Bigdata-facilitated Two-party Authenticated Key Exchange for IoT (full paper)

Bowen Liu[1], Qiang Tang[1] and Jianying Zhou[2]

[1]Luxembourg Institute of Science and Technology (LIST)
5 Avenue des Hauts-Fourneaux, Esch-sur-Alzette, L-4362, Luxembourg
{bowen.liu, qiang.tang}@list.lu
[2]Singapore University of Technology and Design
8 Somapah Rd, Singapore 487372
jianying_zhou@sutd.edu.sg

Abstract. Authenticated Key Exchange (AKE) protocols, by definition, guarantee both session key secrecy and entity authentication. Informally, session key secrecy means that only the legitimate parties learn the established key and mutual authentication means that one party can assure itself the session key is actually established with the other party. Today, an important application area for AKE is Internet of Things (IoT) systems, where an IoT device runs the protocol to establish a session key with a remote server. In this paper, we identify two additional security requirements for IoT-oriented AKE, namely Key Compromise Impersonation (KCI) resilience and Server Compromise Impersonation (SCI) resilience. These properties provide an additional layer of security when the IoT device and the server get compromised respectively. Inspired by Chan et al.'s bigdata-based unilateral authentication protocol, we propose a novel AKE protocol which achieves mutual authentication, session key secrecy (including perfect forward secrecy), and the above two resilience properties. To demonstrate its practicality, we implement our protocol and show that one execution costs about 15.19 ms (or, 84.73 ms) for the IoT device and 2.44 ms (or, 12.51 ms) for the server for security parameter $\lambda = 128$ (or, $\lambda = 256$). We finally propose an enhanced protocol to reduce the computational complexity on the end of IoT by outsourcing an exponentiation computation to the server. By instantiating the signature scheme with NIST's round three alternate candidate *Picnic*, we show that one protocol execution costs about 14.44 ms (or, 58.45 ms) for the IoT device and 12.78 ms (or, 46.34 ms) for the server for security parameter $\lambda = 128$ (or, $\lambda = 256$).

Keywords: Internet of Things · Authenticated Key Exchange · Perfect Forward Secrecy · Key Compromise Impersonation Resilience · Server Compromise Impersonation Resilience

1 Introduction

Two-party key exchange is a fundamental cryptographic primitive that enables two parties to establish secure communication channels over an open network. Furthermore, an authenticated key exchange protocol not only allows two parties to negotiate a session key but also ensures the authenticity of the involved parties [5]. The basic security property of an AKE protocol is that only the legitimate parties can gain access to the established secret key in every protocol execution (i.e. a session). In addition, some AKE protocols guarantee perfect forward secrecy which preserves the session key secrecy even if the long-term credentials of both parties are compromised. Regarding their construction, most existing AKE protocols employ only a single type of authentication factor (e.g. long-term secret keys or digital certificates). Consequently, if the single authentication factor gets compromised, then the AKE protocol's security will be broken.

In this paper, we are interested in two-party AKE protocols for IoT systems, where an IoT device and a server run the protocol to authenticate each other and establish a session key. We generally assume that the IoT device is standalone and no human user is necessarily present when it is engaged in the protocol execution. Note that we do recognize some exceptional scenarios, e.g. when the IoT device is a smart phone, where a human user is able to involve in the AKE protocol. It is well known that IoT devices are constrained with respect to computation capability, network bandwidth, and battery life. This advocates lightweight AKE designs. Regarding security, we would like to emphasize two observations.

- An IoT device is very likely to be compromised and has the stored credentials leaked, e.g. via side-channel attacks. This motivates us to consider key compromise impersonation resilience (see the definition below) to be a valuable property for IoT-oriented AKE protocols.
- Even less likely, the server could also be compromised. Taking into account the fact that the IoT device can be deployed in a critical infrastructure, it is ideal that an attacker should not be able to impersonate the server to the IoT device even if it has compromised the server.

It is worth noting that we assume the attacker only learns some credentials by compromising an entity. Other types of damage (e.g. installing a trapdoor or disabling the entity) are not directly related to AKE and are beyond the scope of our paper.

As a result, our objective is to construct AKE protocols with the following properties in addition to the standard session key security property.

1. *(Prefect) Forward Secrecy. Forward secrecy* means that if one party's long-term key is compromised, the secrecy of its previous session keys should not be affected. *Perfect forward secrecy* (PFS) requires that previous session keys remain secure even if both parties' long-term keys have been compromised.
2. *Key Compromise Impersonation (KCI) resilience.* Even if an attacker has obtained one party's long-term private key, then it still cannot impersonate the other party to this party.

3. *Server Compromise Impersonation (SCI) resilience*. Even if an attacker has compromised the server, then it still cannot impersonate the server to the IoT device.

Our analysis shows that it is very challenging for single-factor-based AKE protocols to achieve all these properties. Hence, in this paper, we will focus on AKE protocols, where the entity authentication is based on two or more factors.

1.1 Related Work

We emphasize that two-party authenticated key exchange is a fruitful research area, with many existing protocols, implementations and standards. For the sake of space, we refer the readers to survey papers/books like [6] for a detailed overview. Regarding Two-Factor Authenticated Key Exchange (2FAKE), Lee et al. [26] proposed a protocol which combines a smart card and a password as authentication factors. Byun [10] proposed a 2FAKE protocol by using a shared common secret and Physical Unclonable Functions (PUFs) as authentication factors. Guo and Chang [21] proposed a chaotic maps-based AKE protocol with password and smart cards as additional authentication factors. Later, Liu and Xue [28] proposed a chaos-based AKE protocol using password as the other authentication factor. Challa et al. [13] proposed an AKE protocol using password, biometric information and a smart card as authentication factors. In 2008, Pointcheval and Zimmer [31] proposed the first Multi-Facor Authenticated Key Exchange (MFAKE) protocol which combines a password, a secret key, and biometric information as the authentication factors. Later, Hao and Clarke [22] pointed out that an adversary can break the protocol by only compromising a single password factor based on the deficiency of its security model (i.e. server impersonation has not been considered). Byun [8, 9, 11] proposed MFAKE protocols by using PUF, biometric template and long-term secret keys as authentication factors. Li et al. [27] proposed a MFAKE protocol by using password, biometric fingerprint and personal identification number (PIN) as authentication factors. Stebila [34] proposed an MFAKE scheme, where multiple short secrets (e.g. one-time response) are used in addition to a password. Besides, Fleischhacker et al. [20] proposed a modular MFAKE framework by combining any subset of multiple low-entropy (one-time) passwords/PINs, high-entropy private/public keys, and biometric factors.

Since we assume there is no human involvement, PIN, password and biometric factors do not fit into our setting. In addition, PUF-based AKE protocols require special type of IoT devices, e.g. it should have PUF embedded. In order to design general purpose two-factor or multi-factor AKE protocols, we need to find other authentication factors. Regarding entity authentication, authentication factors can be classified into three categories: *something you know*, *something you have*, and *something you are* [17]. In addition, Brainard et al. proposed a fourth category: *some one you know* which is the social networking information-based authentication factor [7]. Among all, the *something you have* category fits into our IoT setting, and bigdata could be a candidate of good authentication factor. To this end, Chan et al. [14] proposed a bigdata-based unilateral two-factor authentication protocol. In more detail, their protocol uses all available historical data and relevant tags as an authentication factor, where the tags are generated injectively based on the historical data, in addition to the conventional first authentication factor of the shared long-term key. It is shown that, in a bounded storage model [3, 19], the protocol remains secure since the adversary can only capture limited records of the large amount of full historical data.

1.2 Our Contribution

In this paper, our contribution is multifold. Firstly, inspired by Chan et al.'s work [14], we introduce a new IoT-oriented AKE setting, where bigdata is used as an additional factor in addition to a shared long-term private key for facilitating mutual authentication. We propose a security model to capture all the desired security properties listed in the previous subsection. To our knowledge, no existing AKE protocol achieves all these properties, as shown in Table 1.

Table 1. Comparison among different AKE Protocols

Protocol	Authentication Factor	Security Property						Comm. Pass
		P1	P2	P3	P4	P5	P6	
[31]	Biometic, Password and Secret Key	✗	✓	✓	-	-	-	4
[27]	Biometic, Password and PIN	✗	✓	✓	✓	-	-	4
[26]	Password and Secret Key	✓	✓	✓	✗	-	-	2
[34]	Password and Customized Elements	✓	✓	✓	-	-	-	3
[13]	Biometic, Password and Smart Card	✓	✓	✓	-	-	-	3
[8–11, 28]	PUF/Chaos and Others	✓	✓	✓	✓	-	-	3
[20]	Multiple Customized Elements	✓	✓	✓	-	-	-	2*No. of Factors
Ours	Bigdata and Secret Key	✓	✓	✓	✓	✓	✓	3

'P1': Mutual authentication, 'P2': Session key security, 'P3': Forward secrecy, 'P4': Prefect forward secrecy, 'P5': Key compromise impersonation resilience, 'P6': Server compromise impersonation resilience, '✓': Provides the security property, '✗': Does not prevent the attack, '-': The security property has not been considered.

Secondly, we propose a novel AKE protocol, which uses both long-term private keys and bigdata as its authentication factors. Regarding the processing of bigdata, we distribute the relevant credentials to both the IoT and the server. As a result, we avoid one vulnerability of Chan et al.'s scheme, described in Appendix A.2. Under the general assumption that the Pseudo-Random Functions (PRFs) are secure and the attacker can only retrieve a limited amount of data from a compromised server, we further prove that, our AKE protocol achieves all the desired properties based on the computational Diffie-Hellman (CDH) assumption and the strong Diffie-Hellman (SDH) assumption in the random oracle model with an appropriate parameter setup.

Thirdly, by using Raspberry Pi 3 Model B+ as the IoT device and a PC as the server, we investigate the parameter configurations for the big dataset held by the server and identify the optimal parameters. We run the experiment and show that one protocol execution takes 15.19 ms (or, 84.73 ms) for the IoT device and 2.44 ms (or, 12.51 ms) for the server for security parameter $\lambda = 128$ (or, $\lambda = 256$). Lastly, we propose an enhanced protocol to reduce IoT's computational complexity by outsourcing an exponentiation computation to the server. The enhanced protocol does not increase round complexity (i.e. it still needs three message passes) and only slightly increases the communication complexity. By instantiating the signature scheme with NIST's round three alternate candidate *Picnic*, we show that one protocol execution costs about 14.44 ms (or, 58.45 ms) for the IoT device and 12.78 ms (or, 46.34 ms) for the server for security parameter $\lambda = 128$ (or, $\lambda = 256$). The running time for the IoT device has become 1.05 and 1.45 times faster for $\lambda = 128$ and $\lambda = 256$, respectively. We notice that the improvement can be further enhanced with techniques from [15].

Table 1 shows that our protocol achieves more security properties than the existing ones in the literature. Nevertheless, we compare its complexity with the protocol from [9] which falls into a similar setting to ours. Moreover, we also resolve the scalability question concerning the server needs to serve a large number of IoT devices and store a huge amount of bigdata.

1.3 Paper Organisation

The rest of the paper is organised as follows. In Section 2, we describe the security model. In Section 3 and Section 4, we describe our novel AKE protocol and provide security analyses respectively. In Section 5, we detail our implementation procedure and present the evaluation results, and then present the enhanced protocol and corresponding experimental results. In addition, we make a simple comparison and resolve the scalability question. In Section 6, we conclude the paper.

2 IoT-oriented AKE Security Model

In this section, we first describe our IoT-oriented AKE setting, and then present a security model based on the existing ones, e.g. Bellare-Rogaway [4], Shoup model [32], Canetti-Krawczyk model [12], and particularly that of Pointcheval and Zimmer [31].

2.1 IoT-oriented AKE Setting

For simplicity, we only consider one IoT device and one server, and denote them as $\boldsymbol{c}$ and $\boldsymbol{s}$ respectively. For entity authentication, we assume two factors.

- One is a long-term shared private key mk between the two parties.
- The other is based on a dataset, which contains a large number of data items denoted as d_i $(1 \leq i \leq L)$ for some L. In the initialisation phase, the server processes the dataset with a set of secret keys, which map each data item to a tag. The server sends a subset of the secret keys (denoted as $\mathcal{S}_{\boldsymbol{c}}$) to the IoT device, and keeps some secret keys (denoted as $\mathcal{S}_{\boldsymbol{s}}$) together with the data item and tag tuples (d_i, t_i) $(1 \leq i \leq L)$ locally.

In contrast to Chan et al. [14], the dataset is considered as private information in our design. Furthermore, we assume a bounded retrieval model [2, 18], which implies that an attacker can only obtain a small portion of the the data item and tag tuples (d_i, t_i) $(1 \le i \le L)$ when it compromises the server. To sum up, the long-term private credentials for the IoT device and the server are $(mk, \mathcal{S}_{\boldsymbol{c}})$ and $(mk, \mathcal{S}_{\boldsymbol{s}}, (d_i, t_i)\ (1 \le i \le L))$, respectively.

2.2 Preliminary Notions

For our security model, we will adopt the standard game-based definitions. Below, we briefly introduce the preliminary notions.

For generality, we assume the parties can have concurrent runs of the protocol. Each execution of the protocol is called a session. If the attacker is passive, then a session will be happening between two instances, one from the IoT device $\boldsymbol{c}$ and the other from the server $\boldsymbol{s}$. For a party $\boldsymbol{p} \in \{\boldsymbol{c}, \boldsymbol{s}\}$, we use $\pi^i_{\boldsymbol{p}}$ to denote its i-th instance. Each instance can possess the following essential variables:

- *pid*: the partner identifier, where the server's identifier is denoted as $\boldsymbol{s}$ and $\boldsymbol{c}$ represents the identifier of IoT device.
- *sid*: the session identifier, and each *sid* should be unique.
- *sk*: the session key derived by $\pi^i_{\boldsymbol{p}}$ at the end of the protocol execution. It is initialized as $\perp$.
- *acc*: the state of acceptance $acc \in \{\perp, accepted, rejected\}$, which represents the state of $\pi^i_{\boldsymbol{p}}$ at the end of the protocol execution. It is initialized as $\perp$, will be set as *accepted* if the instance successfully completes the protocol execution, and will be set as *rejected* otherwise.
- *rev*: the status $rev \in \{revealed, unrevealed\}$ of the session key sk of $\pi^i_{\boldsymbol{p}}$. It is initialized as *unrevealed.*

We assume the party $\boldsymbol{p}$ maintains a status variable $cpt \in \{corrupted, uncorrupted\}$ which denotes whether or not it has been compromised or corrupted.

The notion of *partnering*, also called *matching conversation*, happens between two instances: one is the instance of an IoT device $\pi^i_{\boldsymbol{c}}$ and the other instance of a server $\pi^j_{\boldsymbol{s}}$, for some i and j. It requires the following conditions to be satisfied:

- $\pi^i_{\boldsymbol{c}}.acc = \pi^j_{\boldsymbol{s}}.acc = accepted$
- $\pi^i_{\boldsymbol{c}}.sid = \pi^j_{\boldsymbol{s}}.sid$
- $\pi^i_{\boldsymbol{c}}.sk = \pi^j_{\boldsymbol{s}}.sk$
- $\pi^i_{\boldsymbol{c}}.pid = \boldsymbol{s}$ and $\pi^j_{\boldsymbol{s}}.pid = \boldsymbol{c}$

An instance $\pi^i_{\boldsymbol{p}}$ $(\boldsymbol{p} \in \{\boldsymbol{c}, \boldsymbol{s}\})$ is said to be *fresh*, if the following conditions are satisfied: $\pi^i_{\boldsymbol{p}}.acc = accepted$; $\pi^i_{\boldsymbol{p}}.rev = unrevealed$, $\boldsymbol{p}.cpt = uncorrupted$; if it has any partner instance $\pi^j_{\boldsymbol{p}'}$, then $\boldsymbol{p}'.cpt = uncorrupted$ and $\pi^j_{\boldsymbol{p}'}.rev = unrevealed.$

Definition 1. *An AKE protocol is* sound *if, in the presence of any passive attacker, a protocol execution always successfully ends and results in a matching conversation between the IoT device* $\boldsymbol{c}$ *and the server* $\boldsymbol{s}$*.*

2.3 Game-based Security Definitions

For game-based security definitions, an attacker $\mathcal{A}$'s advantage over a security property is evaluated by a game played between the attacker and a challenger $\mathcal{C}$ who simulates the activities of the legitimate players, namely the IoT device $\boldsymbol{c}$ and the server $\boldsymbol{s}$ in our setting. In our security model, we assume that $\mathcal{A}$ is a probabilistic polynomial time (P.P.T.). We further assume that $\mathcal{A}$ fully controls the communication network so that it can intercept, delay, modify and delete the messages sent between any two instances.

Formally, the attacker $\mathcal{A}$'s intervention in a security game is modeled via the following oracle queries submitted to the challenger $\mathcal{C}$.

- $Send(msg, \pi^i_{\boldsymbol{p}})$: $\mathcal{A}$ can send any message msg to an instance $\pi^i_{\boldsymbol{p}}$ via this query. $\pi^i_{\boldsymbol{p}}$ responds according to the protocol specification. For simplicity, we assume the attacker can send a null message for the initiator to start a protocol execution.
- $Corrupt_{\boldsymbol{c}}()$: After receiving this query, the challenger $\mathcal{C}$ returns the long-term key of $\boldsymbol{c}$, namely mk and $\mathcal{S}_{\boldsymbol{c}}$. Simultaneously, the challenger $\mathcal{C}$ sets $\boldsymbol{c}$'s status variable as $\boldsymbol{c}.cpt = corrupted$.
- $Corrupt_{\boldsymbol{s}}(\mathbb{I}_{\mathcal{A}})$: After receiving this query, the challenger $\mathcal{C}$ returns the long-term key of $\boldsymbol{s}$, namely mk and $\mathcal{S}_{\boldsymbol{s}}$, and the (d_i, t_i) whose index i falls inside $\mathbb{I}_{\mathcal{A}}$. In the bounded retrieval model, $\mathbb{I}_{\mathcal{A}}$ has a limited size. Simultaneously, the challenger $\mathcal{C}$ sets $\boldsymbol{s}$'s status variable as $\boldsymbol{s}.cpt = corrupted$.
- $Reveal(\pi^i_{\boldsymbol{p}})$: This query can only be issued to an accepted instance $\pi^i_{\boldsymbol{p}}$. After receiving this query, the challenger $\mathcal{C}$ returns the contents of the session key $\pi^i_{\boldsymbol{p}}.sk$. Simultaneously, the session key status of $\pi^i_{\boldsymbol{p}}$ and its partner $\pi^j_{\boldsymbol{p'}}$ are set to $\pi^i_{\boldsymbol{p}}.rev = \pi^j_{\boldsymbol{p'}}.rev = revealed$.
- $Test(\pi^i_{\boldsymbol{p}})$: The instance $\pi^i_{\boldsymbol{p}}$ should be fresh. After receiving this query, the challenger $\mathcal{C}$ flips a coin $b \in \{0, 1\}$ uniformly at random, and returns the session key if $b = 0$, otherwise, it outputs a random string from the session key space.

Definition 2. *In our security model, an AKE protocol is said to be secure if it is sound and the advantages $Adv^{PFS}(\mathcal{A})$, $Adv^{KCI}_{\boldsymbol{s}}(\mathcal{A})$, $Adv^{KCI}_{\boldsymbol{c}}(\mathcal{A})$ and $Adv^{SCI}(\mathcal{A})$ are negligible for any P.P.T. attacker $\mathcal{A}$. These advantages are defined in the security games in following-up subsections.*

2.3.1 Session Key Security and Forward Secrecy

This game is designed for modeling session key security, including the known key security property (i.e., the knowledge of session keys generated in other sessions should not help the attacker to learn anything more about the session key in a target session.). In more detail, it is defined as follows:

1. $\mathcal{C}$ generates parameters and gives the public parameters to $\mathcal{A}$.
2. Once $\mathcal{A}$ has all public parameters, it can issue a polynomial number of *Send* and *Reveal* queries in any order.

3. At some point, $\mathcal{A}$ chooses a fresh instance $\pi_{\boldsymbol{c}}^{i}$ for some i or $\pi_{\boldsymbol{s}}^{j}$ for some j, and issues a $Test$ query.
4. $\mathcal{A}$ can continue issuing queries as in step 2, but not any *Reveal* query to the tested instance and its partner.
5. Eventually, $\mathcal{A}$ terminates the game and outputs a guess bit b' for b.

$\mathcal{A}$ wins the game if $b' = b$. Formally, $\mathcal{A}$'s advantage is defined as $Adv^{SK}(\mathcal{A}) = \left|\Pr[b = b'] - \frac{1}{2}\right|$.

In order to model (perfect) forward secrecy, we only need to slightly modify the above game.

– If $\mathcal{A}$ is allowed to issue one of $Corrupt_{\boldsymbol{c}}$ and $Corrupt_{\boldsymbol{s}}$ queries in Step 4 of the above game, then we obtain the security game for *forward secrecy*. The attacker $\mathcal{A}$'s advantage is defined as $Adv^{FS}(\mathcal{A}) = \left|\Pr[b = b'] - \frac{1}{2}\right|$.
– If $\mathcal{A}$ is allowed to issue both $Corrupt_{\boldsymbol{c}}$ and $Corrupt_{\boldsymbol{s}}$ queries in Step 4 of the above game, then we obtain the security game for *perfect forward secrecy*. The attacker $\mathcal{A}$'s advantage is defined as $Adv^{PFS}(\mathcal{A}) = \left|\Pr[b = b'] - \frac{1}{2}\right|$.

It is clear that $Adv^{PFS}(\mathcal{A}) \geq Adv^{FS}(\mathcal{A}) \geq Adv^{SK}(\mathcal{A})$. In other words, if $Adv^{PFS}(\mathcal{A})$ is negligible then the others will also be negligible.

2.3.2 Key Compromise Impersonation Resilience

For the KCI property, we consider two scenarios and propose two security games accordingly. In the first scenario, $\mathcal{A}$ impersonates the IoT device $\boldsymbol{c}$ to the corrupted server $\boldsymbol{s}$. In the second scenario, $\mathcal{A}$ impersonates the server $\boldsymbol{s}$ to the corrupted IoT device $\boldsymbol{c}$. Next, we describe the security game for the first scenario.

1. $\mathcal{C}$ generates the parameters and gives the public ones to $\mathcal{A}$.
2. $\mathcal{A}$ sends $Corrupt_{\boldsymbol{s}}$ to $\mathcal{C}$ for the secret information of $\boldsymbol{s}$.
3. $\mathcal{A}$ can issue a polynomial number of *Send* and *Reveal* queries in any order.
4. $\mathcal{A}$ terminates the game and outputs a session identifier $sid_{\mathcal{A}}$ for a selected instance $\pi_{\boldsymbol{s}}^{j}$.

Let $Succ_{\boldsymbol{s}}$ denote an event, defined by the following two conditions. Formally, $\mathcal{A}$'s advantage is defined as $Adv_{\boldsymbol{s}}^{KCI}(\mathcal{A}) = \Pr[Succ_{\boldsymbol{s}}]$.

– $\pi_{\boldsymbol{s}}^{j}$ successfully accepts;
– Not all messages $\pi_{\boldsymbol{s}}^{j}$ receives are identical to what have been sent by some instance $\pi_{\boldsymbol{c}}^{i}$ which also possesses $sid_{\mathcal{A}}$ as its session identifier. This condition excludes the trivial "attack" that $\mathcal{A}$ simply relays the message exchanges between $\pi_{\boldsymbol{c}}^{i}$ and $\pi_{\boldsymbol{s}}^{j}$.

For the second scenario, we only need to make two changes in the above game. In step 2, $\mathcal{A}$ is allowed to issue $Corrupt_{\boldsymbol{c}}$ query instead of $Corrupt_{\boldsymbol{s}}$ query. In step 4, $\mathcal{A}$ outputs a session identifier $sid_{\mathcal{A}}$ for a selected instance $\pi_{\boldsymbol{c}}^{i}$. Let $Succ_{\boldsymbol{c}}$ denote an event, defined by the following two conditions. Formally, $\mathcal{A}$'s advantage is defined as $Adv_{\boldsymbol{c}}^{KCI}(\mathcal{A}) = \Pr[Succ_{\boldsymbol{c}}]$.

- $\pi_{\boldsymbol{c}}^{i}$ successfully accepts;
- Not all messages $\pi_{\boldsymbol{c}}^{i}$ receives are identical to what has been sent by some instance $\pi_{\boldsymbol{s}}^{j}$ which also possesses $sid_{\mathcal{A}}$ as its session identifier.

2.3.3 Server Compromise Impersonation Resilience

For the SCI resilience property, we require that any attacker $\mathcal{A}$ cannot impersonate the server $\boldsymbol{s}$ to the IoT device $\boldsymbol{c}$ even if it can compromise the server, i.e. issuing a $Corrupt_{\boldsymbol{s}}$ query in the game. The following security game captures this property.

1. $\mathcal{C}$ generates the parameters and gives the public ones to $\mathcal{A}$.
2. $\mathcal{A}$ can send $Corrupt_{\boldsymbol{s}}$ to $\mathcal{C}$ for the secret information of $\boldsymbol{s}$.
3. $\mathcal{A}$ can issue a polynomial number of *Send* and *Reveal* queries in any order.
4. $\mathcal{A}$ terminates the game and outputs a session identifier $sid_{\mathcal{A}}$ for a selected instance $\pi_{\boldsymbol{c}}^{i}$.

Let *Succ* denote an event defined by the following two conditions. Formally, $\mathcal{A}$'s advantage is defined as $Adv^{SCI}(\mathcal{A}) = \Pr[Succ]$.

- $\pi_{\boldsymbol{c}}^{i}$ successfully accepts;
- Not all messages $\pi_{\boldsymbol{c}}^{i}$ receives are identical to what have been sent by some $\pi_{\boldsymbol{s}}^{j}$ which also possesses $sid_{\mathcal{A}}$ as its session identifier.

3 The Proposed AKE Protocol

To bootstrap the AKE protocol, an initialisation phase is required for the IoT device and the server to configure their credentials. In practice, these entities can be configured in a secure lab, and then the IoT device is deployed in the remote environment, say in a factory site. In this section, we first introduce this phase, and then describe the proposed AKE protocol in detail.

3.1 Initialisation Phase

In this phase, the server $\boldsymbol{s}$ chooses a security parameter λ (e.g. $\lambda = 128$ or $\lambda = 256$) and initializes the following public parameters: a group $\mathbb{G}$ of prime order q, a generator g of $\mathbb{G}$, a cryptographic hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_q$ and two Pseudo-Random Functions (PRFs) $\mathsf{F} : \{0,1\}^\lambda \times \{0,1\}^* \to \mathbb{Z}_q$, $\mathsf{E} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$. Furthermore, the server $\boldsymbol{s}$ generates a public/private key pair $(pk = g^{sk}, sk)$, where $sk \in \mathbb{Z}_q$, and also generates $sk_1 = \{mk\}$ where $mk \in \{0,1\}^\lambda$ as the long-term shared key and $sk_2 = \{K, K'\}$ for tag generation and data processing where $K \in \mathbb{Z}_q$ and $K' \in \{0,1\}^\lambda$. Suppose the server $\boldsymbol{s}$ possesses a dataset $\mathbb{D}$ which contains L data items d_i $(1 \le i \le L)$. For every data item $d_i \in \mathbb{D}$, the server generates its tag as $t_i = K \cdot \mathsf{H}(d_i) + \mathsf{F}_{K'}(i)$, which is computed in the finite field $\mathbb{Z}_q$. We define a dataset $\mathbb{D}^*$ which contains all data item and tag tuples (d_i, t_i) $(1 \le i \le L)$. In addition, the server also chooses an

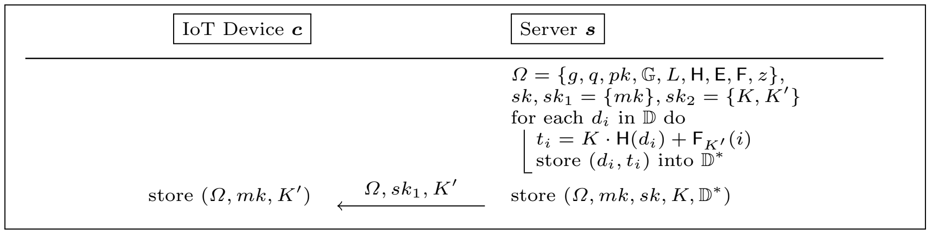
IoT Device $\boldsymbol{c}$
Server $\boldsymbol{s}$
$\Omega = \{g, q, pk, \mathbb{G}, L, \mathsf{H}, \mathsf{E}, \mathsf{F}, z\}$,
$sk, sk_1 = \{mk\}, sk_2 = \{K, K'\}$
for each d_i in $\mathbb{D}$ do
$t_i = K \cdot \mathsf{H}(d_i) + \mathsf{F}_{K'}(i)$
store (d_i, t_i) into $\mathbb{D}^*$
store (Ω, mk, K')
Ω, sk_1, K'
store $(\Omega, mk, sk, K, \mathbb{D}^*)$

Fig. 1. Initialisation Phase

index parameter z which is an integer. For clarity, we summarize the initialisation phase in Figure 1.

As a quick remark, referring to our problem setting described in Section 2.1 and the security model, the IoT device's long-term credentials $\mathcal{S}_{\boldsymbol{c}} = \{mk, K'\}$ and the server's long-term credentials $\mathcal{S}_{\boldsymbol{s}} = \{mk, sk, K, \mathbb{D}^*\}$.

Even though our work is inspired by [14], our initialisation is significantly different in two aspects. Firstly, we assume the dataset $\mathbb{D}$ is pre-configured or randomly generated by the server. This dataset is also treated as secret information for the server. Secondly, the keys for tag generation (namely, sk_2) are split and separately stored in the IoT device $\boldsymbol{c}$ (namely, K) and the server $\boldsymbol{s}$ (namely, K'). Overall, these differences make it impossible for an attacker to forge tags even if it has compromised one party. One specific benefit is that it helps us avoid the vulnerability of Chan et al.'s scheme [14], described in Appendix A.2.

3.2 Description of the Proposed AKE Protocol

Intuitively, the proposed AKE protocol is in the Diffie-Hellman style while the mutual authentication is achieved by (1) asking the IoT device and the server to mutually prove the data-tag relationship via the distributed credentials in the *Initialisation Phase*, and (2) asking the server to prove its knowledge about sk via computing a^*. The protocol is summarized in Figure 2, and its detailed execution is as follows. We use the notation $a \xleftarrow{\$} \mathbb{B}$ to denote selecting a from the set $\mathbb{B}$ uniformly at random.

1. The IoT device $\boldsymbol{c}$ first selects $r_1 \xleftarrow{\$} \mathbb{Z}_q^*$ to compute $a = pk^{r_1}$ and $g' = g^{r_1}$. Then, it selects $r_2 \xleftarrow{\$} \{0,1\}^\lambda$ and a random subset $\mathbb{I}_{\boldsymbol{c}}$ of z distinct indices for the tuples in $\mathbb{D}^*$, and then sends $g', r_2, \mathbb{I}_{\boldsymbol{c}}$ and $M_1 = \mathsf{H}(mk||a||g'||r_2||\mathbb{I}_{\boldsymbol{c}})$ to the server $\boldsymbol{s}$.
2. After receiving the message, the server $\boldsymbol{s}$ first computes $a^* = {g'}^{sk}$ and verifies whether or not $M_1 = \mathsf{H}(mk||a^*||g'||r_2||\mathbb{I}_{\boldsymbol{c}})$ holds. If the verification passes, it randomly selects a subset $\mathbb{I}_{\boldsymbol{s}}$ of z distinct indices which should be disjoint from $\mathbb{I}_{\boldsymbol{c}}$. Next, it computes $r_2' = \mathsf{E}_{mk}(r_2)$, $X = K \cdot \sum_{i \in \mathbb{I}}^{i} (\mathsf{H}(d_i) \cdot \mathsf{F}_{r_2'}(i))$ and $Y = \sum_{i \in \mathbb{I}}^{i} (t_i \cdot \mathsf{F}_{r_2'}(i))$, where $\mathbb{I} = \mathbb{I}_{\boldsymbol{c}} \cup \mathbb{I}_{\boldsymbol{s}}$. Note that we assume X, Y are computed in the finite field $\mathbb{Z}_q$. Besides, the server $\boldsymbol{s}$ randomly selects r_3

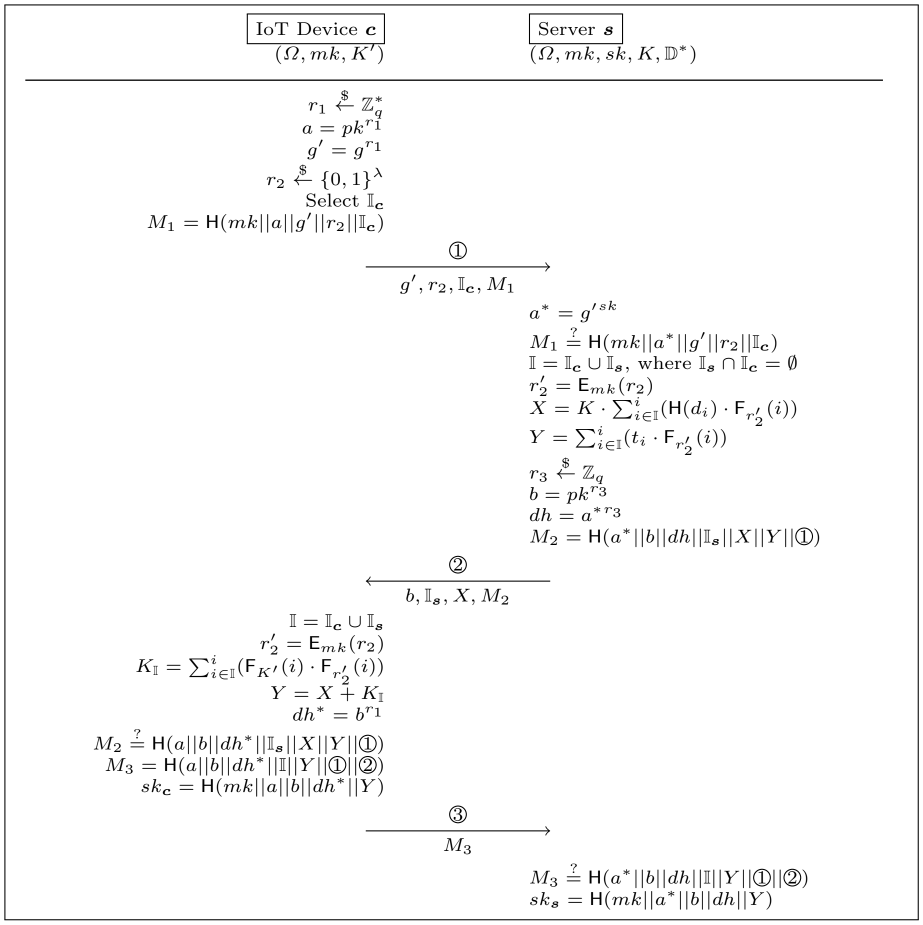
$\boxed{\text{IoT Device } \boldsymbol{c}}$
(Ω, mk, K')
$\boxed{\text{Server } \boldsymbol{s}}$
$(\Omega, mk, sk, K, \mathbb{D}^*)$
$r_1 \xleftarrow{\$} \mathbb{Z}_q^*$
$a = pk^{r_1}$
$g' = g^{r_1}$
$r_2 \xleftarrow{\$} \{0,1\}^\lambda$
Select $\mathbb{I}_{\boldsymbol{c}}$
$M_1 = \mathsf{H}(mk||a||g'||r_2||\mathbb{I}_{\boldsymbol{c}})$
$\xrightarrow[g', r_2, \mathbb{I}_{\boldsymbol{c}}, M_1]{①}$
$a^* = {g'}^{sk}$
$M_1 \overset{?}{=} \mathsf{H}(mk||a^*||g'||r_2||\mathbb{I}_{\boldsymbol{c}})$
$\mathbb{I} = \mathbb{I}_{\boldsymbol{c}} \cup \mathbb{I}_{\boldsymbol{s}}$, where $\mathbb{I}_{\boldsymbol{s}} \cap \mathbb{I}_{\boldsymbol{c}} = \emptyset$
$r_2' = \mathsf{E}_{mk}(r_2)$
$X = K \cdot \sum_{i \in \mathbb{I}}^{i} (\mathsf{H}(d_i) \cdot \mathsf{F}_{r_2'}(i))$
$Y = \sum_{i \in \mathbb{I}}^{i} (t_i \cdot \mathsf{F}_{r_2'}(i))$
$r_3 \xleftarrow{\$} \mathbb{Z}_q$
$b = pk^{r_3}$
$dh = {a^*}^{r_3}$
$M_2 = \mathsf{H}(a^*||b||dh||\mathbb{I}_{\boldsymbol{s}}||X||Y||①)$
$\xleftarrow[b, \mathbb{I}_{\boldsymbol{s}}, X, M_2]{②}$
$\mathbb{I} = \mathbb{I}_{\boldsymbol{c}} \cup \mathbb{I}_{\boldsymbol{s}}$
$r_2' = \mathsf{E}_{mk}(r_2)$
$K_{\mathbb{I}} = \sum_{i \in \mathbb{I}}^{i} (\mathsf{F}_{K'}(i) \cdot \mathsf{F}_{r_2'}(i))$
$Y = X + K_{\mathbb{I}}$
$dh^* = b^{r_1}$
$M_2 \overset{?}{=} \mathsf{H}(a||b||dh^*||\mathbb{I}_{\boldsymbol{s}}||X||Y||①)$
$M_3 = \mathsf{H}(a||b||dh^*||\mathbb{I}||Y||①||②)$
$sk_{\boldsymbol{c}} = \mathsf{H}(mk||a||b||dh^*||Y)$
$\xrightarrow[M_3]{③}$
$M_3 \overset{?}{=} \mathsf{H}(a^*||b||dh||\mathbb{I}||Y||①||②)$
$sk_{\boldsymbol{s}} = \mathsf{H}(mk||a^*||b||dh||Y)$

Fig. 2. Proposed AKE Protocol

to compute $b = pk^{r_3}$ and $dh = a^{*r_3}$. Finally, the server $\boldsymbol{s}$ sends $b, \mathbb{I}_{\boldsymbol{s}}, X$ and $M_2 = \mathsf{H}(a^*||b||dh||\mathbb{I}_{\boldsymbol{s}}||X||Y||①)$ to the IoT device $\boldsymbol{c}$. For the sake of space, we use ① to denote the messages sent in the first round, namely $g', r_2, \mathbb{I}_{\boldsymbol{c}}, M_1$.

3. After receiving the message, the IoT device $\boldsymbol{c}$ computes $r_2' = \mathsf{E}_{mk}(r_2)$ and $K_{\mathbb{I}} = \sum_{i \in \mathbb{I}}^{i} (\mathsf{F}_{K'}(i) \cdot \mathsf{F}_{r_2'}(i))$, where $\mathbb{I} = \mathbb{I}_{\boldsymbol{c}} \cup \mathbb{I}_{\boldsymbol{s}}$. Next, it computes $Y = X + K_{\mathbb{I}}$ and $dh^* = b^{r_1}$, and verifies whether $M_2 = \mathsf{H}(a||b||dh^*||\mathbb{I}_{\boldsymbol{s}}||X||Y||①)$ holds. If the verification passes, it computes $M_3 = \mathsf{H}(a||b||dh^*||\mathbb{I}||Y||①||②)$ and sends it to the server $\boldsymbol{s}$. Finally, the IoT device $\boldsymbol{c}$ computes its session key and session identifier as $sk_{\boldsymbol{c}} = \mathsf{H}(mk||a||b||dh^*||Y)$ and $sid_{\boldsymbol{c}} = \mathsf{H}(①||②)$ respectively.
4. After receiving the message, the server $\boldsymbol{s}$ verifies whether or not the equality $M_3 = \mathsf{H}(a^*||b||dh||\mathbb{I}||Y||①||②)$ holds. If the verification passes, it computes

its session key and session identifier as $sk_{\boldsymbol{s}} = \mathsf{H}(mk||a^*||b||dh||Y)$ and $sid_{\boldsymbol{s}} = \mathsf{H}(①||②)$ respectively.

It is straightforward to verify that the above AKE protocol is sound under Definition 1.

4 Security Analysis

In this section, we first review the Computational Diffie–Hellman (CDH) and Strong Diffie-Hellman (SDH) assumptions and then prove the security of the proposed AKE protocol in our security model.

4.1 CDH and SDH Assumptions

CDH assumption is widely used in the literature. Given a security parameter λ, there exists a polynomial time algorithm which takes λ as input and outputs a cyclic group $\mathbb{G}$ of prime order q. On the input of $(\mathbb{G}, q, g)$ and a CDH challenge (g^a, g^b), where g is a generator of $\mathbb{G}$ and a, b are randomly chosen from $\mathbb{Z}_q^*$, a P.P.T. attacker can only compute g^{ab} with a negligible probability.

Related to the CDH assumption, the SDH assumption has been defined in [1]. The setup is identical to the CDH assumption, except that the attacker has access to an oracle $\mathcal{O}_b$. After receiving a tuple (g_1, g_2), $\mathcal{O}_b$ replies 1 if $g_2 = g_1^b$ and replies 0 otherwise. In this paper, we will use a hashed variant of the SDH assumption where the only difference lies in $\mathcal{O}_b$. Given a hash function H, in this variant, the oracle $\mathcal{O}'_b$ is defined as: after receiving a (g_1, d_1, d_2, h), $\mathcal{O}'_b$ replies 1 if $h = \mathsf{H}(d_1||g_1^b||d_2)$ and replies 0 otherwise.

Suppose the hash function H is modeled as a random oracle, we can prove that the SDH and the hashed variant are equivalent. The equivalence is briefly demonstrated below.

- Given an oracle $\mathcal{O}_b$, we can construct an oracle $\mathcal{O}'_b$ as follows. After receiving a (g_1, d_1, d_2, h), $\mathcal{O}'_b$ replies 1 if (1) there is a query to H such that $h = \mathsf{H}(d_1||x||d_2)$ and (2) when being queried with (g_1, x), $\mathcal{O}_b$ replies 1. Otherwise, $\mathcal{O}'_b$ replies 0.
- Given an oracle $\mathcal{O}'_b$, we can construct an oracle $\mathcal{O}_b$ as follows. After receiving a tuple (g_1, g_2), $\mathcal{O}_b$ replies 1 if the oracle $\mathcal{O}'_b$ replies 1 when being queried with $(g_1, d_1, d_2, \mathsf{H}(d_1||g_2||d_2))$ where d_1 and d_2 are randomly generated. Otherwise, $\mathcal{O}_b$ replies 0.

4.2 Security Proofs

We prove the proposed AKE protocol is secure under Definition 2, by showing the advantages $Adv^{PFS}(\mathcal{A})$, $Adv_{\boldsymbol{s}}^{KCI}(\mathcal{A})$, $Adv_{\boldsymbol{c}}^{KCI}(\mathcal{A})$ and $Adv^{SCI}(\mathcal{A})$ are negligible in the following Lemmas. Note that we generally assume that the Pseudo-Random Functions (PRFs) are secure and the attacker can only retrieve a limited amount of data from a compromised server (i.e. bounded retrieval model). For the sake of simplicity, we avoid repeating them in each Lemma.

Lemma 1. *The proposed protocol achieves KCI resilience in the first scenario defined in Section 2.3.2 (i.e. $\mathcal{A}$ cannot impersonate $\boldsymbol{c}$ to $\boldsymbol{s}$ even after it has compromised $\boldsymbol{s}$). The security property holds based on the CDH assumption in the random oracle as long as $\frac{\binom{n-z}{z}}{\binom{L-z}{z}}$ is negligible.*

Proof. Referring to the security game defined in Section 2.3.2, we assume $\mathcal{A}$ has eavesdropped on m sessions in the game. Hence, $\mathcal{A}$ has all messages transmitted in those sessions (messages in communication ①$_j$,②$_j$,③$_j$, where $j \in [1, m]$). Since $\mathcal{A}$ can issue $Corrupt_{\boldsymbol{s}}$ query, it has some private information of the server $\boldsymbol{s}$: (mk, sk, K) and n tuples of $\mathbb{D}^*$ denoted as $\mathbb{I}_{\mathcal{A}} = \{i_{\mathcal{A}1}, i_{\mathcal{A}2}, \ldots, i_{\mathcal{A}n}\}$.

Let us assume, in step 4 of the attack game, $\mathcal{A}$ has chosen an instance $\pi_{\boldsymbol{s}}^{j}$ which has the session identifier $sid_{\mathcal{A}}$. Next, we analyse the messages received by this instance. Regarding the first message ①, namely $(g', r_2, \mathbb{I}_{\boldsymbol{c}}, M_1)$, there are two scenarios.

– Scenario 1: the message ① is generated by some instance $\pi_{\boldsymbol{c}}^{i}$, and $\mathcal{A}$ just forwards it directly to $\pi_{\boldsymbol{s}}^{j}$ without any modification.
– Scenario 2: the message ① has been modified by $\mathcal{A}$ on the basis of some intercepted message, particularly, $\mathcal{A}$ can generate ① by itself.

In Scenario 1, there are two cases with respect to the action of $\mathcal{A}$ when it receives the message ②, namely $(b, I_{\boldsymbol{s}}, X, M_2)$, from $\pi_{\boldsymbol{s}}^{j}$.

1. $\mathcal{A}$ forwards ② to $\pi_{\boldsymbol{c}}^{i}$ without any modification, then $\pi_{\boldsymbol{c}}^{i}$ will generates a session identifier $sid_{\boldsymbol{c}}$ and $sk_{\boldsymbol{c}}$, then sends a reply ③. In this case, the only possibility to make $\pi_{\boldsymbol{s}}^{j}$ accept is to forward ③ directly to $\pi_{\boldsymbol{s}}^{j}$. According to our definition, $\mathcal{A}$ does not win the game in this case.
2. $\mathcal{A}$ forwards ② to $\pi_{\boldsymbol{c}}^{i}$ with some changes, denoted as ②$'$. If $\pi_{\boldsymbol{c}}^{i}$ does not reject, then it will send ③$'$, which is a hash value based on ②$'$ and some other data. In this case, $\mathcal{A}$ needs to compute a legitimate ③, namely M_3, on its own. In order to compute M_3, $\mathcal{A}$ needs to either (1) recover (a, dh^*, Y) from ③$'$ or ②, or (2) compute these values from scratch. To this end, the success possibility is negligible based on the CDH assumption given that H is a random oracle.

In Scenario 2, without loss of generality, we assume $\mathcal{A}$ has generated ① by itself. $\mathcal{A}$ knows the ephemeral data (r_1, a) as well as ① and ②. Without the knowledge of K', $\mathcal{A}$ needs to compute a legitimate Y in order to compute M_3 to make $\pi_{\boldsymbol{s}}^{j}$ accept. Next, we distinguish two cases.

1. The first case is when $(\mathbb{I}_{\boldsymbol{c}} \cup \mathbb{I}_{\boldsymbol{s}}) \subseteq \mathbb{I}_{\mathcal{A}}$. Because $\mathbb{I}_{\boldsymbol{c}}$ could be chosen by $\mathcal{A}$, then the best strategy to maximize the probability $(\mathbb{I}_{\boldsymbol{c}} \cup \mathbb{I}_{\boldsymbol{s}}) \subseteq \mathbb{I}_{\mathcal{A}}$ is to choose $\mathbb{I}_{\boldsymbol{c}} \subset \mathbb{I}_{\mathcal{A}}$, and then we have $\Pr[\mathbb{I}_{\boldsymbol{s}} \subseteq (\mathbb{I}_{\mathcal{A}} \setminus \mathbb{I}_{\boldsymbol{c}})] = \frac{\binom{n-z}{z}}{\binom{L-z}{z}}$. In this case, $\mathcal{A}$ wins the game as it can compute M_3 successfully.
2. When $\mathbb{I}_{\boldsymbol{s}} \nsubseteq (\mathbb{I}_{\mathcal{A}} \setminus \mathbb{I}_{\boldsymbol{c}})$, $\mathcal{A}$ needs to possess $\mathsf{F}_{K'}(i)$ for all $i \in (\mathbb{I}_{\boldsymbol{c}} \cup \mathbb{I}_{\boldsymbol{s}}) \setminus \mathbb{I}_{\mathcal{A}}$ in order to compute Y and then M_3. Due to the fact that $\mathsf{F}_{K'}(j)$ for any j is always hashed in any transmission, the probability that $\mathcal{A}$ can obtain $\mathsf{F}_{K'}(i)$ for all $i \in (\mathbb{I}_{\boldsymbol{c}} \cup \mathbb{I}_{\boldsymbol{s}}) \setminus \mathbb{I}_{\mathcal{A}}$ is a negligible value.

Summarizing the two scenarios, $Adv_{\boldsymbol{s}}^{KCI}(\mathcal{A})$ is negligible given that CDH assumption holds and H is a random oracle, as long as we can make $\frac{\binom{n-z}{z}}{\binom{L-z}{z}}$ negligible by setting appropriate values for z and L. □

Lemma 2. *The proposed protocol achieves KCI resilience in the second scenario defined in Section 2.3.2 (i.e. $\mathcal{A}$ cannot impersonate $\boldsymbol{s}$ to $\boldsymbol{c}$ even after it has compromised $\boldsymbol{c}$). The security holds based on the SDH assumption in the random oracle.*

Proof. Similar to the proof of Lemma 1, we assume $\mathcal{A}$ knows all messages of m sessions ①$_j$,②$_j$,③$_j$, where $j \in [1, m]$ and the private information (mk, K') of the IoT device $\boldsymbol{c}$ via $Corrupt_{\boldsymbol{c}}$ query.

Let us assume, in step 4 of the attack game, $\mathcal{A}$ has selected an instance $\pi_{\boldsymbol{c}}^{i}$ which has the session identifier $sid_{\mathcal{A}}$. During the game, $\pi_{\boldsymbol{c}}^{i}$ sends the first message ①, namely $(g', r_2, \mathbb{I}_{\boldsymbol{c}}, M_1)$ where $M_1 = \mathsf{H}(mk||a||g'||r_2||\mathbb{I}_{\boldsymbol{c}})$, and it receives the second message ②. There are two scenarios with respect to this message.

– Scenario 1: Some instance $\pi_{\boldsymbol{s}}^{j}$ generates ②, denoted as $(b, I_{\boldsymbol{s}}, X, M_2)$, and $\mathcal{A}$ just forwards it directly to $\pi_{\boldsymbol{c}}^{i}$ without any modification.
– Scenario 2: Some instance $\pi_{\boldsymbol{s}}^{j}$ generates ②, and $\mathcal{A}$ generates a different message ②$'$ based on it.

In Scenario 1, since the only message $\pi_{\boldsymbol{c}}^{i}$ receives is exactly the message sent by some $\pi_{\boldsymbol{s}}^{j}$. Note that $\pi_{\boldsymbol{s}}^{j}$ should have received ① in order to have its ② to be accepted. According to our definition, the attack fails in this scenario, where $\mathcal{A}$'s advantage is 0.

In Scenario 2, $\mathcal{A}$ possesses ①, ② and other credentials, and aims at generating a message ②$'$, which contains a hash value M_2' which should be computed based on a and other data. To this end, $\mathcal{A}$ has two choices.

– One is to compute the pre-images for M_1 and M_2 in ①, ② respectively. To this end, the probability is clearly negligible.
– The other is to compute a from scratch. To this end, $\mathcal{A}$ has $pk = g^{sk}$ and $g' = g^{r_1}$, while a should equal to $g^{sk \cdot r_1}$. In addition, $\mathcal{A}$ can send message in the form of ① to a server's instance to check whether the M_1 value in the message is correctly computed. This is equivalent to an oracle $\mathcal{O}_{sk}$, which takes (g', d_1, d_2, a) as input and outputs 1 if $a = \mathsf{H}(d_1||g'^{sk}||d_2)$. Referring to Section 4.1, $\mathcal{A}$'s advantage is negligible based on the SDH assumption.

Summarizing both scenarios, $\mathcal{A}$'s advantage $Adv_{\boldsymbol{c}}^{KCI}(\mathcal{A})$ is negligible based on the SDH assumption in the random oracle model. □

Lemma 3. *The proposed protocol achieves SCI resilience property defined in Section 2.3.3 (i.e. $\mathcal{A}$ cannot impersonate $\boldsymbol{s}$ to $\boldsymbol{c}$ even after it has compromised $\boldsymbol{s}$). The security property holds in the random oracle as long as $\frac{\binom{n}{z}}{\binom{L}{z}}$ is negligible.*

Proof. As in the proof of Lemma 1 and Lemma 2, we assume $\mathcal{A}$ knows all messages of m sessions ①$_j$,②$_j$,③$_j$, where $j \in [1, m]$. By issuing $Corrupt_{\boldsymbol{s}}(I_{\mathcal{A}} = \{i_{\mathcal{A}1}, i_{\mathcal{A}2}, \ldots, i_{\mathcal{A}n}\})$ query, $\mathcal{A}$ has also obtained the private information (mk, sk, K) and n tuples from $\mathbb{D}^*$ of the server $\boldsymbol{s}$.

Let us assume, in step 4 of the attack game, $\mathcal{A}$ has selected an instance $\pi^i_{\boldsymbol{c}}$ which has the session identifier $sid_{\mathcal{A}}$. From this point, the proof is similar to that of Lemma 2, the only difference is the Scenario 2, we analyze it as follows.

In this scenario, $\mathcal{A}$ possesses ①, ② and other credentials, and aims at generating a message ②$'$ that contains a hash value M_2' which should be computed based on (X, Y) and other data. Overall, $\mathcal{A}$'s probability of successfully computing (X, Y) differs in two cases.

1. When $\mathbb{I}_{\boldsymbol{c}} \nsubseteq \mathbb{I}_{\mathcal{A}}$. In order to derive a legitimate pair of (X, Y), $\mathcal{A}$ needs to possess $\mathsf{F}_{K'}(i)$ for all $i \in (\mathbb{I}_{\boldsymbol{c}} \cup \mathbb{I}_{\boldsymbol{s}}) \setminus \mathbb{I}_{\mathcal{A}}$. As we mentioned in the proof of Lemma 1, the probability that $\mathcal{A}$ can obtain all required $\mathsf{F}_{K'}(i)$ is negligible.
2. When $\mathbb{I}_{\boldsymbol{c}} \subseteq \mathbb{I}_{\mathcal{A}}$. In this case, then the best strategy for $\mathcal{A}$ is to choose $\mathbb{I}_{\boldsymbol{s}} \subset (\mathbb{I}_{\mathcal{A}} \setminus \mathbb{I}_{\boldsymbol{c}})$. In this case, $\mathcal{A}$ can compute a legitimate M_2 and its probability is $\Pr[\mathbb{I}_{\boldsymbol{c}} \subseteq \mathbb{I}_{\mathcal{A}}] = \frac{\binom{n}{z}}{\binom{L}{z}}$.

Summarizing the two scenarios, $Adv^{SCI}(\mathcal{A})$ is negligible given that H is a random oracle, as long as we can make $\frac{\binom{n}{z}}{\binom{L}{z}}$ negligible by setting proper values for z and L. □

It is clear that $\frac{\binom{n}{z}}{\binom{L}{z}} > \frac{\binom{n-z}{z}}{\binom{L-z}{z}}$. To make Lemma 1 and Lemma 3 hold, we (at least) need to make $\frac{\binom{n}{z}}{\binom{L}{z}}$ negligible.

Lemma 4. *The proposed protocol achieves the perfect forward secrecy (PFS) property based on the SDH assumption in the random oracle as long as* $\frac{\binom{n-z}{z}}{\binom{L-z}{z}}$ *is negligible.*

Proof. We first assume that $\mathcal{A}$ issues the $Test$ query to an instance $\pi^i_{\boldsymbol{c}}$ in step 3 of the game. We carry out the proof via the standard game hopping technique.

Game 0 is the original session key security game described in Section 2.3.1. Let E_0 be the event that $b = b'$ at the end of the game. By definition, we have $Adv^{PFS}(\mathcal{A}) = \left|\Pr[E_0] - \frac{1}{2}\right|$.

Game 1. In this game, we rewrite **Game** 0 with the following definitions. We define the event Evn^+ as: there is an instance $\pi^j_{\boldsymbol{s}}$ which receives $\pi^i_{\boldsymbol{c}}$'s message ① and has its reply ② received by $\pi^i_{\boldsymbol{c}}$. While, we define Evn^* as the event that Evn^+ does not occur. Clearly, $\Pr[Evn^*] + \Pr[Evn^+] = 1$. Next, we define the event E_1^* as $b = b'$ when Evn^* happens and E_1^+ as $b = b'$ when Evn^+ happens. Hence, we have

$$\Pr[E_0] = \Pr[E_1] = \Pr[E_1^*] \cdot \Pr[Evn^*] + \Pr[E_1^+] \cdot \Pr[Evn^+].$$

Based on Lemma 2, we know that $\Pr[Evn^*]$ is negligible based on the SDH assumption in the random oracle.

Game 2. In this game, we add an abort rule. $\mathcal{C}$ aborts if the event Evn^* happens. Otherwise, the game continues as following. Define the event E_2 as $b = b'$ when the game does not abort. Now we analyse $\Pr[E_2]$, which equals to $\Pr[E_1^+]$ in **Game** 1. Suppose there are q_h queries to H, and define the following events: Evn' as the event that a query of the form $*||dh||*$ has been queried, Evn'' as Evn' does not happen, E_2' as $b = b'$ when Evn' happens, and E_2'' as $b = b'$ when Evn'' happens. Therefore, we have the following result,

$$\begin{aligned}\Pr[E_2] &= \Pr[E_2'] \cdot \Pr[Evn'] + \Pr[E_2''] \cdot \Pr[Evn''] \\ &= \Pr[E_2'] \cdot \Pr[Evn'] + \frac{1}{2} \cdot (1 - \Pr[Evn']) \,. \\ &= \Pr[Evn'] \cdot (\Pr[E_2'] - \frac{1}{2}) + \frac{1}{2}\end{aligned}$$

If Evn' happens, then we can solve the CDH problem with probability $\frac{1}{q_h}$, then $\left(\frac{1}{q_h}\right) \cdot Pr[Evn'] \leq \epsilon_{CDH}$, where ϵ_{CDH} is the upper bound of solving CDH problem by a P.P.T. adversary.

Summarizing the results from the above three games, we have the following:

$$\begin{aligned}\Pr[E_0] &= \Pr[E_1^*] \cdot \Pr[Evn^*] + \Pr[E_1^+] \cdot \Pr[Evn^+] \\ &= \Pr[E_1^*] \cdot \Pr[Evn^*] + \Pr[E_1^+] \cdot (1 - \Pr[Evn^*]) \\ &= \Pr[Evn^*] \cdot (\Pr[E_1^*] - \Pr[E_1^+]) + \Pr[E_1^+]\end{aligned}$$

$$\begin{aligned}Adv^{SK}(\mathcal{A}) &= \left|\Pr[E_0] - \frac{1}{2}\right| \\ &= \left|\Pr[Evn^*] \cdot (\Pr[E_1^*] - \Pr[E_1^+]) + \Pr[E_1^+] - \frac{1}{2}\right| \\ &\leq \left|\Pr[Evn^*] \cdot (\Pr[E_1^*] - \Pr[E_1^+])\right| + \left|\Pr[E_1^+] - \frac{1}{2}\right| \\ &= \left|\Pr[Evn^*] \cdot (\Pr[E_1^*] - \Pr[E_1^+])\right| + \left|\Pr[Evn'] \cdot (\Pr[E_2'] - \frac{1}{2}) + \frac{1}{2} - \frac{1}{2}\right| \\ &= \left|\Pr[Evn^*] \cdot (\Pr[E_1^*] - \Pr[E_1^+])\right| + \Pr[Evn'] \cdot \left|(\Pr[E_2'] - \frac{1}{2})\right| \\ &\leq \left|\Pr[Evn^*] \cdot (\Pr[E_1^*] - \Pr[E_1^+])\right| + q_h \cdot \epsilon_{CDH} \cdot \left|(\Pr[E_2'] - \frac{1}{2})\right|\end{aligned}$$

Since $Pr[Evn^*]$ is negligible based on the SDH assumption in the random oracle, then $Adv^{PFS}(\mathcal{A})$ is negligible based on the same assumption.

If we assume that $\mathcal{A}$ issues the $Test$ query to an instance $\pi_{\boldsymbol{s}}^j$ in step 3 of the game. The proof procedure is similar, except that $Pr[Evn^*]$ is negligible in **Game** 1 based on the CDH assumption in the random oracle as long as $\frac{\binom{n-z}{z}}{\binom{L-z}{z}}$ is negligible (namely, it is based on Lemma 1). As a result, $Adv^{SK}(\mathcal{A})$ is negligible based on the same assumption.

To sum up both cases, $Adv^{SK}(\mathcal{A})$ is negligible based on the SDH assumption (which is clearly stronger than CDH) in the random oracle as long as $\frac{\binom{n-z}{z}}{\binom{L-z}{z}}$ is negligible.

□

5 Performance Evaluation and Enhancements

In Table 2, we summarize the asymptotic complexity (i.e. the number of different types of computations) of the proposed protocol.

Table 2. Complexity of the Proposed Protocol

	Modular Exponentiation	**Multiplication**	**Addition**	**PRF** E	**PRF** F	**Hash** H
Tag Generation	-	L	L	-	L	L
IoT Device	3	2z	2z	1	4z	5
Server	3	4z + 1	4z - 2	1	2z	2z + 5

In the rest of this section, we implement our protocol and provide the detailed running time. Furthermore, we show how to reduce the running time for the IoT device. At last, we make a comparison to the protocol from [9] and resolve the scalability question.

5.1 Parameter Selection and Implementation Results

We consider two security levels, namely $\lambda = 128$ and $\lambda = 256$. Next, we first describe how to set up the parameters and then present the implementation results.

5.1.1 Parameter Setup

With respect to the instantiation of group $\mathbb{G}$, we use the Koblitz curve secp256k1 and secp521r1, respectively. These curves are recommended parameters defined in Standards for Efficient Cryptography [33], and the parameters can be found in Appendix B. When $\lambda = 256$, we use SHA-256 to implement the function H and use HMAC-SHA256 to instantiate the PRF E. For the PRF F, we can also use HMAC-SHA256 by truncating its output size to q. When $\lambda = 128$, we can further truncate the outputs of these functions to fit into the required domain. We skip the detail here.

Given a security parameter, we take the following approach to determine the parameters L, n, z.

1. Since we rely on the bounded retrieval model, we need to first set a threshold τ, which limits how much data an attacker can retrieve if it has compromised the server. For our implementation, we suppose the attacker can only retrieve $\tau = 100$ MB data. It will be straightforward to adapt our discussions to other τ values.
2. With τ, we enumerate some potential sizes for a single data item in $\mathbb{D}$. Let the sizes be denoted as x_i $(1 \leq i \leq T)$ for some integer T. For every x_i, we do the following.
   (a) Compute $n_i = \frac{\tau}{x_i}$, which represents the number of tuples an attacker can retrieve if it has compromised the server.

(b) With n_i, we need to try different (z, L) pairs so that $\frac{\binom{n}{z}}{\binom{L}{z}}$ is negligible w.r.t the security parameter. Note that a smaller z requires a larger L.

(c) Evaluate the the obtained (z, L) pairs, and try to find the one which results in a good balance between the size of $\mathbb{D}$ (i.e. $L \times x_i$) and the complexity of $2z$ hash computations. In another word, both z and L should not be too large. It is also worth noting that if z is very large, there is also the cost of multiplications in the computation of X and Y for the server (this makes a difference in our case of $\lambda = 256$, see below).

3. Further evaluate the (z, L) pairs for all x_i $(1 \leq i \leq T)$ obtained at the end of last step, and select the most suitable one.

In the following table, we enumerate five options for the size of a single data item in $\mathbb{D}$ and present the hashing time and the value for n correspondingly. All the computations are done with a PC, with its configurations described in the next subsection.

Table 3. Data Item Sizes

Data item size (MB)	0.0005	0.001	0.01	0.1	1
Hash one data item (ms)	0.004	0.007	0.068	0.689	6.698
Value of n	10^6	10^5	10^4	10^3	10^2

For the security parameter $\lambda = 128$, we compute the (z, L) pairs shown in Table 4, where the top row lists different data item size and n value tuples. Each pair guarantees that $\frac{\binom{n}{z}}{\binom{L}{z}} \leq \frac{1}{2^{128}}$. In the table, we have also presented the computation time for $2z$ hashes and the storage for the server. From the table, it seems the most appropriate pair is $(z = 50, L = 5896957)$, which achieves a better balance for hashing time and storage for the server. If storage is more important for the server, then $(z = 50, L = 589588)$ can be the alternative.

Table 4. Determine (z, L) when $\lambda = 128$

	(0.0005, 10^6)	**(0.001, 10^5)**	**(0.01, 10^4)**	**(0.1, 10^3)**	**(1, 10^2)**
Value of L when $z = 10$	7131518127	713122933	71283411	7099433	680759
Size of $\mathbb{D}$ (GB)	3565.76	713.12	712.83	709.94	680.76
Hash $2z$ data items (ms)	0.075	0.14	1.38	13.91	134.02
Value of L when $z = 20$	84447713	8444058	843693	83655	7637
Size of $\mathbb{D}$ (GB)	42.22	8.44	8.43	8.37	7.64
Hash $2z$ data items (ms)	0.15	0.27	2.73	26.89	267.91
Value of L when $z = 50$	**5896957**	589588	58851	5777	462
Size of $\mathbb{D}$ (GB)	**2.95**	0.59	0.59	5.78	0.46
Hash $2z$ data items (ms)	**0.36**	0.72	6.77	67.49	669.29
Value of L when $z = 100$	2428319	242769	24214	2357	147
Size of $\mathbb{D}$ (GB)	1.21	0.24	0.24	0.24	0.15
Hash $2z$ data items (ms)	0.73	1.45	13.73	135.31	1342.25

For the security parameter $\lambda = 256$, we compute the (z, L) pairs shown in Table 5. Each pair guarantees that $\frac{\binom{n}{z}}{\binom{L}{z}} \leq \frac{1}{2^{256}}$. It may seem that $(z = 100, L = 5896835)$ is a good choice. However, considering also the costs in multiplications (see our explanation in Step 2.(c)) from the aforementioned approach description, the most appropriate pair is $(z = 50, L = 3476724)$ in this case. In addition, we do not choose $(z = 50, L = 34774689)$ because the storage is too high.

Table 5. Determine (z, L) when $\lambda = 256$

	(0.0005, 10^6)	**(0.001, 10^5)**	**(0.01, 10^4)**	**(0.1, 10^3)**	**(1, 10^2)**
Value of L when $z = 10$	50858779596503	5085671978593	508361198100	50629932185	4854873253
Dataset size (GB)	25429389.80	5085671.98	5083611.98	5062993.22	4854873.25
Hash $2z$ data items (ms)	0.081	0.14	1.37	14.11	142.14
Value of L when $z = 20$	7131482474	713087280	71247750	7063691	644102
Size of $\mathbb{D}$ (GB)	3565.74	713.09	712.48	706.37	644.10
Hash $2z$ data items (ms)	0.16	0.29	2.76	25.47	265.13
Value of L when $z = 50$	34774689	**3476724**	346928	33945	2601
Size of $\mathbb{D}$ (GB)	17.39	**3.48**	3.47	3.39	2.60
Hash $2z$ data items (ms)	0.36	**0.89**	6.67	64.22	673.19
Value of L when $z = 100$	5896835	589466	58729	5653	276
Size of $\mathbb{D}$ (GB)	2.95	0.59	0.06	0.57	0.28
Hash $2z$ data items (ms)	0.71	1.44	14.21	139.64	1324.91

5.1.2 Implementation Results

With the selected parameters from the previous subsection, we implement the proposed AKE protocol in C with the MIRACL cryptographic library [30][1]. In the experiment, we use a PC as the server. It has an Intel® Core™ i7-4770 CPU @ 3.4 GHz processor with 16 GB RAM. In the literature, most benchmarks are implemented by using a single-board computer to simulate an IoT device, like Arduino, BeagleBone Black, Raspberry Pi, etc. [14, 24, 25]. Therefore, we use a Raspberry Pi 3 Model B+ with ARM Cortex-A53 @ 1.4 GHz processor and 1 GB RAM as the IoT device. To obtain fair execution results, we execute the codes ten times and take the average. Table 6 depicts the results. From the table, we observe that the running time of the server is much smaller than that of the IoT device. This may look strange, but it could be preferable in practice given that the server may need to support a large number of IoT devices. For the IoT device, the running time 15.19 ms (when $\lambda = 128$) and 84.73 ms (when $\lambda = 256$) could be acceptable in many application scenarios. But, it will be interesting to reduce this complexity, particularly for IoT devices which have less computing power than the Raspberry Pi.

[1] Source code is available at https://github.com/n00d1e5/Demo_Bigdata-facilitated_Two-party_AKE_for_IoT

Table 6. Running Time (ms)

	Modular Exponentiation	**Multiplication**	**Addition**	**PRF** E	**PRF** F	**Hash** H	**Total**
$\lambda = 128$, Elliptic Curve: secp256k1, $L = 5896957, n = 10^6, z = 50$							
IoT Device	13.61	0.16	0.03	0.04	0.66	0.06	**15.19**
Server	1.69	0.04	0.01	0.04	0.06	0.37	**2.44**
$\lambda = 256$, Elliptic Curve: secp521r1, $L = 3476724, n = 10^5, z = 50$							
IoT Device	81.96	0.34	0.05	0.04	0.96	0.12	**84.73**
Server	11.08	0.09	0.02	0.02	0.07	0.73	**12.51**

5.2 Efficiency Enhancement for the IoT

Referring to the AKE protocol in Figure 2 from Section 3.2, the values of a and g' can be computed in advance. By doing so, the IoT device can avoid about two-thirds of the computations required in the protocol execution.

Besides the "trivial" pre-computation strategy, we can try to offload one exponentiation from IoT to the server. For the enhanced AKE protocol, the initialisation phase stays the same except the following.

- IoT device is configured with a set $\mathbb{S}$ which contains tuples in the form of (g^u, u) where $u \xleftarrow{\$} \mathbb{Z}_q^*$. This allows us to outsource the computation of g^{r_1} to the server without revealing r_1.
- The original (pk, sk) is discarded and a new key pair (spk, ssk) is generated for a signature scheme $(\mathsf{Sign}, \mathsf{Verify})$ which achieves *existential unforgeability under chosen-message attacks* (EUF-CMA). This is necessary to achieve KCI resilience when the attacker compromises the IoT device.

The Enhanced AKE protocol is summarized in Figure 3.

Regarding the security of the enhanced protocol, we sketch below how the Lemmas from Section 4 still hold. First of all, the results of Lemma 1 and Lemma 3 will not be affected because compromising the server does not give the attacker any more privileges. Intuitively, our modification against the original AKE protocol from Section 3 does not give any more power to the attacker when it compromises the server, i.e. in the original case, it obtains sk while in the enhanced protocol it obtains ssk. In addition, the security results mainly come from the inability for the attacker to forge bigdata-related information. The new security proofs can be carried out in a very similar manner, we skip the details here. Regarding Lemma 2, the original proof methodology does not work anymore. In fact, this is why we have introduced the digital signature scheme. Since the server is required to send σ which is a signature for the exchanged messages, i.e. ① and M_2 which embeds $(b, \mathbb{I}_{\boldsymbol{s}}, X)$ inside. Based on the EUF-CMA property, the attacker cannot impersonate the server to the IoT device even if it has compromised the latter. Give that the results of Lemma 1, Lemma 2, and Lemma 3 still hold, then Lemma 4 also holds and the proof stays very similar. We skip the details here as well. It worth pointing out that, in comparison to the original protocol, one potential drawback for this enhanced protocol is the lack of *backward secrecy*, which means that an attacker can obtain the session keys

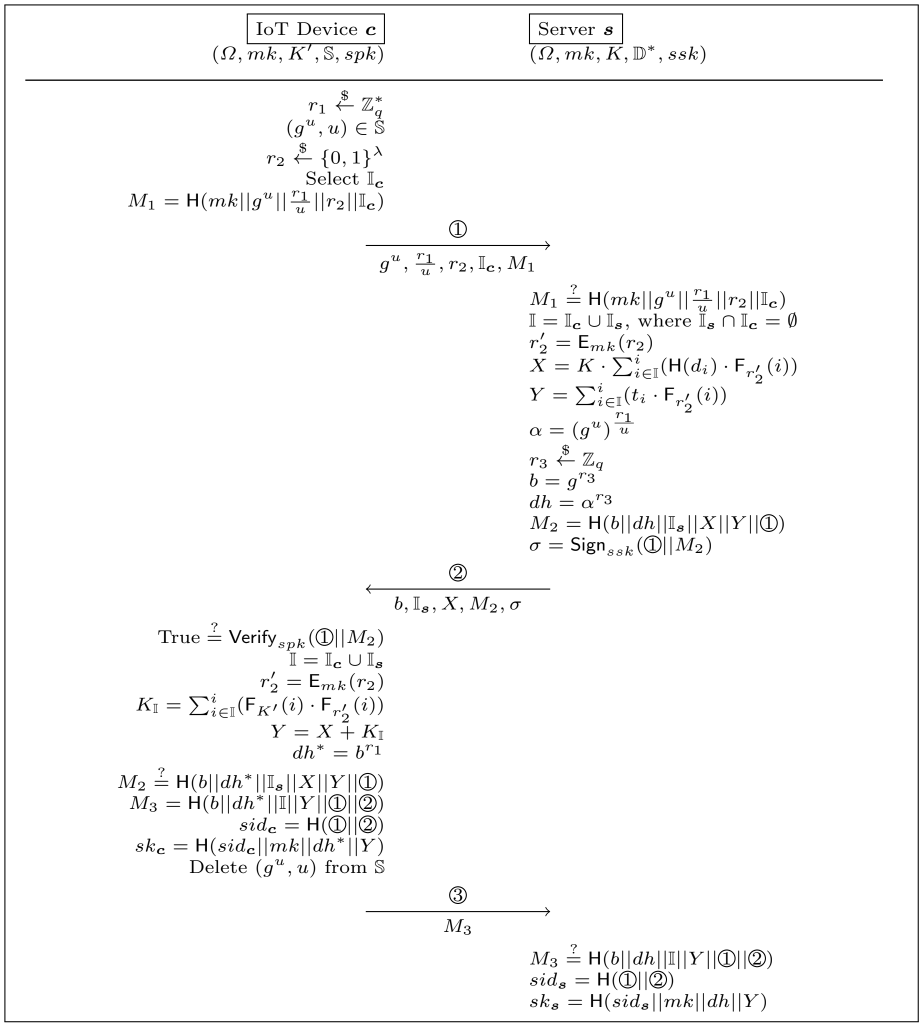
IoT Device $\boldsymbol{c}$
$(\Omega, mk, K', \mathbb{S}, spk)$
Server $\boldsymbol{s}$
$(\Omega, mk, K, \mathbb{D}^*, ssk)$
$r_1 \xleftarrow{\$} \mathbb{Z}_q^*$
$(g^u, u) \in \mathbb{S}$
$r_2 \xleftarrow{\$} \{0,1\}^\lambda$
Select $\mathbb{I}_{\boldsymbol{c}}$
$M_1 = \mathsf{H}(mk||g^u||\frac{r_1}{u}||r_2||\mathbb{I}_{\boldsymbol{c}})$
①
$g^u, \frac{r_1}{u}, r_2, \mathbb{I}_{\boldsymbol{c}}, M_1$
$M_1 \overset{?}{=} \mathsf{H}(mk||g^u||\frac{r_1}{u}||r_2||\mathbb{I}_{\boldsymbol{c}})$
$\mathbb{I} = \mathbb{I}_{\boldsymbol{c}} \cup \mathbb{I}_{\boldsymbol{s}}$, where $\mathbb{I}_{\boldsymbol{s}} \cap \mathbb{I}_{\boldsymbol{c}} = \emptyset$
$r_2' = \mathsf{E}_{mk}(r_2)$
$X = K \cdot \sum_{i \in \mathbb{I}}^{i} (\mathsf{H}(d_i) \cdot \mathsf{F}_{r_2'}(i))$
$Y = \sum_{i \in \mathbb{I}}^{i} (t_i \cdot \mathsf{F}_{r_2'}(i))$
$\alpha = (g^u)^{\frac{r_1}{u}}$
$r_3 \xleftarrow{\$} \mathbb{Z}_q$
$b = g^{r_3}$
$dh = \alpha^{r_3}$
$M_2 = \mathsf{H}(b||dh||\mathbb{I}_{\boldsymbol{s}}||X||Y||①)$
$\sigma = \mathsf{Sign}_{ssk}(①||M_2)$
②
$b, \mathbb{I}_{\boldsymbol{s}}, X, M_2, \sigma$
True $\overset{?}{=} \mathsf{Verify}_{spk}(①||M_2)$
$\mathbb{I} = \mathbb{I}_{\boldsymbol{c}} \cup \mathbb{I}_{\boldsymbol{s}}$
$r_2' = \mathsf{E}_{mk}(r_2)$
$K_{\mathbb{I}} = \sum_{i \in \mathbb{I}}^{i} (\mathsf{F}_{K'}(i) \cdot \mathsf{F}_{r_2'}(i))$
$Y = X + K_{\mathbb{I}}$
$dh^* = b^{r_1}$
$M_2 \overset{?}{=} \mathsf{H}(b||dh^*||\mathbb{I}_{\boldsymbol{s}}||X||Y||①)$
$M_3 = \mathsf{H}(b||dh^*||\mathbb{I}||Y||①||②)$
$sid_{\boldsymbol{c}} = \mathsf{H}(①||②)$
$sk_{\boldsymbol{c}} = \mathsf{H}(sid_{\boldsymbol{c}}||mk||dh^*||Y)$
Delete (g^u, u) from $\mathbb{S}$
③
M_3
$M_3 \overset{?}{=} \mathsf{H}(b||dh||\mathbb{I}||Y||①||②)$
$sid_{\boldsymbol{s}} = \mathsf{H}(①||②)$
$sk_{\boldsymbol{s}} = \mathsf{H}(sid_{\boldsymbol{s}}||mk||dh||Y)$

Fig. 3. Enhanced AKE Protocol

established after it compromised the IoT (due to the fact that it can obtains r_1 through u). How to address this issue is an interesting future work.

Table 7 shows the complexity of the enhanced protocol. Comparing to Table 2, we can conclude that the signature verification operation should be very efficient in order to make the outsourcing meaningful.

To implement the enhanced AKE protocol, all parameters can stay the same except that we need to choose an appropriate digital signature scheme. According to NIST's benchmarking [16], we choose *Picnic*[29]. We benchmark the schemes

Table 7. Complexity of the Enhanced Protocol

	Sign	**Verify**	**Modular Exponentiation**	**Multiplication**	**Addition**	**PRF** E	**PRF** F	**Hash** H
IoT Device	-	1	1	2z	2z	1	4z	5
Server	1	-	3	4z + 1	4z - 2	1	2z	2z + 5

both for 128 and 256-bit security[2] on our Raspberry Pi and get the results in Table 8. For 128-bit security, the running time remains almost the same for the IoT device as the original solution, while the running time for the server has increased significantly. But, for 256-bit security, the execution time is 1.45 times faster for the IoT device while the running time for the server has increased moderately.

Table 8. Running Time (ms)

	Sign	**Verify**	**Modular Exp.**	**Multiplication**	**Addition**	**PRF** E	**PRF** F	**Hash** H	**Total**
$\lambda = 128$, Elliptic Curve: secp256k1, $L = 5896957, n = 10^6, z = 50$									
IoT Device	-	8.32	4.54	0.16	0.03	0.04	0.66	0.06	**14.44**
Server	10.34	-	1.69	0.04	0.01	0.04	0.06	0.37	**12.78**
$\lambda = 256$, Elliptic Curve: secp521r1, $L = 3476724, n = 10^5, z = 50$									
IoT Device	-	28.36	27.32	0.34	0.05	0.04	0.96	0.12	**58.45**
Server	33.83	-	11.08	0.09	0.02	0.02	0.07	0.73	**46.34**

Furthermore, it is clear that if there is a more efficient signature scheme, then the efficiency gain will be more for the enhanced AKE protocol. To further improve its efficiency, we can also try to outsource the computation of $dh^* = b^{r_1}$ to the server. To this end, *Protocol 5* from [15] can be employed. A detailed investigation of this direction is an interesting future work.

5.3 Comparison with Existing Protocol(s)

Regarding the setting mentioned in the beginning of Section 1, the protocol from [9] is similar to ours, even though it does not achieve the KCI and SCI properties. We choose 128-bit AES to instantiate the encryption algorithm and use the same group to implement this protocol for the security level $\lambda = 128$, and summarize the results in Table 9. In comparison to the results from Table 6, it is clear that the complexities for the IoT device are very close while the complexity for the server is slightly higher in our protocol. Note also the fact that we have not taken into account the PUF operations, which may increase the complexity for the IoT device for the protocol from [9].

In this paper, we have assumed a setting with one IoT device and the server. In practice, the server may serve a large number of IoT devices, e.g. thousands

[2] Source code of both schemes picnic-L1-full for 128-bit security and picnic-L5-full for 256-bit security is available at https://github.com/IAIK/Picnic

Table 9. Complexity and Running Time (ms) of [9]

	Modular Exponentiation	**Encryption**	**Decryption**	**Hash** H	**Total**
IoT Device	13.61	0.00006	-	0.06	$\approx$ 13.67
Server	1.69	0.000006	0.000006	0.02	$\approx$ 1.71

of them. In this case, the security properties will not be affected in any manner, but there are potential scalability concerns. From our implementation results, the running time of the server can scale to a considerable number of IoT devices, and the main concern is the storage. Below, we propose a simple solution to resolve this question.

Instead of storing an individual dataset $\mathbb{D}$ for every IoT device, the server can store a global dataset $\tilde{\mathbb{D}}$ which contains L data items $\tilde{d_i}$ $(1 \leq i \leq L)$. In addition, the server can generate a global secret key gk for dataset configuration. Consider an IoT device, which has the identifier id. For both the *Initialisation Phase* and the AKE protocol, the server can construct a dataset $\mathbb{D}$ for this IoT device on-the-fly, where every element d_i is derived from $\tilde{d_i}$ as follows:

$$d_i = \tilde{\mathsf{H}}(id||gk||\tilde{d_i})$$

where $\tilde{\mathsf{H}}$ is a hash function. By doing so, the server only needs to store the global dataset $\tilde{\mathbb{D}}$ and ephemerally generate $\mathbb{D}$ when necessary. When the cryptographic hash function $\tilde{\mathsf{H}}$ is modeled as a random oracle, it is straightforward to verify that the security properties of the proposed protocols will not be affected. Due to the space limitation, we skip the details here.

6 Conclusion

Motivated by Chan et al.'s unilateral authentication scheme [14], we have proposed a bigdata-facilitated two-party AKE protocol for IoT systems. The proposed protocol achieves a wide range of security properties including PFS, KCI resilience and SCI resilience. In particular, the KCI and SCI resilience properties are well demanded by the IoT environment, and cannot be satisfied by existing AKE protocols. Furthermore, we have presented an enhanced protocol, which can significantly reduce the computation load for the IoT with an appropriate signature scheme. Our work has left a number of future research directions. As mentioned in Section 5.2, it is an immediate future work to give a formal proof of the enhanced AKE protocol even if it is almost straightforward. Furthermore, it is worth investigating to further improve the efficiency of the protocol by integrating the *Protocol 5* from [15] and evaluate its efficiency gain vs the added communication complexity. Along this direction, it is also worth exploring other signature schemes which have better verification efficiency.

Acknowledgement

This paper is supported in the context of the project CATALYST funded by Fonds National de la Recherche Luxembourg (FNR, reference 12186579).

References

1. Abdalla, M., Bellare, M., Rogaway, P.: The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES. In: Naccache, D. (ed.) Topics in Cryptology - CT-RSA 2001. LNCS, vol. 2020, pp. 143–158. Springer (2001)
2. Alwen, J., Dodis, Y., Wichs, D.: Leakage-resilient public-key cryptography in the bounded-retrieval model. In: Annual International Cryptology Conference. pp. 36–54. Springer (2009)
3. Aumann, Y., Ding, Y.Z., Rabin, M.O.: Everlasting security in the bounded storage model. IEEE Transactions on Information Theory **48**(6), 1668–1680 (2002)
4. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Annual international cryptology conference. pp. 232–249. Springer (1993)
5. Blake-Wilson, S., Menezes, A.: Authenticated Diffe-Hellman key agreement protocols. In: International Workshop on Selected Areas in Cryptography. pp. 339–361. Springer (1998)
6. Boyd, C., Mathuria, A., Stebila, D.: Protocols for Authentication and Key Establishment. Springer Berlin, Heidelberg (2020)
7. Brainard, J., Juels, A., Rivest, R.L., Szydlo, M., Yung, M.: Fourth-factor authentication: somebody you know. In: Proceedings of the 13th ACM conference on Computer and communications security. pp. 168–178 (2006)
8. Byun, J.W.: A generic multifactor authenticated key exchange with physical unclonable function. Security and Communication Networks **2019** (2019)
9. Byun, J.W.: An efficient multi-factor authenticated key exchange with physically unclonable function. In: 2019 International Conference on Electronics, Information, and Communication (ICEIC). pp. 1–4. IEEE (2019)
10. Byun, J.W.: End-to-end authenticated key exchange based on different physical unclonable functions. IEEE Access **7**, 102951–102965 (2019)
11. Byun, J.W.: PDAKE: a provably secure PUF-based device authenticated key exchange in cloud setting. IEEE Access **7**, 181165–181177 (2019)
12. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 453–474. Springer (2001)
13. Challa, S., Wazid, M., Das, A.K., Kumar, N., Reddy, A.G., Yoon, E.J., Yoo, K.Y.: Secure signature-based authenticated key establishment scheme for future IoT applications. IEEE Access **5**, 3028–3043 (2017)
14. Chan, A.C.F., Wong, J.W., Zhou, J., Teo, J.: Scalable two-factor authentication using historical data. In: European Symposium on Research in Computer Security. pp. 91–110. Springer (2016)
15. Chevalier, C., Laguillaumie, F., Vergnaud, D.: Privately Outsourcing Exponentiation to a Single Server: Cryptanalysis and Optimal Constructions. Algorithmica **83**(1), 72–115 (2021)
16. Dang, V.B., Farahmand, F., Andrzejczak, M., Mohajerani, K., Nguyen, D.T., Gaj, K.: Implementation and benchmarking of round 2 candidates in the nist

post-quantum cryptography standardization process using hardware and software/hardware co-design approaches. Cryptology ePrint Archive: Report 2020/795 (2020)

17. Davies, S.G.: Touching Big Brother: How biometric technology will fuse flesh and machine. Information Technology & People **7**(4), 38–47 (1994)
18. Di Crescenzo, G., Lipton, R., Walfish, S.: Perfectly secure password protocols in the bounded retrieval model. In: Theory of Cryptography Conference. pp. 225–244. Springer (2006)
19. Dziembowski, S.: Intrusion-resilience via the bounded-storage model. In: Theory of Cryptography Conference. pp. 207–224. Springer (2006)
20. Fleischhacker, N., Manulis, M., Azodi, A.: A modular framework for multi-factor authentication and key exchange. In: International Conference on Research in Security Standardisation. pp. 190–214. Springer (2014)
21. Guo, C., Chang, C.C.: Chaotic maps-based password-authenticated key agreement using smart cards. Communications in Nonlinear Science and Numerical Simulation **18**(6), 1433–1440 (2013)
22. Hao, F., Clarke, D.: Security analysis of a multi-factor authenticated key exchange protocol. In: International Conference on Applied Cryptography and Network Security. pp. 1–11. Springer (2012)
23. Jin, C., Yang, Z., Adepu, S., Zhou, J.: HMAKE: Legacy-Compliant Multi-factor Authenticated Key Exchange from Historical Data. IACR Cryptology ePrint Archive **2019**, 450 (2019)
24. Kruger, C.P., Hancke, G.P.: Benchmarking Internet of things devices. In: 2014 12th IEEE International Conference on Industrial Informatics (INDIN). pp. 611–616. IEEE (2014)
25. Krylovskiy, A.: Internet of things gateways meet linux containers: Performance evaluation and discussion. In: 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT). pp. 222–227. IEEE (2015)
26. Lee, Y., Kim, S., Won, D.: Enhancement of two-factor authenticated key exchange protocols in public wireless LANs. Computers & electrical engineering **36**(1), 213–223 (2010)
27. Li, Z., Yang, Z., Szalachowski, P., Zhou, J.: Building Low-Interactivity Multi-Factor Authenticated Key Exchange for Industrial Internet-of-Things. IEEE Internet of Things Journal (2020)
28. Liu, Y., Xue, K.: An improved secure and efficient password and chaos-based two-party key agreement protocol. Nonlinear Dynamics **84**(2), 549–557 (2016)
29. Microsoft: The Picnic Signature Algorithm, https://github.com/microsoft/Picnic/
30. MIRACL Ltd.: Multiprecision Integer and Rational Arithmetic Cryptographic Library – the MIRACL Crypto SDK (2019), https://github.com/miracl/MIRACL
31. Pointcheval, D., Zimmer, S.: Multi-factor authenticated key exchange. In: International Conference on Applied Cryptography and Network Security. pp. 277–295. Springer (2008)
32. Shoup, V.: On Formal Models for Secure Key Exchange . Cryptology ePrint Archive, Report 1999/012 (1999), https://eprint.iacr.org/1999/012
33. Standards for Efficient Cryptography (SEC): SEC 2: Recommended elliptic curve domain parameters (2000)
34. Stebila, D., Udupi, P., Chang Shantz, S.: Multi-factor password-authenticated key exchange. Information Security 2010 pp. 56–66 (2010)

A Review of Chan et al. [14]

A.1 The Authentication Scheme of Chan et al.

The following arithmetic is done in a certain finite field $\mathbb{F}$. Their protocol has a cryptographic hash function $\mathsf{H} : \{0,1\}^* \rightarrow \mathbb{F}$ and two Pseudorandom Functions (PRFs) $\mathsf{F} : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \mathbb{F}$, $\mathsf{E} : \{0,1\}^\lambda \times \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda$ where λ is a security parameter. In the initialization phase, a secret key $sk_1 = mk \in \{0,1\}^\lambda$ is randomly generated and shared between a verifier $\boldsymbol{v}$ and a prover $\boldsymbol{p}$. In our setting, $\boldsymbol{v}$ can be an IoT device and $\boldsymbol{p}$ can be a server. In addition, $\boldsymbol{v}$ also holds the the other secret keys $sk_2 = (K, K')$, where $K \in \mathbb{F}^*$ and $K' \in \{0,1\}^\lambda$ ($K' \in \{0,1\}^*$ in [14]). Assume there is a big dataset D, we briefly summarize the tag generation in Figure 4 and the authentication protocol execution in Figure 5. Note that the cardinality L of the big dataset could be generated by some random number generator with a seed s chosen by $\boldsymbol{v}$ or just indices spaced equally.

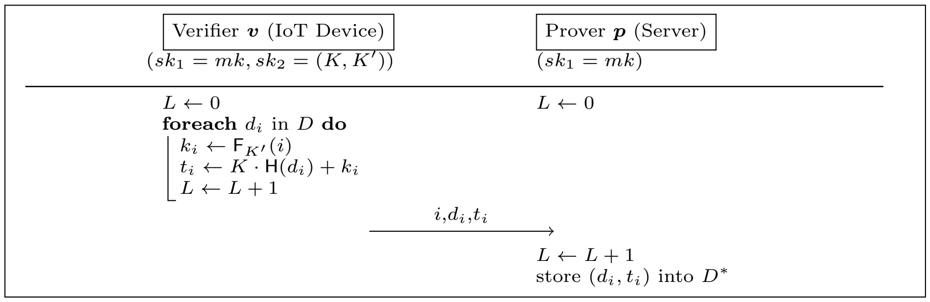
Verifier $\boldsymbol{v}$ (IoT Device)
$(sk_1 = mk, sk_2 = (K, K'))$
Prover $\boldsymbol{p}$ (Server)
$(sk_1 = mk)$
$L \leftarrow 0$
foreach d_i in D **do**
$k_i \leftarrow \mathsf{F}_{K'}(i)$
$t_i \leftarrow K \cdot \mathsf{H}(d_i) + k_i$
$L \leftarrow L + 1$
$\xrightarrow{i, d_i, t_i}$
$L \leftarrow 0$
$L \leftarrow L + 1$
store (d_i, t_i) into D^*

Fig. 4. Tag Generation of Chan et al.'s Authentication Protocol [14]

A.2 Tag Stealing Attack

In [23], a vulnerability is shown that all tags can be leaked to adversaries. Once an adversary $\mathcal{A}$ has corrupted the first authentication factor mk and knows one tuple (d_i, t_i) (or $(\mathsf{H}(d_i), t_i)$) of D^*, then $\mathcal{A}$ can launch the stealing attack according to the following steps:

1. $\mathcal{A}$ generates a set I_1 with t random indices, then let I_2 denote the other set by replacing one chosen index (e.g., j) with i.
2. $\mathcal{A}$ sends (r, I_1) to $\boldsymbol{p}$ and receives (X_1, Y_1).
3. $\mathcal{A}$ sends (r, I_2) to $\boldsymbol{p}$ in another session and receives (X_2, Y_2).
4. $\mathcal{A}$ can now gain the knowledge of the tag $t_j = \frac{Y_2 - Y_1 + \mathsf{F}_{r'}(i) \cdot t_i}{\mathsf{F}_{r'}(j)}$ and $\mathsf{H}(d_j) = \frac{X_2 - X_1 + \mathsf{F}_{r'}(i) \cdot \mathsf{H}(d_i)}{\mathsf{F}_{r'}(j)}$, where $r' = \mathsf{E}_{mk}(r)$.
5. By repeating the steps above, $\mathcal{A}$ is able to steal all tags and hash values of all original data items.

$$
\begin{array}{rcl}
\boxed{\text{Verifier } \boldsymbol{v} \text{ (IoT Device)}} & & \boxed{\text{Prover } \boldsymbol{p} \text{ (Server)}} \\
(sk_1 = mk, sk_2 = (K, K'), L) & & (sk_1 = mk, L, D^*) \\
\hline
\text{a set with } t \text{ random indices } I \overset{\$}{\leftarrow} L & & \\
r \overset{\$}{\leftarrow} \{0,1\}^\lambda & & \\
& \xrightarrow{\quad I, r \quad} & \\
& & r' \leftarrow \mathsf{E}_{mk}(r) \\
& & X \leftarrow \sum_{i \in I}^{i} (\mathsf{F}_{r'}(i) \cdot \mathsf{H}(d_i)) \\
& & Y \leftarrow \sum_{i \in I}^{i} (\mathsf{F}_{r'}(i) \cdot t_i) \\
& \xleftarrow{\quad X, Y \quad} & \\
r'' \leftarrow \mathsf{E}_{mk}(r) & & \\
K_I \leftarrow \sum_{i \in I}^{i} (\mathsf{F}_{K'}(i) \cdot \mathsf{F}_{r''}(i)) & & \\
Y \overset{?}{=} K \cdot X + K_I & &
\end{array}
$$

Fig. 5. Execution of Chan et al.'s Authentication Protocol [14]

B Implementation Parameters

The Koblitz curve secp256k1 is given by $E/\mathbb{F}_q : y^2 = x^3 + ax + b$ with prime group order q, while the corresponding parameters are stated as

```
p :=  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F,
a :=  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000,
b :=  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000007,
G := (79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798,
      483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8),
q :=  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141,
h :=  01.
```

And the curve secp521r1 is defined by changing the following parameters

```
p :=  01FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
           FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF,
a :=  01FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
           FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFC,
b :=  0051 953EB961 8E1C9A1F 929A21A0 B68540EE A2DA725B 99B315F3 B8B48991 8EF109E1
           56193951 EC7E937B 1652C0BD 3BB1BF07 3573DF88 3D2C34F1 EF451FD4 6B503F00,
G := (00C6 858E06B7 0404E9CD 9E3ECB66 2395B442 9C648139 053FB521 F828AF60 6B4D3DBA
           A14B5E77 EFE75928 FE1DC127 A2FFA8DE 3348B3C1 856A429B F97E7E31 C2E5BD66,
      0118 39296A78 9A3BC004 5C8A5FB4 2C7D1BD9 98F54449 579B4468 17AFBD17 273E662C
           97EE7299 5EF42640 C550B901 3FAD0761 353C7086 A272C240 88BE9476 9FD16650),
q :=  01FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFA
           51868783 BF2F966B 7FCC0148 F709A5D0 3BB5C9B8 899C47AE BB6FB71E 91386409.
```