(i) Key generator. The key pairs are computed as follows: Randomly choose x, where $sk = x \in \mathbb{Z}_q$. Then compute $pk = h = g^x$

(ii) Encryption algorithm. Given $pk = h \in \mathbb{G}$ and $m \in \mathcal{M}$, the algorithm encrypts the message m as follows: Randomly choose y, where $y \in \mathbb{Z}_q$. Compute $s = h^y$, $c_1 = g^y$, and $c_2 = m \cdot s$. Finally, the ciphertext is (c_1, c_2)

(iii) Decryption algorithm. Given $sk = x \in \mathbb{Z}_q$ and (c_1, c_2), the algorithm decrypts the ciphertext as follows: Compute $s = {c_1}^x$ and $m = c_2 \cdot s^{-1}$

2.5. Digital Signature Algorithm (DSA). In this part, we introduce the digital signature scheme used in our protocols. Actually, the digital signature algorithm (DSA) is a variant of ElGamal signature schemes. DSA is also a digital signature standard proposed by the National Institute of Standards and Technology (NIST) [21]. Following the standard, we describe a DSA digital signature as follows:

(i) Parameters. There are a group of domain parameters used by a DSA as follows: A prime modulus p has a bit length of L, where $2^{L-1} < p < 2^L$; a prime divisor q has a bit length of N, where $q|p-1$ and $2^{N-1} < q < 2^N$; a multiplicative group $GF(p)$; a subgroup $SubG(q)$ in $GF(p)$ with a order of q; a generator g of $SubG(q)$, where $1 < g < p$; a randomly choosed private key x, where $x \in \mathbb{Z}_q$; the corresponding public key y, where $y = g^x$; and a randomly chosen number k, where k is unique to each message and $k \in \mathbb{Z}_q$. The choices for the pair L and N follow the standard

(ii) Signature generation. Given private key x and message m, the algorithm generates the signature for m as follows: $r = (g^k \bmod p) \bmod q$; $z = lm(\min(N, outlen), Hash(m))$; $s = (k^{-1}(z + xr)) \bmod q$. Output the signature (r, s). Here, lm means the leftmost $\min(N, outlen)$ bits of the hash output, where $outlen$ is the bit length of the hash function output block

(iii) Signature verification. Given a triplet (m', r', s'), the algorithm verifies the signature as follows: Check that $0 < r' < q$ and $0 < s' < q$; $w = (s'^{-1} \bmod q)$; $z = lm(\min(N, outlen), Hash(m'))$; $u_1 = (zw) \bmod q$; $u_2 = ((r')w) \bmod q$; and $v = ((g^{u_1} y^{u_2}) \bmod p) \bmod q$. If $v = r'$, then the signature is verified

3. DAKEs: The Framework

3.1. Overview. As is shown in Figure 1, we give a decentralized framework for designing authenticated key exchange protocols. In this framework, we have implemented five protocols with different properties. These protocols are DAKE1 to DAKE5. There are two kinds of users in this framework, namely, the originator who starts a protocol at the beginning and the counterparty who communicates with the originator. Thus, O denotes the originator and C denotes the counterparty. The originator O is on the left side of the figure, while the counterparty C is on the right side. The direction of each line with an arrow represents the direction of the data flow. For example, the first line of DAKE1 represents that a random number r is sent from the originator O into the blockchain. And then a group of information r, pk_o, id_o is by the counterparty C from the blockchain. Before giving the details of each protocol, we will explain the whole design of the framework and the basic notations as follows:

(i) Random number. r represents a random number generated under the domain parameters

(ii) Public key. pk with a subscript denotes the public key of a user. For example, pk_o denotes the originator's public key. Especially, pk_R denotes a random public key generated by the user. Since we used the ElGamal encryption and DSS in our framework, the authenticated public key of the originator and the counterparty can also be denoted as the g^α and g^β separately. Other notations like g^ν, g^μ, g^γ, and g^σ are the exponential computation in the ElGamal encryption and DSS. We will give more explanation when describing the specific protocol

(iii) Identity. id with a subscript denotes the public key of a user. For example, id_o denotes the originator's identity

(i) Encryption algorithm. Enc with a subscript denotes an asymmetric encryption algorithm with a public key. For example, $Enc_o(m)$ denotes encrypting the message m using the originator's public key. $Enc_{pk_R}(m)$ encrypts the message m using the randomly generated public key. E with a subscript denotes a symmetric encryption algorithm using a symmetric secret key. For example, $E_k(m)$ denotes encrypting the message m using a symmetry secret key k

(iv) Digital signature algorithm. Sig with a subscript denotes a signature algorithm generated by a user. For example, $Sig_o(m)$ denotes generating a signature on the message m using the originator's secret key. And the notation δ represents a signature

(v) Others. k is the session key shared between O and C. H is a hash function

In this framework, we have implemented three smart contracts for the users in the protocols to interact with the blockchain. In the initialization stage, all users

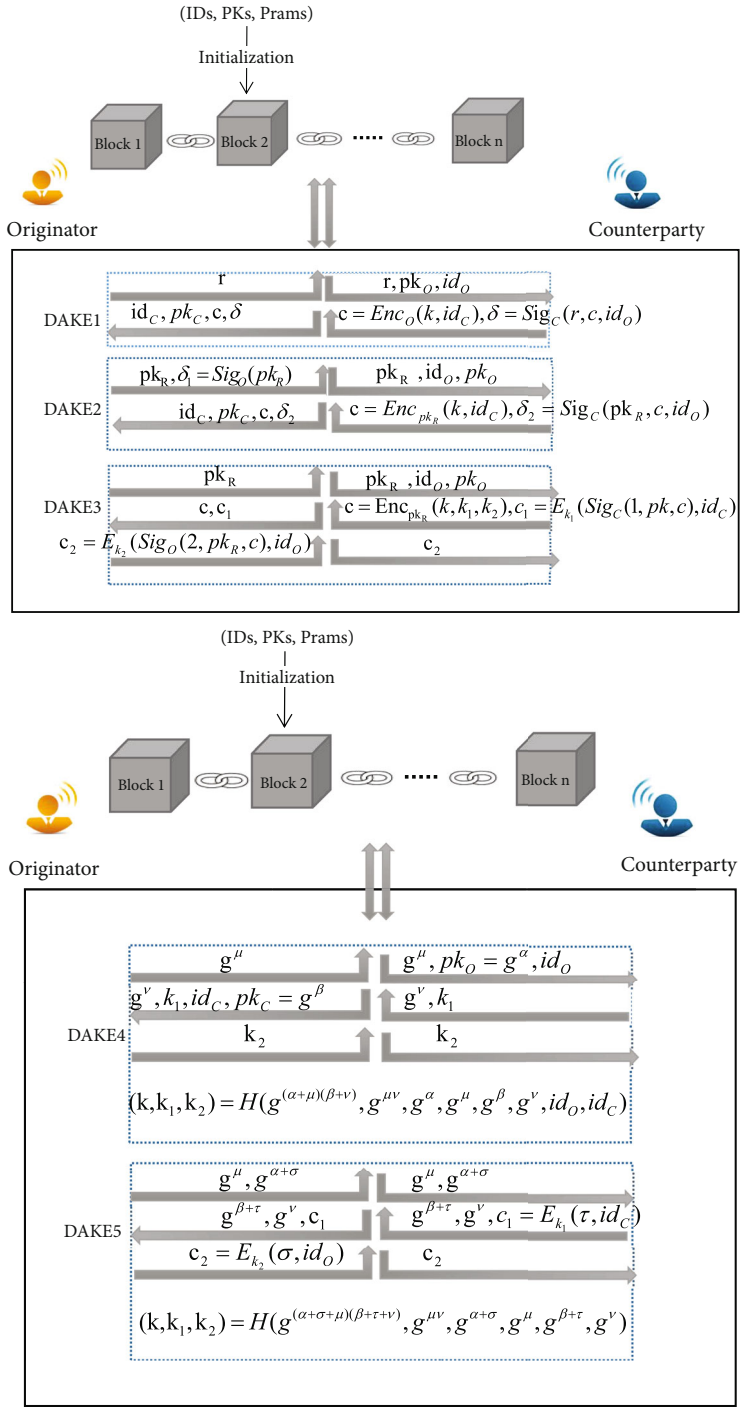

Figure 1: The framework of the proposed decentralized authenticated key exchange protocols.

submit their identification documents (e.g., passport or drivers license) and their public key for the system. They agree on the security parameters (G, g, p, q, N, L) mentioned in Sections 2.4 and 2.5. Therefore, at the beginning of each protocol, there have been $(IDs, PKs, Params)$ on the blockchain where *IDs* denotes the identities of the users, *PKs* denotes the public keys, and *Params* denotes (G, g, p, q, N, L)

3.2. Protocol Design. After the initialization, we can start to design the protocols. Note that when we say send the message to the blockchain "explicitly," anyone in the system can find out the identity of the message receiver. When we say send the message to the blockchain "implicitly," any others cannot get the identity of the message receiver. We will explain how to realize this in the Section 3.3. In this part, we focus the process of the protocols.

In DAKE1, the detailed description of the first protocol is as follows:

(1) O chooses a random number r, where $r \in \mathscr{R}$. Then, he/she sends r to the blockchain explicitly

(2) C gets the r, pk_O, id_O from the blockchain; chooses a random session key k, where $k \in \mathscr{K}$; encrypts (k, id_C) using $O's$ public key, namely $c = Enc_O(k, id_C)$; signs on (r, c, id_O), namely, $\delta = Sig_C(r, c, id_O)$; and sends (c, δ) to the blockchain explicitly. At this time, C terminates. He/she gets the session key k and his/her counterparty identity id_O

(3) O gets the c, δ, pk_C, id_C from the blockchain and then verifies the validity of δ using $C's$ public key. If δ is a valid signature on the message (r, c, id_O), O decrypts the ciphertext c. If the decryption is successful, O will gets the session key k and his/her counterparty identity id id_C

Here, the encryption $Enc_O(k, id_C)$ is used to bind the identity id_C and the session key k to the ciphertext c. To avoid a misbinding attack, the encryption algorithm should be able to resist the chosen ciphertext attack (CCA). We use the ElGamal encryption scheme (see Section 2.4) to encapsulate the session key. Let $\mathbb{G}$ be a cyclic group, q be the prime order of $\mathbb{G}$, g is a generator of $\mathbb{G}$, $IDSpace$ be the user identity space, and H be a hash function $H : \mathbb{G}^3 \times IDS \longrightarrow \mathscr{K}$. Then, we get the implementation of the key exchange protocol DAKE1 as follows:

(1) O chooses a random number r, where $r \in \mathscr{R}$. Then, he/she sends r to the blockchain explicitly

(2) C gets the r, pk_O, id_O from the blockchain, where $p k_O = g^{\alpha}$ and α denotes the secret key of O and then signs on (r, pk_C, id_O), namely, $\delta = Sig_C(r, pk_C, id_O)$ and sends δ to the blockchain explicitly. At this time, C terminates. He/she gets the session key by computing $k = H(pk_O, pk_C, {pk_O}^{\beta}, id_C)$

(3) O gets the δ, pk_C, id_C from the blockchain, where $p k_C = g^{\beta}$ and β denotes the secret key of O. Then, O verifies that δ is a valid signature; if not, O aborts; otherwise, O terminates and gets the session key by computing $k = H(pk_O, pk_C, {pk_C}^{\alpha}, id_C)$

In DAKE2, the detailed description of the second protocol is as follows:

(1) O uses a key generator algorithm to generate a new random key par (pk_R, sk_R). Then he/she signs on $p k_R$ and gets $\delta_1 = Sig_O(pk_R)$. At last, he sends (pk_R, δ_1) to the blockchain explicitly

(2) C gets the $pk_R, pk_O, \delta_1, id_O$ from the blockchain and verifies the validity of δ_1. If it is valid, C randomly choose a session key k, where $k \in \mathscr{K}$. Then, he/she encrypts (k, id_C) using the public key pk_R and gets $c = Enc_{pk_R}(k, id_C)$. Then, he/she signs on (pk_R, c, id_O) and gets $\delta_2 = Sig_C(pk_R, c, id_O)$. At last, he/she sends (c, δ_2) to the blockchain explicitly

(3) O gets the c, δ_2, pk_C, id_C from the blockchain and then verifies the validity of δ_2. If it is valid, O decrypts the ciphertext c. If the decryption is successful, O will get the session key k and the counterparty identity id id_C

Similarly, we encapsulate the key by a mechanism corresponding to the ElGamal encryption scheme. Then, we get the implementation of the key exchange protocol DAKE2 as follows:

(1) O generates a random public key $u = g^{\gamma}$; signs on u and gets $\delta_1 = Sig_O(u)$; and then sends (u, δ_1) to the blockchain explicitly

(2) C gets the u, δ_1, pk_O, id_O from the blockchain, verifies the validity of δ_1, and then generates the signatures $\delta_2 = Sig_C(u, v = g^{\sigma}, id_O)$. At last, he/she sends it to the blockchain explicitly. At this time, C gets the session key k by computing $k = H(u, v, u^{\sigma}, id_C)$

(3) O gets the δ_2, pk_C, id_C from the blockchain. Then, he/she verifies the validity of δ_2. If it is valid, O gets the session key k by computing $k = H(u, v, v^{\gamma}, id_C)$

In DAKE3, yhe detailed description of the third protocol is as follows:

(1) O uses a key generator algorithm to generate a new random key par (pk_R, sk_R) and sends pk_R to the blockchain implicitly

(2) C generates a random session key k and two other random keys k_1, k_2 for the symmetric encryption E; encrypts (k, k_1, k_2) and gets $c = E_{pk_R}(k, k_1, k_2)$; signs on $(1, pk_R, c)$ and gets $\delta_1 = Sig_C(1, pk_R, c)$; encrypts (δ_1, id_C) and gets $c_1 = E_{k_1}(\delta_1, id_C)$; and at last, sends (c, c_1) to the blockchain implicitly

(3) O gets the c, c_1 from the blockchain, decrypts c using the key sk_R, and gets k, k_1, k_2. Then, O decrypts c_1 using k_1 and gets δ_1, id_C. Then, O gets pk_C corresponding to id_C from the blockchain and verifies the validity of δ_1. If it is valid, O computes $\delta_2 = Si g_O(2, pk_R, c)$, $c_2 = E_{k_2}(\delta_2, id_O)$ and sends c_2 to the blockchain implicitly

(4) C decrypts c_2 using the key k_2 and gets δ_2 and id_O; C gets pk_O corresponding to id_O from the blockchain and verifies the validity of δ_2

As we did for protocols DAKE1 and DAKE2, we can implement protocol AKE3 using ElGamal encryption as follows:

(1) O generates a random public key $u = g^{\gamma}$ and sends u to the blockchain implicitly

(2) C gets the u from the blockchain and generates $v = g^{\sigma}$. C computes $(k, k_1, k_2) = H(g^{\gamma}, g^{\sigma}, g^{\gamma\sigma})$, $\delta_1 = Sig_C(1, u, v)$, and $c_1 = E_{k_1}(\delta_1, id_C)$ and sends v, c_1 to the blockchain implicitly

(3) O gets the v, c_1 from the blockchain and computes $(k, k_1, k_2) = H(g^{\gamma}, g^{\sigma}, g^{\gamma\sigma})$. O decrypts c_1 using the key k_1 and gets δ_1, id_C. Then, O gets pk_C, id_C from the blockchain and verifies the validity of δ_1. If it is valid, O computes $\delta_2 = Sig_O(2, u, v)$, $c_2 = E_{k_2}(\delta_2, id_O)$ and sends c_2 to the blockchain

(4) C decrypts c_2 using the key k_2 and gets δ_2 and id_O; C gets pk_O corresponding to id_O from the blockchain and verifies the validity of δ_2

In DAKE4, the detailed description of the forth protocol is as follows:

(1) O generates a random public key g^{μ} and sends it to the blockchain explicitly

(2) C gets the $g^{\mu}, pk_O = g^{\alpha}, id_O$ from the blockchain and generates g^{ν}. C computes $(k, k_1, k_2) = H((g^{\alpha}g^{\mu})^{\beta+\nu}, (g^{\mu})^{\nu}, g^{\alpha}, g^{\mu}, g^{\beta}, g^{\nu}, id_O, id_C)$, where g^{β} is C's public key, and sends g^{ν}, k_1 to the blockchain explicitly

(3) O gets the $g^{\nu}, k_1, pk_C = g^{\beta}, id_C$ from the blockchain and computes $(k, k_1, k_2) = H((g^{\beta}g^{\nu})^{\alpha+\mu}, (g^{\nu})^{\mu}, g^{\alpha}, g^{\mu}, g^{\beta}, g^{\nu}, id_O, id_C)$; then, O compares its computed value of k_1 to the value it received from the blockchain; if these match, O sends k_2 to the blockchain explicitly

(4) C gets k_2 from the blockchain and compares its computed value of k_2 to it

In DAKE5, the detailed description of the fifth protocol is as follows:

(1) O generates random public keys g^{σ} and g^{μ}; O computes $\alpha' = \alpha + \sigma$, where g^{α} is O's initial public key on the blockchain. Then, O sends $g^{\alpha'}, g^{\mu}$ to the blockchain implicitly

(2) C gets the $g^{\alpha'}, g^{\mu}$ from the blockchain and generates (g^{τ}, g^{ν}); C computes $\beta' = \beta + \tau$ where g^{β} is C's initial public key on the blockchain. Then, C computes $(k, k_1, k_2) = H((g^{\alpha'}g^{\mu})^{(\beta'+\nu)}, g^{\mu\nu}, g^{\alpha'}, g^{\mu}, g^{\beta'}, g^{\nu})$ and $c_1 \longleftarrow^{R} E_{k_1}(\tau, id_C)$ and sends $g^{\beta'}, g^{\nu}, c_1$ to the blockchain implicitly

(3) O gets the $g^{\beta'}, g^{\nu}, c_1$ from the blockchain and computes $(k, k_1, k_2) = H((g^{\beta'}g^{\nu})^{(\alpha'+\mu)}, g^{\nu\mu}, g^{\alpha'}, g^{\mu}, g^{\beta'}, g^{\nu})$, $c_2 \longleftarrow^{R} E_{k_2}(\sigma, id_O)$. O decrypts c_1 using the key k_1 and gets (τ, id_C). Then, O gets pk_C corresponding to id_C from the blockchain and verifies $g^{\beta} \cdot g^{\tau} = g^{\beta'}$; if it is valid, O sends c_2 to the blockchain implicitly

(4) C gets the c_2 from the blockchain decrypts it using the key k_2; C obtains (σ, id_O) and then gets the corresponding pk_O from the blockchain. C verifies $g^{\alpha} \cdot g^{\sigma} = g^{\alpha'}$

3.3. Smart Contracts. We have given three smart contracts noted as contracts 1, 2, and 3. All the protocols should use the contracts to interact with the blockchain. Specifically, contract 1 provides a way to send the message to the blockchain "explicitly" for DAKE1, DAKE2, and DAKE4. Contract 2 provides a way to send the message to the blockchain "implicitly" for DAEK3. Contract 3 provides a way to send the message to the blockchain "implicitly" for DAEK5. Here, "explicitly" means that anyone in the system can find out the identity of the message receiver, while "implicitly" means that any others cannot get the identity of the message receiver. Now, let us see how to achieve this.

As is shown in Table 1, the three smart contracts have common parts in their user data structures. For a user in the system, there is a public key "PK," an identity string "ID," an Ethereum account address "user," and a "timestamp" which is used to indicate whether the public key is expired. These public parameters are set up in the initialization stage and cannot be modified later. The data type "mapping(address ⇒ string)" makes contract 1 different from the other two contracts. We have implemented "set" and "get" interfaces in these contracts separately. For simplicity, we use the pseudocode shown in Algorithms 2 and 3 to give the overall design. An originator can set a communicate message for his/her counterparty by using Algorithm 2. And the counterparty can read the message set by his/her originator by using Algorithm 3.

For contract 1, the inputs of Algorithm 2 are $(iO, iC, r, version)$, where iO, iC are the public account addresses of the originator and the counterparty separately. r is the data to be exchanged, for example, a random number in DAKE1. Lines 1 to 4 will be executed for contract 1. In Algorithm 3, lines 1 to 4 will be executed for contracts 1 and line 2 to make sure that only the designated counterparty can read the message. Since the data on the blockchain is thought to be public, everyone will find out the identity of the counterparty in the contract 1. Therefore, we say that contract 1 provides a way to send the message to the blockchain "explicitly."

Contrarily, for contracts 2 and 3, Algorithm 2 does not designate the account address of the counterparty. The difference between 2 and 3 is that 3 has two other message to record in the blockchain, namely, $token1$ and $token2$. $c, token1$, and $tonken2$ represent the data to be exchanged. For example, in DAKE5 for O, $token1$ is $g^{\alpha'}$, $token2$ is g^{μ}, and c is c_2. And for C, $token1$ is $g^{\beta'}$, $token2$ is g^{ν}, and c is c_1. This will make sense when we take a look at the detail of the protocols. And since we set the message without pointing out the counterparty address, we say that they provide a way

Table 1: The user data structure of the smart contracts.

Contract name	Data type	Name
Contract I	address	user
	string	PK
	string	ID
	uint	timestamp
	mapping(address ⇒ string)	r
	mapping(address ⇒ string)	sig
Contract II	address	user
	string	PK
	string	ID
	uint	timestamp
	string	c
Contract III	address	user
	string	PK
	string	ID
	uint	timestamp
	string	token1
	string	token2
	string	c

Input (*iO*, *iC*, *r*, *c*, *token*1, *token*2, *version*, *type*)
1: require(users[iP].user == msg.sender);
2: **if** version ==1 **then**
3: users[iO].r[iC] = r; /* Contract i */
4: **end if**
5: **if** version ==2 **then**
6: users[iO].c = c; /* Contract ii */
7: **end if**
8: **if** version ==3 **then**
9: **if** 100 == type **then**
10: users[iO].token1 = token1; /* Contract iii */
11: **end if**
12: **if** 200 == type **then**
13: users[iO].token2 = token2; /* Contract iii */
14: **end if**
15: **if** 300 == type **then**
16: users[iO].c = c; /* Contract iii */
17: **end if**
18:**end if**

Algorithm 1: set

to send the message to the blockchain "implicitly" in contracts 2 and 3.

4. Soundness and Security Property Analysis

4.1. Soundness. The soundness of each protocol is proven if the originator *O* and the counterparty *C* obtain an identical session key *k* at the end of the protocol. In this sense, the soundness of DAKE3, DAKE4, and DAKE5 are quite straight since *O* and *C* use the same input of a hash function to compute

Input (*iO*, *iC*, *type*)
1: **if** version ==1 **then**
2: require(users[iC].user == msg.sender);
3: **return** users[iP].r[iC];
4: **end if**
5: **if** version ==2 **then**
6: **return** users[iO].c;
7: **end if**
8: **if** version ==3 **then**
9: **if** 100 == type **then**
10 **return** users[iO].token1;
11: **end if**
12: **if** 200 == type **then**
13: **return** users[iO].token2;
14: **end if**
15: **if** 300 == type **then**
16: **return** users[iO].c;
17: **end if**
18:**end if**

Algorithm 2: get

their session keys. In the implementation of DAKE1, *O* will get a session key $k = H(pk_O, pk_C, {pk_C}^{\alpha}, id_C)$, and *C* will get $k = H(pk_O, pk_C, {pk_O}^{\beta}, id_C)$. Since ${pk_C}^{\alpha} = g^{\beta\alpha} = g^{\alpha\beta} = {pk_O}^{\beta}$, *O* and *C* will hold an identical session key at last. In the implementation of DAKE2, *O* will get a session key $k = H(u, v, v^{\gamma}, id_C)$, and *C* will get $k = H(u, v, u^{\sigma}, id_C)$. Since $v^{\gamma} = g^{\sigma\gamma} = g^{\gamma\sigma} = u^{\sigma}$, *O* and *C* will hold an identical session key at last.

4.2. Security Property. Now, we can compare their security properties shown in Table 2 with the other decentralized protocols. SmartDHX [12] only has a security property of Static secrecy since it is actually a Diffie-Hellman key exchange based on blockchain. Apparently, it should assume an honest participant's long-term secret key can never be compromised by the adversary. Besides, smartDHX does not provide any way for the participants to authenticate the identity of their counterparties. Thus, it cannot obtain other security properties.

In Bitcoin-based AKE [4], the authors provide two protocols over Bitcoin. One of them called Diffie-Hellman-over-Bitcoin is a non-interactive protocol without perfect forward secrecy, while another one called YAK-over-Bitcoin is an interactive protocol with perfect forward secrecy. Bitcoin-based AKE uses a Schnorr zero knowledge proof algorithm to provide identity protection and authenticity. However, the deniability is not taken into account.

Protocol DAKE1 only provides static secrecy and implicit authenticity since we are assuming that the long-term keys (initial public key pairs) are never compromised. When *O* finishes the protocol, he can be confident that *C* was online since *C* must have signed the message containing *O*'s random number. However, when *C* finishes the protocol, he has no such guarantee. Protocol DAKE2 provides perfect forward secrecy since a new key pair for the encryption scheme is generated with each run of the protocol. And user long-term keys are used only for signing, not encrypting. So the adversary cannot

Table 2: Security property comparison.

Protocol	Static secrecy	Perfect forward secrecy	Implicit authenticity	Explicit authenticity	Identity protection	Deniability
smartDHX [12]	√					
Bitcoin-based AKE [14]	√	√	√	√		
DAKE1	√		√			
DAKE2	√	√	√			
DAKE3	√	√	√	√	√	
DAKE4	√	√	√	√		√
DAKE5	√	√	√	√	√	√

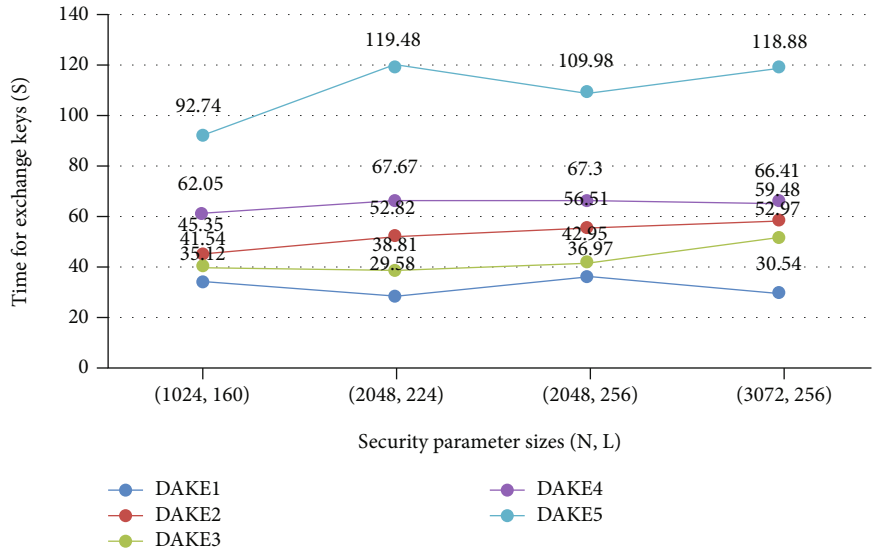

Figure 2: The overall time for key exchange.

Table 3: Number of interactions with blockchain.

Protocol	Originator	Counterparty	Sum
DAKE1	$2q + 1t$	$2q + 1t$	$4q + 2t$
DAKE2	$2q + 2t$	$3q + 1t$	$5q + 3t$
DAKE3	$3q + 1t$	$2q + 1t$	$5q + 2t$
DAKE4	$3q + 2t$	$3q + 2t$	$6q + 4t$
DAKE5	$4q + 3t$	$4q + 3t$	$8q + 6t$

decrypt any messages by compromising the signing key. DAKE3 provides identity protection and explicit authenticity. If the adversary is not one of the participants, he/she cannot learn the identities of both the participants since they are encrypted in the communication. Although each participant will eventually learn the identity of the other, O can withhold his/her identity until confirming the identity of C. Thus, we say that DAKE3 has the explicit authenticity property. Note that we do not consider that DAKE3 provides deniability, since C and O have signed on the message. Even though all messages are encrypted, one of the participants can still collect all the data to prove the existence of another one. Since both the participants sign nothing, DAKE4 provides deniability. DAKE5 has an extra property of Deniability. DAKE4 sends g^{α} in the clear, while in DAKE5, O sends $g^{\alpha'}$ for blinding the value. And the exponent σ along with id_O are encrypted. C can verify the blinding value and use a symmetric method.

5. Implementation and Performance Evaluation

5.1. Implementation. In our implementation, we use smart contracts written in solidity to store and query the public data in the blockchain. The other functions are implemented as Web APIs written in JavaScript. There are several JavaScript open libraries we used. "ethers.js" is used to interact with the Ethereum blockchain. To provide the necessary cryptographic functions, we also imported two other libraries "Crypto.js" and "jsbn.js." Our test environment is built by a professional tool called HardHat. You can compile, deploy, test, and debug your Ethereum software with HardHat. We

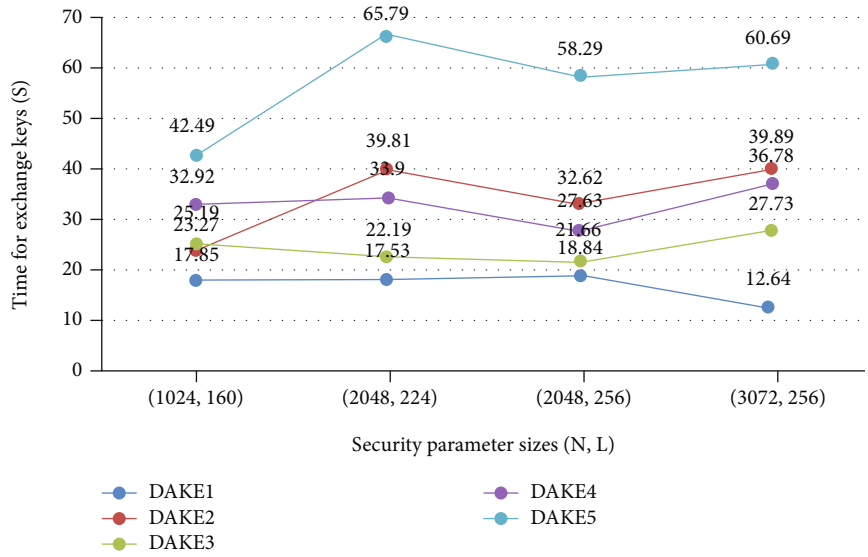

Figure 3: Time for the originator to exchange keys.

run our test cases in Hardhat local network and Rinkeby test network. Once weâ€™re ready to share the decentralized services with other people, what we may want to do is deploy it to a live network. To see the performance of our protocols, we provide the tests on a live network. Rinkeby is the live network we chose. It is a testnet of Ethereum based on proof of authority (PoA) [22]. We used a PC with the OS of Ubuntu Desktop 18.04 64× as a client PC to run the tests. The CPU of this computer is dual-core with intel(R) Core(TM) i7-10510 U CPU @ 1.80GHz 2.30GHz on each. The memory is 4G.

5.2. Efficiency Analysis. In order to give a comprehensive estimation for our authenticated key exchange framework, we first gave the overall performance of each protocol and then provide the test results of the originator and counterparty separately. The reason is that in some applications, the computation and network capability are quite different for the users of the key exchange protocols. For instance, in a payment application, the bank as a server has a more large capability than the clients of the application. In this kind of application, the detailed analysis can help you to choose a proper protocol in our framework. The originator as a client can get a similar efficiency as we showed in the test, while the counterparty as a server may obtain a more efficient performance since he/she can increase his/her computation and network resource.

As is shown in Figure 2, the overall performance of each protocol is quite stable when the size of the security parameters increases from 1024 bits to 3072 bits. According to Table 2, the security properties of the protocols from DAKE1 to DAKE5 increase one by one. Intuitively, a protocol should cost more exchange time for obtaining more properties as well. However, we can find out that there is an exception where protocol DAKE2 has a better performance than DAKE3 in general. To figure out this, we then listed the number of interactions with Ethereum blockchain for each protocol. As shown in Table 3, the letter "q" represents an operation to query data from the blockchain, and the letter "t" means sending a transaction to the blockchain to write data on it. Thus, the "t" operation will take more time than the "q" operation. Although protocol DAKE3 has an extra cryptographic encryption function than protocol DAKE2, it needs fewer transactions. Besides, we can infer that the time to perform a cryptographic function locally is quite less than to execute a transaction on the blockchain.

Furthermore, when we see the time for the originator and the counterparty separately, we find out that more than one protocol having fewer security properties can obtain quite better performance for one of the participants. The results are given in Figures 3 and 4. Except for the protocol DAKE5, all protocols have obviously distinct costs between the originator and the counterparty. Thus, when the users of the application play different roles and have distinct computation resources, one can choose a more proper protocol according to the performance of the originator and the counterparty. Table 3 can also be used to explain this result.

5.3. Gas Cost Analysis. Like the efficiency analysis, we first gave the overall gas cost of each protocol and then provide the gas cost for the originator and counterparty separately. The gas cost includes deploying a smart contract on the Ethereum blockchain and sending transactions to it. Note that it is costless to query data from the blockchain, namely, the "q" operation is gas-free.

As is shown in Table 4, the gas costs for deployment in protocols DAKE1, DAKE2, and DAKE4 are equal since they use the same smart contract. When the size of the security parameters increases for each protocol, the gas cost for exchange will go higher. It is easy to understand, for a large parameter size means a longer string to exchange. Then, we can take a look at different protocols with identical parameter

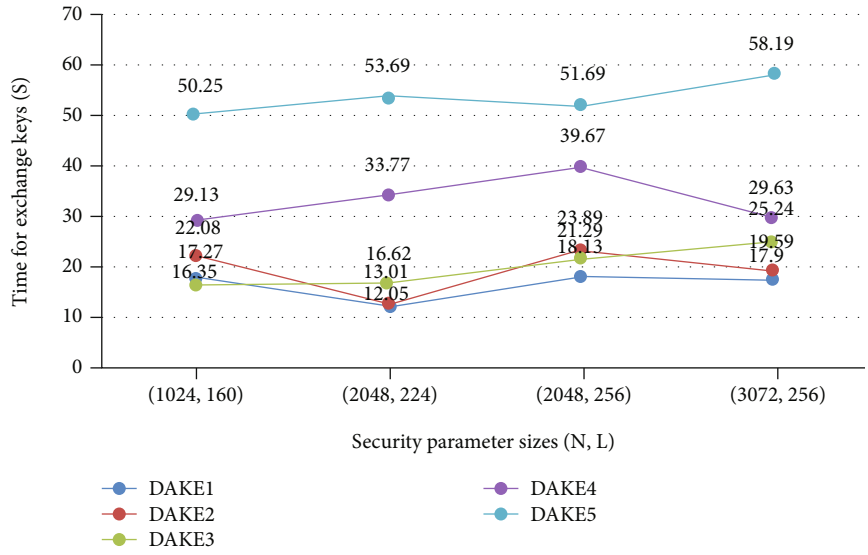

Figure 4: Time for the counterparty to exchange keys.

Table 4: The overall gas cost comparison.

Protocol	Deployment	Exchange	Sum
DAKE1 (1024, 160)	1,254,118	208,154	1,462,272
DAKE1 (2048, 224)	1,254,118	230,836	1,484,954
DAKE1 (2048, 256)	1,254,118	253,326	1,507,444
DAKE1 (3072, 256)	1,254,118	253,326	1,507,444
DAKE2 (1024, 160)	1,254,118	459,585	1,713,703
DAKE2 (2048, 224)	1,254,118	686,405	1,940,523
DAKE2 (2048, 256)	1,254,118	731,385	1,985,503
DAKE2 (3072, 256)	1,254,118	912,841	2,166,959
DAKE3 (1024, 160)	926,738	2,316,224	3,242,962
DAKE3 (2048, 224)	926,738	4,493,720	5,420,458
DAKE3 (2048, 256)	926,738	4,493,696	5,420,434
DAKE3 (3072, 256)	926,738	6,671,180	7,597,918
DAKE4 (1024, 160)	1,254,118	643,464	1,897,582
DAKE4 (2048, 224)	1,254,118	1,006,376	2,260,494
DAKE4 (2048, 256)	1,254,118	1,006,376	2,260,494
DAKE4 (3072, 256)	1,254,118	1,369,288	2,623,406
DAKE5 (1024, 160)	1,078,513	2,505,604	3,584,117
DAKE5 (2048, 224)	1,078,513	3,862,800	4,941,313
DAKE5 (2048, 256)	1,078,513	4,683,076	5,761,589
DAKE5 (3072, 256)	1,078,513	6,860,536	7,939,049

sizes. We can find out the two highest gas cost protocols are DAKE3 and DAKE5. To figure out this, we can review the protocols in Section 3. The difference between these two protocols with others is that they both need an encryption function which makes their exchange data get larger than others.

As is shown in Table 5, the gas cost for the originator and the counterparty of the same protocol in DAKE3, DAKE4, and DAKE5 are equal. Note that the situation in gas cost seems a slice different from that in the efficiency analysis where most protocols have obviously distinct time costs between the originator and the counterparty. It is because both the computation and blockchain network capability will affect the efficiency and the time for completing the protocol may be a little fluctuated. However, the gas cost for executing a transaction is only related to the size of the transaction and is quite stable.

Table 5: Gas cost for the originator and the counterparty.

Protocol	Originator	Counterparty
DAKE1 (1024, 160)	92,815	115,339
DAKE1 (2048, 224)	92,815	138,021
DAKE1 (2048, 256)	92,815	160,511
DAKE1 (3072, 256)	92,815	160,511
DAKE2 (1024, 160)	344,246	115,339
DAKE2 (2048, 224)	548,384	138,021
DAKE2 (2048, 256)	570,874	160,511
DAKE2 (3072, 256)	752,330	160,511
DAKE3 (1024, 160)	1,158,112	1,158,112
DAKE3 (2048, 224)	2,246,860	2,246,860
DAKE3 (2048, 256)	2,246,848	2,246,848
DAKE3 (3072, 256)	3,335,596	3,335,584
DAKE4 (1024, 160)	321,732	321,732
DAKE4 (2048, 224)	503,188	503,188
DAKE4 (2048, 256)	503,188	503,188
DAKE4 (3072, 256)	684,644	684,644
DAKE5 (1024, 160)	1,252,808	1,252,796
DAKE5 (2048, 224)	1,521,256	2,341,544
DAKE5 (2048, 256)	2,341,532	2,341,544
DAKE5 (3072, 256)	3,430,280	3,430,256

6. Conclusion

In this paper, we propose a novel decentralized authenticated key exchange framework via blockchain. In this framework, we implement five protocols with different security properties. To the best of our knowledge, our framework is the first one to take different kinds of security properties into account and construct a decentralized authenticated key exchange protocol framework which can be used in different applications in the smart city. To resist the potential threat, we combine the CCA-secure ElGamal encryption algorithm and the digital signature algorithm (DSA) in some of the protocols. Compared with other decentralized key exchange protocols, our protocols enjoy more security properties and get rid of the problem caused by the availability of Bitcoin-like blockchain. Finally, we conduct our test cases in the live testnet of Ethereum and give a comprehensive analysis. The results can help someone who wants to use the proposed framework to choose which protocol is suitable for his application.

Since the decentralization of the proposed protocol framework is based on the blockchain technology, a natural progression of this work is to improve the framework as the blockchain technology evolves. For example, an improved blockchain technology [23] [24] may provide specialized features for IoT to design a more efficient protocol. A further study could assess the compatibility of the protocols in the applications of the smart city. More information about the compatibility of the protocols in the applications of the smart city would help us to establish a greater degree of availability on the protocols.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was partially supported by the National Key Research and Development Program of China (Grant No. 2019YFE0118700) and Science and Technology Project of China Southern Power Grid Corporation (Grant No. 080036KK52190007).

References

[1] R. C. Merkle, "Secure communications over insecure channels," *Communications of the ACM*, vol. 21, no. 4, pp. 294–299, 1978.

[2] M. Rana, Q. Mamun, and R. Islam, "Current lightweight cryptography protocols in smart city iot networks: a survey," 2020, http://arxiv.org/abs/2010.00852.

[3] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Workshop on the theory and application of cryptographic techniques*, pp. 47–53, Springer, 1985.

[4] P. McCorry, S. F. Shahandashti, D. Clarke, and F. Hao, "Authenticated key exchange over bitcoin," in *International Conference on Research in Security Standardisation*, pp. 3–20, 2015.

[5] *Bitcoin: A peer-to-peer electronic cash system*, Social Science Electronic Publishing, 2008.

[6] M. Yavari, M. Safkhani, S. Kumari, S. Kumar, and C.-M. Chen, "An improved blockchain-based authentication protocol for IoT network management," *Security and Communication Networks*, vol. 2020, Article ID 8836214, 16 pages, 2020.

[7] J. Yin, Y. Xiao, Q. Pei et al., "SmartDID: a novel privacy-preserving identity based on blockchain for iot," *IEEE Internet of Things Journal*, 2022.

[8] Q. Mei, H. Xiong, Y.-C. Chen, and C.-M. Chen, "Blockchain-enabled privacy-preserving authentication mechanism for transportation cps with cloud-edge computing," *IEEE Transactions on Engineering Management*, pp. 1–12, 2022.

[9] F. Tschorsch and B. Scheuermann, "Bitcoin and beyond: a technical survey on decentralized digital currencies," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2084–2123, 2016.

[10] Y. Hu, Y. Xiong, W. Huang, and X. Bao, "Keychain: blockchain-based key distribution," in *2018 4th International Conference on Big Data Computing and Communications (BIGCOM)*, pp. 126–131, 2018.

[11] V. Buterin, "Ethereum: a next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, 2014.

[12] R. Muth and F. Tschorsch, "Smartdhx: Diffie-hellman key exchange with smart contracts," in *2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pp. 164–168, Oxford, UK, 2020.

[13] J. Chen, W. Gan, M. Hu, and C.-M. Chen, "On the construction of a post-quantum blockchain for smart city," *Journal of Information Security and Applications*, vol. 58, article 102780, 2021.

[14] B. LaMacchia, K. Lauter, and A. Mityagin, "Stronger security of authenticated key exchange," in *International conference on provable security*, pp. 1–16, Berlin, Heidelberg, 2007.

[15] F. Callegati, W. Cerroni, and M. Ramilli, "Man-in-the-middle attack to the https protocol," *IEEE Security & Privacy*, vol. 7, no. 1, pp. 78–81, 2009.

[16] F. Miao, M. Pajic, and G. J. Pappas, "Stochastic game approach for replay attack detection," in *52nd IEEE conference on decision and control*, pp. 1854–1859, Firenze, Italy, 2013.

[17] H. Krawczyk, "SIGMA: the 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE protocols," in *Annual international cryptology conference*, pp. 400–425, Berlin, Heidelberg, 2003.

[18] Y. Tsiounis and M. Yung, "On the security of Elgamal based encryption," in *International Workshop on Public Key Cryptography*, pp. 117–134, Springer, 1998.

[19] D. Boneh and V. Shoup, "A graduate course in applied cryptography," *Draft 0.5*, 2020, https://crypto.stanford.edu/~dabo/cryptobook/BonehShoup_0_4.pdf.

[20] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[21] C. F. Kerry and C. R. Director, *Fips Pub 186-4 Federal Information Processing Standards Publication Digital Signature Standard (Dss)*, Citeseer, 2013.

[22] O. Hasan, L. Brunie, and E. Bertino, "Privacy-preserving reputation systems based on blockchain and other cryptographic building blocks: a survey," *ACM Computing Surveys (CSUR)*, vol. 55, no. 2, pp. 1–37, 2023.

[23] E. K. Wang, R. Sun, C.-M. Chen, Z. Liang, S. Kumari, and M. K. Khan, "Proof of X-repute blockchain consensus protocol for IoT systems," *Computers & Security*, vol. 95, article 101871, 2020.

[24] Q. Pei, E. Zhou, Y. Xiao, D. Zhang, and D. Zhao, "An efficient query scheme for hybrid storage blockchains based on Merkle semantic trie," in *2020 International symposium on reliable distributed systems (SRDS)*, pp. 51–60, Shanghai, China, 2020.