symmetry

MDPI

Article

CAKE: Compatible Authentication and Key Exchange Protocol for a Smart City in 5G Networks

Yun-Hsin Chuang and Yuh-Min Tseng *

Department of Mathematics, National Changhua University of Education, Changhua 500, Taiwan; d00921014@ntu.edu.tw

* Correspondence: ymtseng@cc.ncue.edu.tw

Abstract: In a smart city, there are different types of entities, such as nature persons, IoT devices, and service providers, which have different computational limitations and storage limitations. Unfortunately, all of the existing authentication and key exchange (AKE) protocols are designed for either client–server or client–client authentication, including the ones designed for smart cities. In this paper, we present the idea of a compatible authentication and key exchange (CAKE) protocol which provides cross-species authentication. We propose the first CAKE protocol for a smart city that any two valid entities can authenticate with each other and create a secure session key without the help of any third party, while there is also no password table and no public key issuing problem. The entity can be a natural person having biometrics, an IoT device embedded with a physical unclonable function (PUF), or a service provider. Moreover, we extend the CAKE protocol to an anonymous CAKE (ACAKE) protocol, which provides natural persons an anonymous option to protect their privacy. In addition, both the proposed CAKE and ACAKE protocols can deal with the entity revocation problem. We define the framework and the security model of CAKE and ACAKE protocols. Under the security model, we formally prove that the proposed protocols are secure under the elliptic curve computational Diffie–Hellman (ECCDH) problem, the decisional bilinear Diffie–Hellman (DBDH) problem, and hash function assumptions. Comparisons with the related protocols are conducted to demonstrate the benefits of our protocols. Performance analysis is conducted and the experience results show that the proposed protocols are practical in a smart city.

Keywords: anonymity; biometric; IoT; key agreement; list-free; multi-factor authentication; multi-server; privacy; PUF; revocation

check for updates

Citation: Chuang, Y.-H.; Tseng, Y.-M. CAKE: Compatible Authentication and Key Exchange Protocol for a Smart City in 5G Networks. *Symmetry* **2021**, *13*, 698. https://doi.org/10.3390/sym13040698

Academic Editor: Basil Papadopoulos

Received: 25 March 2021
Accepted: 15 April 2021
Published: 16 April 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Smart technologies play an important role in our daily lives, such as smart cities, smart grids, smart homes, autonomous vehicles, drones, telemedicine, teleportation, and remote education. With the development of smart technologies, the number of Internet of Things (IoT)-connected devices is currently around 30 billion and will increase to around 75 billion in 2025 [1]. The fifth generation (5G) technology standard for cellular networks provides high speed, high capacity, extremely low latency, and a significant improvement in users' perceived quality of service (QoS), as compared to current 4G LTE networks [2]. Fifth generation networks are expected to be the panacea of the network issues of IoT [3–5]. Fifth generation-based IoT middleware systems have several security challenges [6], in which the authentication and key exchange (AKE) protocol is the most significant security issue, since the entities communicate to each other in untrustworthy networks (i.e., the Internet).

An AKE protocol allows two parties to authenticate mutually and create a secure common session key to communicate with each other. All of the existing AKE protocols are designed for either client–server or client–client authentication. None of the existing AKE protocols provide cross-species authentication, which allows cross-species entities to

authenticate mutually and create a secure session key. We first propose the concept of the compatible AKE (CAKE) protocol, which enables any two entities to mutually authenticate with each other over untrustworthy channels, for example, not only for client–server but also for client–client and server–server authentication.

1.1. Contributions

In this paper, we propose the first compatible AKE (CAKE) protocol for smart cities, which provides cross-species authentication; meanwhile, the proposed protocol achieves multi-factor authentication, suitability for a multi-server environment, independent authentication, member revocation, being password table-free, and being public key management-free. According to the guidelines for designing a secure anonymous AKE (AAKE) protocol [7], we furthermore extend the proposed CAKE protocol to an anonymous CAKE (ACAKE) protocol, which achieves user anonymity and user untraceability/unlinkability, in order to protect the privacy of natural persons in a smart city.

We define the adversary model of CAKE/ACAKE protocols and provide the formal security proofs of the proposed protocols. We compare the proposed CAKE/ACAKE protocols with the related AKE protocols and analyze the performance to show that the proposed CAKE/ACAKE protocols are efficient and practical in smart cities. We estimate the executing costs of an IoT device on a 32-bit Philips HiPerSmart^{TM} smart card with a maximum clock speed of 36MHz, a natural person on a Linux personal digital assistant equipped with a 206-MHz strong ARM processor and on an Intel PXA270 processor at 624-MHz installed on the Linux personal digital assistant, and a service provider and the registration center (RC) on a PIV 3-GHZ processor with 512-MB memory and a Windows XP operation system.

1.2. Paper Organization

The rest of the paper is organized as below. Section 2 presents the related work, and the preliminaries are presented in Section 3. We present the proposed protocols in Section 4 and analyze the security of the proposed protocols in Section 5. The comparisons and the performance analysis are given in Section 6, and the conclusion is drawn in Section 7.

2. Related Work

In the past, a large number of AKE protocols have been proposed. Here, we introduce some properties of these protocols that include multi-factor authentication, multi-server environment, independent authentication, user anonymity, user untraceability/unlinkability, and member revocation.

1. ***Multi-factor authentication:*** Remote authentication schemes were developed from the one-factor, i.e., password, authentication [8] to the two-factor and multi-factor authentication. In general, a two-factor authentication adopts smartcards as the additional second factor to avoid replay attacks, and a three-factor authentication adopts biometrics or physical unclonable functions (PUFs) as the additional third factor to avoid smartcard stolen attacks.
2. ***Suitability for a multi-server environment:*** Traditional client–server AKE protocols were proposed for single-server environments. Nowadays, there are many systems constructed in multi-server architectures. Many AKE protocols for multi-server environments were proposed and users need not to register at different servers.
3. ***Independent authentication:*** There is a trusted third party who needs to involve the authentication phase in many AKE protocols. On the contrary, some AKE protocols can achieve independent authentication in the sense that two entities can freely authenticate with each other by themselves without the help of any third party.
4. ***User anonymity:*** In an AAKE protocol, the real identity of a user would not be revealed to others when he/she logs into a server through an untrustworthy channel.
5. ***User untraceability/unlinkability:*** In ordinary AAKE schemes, even though a user uses an anonymous identity to log in, the relationship between each login is exposed

to a third party since the user uses a duplicate anonymous identity in each login. Some AAKE protocols not only achieve user anonymity but also achieve user untraceability/unlinkability, where a user cannot be traced by the login transmissions, i.e., no third party can derive the relationship between any two login transmissions. Without loss of generality, to avoid the physical location tracking in an AAKE scheme, users either can simply forge their location or use physical layer signatures [9] to simultaneously achieve privacy-preserving location authentication and user untraceability.

6. ***Member revocation:*** Many AKE protocols did not deal with the member revocation problem when a member is disabled or leaving. As compared with traditional revocable AKE protocols using a certificate revocation list (CRL), the advanced ones adopt time periods to deal with revocation problems, namely, a revoked or disabled member could not obtain the latest secret key in the next time period.
7. ***Table-free:*** No password table or verification list needs to be stored or maintained.
8. ***Public key issuing-free:*** No public key needs to be announced; that is, either there are no public keys in the system or the identity can be used as the public key.
9. ***Leakage-resilient:*** This property helps to resist side channel attacks.

According to the survey proposed by Chuang and Lei [10], there is no AKE protocol that simultaneously achieves all the properties mentioned above, and none of them provide cross-species authentication, which allows cross-species entities to authenticate mutually and create a secure session key.

Li et al. [11] proposed an authentication and key agreement scheme with user anonymity for a roaming service in a smart city in 2017, and their protocol did not provide multi-factor authentication, suitability for a multi-server environment, independent authentication, and member revocation. After that, both Li et al. [12] and Reddy et al. [13] proposed privacy-preserving authentication protocols for smart cities, which additionally achieved independent authentication as compared with Li et al.'s [11] protocol. In 2019, Xie and Hwang [14] proposed a two-factor anonymous roaming authentication scheme in a smart city, which additionally achieved suitability for a multi-server environment as compared with Li et al.'s [11] protocol. Jegadeesan et al. [15] proposed an anonymous mutual authentication technique for mobile cloud computing in a smart city in 2019, which additionally achieved independent authentication as compared with Xie and Hwang's protocol [14]. All of these schemes [11–15] do not provide compatible authentication. Although many existing anonymous AKE protocols [16–25] are suitable for a smart city, none of them provide cross-species authentication and none of them achieve all the properties mentioned above [10]. No CAKE protocol for a smart city has been proposed up to date.

3. Preliminaries

Here, we introduce physical unclonable functions (PUFs), fuzzy extractors, bilinear pairing, and hard problem assumptions.

3.1. Physical Unclonable Functions

A physical unclonable function (PUF) is a physical entity embedded in a physical device, such as a microprocessor, that provides a physically defined output (response) for a given input (challenge).

The challenge and the corresponding response are called the challenge response pair (CRP). PUFs provide alternate hardware fingerprints which can be used for lightweight authentication [26,27]. PUFs can be generally classified into strong PUFs and weak PUFs according to the number of possible CRPs [28]. A weak PUF only supports a small amount of CRPs in the sense that an attacker could obtain all the responses when it obtains temporary access to the device. Weak PUFs are typically used for key storage, since the responses are stable in the sense that a challenge always yields the same response in various environments and conditions. A strong PUF can support a large number of CRPs; hence, it is generally employed for authentication.

3.2. Fuzzy Extractors

A fuzzy extractor [29] or helper data algorithm can generate stable cryptographic keys with appropriate entropy from noisy and non-uniformly distributed random PUF responses [30–32].

Using the Bose–Chaudhuri–Hocquenghem (BCH) codes and the hash function can guarantee the elimination of noise from the collected noisy data and the uniform distribution of the derived key bits [29–32]. Kang et al.'s article [30] illustrates the implementation diagram for the efficient fuzzy extractor using the BCH code and syndrome concept (N = 255). The helper data along with a BCH decoder can be used to re-generate the correct response from the actual response of a PUF for a specific challenge.

Definition 1. *Let M be a metric space on N points with distance function* ***dis****, and* U_l *denote the uniform distribution on l-bit binary strings. An (M, δ, l, d, ε) fuzzy extractor is a pair of procedures (Gen, Rep) [29,30].*

1. *Gen* is a probabilistic generation procedure, which takes an input $R \in$ M and outputs an extracted string $SS \in \{0, 1\}^l$ and a helper string $HLP \in \{0, 1\}^*$. For any distribution D on M of min-entropy δ, if $(SS, HLP) \leftarrow Gen(D)$, we have **SD** $((SS, HLP), (U_l, HLP)) \leq \varepsilon$.
2. *Rep* is a deterministic reproduction procedure that can recover SS from the corresponding helper string HLP and any vector m' close to m, namely, for all $R', R \in$ M satisfying **dis** $(R', R) \leq d$, if $(SS, HLP) \leftarrow Gen(R)$, then $Rep\ (R', HLP) = SS$.

3.3. Bilinear Pairing

Let E be an elliptic curve over a finite field F_q, which is a field of integers modulo, and a large prime number q, and $E(F_q)$ denote the set of all the points on E. Let G_1 be an additive cyclic subgroup of points on $E(F_q)$, a point P be a generator of G_1, and G_2 be a multiplicative group with the same order q. A bilinear pairing [33–36] is a map, $\hat{e}$: $G_1 \times G_1 \rightarrow G_2$, which satisfies the following properties:

1. *Bilinear*: $\hat{e}(aQ, bR) = \hat{e}(Q, R)^{ab}$ for all $a, b \in {Z_q}^*$ and all $Q, R \in G_1$.
2. *Non-degenerate*: There are two points $Q, R \in G_1$ such that $\hat{e}(Q, R) \neq 1$.
3. *Computability*: There is an efficient algorithm to compute $\hat{e}(Q, R)$ for all $Q, R \in G_1$.

3.4. Hard Problem Assumptions

Three hard mathematical problems [35,36] used in this paper are listed below.

1. ***Elliptic curve discrete logarithm (ECDL)*** *problem*: Given a point aP in G_1, find the integer a.
2. ***Elliptic curve computational Diffie–Hellman (ECCDH)*** *problem*: Given P, aP, and bP in G_1, find abP in G_1.
3. ***Decisional bilinear Diffie–Hellman (DBDH)*** *problem*: Given P, aP, bP, and cP in G_1 and $\hat{e}(P, P)^d$ in G_2, determine whether $\hat{e}(P, P)^d = \hat{e}(P, P)^{abc}$.

Four security assumptions are listed below.

1. ***ECDL assumption:*** No probabilistic polynomial time (PPT) algorithm can solve the ECDL problem with a non-negligible advantage.
2. ***ECCDH assumption:*** No PPT algorithm can solve the ECCDH problem with a non-negligible advantage.
3. ***DBDH assumption:*** No PPT algorithm can solve the DBDH problem with a non-negligible advantage.
4. ***Hash function assumption:*** There is a one-way and collision-resistant cryptographic hash function H: $A \rightarrow B$ satisfying the following three conditions:
   - *Preimage resistance:* For any given $b \in B$, it is hard to find $a \in A$ such that $H(a) = b$.
   - *Second preimage resistance:* For any given $a \in A$, it is hard to find $c \in A$ such that $a \neq c$ and $H(a) = H(c)$.

- *Collision resistance:* It is hard to find $a, c \in A$ such that $a \neq c$ and $H(a) = H(c)$.

4. Proposed Protocols

In this section, the CAKE and ACAKE protocols for a smart city are proposed. In the proposed protocols, there are four kinds of components, namely, one registration center (*RC*), numerous service providers (Server), plenty of natural persons (*NP*), and a large number of IoT devices, which are depicted in Figure 1.

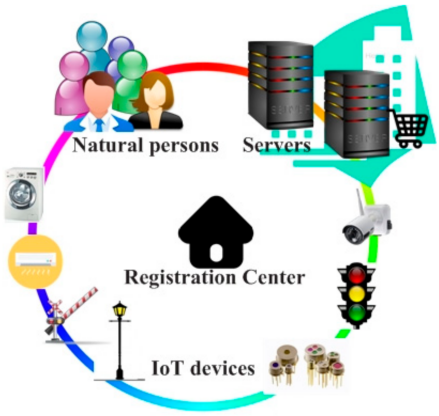


Figure 1. Components of CAKE/CAKE protocols.

RC is a trusted third party, which is responsible for the system setup and performing the registration, revocation, and key updating for members. Servers can have a variety of functions, such as providing services for *NP*s, monitoring and controlling IoT devices, and gathering and analyzing data from IoT devices. A server may be a cashier, an automated teller machine (ATM), a housekeeper, a web service provider, a cloud service provider, a medical service provider, a library, a school, a company, a bank, a city control center, a government department, etc. An IoT device is assumed to have embedded physical unclonable functions (PUFs) and low computational capability; it may be a sensor, electrical appliance, traffic signal, streetlight, surveillance camera, vehicle, etc. In addition, we summarize the notations in Table 1.

Table 1. Notations.

Symbol	Meaning	Symbol	Meaning	Symbol	Meaning
RC	Registration center	MSK	Master secret key	ID_α	Identity of member α
NP_i	i-th natural person	MRK	Master private key	pw_α	Password of member α
S_j	j-th server	MPK	Master public key	B_i	Biometric of NP_i
D_l	l-th IoT device	t	Index of time period	SK	Session key
C_A/R_A	Challenge/response string of PUF function	Gen/Rep	Generation/reproduction procedure of fuzzy extractor	K_{α_t}	Secret key of member α in t-time period
SS_i/HLP_i	Secret/helper string	$\|\|$	Concatenation operation	$\oplus$	XOR operation

4.1. Framework of CAKE and ACAKE Protocols

A fuzzy extractor [29] or helper data algorithm can generate stable cryptographic keys with appropriate entropy from noisy and non-uniformly distributed random PUF responses [30–32]. Here, we present the framework of the proposed CAKE and ACAKE protocols. The ACAKE protocol is an extended CAKE protocol, which additionally provides an anonymity option for natural persons to protect their privacy. The framework

of the proposed CAKE and ACAKE protocols consists of four phases, namely, initialization phase, registration phase, online update phase, and CAKE/ACAKE phase. Figure 2 presents the framework of the proposed CAKE and ACAKE protocols.

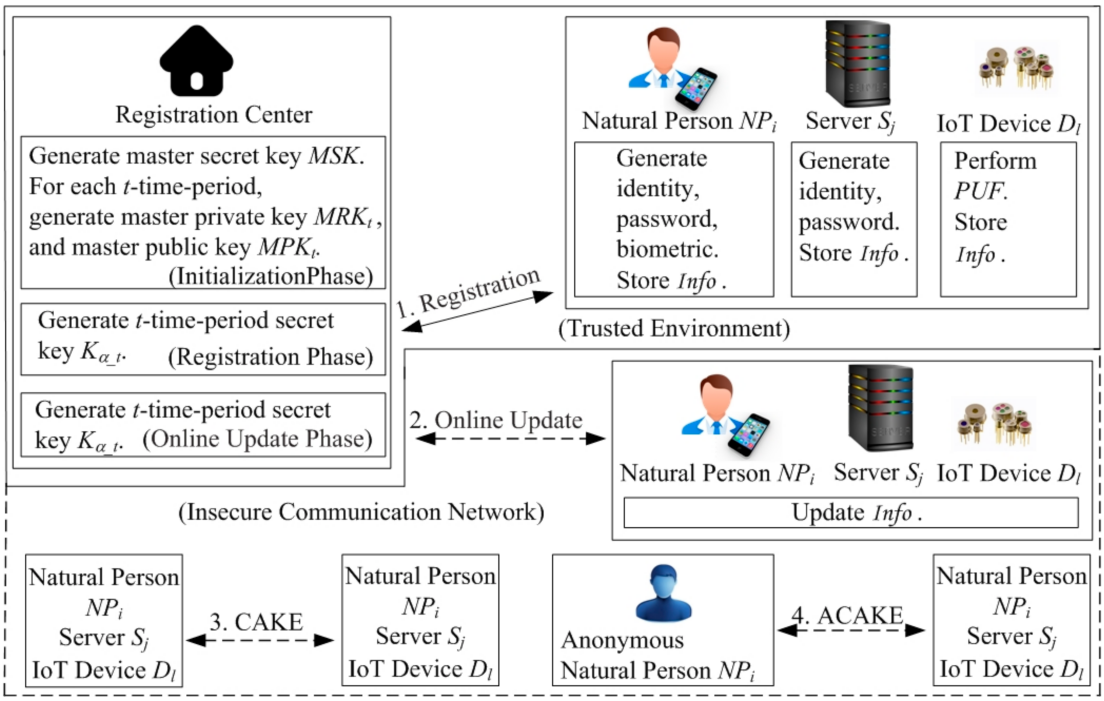


Figure 2. Framework of the proposed CAKE and ACAKE protocols.

In both the CAKE and ACAKE protocols, every member has to register at the registration center (RC) in a secure environment when it joins the system. In the CAKE phase, each member can mutually authenticate with the other member and establish a secure session key through an untrustworthy channel without the help of any third party, even if they are different kinds of members. NP has an anonymity option to run the ACAKE phase with the other NP, Server, or IoT device. In the ACAKE protocol, the real identity of NP would not be revealed in the transmitted channel, and the relation between NP's sessions is untraceable.

For the member revocation problem, both the CAKE and ACAKE protocols employ a key update method to refresh the current secret key for each new time period. In such a case, every member must run the online update phase to update its secret key. Each member has to update its secret key with the help of RC through an open channel at each time period. Once a member is revoked or disabled, it will not be able to receive the new updated key in the next time period. The length of a time period, which is decided by the supervisor, may be days, weeks, or months.

4.2. Proposed CAKE Protocol

The proposed CAKE protocol comprises four phases, namely, initialization, registration, online update, and compatible authentication and key exchange (CAKE) phases. Note that in both the online update and CAKE phases, we define an algorithm GETKEY(A) to be a procedure obtaining the secret key of member A.

4.2.1. Initialization Phase

RC chooses a large prime q and defines a finite field F_q and an elliptic curve $E(F_q)$ by an equation of the form $y^2 = x^3 + mx + n$, where $m, n \in F_q$ satisfy $4\,m^3 + 27\,n^2 \neq 0 \pmod{q}$. *RC* produces a bilinear map, $\hat{e}$: $G_1 \times G_1 \rightarrow G_2$, where G_1 is a subgroup of the additive cyclic group of points on E with order q, and G_2 is a multiplicative cyclic subgroup over F_q

with order q. RC picks a generator point P in G_1 and chooses four distinct one-way hash functions, namely, H_1:{0, 1}$^* \rightarrow G_1$, H_2:{0, 1}$^* \rightarrow G_1$, h_1:{0, 1}$^* \rightarrow$ {0, 1}^{l}, and h_2:{0, 1}$^* \rightarrow$ {0, 1}^{l}, where l is the fixed length of the output. RC also generates the master secret key $MSK \in Z_p^*$. For each t-time period, RC randomly generates a seed $s_t \in Z_q^*$, computes the master private key $MRK_t = h_1(MSK||s_t)$ and master public key $MPK_t = MRK_t \cdot P$, and appends (t, s_t, MRK_t, MPK_t) to list L_{seed} and keeps it secret. RC sets the public parameters Pub = {q, G_1, G_2, P, $\hat{e}$, $H_1()$, $H_2()$, $h_1()$, $h_2()$}.

4.2.2. Registration Phase: Natural Person

When a natural person NP_i with a smart card wants to join the system, NP_i interacts with RC as the following steps, which are illustrated in Figure 3.

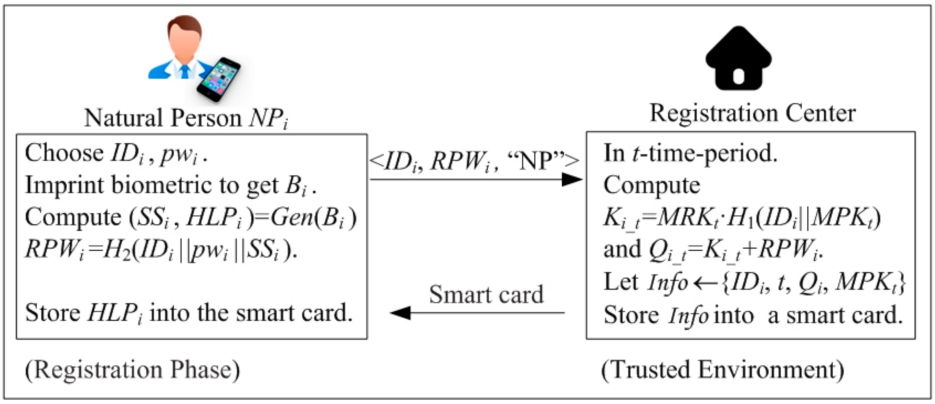


Figure 3. Registration phase of a natural person.

Step 1: NP_i chooses his/her identity ID_i and password pw_i and imprints his/her biometric to the device to obtain B_i. The device performs the generation procedure Gen of the fuzzy extractor on B_i to obtain the secret string SS_i and helper string HLP_i, i.e., (SS_i, HLP_i) = $Gen(B_i)$. The device computes $RPW_i = H_2(ID_i||pw_i||SS_i)$ and gives <ID_i, RPW_i, "NP"> to RC.

Step 2: RC checks the current time period index t and verifies the validity of NP_i. If RC wants to authorize NP_i, then RC computes $K_{i_t} = MRK_t \cdot H_1(ID_i||MPK_t)$ and $Q_{i_t} = K_{i_t} + RPW_i$, lets Info $\leftarrow$ {ID_i, t, Q_i, MPK_t}, and issues a new smart card with Info to NP_i.

Step 3: NP_i inserts the received smart card into the device, and the device appends HLP_i to the smart card.

4.2.3. Registration Phase: Server

When a new server S_j wants to join the system, it has to register at RC and perform the following steps, which are illustrated in Figure 4, in a trusted environment.

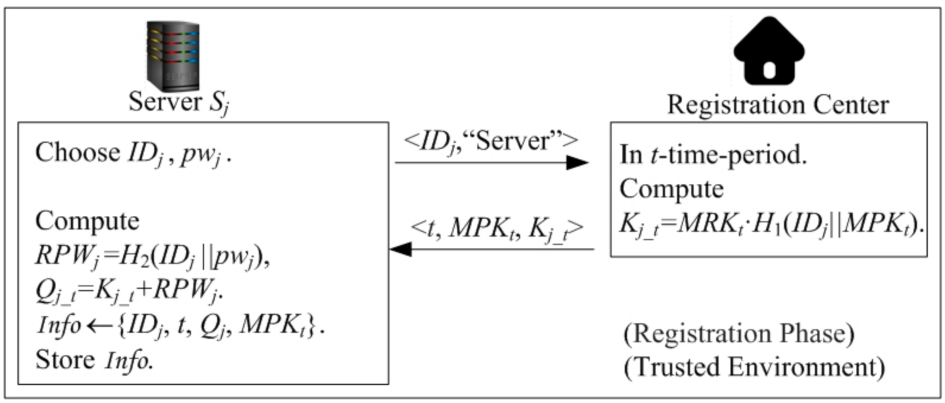


Figure 4. Registration phase of a server.

Step 1: S_j chooses its identity ID_j and password pw_j and gives <ID_j, "Server"> to *RC*.

Step 2: *RC* checks the current time period index t and verifies the validity of S_j. If *RC* wants to authorize S_j, then *RC* computes S_j's secret key $K_{j_t} = MRK_t \cdot H_1(ID_j || MPK_t)$ and sends <t, MPK_t, K_{j_t}> to S_j.

Step 3: After receiving <K_{j_t}>, S_j computes $RPW_j = H_2(ID_j || pw_j)$ and $Q_{j_t} = K_{j_t} + RPW_j$ and lets Info ← {ID_j, t, Q_j, MPK_t}. S_j then stores Info.

4.2.4. Registration Phase: IoT Device

The following steps, which are illustrated in Figure 5, are performed in a trusted environment before adding an IoT device D_l into the system.

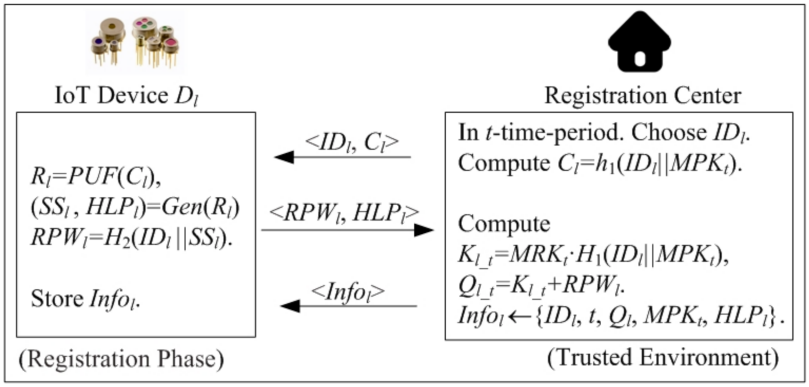


Figure 5. Registration phase of an IoT device.

Step 1: *RC* checks the current time period index t, chooses D_l's identity ID_l, computes the challenge string $C_l = h_1(ID_l || MPK_t)$, gives <ID_l, C_l> to D_l, and waits for the response.

Step 2: D_l performs a PUF on the challenge string C_l to obtain the output response R_l and performs the generation procedure *Gen* of the fuzzy extractor with the BCH encoder [30] on B_i to obtain the secret string SS_l and helper string HLP_l, i.e., $R_l = PUF(C_l)$ and $(SS_l, HLP_l) = Gen(R_l)$. D_l computes $RPW_l = H_2(ID_l || SS_l)$ and sends <RPW_l, HLP_l> back to *RC*.

Step 3: *RC* computes D_l's secret key $K_{l_t} = MRK_t \cdot H_1(ID_l || MPK_t)$ and $Q_{l_t} = K_{l_t} + RPW_l$, and lets Info ← {$ID_l$, t, Q_l, MPK_t, HLP_l}. *RC* then sends <$Info_l$> to D_l. D_l then stores $Info_l$.

4.2.5. GETKEY(A) Algorithm

If A is NP, then A takes as input ID_A and pw_A and imprints the biometric to obtain BA, and the smart card performs the deterministic reproduction procedure *Rep* on B_i to recover SS_A from the corresponding helper string HLP_A, i.e., $SS_A = Re(B_A, HLP_A)$, computes $RPW_A = H_2(ID_A || pw_A || SS_A)$ and secret key $K_A = Q_A - RPW_A$, and outputs K_A.

If A is Server, then A inputs ID_A and pw_A, computes $RPW_A = H_2(ID_A || pw_A)$ and secret key $K_{A_t} = Q_A - RPW_A$, and outputs K_A.

If A is IoT, then A computes the challenge string $C_A = h_1(ID_A || MPK)$, performs a *PUF* on C_A to obtain the response string R_A, performs the deterministic reproduction procedure *Rep* on R_A to recover SS_A from the corresponding helper string HLP_A, i.e., $R_A = PUF(C_A)$ and $SS_A = Rep(R_A, HLP_A)$, computes $RPW_A = H_2(ID_A || SS_A)$ and secret key $K_A = Q_A - RPW_A$, and outputs K_A.

4.2.6. Online Update Phase

Each *NP*, Server, and IoT device needs to perform the following steps, which are illustrated in Figure 6, through an untrustworthy channel to online update its secret key before it authenticates with another member in t-time period.

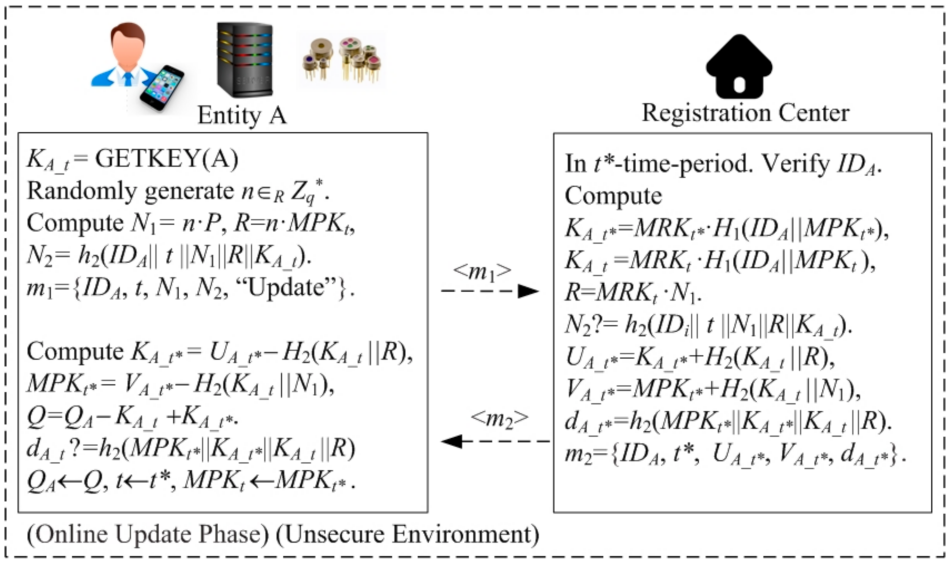


Figure 6. Online update phase.

Step 1: The entity, say Entity A, executes $K_{A_t} \leftarrow$ GETKEY(A), randomly generates $n \in_R Z_q^*$, computes $N_1 = n \cdot P$, $R = n \cdot MPK_t$, $N_2 = h_2(ID_A || t || N_1 || R || K_{A_t})$, and sends <$ID_A$, t, N_1, N_2, “Update”> to RC.

Step 2: RC checks the current time period index t^*. RC verifies ID_A and computes $K_{A_t^*} = MRK_{t^*} \cdot H_1(ID_A || MPK_{t^*})$, $K_{A_t} = MRK_t \cdot H_1(ID_A || MPK_t)$, $R = MRK_t \cdot N_1$. RC verifies if $N_2 = h_2(ID_A || t || N_1 || R || K_{A_t})$. If not, it aborts it. Otherwise, it computes $U_{A_t^*} = K_{A_t^*} + H_2(K_{A_t} || R)$, $V_{A_t^*} = MPK_{t^*} + H_2(K_{A_t} || N_1)$, and $d_{A_t} = h_2(MPK_{t^*} || K_{A_t^*} || K_{A_t} || R)$ and sends <ID_A, t^*, $U_{A_t^*}$, $V_{A_t^*}$, $d_{A_t^*}$> to A.

Step 3: A computes $K_{A_t^*} = U_{A_t^*} - H_2(K_{A_t} || R)$, $MPK_{t^*} = V_{A_t^*} - H_2(K_{A_t} || N_1)$, and $Q = Q_A - K_{A_t} + K_{A_t^*}$. A verifies if $d_{A_t^*} = h_2(MPK_{t^*} || K_{A_t^*} || K_{A_t} || R)$. If so, then it rewrites $Q_i \leftarrow Q$, $t \leftarrow t^*$, and $MPK_t \leftarrow MPK_{t^*}$.

4.2.7. CAKE Phase

When two entities, say Entity A and Entity B, want to mutually authenticate with each other in t-time period and create a secure session key, they perform the following steps, which are illustrated in Figure 7, through an untrustworthy channel. Note that the entity may be a natural person, server, or IoT device.

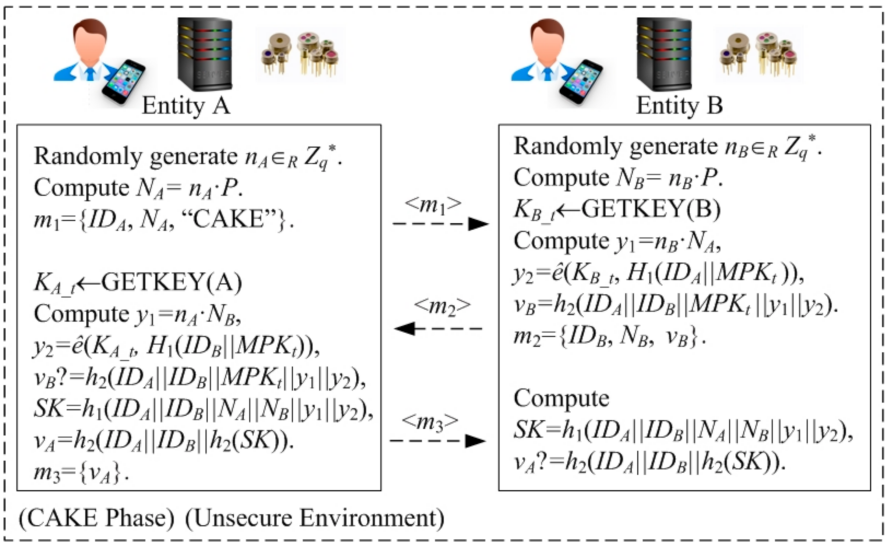


Figure 7. CAKE phase.

Step 1: A randomly generates $n_A \in_R Z_q^*$, computes $N_A = n_A \cdot P$, and sends <ID_A, N_A, "CAKE"> to B.

Step 2: B randomly generates $n_B \in_R Z_q^*$ and computes $N_B = n_B \cdot P$, $K_{B_t} \leftarrow$ GETKEY(B), $y_1 = n_B \cdot N_A$, $y_2 = \hat{e}(K_{B_t}, H_1(ID_A||MPK_t))$, and $v_B = h_2(ID_A||ID_B||MPK_t||y_1||y_2)$. B then sends <ID_B, N_B, v_B> to A.

Step 3: A performs $K_{A_t} \leftarrow$ GETKEY(A), computes $y_1 = n_A \cdot N_B$ and $y_2 = \hat{e}(K_{A_t}, H_1(ID_B||MPK_t))$, and verifies if $v_B = h_2(ID_A||ID_B||MPK_t||y_1||y_2)$. If so, then the authority of B is confirmed, and A computes session key $SK = h_1(ID_A||ID_B||N_A||N_B||y_1||y_2)$ and $v_A = h_2(ID_A||ID_B||h_2(SK))$, and sends <$v_A$> to B.

Step 4: B computes $SK = h_1(ID_A||ID_B||N_A||N_B||y_1||y_2)$ and verifies if $v_A = h_2(ID_A||ID_B||h_2(SK))$. If so, then the authority of A is confirmed, and B accepts SK as the session key.

4.3. Proposed ACAKE Protocol

The propose ACAKE protocol is an extension of the CAKE protocol, which has an extra anonymous compatible authentication and key exchange (ACAKE) phase for natural persons. The GETKEY algorithm and initialization, registration, online update, and CAKE phases are the same as those in the proposed CAKE protocol. NP can additionally perform the ACAKE phase as follows, which is illustrated in Figure 8, to make AKE with another member while keeping the secrecy of his/her real identity.

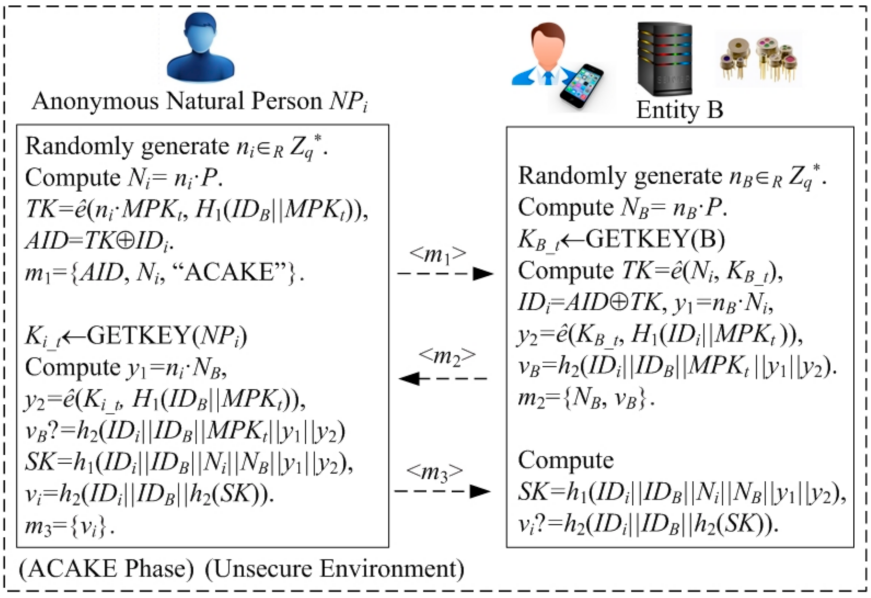


Figure 8. ACAKE phase.

ACAKE phase

When a natural person, say NP_i, wants to anonymously authenticate with Entity B, whose identity is known by NP_i, they perform the following steps to mutually authenticate with each other and establish a secure common session key through an untrustworthy channel.

Step 1: NP_i randomly generates $n_i \in_R Z_q^*$, computes $N_i = n_i \cdot P$, $TK = \hat{e}(n_i \cdot MPK_t, H_1(ID_B||MPK_t))$, and anonymous identity $AID = TK\ ID_i$, and sends <AID, N_i, "ACAKE"> to B.

Step 2: B randomly generates $n_B \in_R Z_q^*$ and computes $N_B = n_B \cdot P$, $K_{B_t} \leftarrow$ GETKEY(B), $TK = \hat{e}(N_i, K_{B_t})$, $ID_i = AID\ TK$, $y_1 = n_B \cdot N_i$, $y_2 = \hat{e}(K_{B_t}, H_1(ID_i||MPK_t))$, and $v_B = h_2(ID_i||ID_B||MPK_t||y_1||y_2)$. B then sends <N_B, v_B> to NP_i.

Step 3: NP_i performs K_{i_t} GETKEY(NP_i), computes $y_1 = n_i \cdot N_B$ and $y_2 = \hat{e}(K_{i_t}, H_1(ID_B||MPK_t))$, and verifies if $v_B = h_2(ID_i||ID_B||MPK_t||y_1||y_2)$. If so, then the author-

ity of B is confirmed, and NP_i computes session key $SK = h_1(ID_i||ID_B||N_i||N_B||y_1||y_2)$ and $v_i = h_2(ID_i||ID_B||h_2(SK))$ and sends $<v_i>$ to B.

Step 4: B computes $SK = h_1(ID_i||ID_B||N_i||N_B||y_1||y_2)$ and verifies if $v_i = h_2(ID_i||ID_B||h_2(SK))$. If so, then the authority of NP_i is confirmed and SK is accepted as the session key.

5. Security Analysis

Here, we propose threat assumptions, and construct an adversarial model for a CAKE/ACAKE protocol. We then formally prove that our CAKE and ACAKE protocols achieve existential session key secrecy and perfect forward secrecy under ECCDH and hash function assumptions and existential unforgeability under the DBDH and hash function assumptions in Theorem 1 and Theorem 2, respectively. We also formally prove that our ACAKE protocol achieves existential anonymity under DBDH and hash function assumptions in Theorem 3.

5.1. Threat Assumptions

Assume that an adversary might have the following attack capabilities in a CAKE/ACAKE protocol.

1. AC1: Be an outsider or any one of the legitimate members [8].
2. AC2: Eavesdrop, delete, replay, or modify any message transmitted over an untrustworthy channel [8].
3. AC3: Offline, try all the (identity, password) pairs within probabilistic polynomial time [37].
4. AC4: Steal the smart card and analyze the instruction power consumption to extract the secret data from the smart card [8,38,39].
5. AC5: Fake the biometric [40].
6. AC6: Correctly predict the PUF's responses to arbitrary challenges with high probability [41].
7. AC7: Individually obtain the (identity, password) pair, obtain the secret data in the smart card, fake the biometric, and correctly predict the PUF's responses. However, breaking them all in polynomial time is not feasible.

5.2. Adversarial Model of CAKE Protocols

The adversarial model of the CAKE protocol is defined in a random oracle model [42]. There are four kinds of members in the adversarial model: a trusted registration center (RC), m Server, n natural persons (NP), and d IoT devices. Each entity has a unique identity. Let Π_α^s denote the s-session of entity α. Π_α^s and Π_β^t are ***partners*** if they are the same session, i.e., α and β authenticate with each other and accept a common session key in α's s-session and in β's t-session. Let A be a probabilistic polynomial time (PPT) algorithm that simulates the behavior of an adversary, who can potentially control all the communications by asking the queries described below, and the queries are responded to by a Challenger B (oracle).

1. ***Hash*** (M): B maintains a hash list to ensure the identical responses and avoid the collision. If ***Hash*** (M) has been asked before, then B returns the same response. Otherwise, B generates a random value h, appends (M, h) to the hash list, and returns h.
2. ***Extract*** (α): This query can be asked for entity α, who is not one of the existing members. B executes the registration phase of the protocol and responds to the corresponding results. (AC1)
3. ***Send*** (Π_α^s, M): B responds to the corresponding results, which should be responded to by α after α receives the message M, by executing the protocol. It models the active attack. (AC1) (AC2)
4. ***Execute*** (Π_α^s, Π_β^t): B returns the complete transcripts of an honest execution between entity α in its s-session and its partner Π_β^t. It models the passive attack. (AC2)

5. ***Reveal*** (Π_{α}^{s}): If entity α has accepted a session key, say SK, in its s-session, then B returns SK. Otherwise, B returns "NULL". It models the known session key attack.
6. ***Corrupt*** (α): B returns the secret key of α, who is one of the existing members. (AC1) (AC3) (AC4) (AC5) (AC6)
7. ***Test$_{SK}$*** (Π_{α}^{s}): B flips an unbiased bit $b \in \{0,1\}$. If $b = 1$, then B returns the session key of entity α in its s-session. If $b = 0$, then B returns a random value.

An oracle Π_{α}^{s} with its partner Π_{β}^{t} is said to be ***fresh*** if the common session key $SK \neq$ NULL between them is confirmed, both ***Reveal*** (Π_{α}^{s}) and ***Reveal*** (Π_{β}^{t}) queries have not been asked, and there is no Corrupt (α) or Corrupt (β) query asked before the Send (Π_{α}^{s}, M) or Send (Π_{β}^{t}, M) query.

A can ask the ***Test$_{SK}$*** (Π_{α}^{s}) query only once at any time during the game and may ask other queries when asking the ***Test$_{SK}$*** (Π_{α}^{s}) query whenever Π_{α}^{s} is fresh. A outputs its guess b' for the bit b in the ***Test$_{SK}$*** (Π_{α}^{s}) query eventually. Let A's distinguishing ability in the ***Test*** query measure the advantage of A, and let ***Succ*** denote the event that A successfully guesses the unbiased bit b in the ***Test*** query, and the ***advantage*** of A attacking the protocol Ψ is defined by Adv_{Ψ} (A) = $|2 \cdot \Pr(\textbf{\textit{Succ}}) - 1|$. Note that if A guesses b totally in random, then Pr(***Succ***) would approach $1/2$ and Adv_{Ψ} (A) is negligible. The adversarial model is a random oracle model where oracles behave with a hash function as a true random function, which produces a random value for each new query.

5.3. Adversarial Model of ACAKE Protocols

The adversarial model of ACAKE protocols is the same as the adversarial model of a CAKE protocol, which additionally has an extra ***Test$_{ID}$*** (Π_{α}^{s}) query as follows.

1. ***Test$_{ID}$*** (Π_{α}^{s}): B flips an unbiased bit b in {0, 1} and returns the real identity of entity α if $b = 1$. If $b = 0$, then a random value is returned.

A can ask the ***Test$_{ID}$*** (Π_{α}^{s}) query only once at any time during the game and may ask other queries when asking the ***Test$_{ID}$*** (Π_{α}^{s}) query whenever Π_{α}^{s} is fresh. A outputs its guess b' for the bit b in the ***Test$_{ID}$*** (Π_{α}^{s}) query eventually. A's ability of distinguishing the session key/identity from a random value in the ***Test*** query measures the advantage of A.

Definition 2. *A CAKE/ACAKE protocol achieves existential* ***perfect forward secrecy, secrecy of the session key****, and* ***unforgeability*** *if there is no probabilistic polynomial time adversary A, who has a non-negligible advantage in the following game played between A and an infinite set of oracles.*

1. The system is set up according to the initialization and registration phases of the protocol.
2. A may ask the following queries and obtain the corresponding results: ***Test***, ***Hash***, ***Execute***, ***Send***, ***Extract***, ***Corrupt***, and ***Reveal***.
3. There is no *Corrupt* (α) or *Reveal* (Π_{α}^{s}) query asked before the *Test* (Π_{α}^{s}) query.

A may ask other queries when asking the *Test* (Π_{α}^{s}) query whenever Π_{α}^{s} is fresh and eventually outputs its guess b' for the unbiased bit b in the *Test* (Π_{α}^{s}) query. The game is terminated.

5.4. Formal Proof

Theorem 1. *The proposed CAKE and ACAKE protocols achieve existential perfect forward secrecy and secrecy of the session key under the ECCDH and hash function assumptions.*

Theorem 2. *The proposed CAKE and ACAKE protocols achieve existential unforgeability under DBDH and hash function assumptions.*

Theorem 3. *The proposed ACAKE protocol achieves existential anonymity under DBDH and hash function assumptions.*

The proofs of Theorems 1, 2, and 3 are presented in Appendix A.

6. Comparisons and Performance Analysis

Biometrics and physical unclonable functions (PUFs) are, respectively, adopted as the third authentication factor of a nature person and an IoT device in the proposed CAKE/ACAKE protocols; hence, our protocols achieve multi-factor authentication. In the proposed CAKE/ACAKE protocols, servers are regarded as independent entities that have distinct secret keys, and no third party is involved in the CAKE/ACAKE phase; hence, our protocols are suitable for a multi-server environment and achieve the independent authentication property. Each member periodically executes the online update phase to achieve the member revocation property that a member cannot authenticate with another member if he/she does not update his/her secret key in the present period. In the proposed ACAKE protocols, anonymous nature persons use dynamic values to mask their real identities to achieve anonymity and untraceability/unlinkability.

The comparisons between our CAKE and ACAKE protocols and the related AKE protocols are listed in Table 2. The protocols in [11–15] are the related anonymous AKE protocols designed for smart cities, and all of them are two-factor authentication and do not provide compatible authentication and member revocation properties. The protocols in [16–25] are the related AKE protocols, which are suitable for multi-server environments; the protocols in [18–22] achieve independent authentication, but the protocols in [18,21] do not provide user anonymity. The protocols in [22–25] are three-factor authentication schemes, which are suitable for multi-server environments and provide user privacy protection (user anonymity and untraceability/unlinkability); however, the protocols in [23–25] do not provide independent authentication. As shown in Table 2, only our ACAKE protocol achieves all the listed properties; moreover, there is no password table or public key issuing problem in our CAKE/ACAKE protocols.

Table 2. Comparisons.

	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]	[22]	[23]	[24]	[25]	**Our ACAKE**
Compatible authentication	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	**Y**
Multi-factor authentication [a]	N	N	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	**Y**
Multi-server environment [b]	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	**Y**
Independent authentication [c]	N	Y	Y	N	Y	N	N	Y	Y	Y	Y	Y	N	N	N	**Y**
User anonymity	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	N	Y	Y	Y	Y	**Y**
User untraceability/unlinkability	Y	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y	**Y**
Entity revocation	N	N	N	N	N	N	Y	N	N	Y	Y	N	Y [d]	N	N	**Y**

[a] using three or more than three factors to authenticate; [b] users do not need to register at each server; [c] without the help of any third party; [d] revocation list.

For convenience, let $TG_{\hat{e}}$, TG_{mul}, TG_{add}, T_H, and T_h, respectively, denote the time cost of executing bilinear pairing, multiplication on ECC, point addition on ECC, hash function mapping a string to a point on ECC, and hash function mapping a string to a string. Scott et al. [43] implemented the standard Tate pairing ($k = 2$) over a large 512-bit field on a 32-bit Philips HiPerSmart^{TM} smart card with a maximum clock speed of 36 MHz, an instantiation of the MIPS-32 based SmartMIPS^{TM} architecture. Cao et al. [44] implemented the standard Tate pairing over a large 512-bit field and the 160-bit hash in group G on a PIV 3-GHz processor with 512-MB memory and a Windows XP operation system, and on a

Linux personal digital assistant equipped with a 206-MHz Strong ARM processor. Xiong and Qin [45] implemented the same experiment on an Intel PXA270 processor at 624-MHz installed on the Linux personal digital assistant. The time costs of T_H on a 36-MHz device and on a 3 GHz device are estimated by 3.04 × 3000/624 = 14.62 and 3.04 × 3000/36 = 253 milliseconds, respectively.

Cavalieri and Cutuli [46] implemented a hashing algorithm on the MSP430 family, assuming a frequency of 8 MHz, and the execution time was 0.065 milliseconds, which is negligible. We ignore the computational costs of the hash function, the point addition on ECC, generating a random number, and XOR and concatenation operations here.

In the online update phase, the time cost of cryptographic operations of a member and *RC* is $2TG_{mul} + 3TG_H + 4TG_{add} + 2T_h$ and $3TG_{mul} + 4TG_H + 2TG_{add} + 2T_h$, respectively. The estimated execution times of an IoT member with a 36-MHz device, a natural person with a 206-MHz device, a natural person with a 624-MHz device, a server with a 3 GHz device, and RC with a 3 GHz device are approximately 2 × 270 ms +3 × 253 ms = 1299 ms, 2 × 92.91 ms + 3 × 44.27 ms = 318.63 ms, 2 × 30.67 ms +3 × 14.62 ms = 105.2 ms, 2 × 6.38 ms +3 × 3.04 ms = 21.88 ms, and 3 × 6.38 ms +4 × 3.04 ms = 31.3 ms, respectively.

The time cost of cryptographic operations of each member is $TG_{\hat{e}} + 2TG_{mul} + 2TG_H + TG_{add} + 4T_h$ in the CAKE phase. The estimated execution times of an IoT member with a 36-MHz device, a natural person with a 206-MHz device, a natural person with a 624-MHz device, a server with a 3 GHz device, and RC with a 3 GHz device are approximately 1 × 290 ms +2 × 270 ms +2 × 253 ms = 1336 ms, 1 × 291.84 ms +2 × 92.91 ms +2 × 44.27 ms = 566.2 ms, 1 × 96.2 ms +2 × 30.67 ms +2 × 14.62 ms = 186.78 ms, and 1 × 20.04 ms +2 × 6.38 ms +2 × 3.04 ms = 38.88 ms, respectively.

In the ACAKE phase, the time costs of cryptographic operations of *NP* and his/her partner member are $2TG_{\hat{e}} + 3TG_{mul} + 2TG_H + TG_{add} + 4T_h$ and $2TG_{\hat{e}} + 2TG_{mul} + 2TG_H + TG_{add} + 4T_h$, respectively. The estimated execution times of an IoT member with a 36-MHz device, a natural person with a 206-MHz device, a natural person with a 624-MHz device, a server with a 3 GHz device, and RC with a 3 GHz device are approximately 2 × 290 ms +3 × 270 ms + 2 × 253 ms = 1896 ms, 2 × 291.84 ms + 3 × 92.91 ms + 2 × 44.27 ms = 950.95 ms, 2 × 96.2 ms + 3 × 30.67 ms + 2 × 14.62 ms = 313.65 ms, and 2 × 20.04 ms + 3 × 6.38 ms + 2 × 3.04 ms = 65.3 ms, respectively.

Table 3 summarizes the times of executing cryptographic operations and the times of *RC* and each kind of member executing cryptographic operations during three phases in the proposed protocols: online updating, CAKE, and ACAKE phases. Note that TG_{add} and T_h are negligible and ignored here. As shown in Table 3, the execution times are appropriate.

Table 3. Execution times of cryptographic operations (in milliseconds).

	IoT Device	**Natural Person**		**Server**	***RC***
	(36-MHz) [43]	**(206-MHz) [44]**	**(624-MHz) [45]**	**(3 GHz) [44]**	
$TG_{\hat{e}}$	290	291.84 (estimated)	96.2	20.04	
TG_{mul}	270	92.91	30.67	6.38	
T_H	253 (estimated)	44.27	14.62 (estimated)	3.04	
Executing updating	1299	318.63	105.2	21.88	31.3
Executing CAKE	1336	566.2	186.78	38.88	-
Executing ACAKE	1896	950.95	313.65	65.3	-

Assume that *RC* has 100 computers, with a frequency of 3 GHz, simultaneously performing members' registration and online updating. Then, *RC* can perform registration for 10,615 entities in one second, where 1000 ms/((6.38 ms + 3.04 ms)/entity) × 100 = 10,615 entities, or perform online updating for 3194 members in one second, where 1000 ms/(31.3 ms/member) × 100 = 3194 members. Thus, *RC* has the ability to perform the online update for all members in each time period to deal with the member revocation issue. The proper length of a time period, which is decided by the supervisor, may be hours, days, or weeks.

7. Conclusions

In this paper, we first proposed the concept of compatible authentication, which provides an efficient cross-species authentication. We proposed the first CAKE and ACAKE protocols for a smart city in 5G networks. The proposed ACAKE protocol is the first protocol that simultaneously achieves all the presented properties: probably secure, compatible authentication, multi-factor authentication, suitability for multi-server environments, independent authentication, user anonymity, user untraceability/unlinkability, member revocation, list-free, public key management-free, and efficiency. We formally analyzed the security of the proposed protocols and demonstrated the efficiency.

In the proposed CAKE and ACAKE protocols, the registration center (RC) is assumed to be a powerful party working in a secure and trusted environment meaning that it does not matter if the RC cannot resist side channel attacks; members can partially resist side channel attacks since they periodically execute the online updating phase to change their secret keys. Our future work will enhance the proposed protocols to establish a protocol which resists side channel attacks, and we will consider the applicability in systems based on blockchains.

Author Contributions: Y.-H.C. designed the scheme, proposed the adversary model, and performed the formal analysis; Y.-M.T. performed the security analysis and revised the article. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially supported by the Ministry of Science and Technology, Taiwan, under contract no. MOST108-2221-E-018-04-MY2.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No data is reported.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Proof of Theorem 1. Suppose that there are one trusted third-party registration center (*RC*), *m* Server, *n* natural persons (*NP*), and *d* IoT devices in the smart city system. Each entity has a unique identity. Suppose that there is an adversary A, who can break the perfect forward secrecy or the secrecy of the session key of the proposed CAKE/ACAKE protocols with a non-negligible advantage in probabilistic polynomial time. Then, we can construct a Challenge Algorithm B to solve the ECCDH problem with a non-negligible advantage by using A's ability of breaking the protocol. An instance (G_1, G_2, P, $\hat{e}$, q, $X = xP$, $Y = yP$) of the ECCDH problem is given to B, and B's goal is to output xyP. B permeates the ECCDH problem into the queries, which are asked by A in the l-session for Π_U^S and its partner Π_V^T. B manages a time seed list L_{seed}, and four hash lists L_{H1}, L_{H2}, L_{h1}, and L_h, which are empty initially. B generates a master secret key $MSK \in {Z_p}^*$. For each t-time period, B randomly generates s_t, $r_t \in {Z_q}^*$, sets the master private key $MRK_t = h_1(MSK||s_t)$ and master public key $MPK_t = MRK_t \cdot P$, and appends $((MSK||s_t), r_t, r_t \cdot P)$ to list L_{h1} and (t, s_t, MRK_t, MPK_t) to list L_{seed}. B sets Pub = $\{q, G_1, G_2, P, \hat{e}\}$ to be public parameters. B simulates the oracle queries:

1. ***Hash*** (M): B checks the hash lists and returns the corresponding value if it is in the list. Otherwise, B randomly generates $r \in Z_q^*$ and $m \in \{0,1\}^l$ and returns the value by the following three different types of hash queries.
   a. ***Hash*$_{H1}$** (M): Compute $R = r \cdot P$, append (M, r, R) to list L_{H1}, and return R.
   b. ***Hash*$_{H2}$** (M): Compute $R = r \cdot P$, append (M, r, R) to list L_{H2}, and return R.
   c. ***Hash*$_{h1}$** (M): Append (M, m) to list L_{h1} and return m.
   d. ***Hash*$_{h2}$** (M): Append (M, m) to list L_{h2} and return m.
2. GETKEY(A) algorithm: If A is NP, then it inputs ID_A, pw_A, and B_i, computes $SS_A = Rep(B_A, HLP_A)$, asks the ***Hash*$_{h1}$** query to obtain $RPW_A = H_2(ID_A || pw_A || SS_A)$, computes $K_A = Q_A - RPW_A$, and returns K_A. If A is a server, then it inputs ID_A and pw_A, asks the ***Hash*$_{H}$** query to obtain $RPW_A = H_2(ID_A || pw_A)$, computes $K_{A_t} = Q_A - RPW_A$, and returns K_A. If A is an IoT, then A computes $C_A = h_1(ID_A || MPK)$, performs $R_A = PUF(C_A)$, computes $SS_A = Rep(R_A, HLP_A)$, asks the ***Hash*$_{h1}$** query to obtain $RPW_A = H_2(ID_A || SS_A)$, computes $K_A = Q_A - RPW_A$, and returns K_A.
3. ***Extract*** (α, M): If α is one of the members, then it returns "Reject". Otherwise, it returns the corresponding registration parameters as follows. B checks the current time period t and list L_{seed} to obtain (t, s_t, MRK_t, MPK_t).
   a. M = <ID_α, RPW_α, "NP">: B asks the ***Hash***$_{H1}$ query to obtain $H_1(ID_\alpha || MPK_t)$, computes $K_{\alpha_t} = MRK_t \cdot H_1(ID_\alpha || MPK_t)$ and $Q_{\alpha_t} = K_{\alpha_t} + RPW_\alpha$, lets Info $\leftarrow$\{ID_α, t, Q_α, MPK_t\}, and returns Info.
   b. M = <ID_α, "Server">: B asks the ***Hash***$_{H1}$ query to obtain $H_1(ID_\alpha || MPK_t)$, computes $K_{\alpha_t} = MRK_t \cdot H_1(ID_\alpha || MPK_t)$, and returns <$t$, MPK_t, K_{α_t}>.
   c. M = <"IoT">: B chooses ID_α, asks the ***Hash*$_{h1}$** query to obtain $C_\alpha = h_1(ID_\alpha || MPK_t)$, returns <$C_\alpha$>, and waits for the input <RPW_α, HLP_α>. B asks the ***Hash***$_{H1}$ query to obtain $H_1(ID_\alpha || MPK_t)$, computes $K_{\alpha_t} = MRK_t \cdot H_1(ID_\alpha || MPK_t)$ and $Q_{\alpha_t} = K_{\alpha_t} + RPW_\alpha$, lets Info $\leftarrow$\{ ID_α, t, Q_α, MPK_t, HLP_α\}, and returns Info.
4. ***Send*** (Π_β^n, M): There are three different types of ***Send*** query in the proposed protocols: Update, CAKE, and ACAKE. Assume that the current time is in t^*-time period. B checks list L_{seed} to obtain (t^*, s_{t^*}, MRK_{t^*}, MPK_{t^*}) and responds according to the type of ***Send*** query.
   a. ***Send*** (RC, <ID_A, t, N_1, N_2, "Update">): B checks list L_{seed} to obtain (t, s_t, MRK_t, MPK_t), asks the ***Hash*$_{H1}$** query to obtain $H_1(ID_A || MPK_t)$ and $H_1(ID_A || MPK_{t^*})$, and computes $K_{A_t} = MRK_t \cdot H_1(ID_A || MPK_t)$, $K_{A_t^*} = MRK_{t^*} \cdot H_1(ID_A || MPK_{t^*})$, and $R = MRK_t \cdot N_1$. B asks the ***Hash*$_{H2}$** query to obtain $H_2(K_{A_t} || R)$, computes $U_{A_t^*} = K_{A_t^*} + H_2(K_{A_t} || R)$ and $V_{A_t^*} = MPK_{t^*} + H_2(K_{A_t} || N_1)$, and asks the ***Hash*$_{h2}$** query to obtain $d_{A_t^*} = h_2(MPK_{t^*} || K_{A_t^*} || K_{A_t} || R)$. B then returns <$ID_A$, t^*, $U_{A_t^*}$, $V_{A_t^*}$, $d_{A_t^*}$>.
   b. ***Send*** (Π_β^n, <ID_A, N_A, "CAKE">): B randomly generates $n_\beta \in_R Z_q^*$, computes $N_\beta = n_\beta \cdot P$, conducts the GETKEY($\beta$) algorithm to obtain K_{β_t}, and computes $y_1 = n_\beta \cdot N_A$. B asks the ***Hash*$_{H1}$** query to obtain $H_1(ID_A || MPK_t)$, computes $y_2 = \hat{e}(K_{\beta_t}, H_1(ID_A || MPK_t))$, and asks the ***Hash*$_{h2}$** query to obtain $v_\beta = h_2(ID_A || ID_\beta || MPK_t || y_1 || y_2)$. B returns <$ID_\beta$, N_β, v_β>.
   c. ***Send*** (Π_β^n, <AID, N_A, "ACAKE">): B randomly generates $n_\beta \in_R Z_q^*$, computes $N_\beta = n_\beta \cdot P$, conducts the GETKEY($\beta$) algorithm to obtain K_{β_t}, and computes $TK = \hat{e}(N_A, K_{\beta_t})$, $ID_A = AID \oplus TK$, and $y_1 = n_\beta \cdot N_A$. B asks the ***Hash*$_{H1}$** query to obtain $H_1(ID_A || MPK_t)$, computes $y_2 = \hat{e}(K_{\beta_t}, H_1(ID_A || MPK_t))$, and asks the ***Hash*$_{h2}$** query to obtain $v_\beta = h_2(ID_A || ID_\beta || MPK_t || y_1 || y_2)$. B then returns <$ID_\beta$, N_β, v_β>.
5. ***Execute*** (Π_α^m, Π_β^n): If $\Pi_\beta^m = \Pi_U^S$. and $\Pi_\beta^n = \Pi_V^T$, then B lets $N_\alpha = X$, $N_\beta = Y$, y_1= $NULL$, and $Tag = ID_U || ID_V || X || Y$ and randomly chooses $v_\beta \in_R \{0, 1\}^l$. Otherwise, B randomly generates n_α, $n_\beta \in_R Z_q^*$ and computes $N_\alpha = n_\alpha \cdot P$, $N_\beta = n_\beta \cdot P$, and y_1 =

$n_\beta \cdot N_\alpha$. Suppose that the current time is in t-time period. B checks list L_{seed} to obtain (t, s_t, MRK_t, MPK_t).

a. ***Execute*$_{UD}$** (α, RC): B chooses t^{OLD}-time period, lets $N_1 = N_\alpha$, computes $R = n_\alpha \cdot MPK_{(t\ \mathrm{OLD})}$, asks the ***Hash*$_{h2}$** query to obtain $N_2 = h_2(ID_A \| t^{OLD} \| N_1 \| R \| K_{A_(t\ \mathrm{OLD})})$, and asks the ***Send*** (RC, <ID_α, t^{OLD}, N_1, N_2, "Update">) query to obtain <ID_α, t, U_{α_t}, V_{α_t}, $d_{_t}$>. B then returns <ID_α, t^{OLD}, N_1, N_2, "Update"> and <ID_α, t, U_{α_t}, V_{α_t}, d_{α_t}>.

b. ***Execute*$_{CAKE}$** (Π_α^m, Π_β^n): B conducts the GETKEY(β) algorithm to obtain K_{β_t}, asks the ***Hash*$_H$** query to obtain $H(ID_\alpha \| MPK_t)$, and computes $y_2 = \hat{e}(K_{\beta_t}, H_1(ID_\alpha \| MPK_t))$. If $\Pi_\beta^m \neq \Pi_U^S$ or $\Pi_\beta^n \neq \Pi_V^T$, then B asks the ***Hash*$_{h2}$** query to obtain $v_\beta = h_2(ID_\alpha \| ID_\beta \| MPK_t \| y_1 \| y_2)$ and asks the ***Hash*$_{h1}$** query to obtain $SK = h_1(ID_\alpha \| ID_\beta \| N_\alpha \| N_\beta \| y_1 \| y_2)$. B then asks the ***Hash*$_{h2}$** query to obtain $h_2(SK)$ and $v_\alpha = h_2(ID_\alpha \| ID_\beta \| h_2(SK))$ and returns <ID_α, N_α, "CAKE">, <ID_β, N_β, v_β>, <v_α>.

c. ***Execute*$_{ACAKE}$** (Π_α^m, Π_β^n): B computes $TK = \hat{e}(n_\alpha \cdot MPK_t, H_1(ID_\beta \| MPK_t))$ *and* AID = TK $\oplus$ ID_α. *B conducts the GETKEY(β) algorithm to obtain* K_{β_t}, *asks the* **Hash$_{H1}$** *query to obtain* $H_1(ID_\alpha \| MPK_t)$, *and computes* $y_2 = \hat{e}(K_{\beta_t}, H_1(ID_\alpha \| MPK_t))$. *If* $\Pi_\beta^m \neq \Pi_U^S$ *or* $\Pi_\beta^n \neq \Pi_V^T$, *then B asks the* ***Hash*$_{h2}$** query to obtain $v_\beta = h_2(ID_\alpha \| ID_\beta \| MPK_t \| y_1 \| y_2)$ and asks the ***Hash*$_{h1}$** query to obtain $SK = h_1(ID_\alpha \| ID_\beta \| N_\alpha \| N_\beta \| y_1 \| y_2)$. B then asks the ***Hash*$_{h2}$** query to obtain $v_\alpha = h_2(ID_\alpha \| ID_\beta \| h_2(SK))$ and returns <AID, N_α, "ACAKE">, <ID_β, N_β, v_β>, and <v_α>.

6. ***Reveal*** (Π_α^s): If entity α has accepted a session key, say SK, in its s-session, then B returns SK. Otherwise, B returns "NULL".
7. ***Corrupt*** (α): If entity α is one of the system members, then B runs the same procedures as the ***Extract*** (α, M) query and returns the corresponding result.
8. ***Test*$_{SK}$** (Π_α^s): B flips an unbiased bit $b \in \{0,1\}$. B returns entityα's real session key in its s-session if $b = 1$ and returns a random value if $b = 0$.
9. ***Test*$_{ID}$** (Π_α^s): B flips an unbiased bit $b \in \{0,1\}$. B returns entityα's real identity if $b = 1$ and returns a random value if $b = 0$.

If A can successfully break the perfect forward secrecy or the secrecy of the session key of the proposed protocols, then xyP should appear in the L_{h1} list. B uses Tag to find ($M = (Tag \| y_1 \| y_2)$, r, R) in L_{h1}, where $y_1 = xyP$ and $R = SK$. Then, B answers with xyP to the ECCDH problem. The successful probability of B solving the ECCDH problem depends on the successful probability of A asking the $Test_{SK}$ query in the l-session for Π_U^S or its partner Π_U^S and breaking the perfect forward secrecy or the secrecy of the session key of the proposed protocols. The successful probability of B solving the ECCDH problem depends on the event that A asks the $Test_{SK}$ query in the l-session for Π_U^S or its partner Π_U^S and breaks the session key secrecy or the perfect forward secrecy of the proposed protocol. The probability of A asking the $Test_{SK}$ query in the l-session for Π_U^S or its partner Π_U^S is $1/q_n$, where q_n is the total number of sessions. If A successfully guesses b in the $Test_{SK}$ query with advantage δ, which is non-negligible, then B can solve the ECCDH problem with advantage δ/q_n, which is also non-negligible. It contradicts the ECCDH assumption. Thus, there is no polynomial time adversary that can break the perfect forward secrecy or the secrecy of the session key of the proposed protocols with a non-negligible advantage. □

Proof of Theorem 2. Suppose that there are one trusted third-party registration center (RC), m Server, n natural persons (NP), and d IoT devices in the smart city system, and each entity has a unique identity. Suppose that there is an adversary A, who can forge a member to cheat another member in the proposed CAKE/ACAKE protocol with a non-negligible advantage in probabilistic polynomial time. Then, we can construct a Challenge Algorithm B to solve the DBDH problem with a non-negligible advantage by using A's ability of forging a member. An instance (G_1, G_2, P, $\hat{e}$, q, $X = xP$, $Y = yP$, $Z = zP$, g) of the DBDH

problem is given to B, and B's goal is to determine if $g = \hat{e}(P, P)^{xyz}$. B guesses that A tends to forge member U to cheat member V and permeates the DBDH problem into the queries. A cannot ask the Corrupt query on U or on V. B manages a time seed list L_{seed} and four hash lists L_{H1}, L_{H2}, L_{h1}, and L_{h2}, which are empty initially. For each t-time period, B randomly generates a_t, $r_t \in_R Z_q^*$, computes $MPK_t = a_t \cdot X$, appends (t, a_t, MPK_t) to list L_{seed}, and appends $((ID_U \| \| MPK_t), r_t, r_t \cdot Y)$ and $((ID_V \| \| MPK_t), r_t, r_t \cdot Z)$ to list L_{H1}; i.e., let $H_1(ID_U \| \| MPK_t) = r_t \cdot Y$ and $H_1(ID_V \| \| MPK_t) = r_t \cdot Z$. Note that $MRK_t = a_t \cdot x$ is unknown here. B sets Pub = $\{q, G_1, G_2, P, \hat{e}\}$ to be public parameters. B simulates the oracle queries:

1. The ***Hash***, ***Reveal***, ***Corrupt***, and ***Test*** queries are identical to those in the proof of Theorem 1.
2. ***Extract*** (α, M): If α is one of the members, then it returns "Reject". Otherwise, it returns the corresponding registration parameters as follows. B checks the current time period t and list L_{seed} to obtain (t, a_t, MPK_t).
   a. M = <ID_α, RPW_α, "NP">: B asks the $\boldsymbol{Hash_{H1}}$ query for $(ID_\alpha \| \| MPK_t)$, finds $((ID_\alpha \| \| MPK_t), r, H_1(ID_\alpha \| \| MPK_t))$ in L_H to obtain r, and computes $K_{\alpha_t} = a_t \cdot r \cdot X$. Note that $K_{\alpha_t} = MRK_t \cdot H_1(ID_\alpha \| \| MPK_t) = a_t \cdot r \cdot X$. B computes $Q_{\alpha_t} = K_{\alpha_t} + RPW_\alpha$, lets Info $\leftarrow\{ ID_\alpha, t, Q_\alpha, MPK_t\}$, and returns Info.
   b. M = <ID_α, "Server">: B asks the $\boldsymbol{Hash_{H1}}$ query for $(ID_\alpha \| \| MPK_t)$, finds $((ID_\alpha \| \| MPK_t), r, H_1(ID_\alpha \| \| MPK_t))$ in L_{H1} to obtain r, and computes $K_{\alpha_t} = a_t \cdot r \cdot X$. B returns <$t$, MPK_t, K_{α_t}>.
   c. M = <"IoT">: B chooses ID_α, asks the $\boldsymbol{Hash_{h1}}$ query to obtain $C_\alpha = h_1(ID_\alpha \| \| MPK_t)$, returns <$C_\alpha$>, and waits for the input <RPW_α, HLP_α>. B asks the $\boldsymbol{Hash_{H1}}$ query for $(ID_\alpha \| \| MPK_t)$, finds $((ID_\alpha \| \| MPK_t), r, H_1(ID_\alpha \| \| MPK_t))$ in L_{H1} to obtain r, computes $K_{\alpha_t} = a_t \cdot r \cdot X$ and $Q_{\alpha_t} = K_{\alpha_t} + PW_l$, lets Info$\leftarrow\{ ID_\alpha, t, Q_\alpha, MPK_t, HLP_\alpha\}$, and returns Info.
3. ***Send*** (Π_B^n, M): There are three different types of ***Send*** query in the proposed protocols: Update, CAKE, and ACAKE. Assume that the current time is in t-time period. B checks list L_{seed} to obtain (t, a_t, MPK_t) and responds according to the type of ***Send*** query.
   a. ***Send*** (RC, <ID_α, t^{OLD}, N_1, N_2, "Update">): B checks list L_{seed} to obtain $(t^{OLD}, a_{t\ OLD}, MPK_t{}^{OLD})$, asks the $\boldsymbol{Hash_{H1}}$ query for $(ID_\alpha \| \| MPK_t)$, and finds $((ID_\alpha \| \| MPK_t), r_t, H_1(ID_\alpha \| \| MPK_t))$ and $((ID_\alpha \| \| MPK_t{}^{OLD}), r_{tOLD}, H_1(ID_\alpha \| \| MPK_t{}^{OLD}))$ in L_{H1} to obtain r_t and r_{tOLD}. If $\alpha \in \{U, V\}$, then B finds $((ID_A \| \| t^{OLD} \| \| N_1 \| \| R \| \| K_{\alpha_(t\mathrm{OLD})}), N_2)$ in list L_{h2} to obtain R and $K_{\alpha_(t\mathrm{OLD})}$ and computes $K_{\alpha_t} = a_t \cdot a_{tOLD}{}^{-1} \cdot r_t \cdot r_{tOLD}{}^{-1} \cdot K_{\alpha_(t\mathrm{OLD})}$. If $\alpha \notin \{U, V\}$, then B computes $K_{\alpha_(t\mathrm{OLD})} = a_t \cdot r_{t\mathrm{OLD}} \cdot X$ and $K_{\alpha_t} = a_t \cdot r_t \cdot X$. B then asks the $\boldsymbol{Hash_{H2}}$ query to obtain $H_2(K_{\alpha_(t\mathrm{OLD})} \| \| R)$ and $d_{\alpha_t} = H_2(MPK_t \| \| K_{\alpha_t} \| \| K_{\alpha_(t\mathrm{OLD})} \| \| R)$, computes $U_{\alpha_t} = K_{\alpha_t} + H_2(K_{\alpha_(t\mathrm{OLD})} \| \| R)$ and $V_{\alpha_t} = MPK_t + H_2(K_{\alpha_(t\mathrm{OLD})} \| \| N_1)$, and returns <$ID_\alpha$, t, U_{α_t}, V_{α_t}, d_{α_t}>.
   b. ***Send*** (Π_β^n, <ID_α, N_α, "CAKE">): B randomly generates $n_\beta \in_R Z_q^*$, computes $N_\beta = n_\beta \cdot P$ and $y_1 = n_\beta \cdot N_\alpha$, and asks the $\boldsymbol{Hash_H}$ query to obtain $H_1(ID_\alpha \| \| MPK_t)$ and $H_1(ID_\beta \| \| MPK_t)$. If $\alpha \in \{U, V\}$ and $\beta \in \{U, V\}$, then B randomly chooses $y_2 = g$. If $\alpha \notin \{U, V\}$ and $\beta \in \{U, V\}$, then B finds $((ID_\alpha \| \| MPK_t), r, H_1(ID_\alpha \| \| MPK_t))$ in L_{H1} to obtain r and computes $K_{\alpha_t} = a_t \cdot r \cdot X$ and $y_2 = \hat{e}(K_{\alpha_t}, H_1(ID_\beta \| \| MPK_t))$. If $\beta \notin \{U, V\}$, then B finds $((ID_\beta \| \| MPK_t), r, H_1(ID_\beta \| \| MPK_t))$ in L_{H1} to obtain r and computes $K_{\beta_t} = a_t \cdot r \cdot X$ and $y_2 = \hat{e}(K_{\beta_t}, H_1(ID_\alpha \| \| MPK_t))$. B then asks the $\boldsymbol{Hash_{h2}}$ query to obtain $v_\beta = h_2(ID_\alpha \| \| ID_\beta \| \| MPK_t \| \| y_1 \| \| y_2)$ and returns <ID_β, N_β, v_β>.
   c. ***Send*** (Π_β^n, <AID, N_α, "ACAKE">): B randomly generates $n_\beta \in_R Z_q^*$, computes $N_\beta = n_\beta \cdot P$ and $y_1 = n_\beta \cdot N_\alpha$, and asks the $\boldsymbol{Hash_{H1}}$ query to obtain $H_1(ID_\alpha \| \| MPK_t)$ and $H_1(ID_\beta \| \| MPK_t)$. If $\beta \in \{U, V\}$, then B randomly chooses $y_2 = g$. If $\beta \notin \{U, V\}$, then B finds $((ID_\alpha \| \| MPK_t), r, H_1(ID_\alpha \| \| MPK_t))$ in L_{H1} to obtain r and computes $K_{\beta_t} = a_t \cdot r \cdot X$, $TK = \hat{e}(N_\alpha, K_{\beta_t})$, $ID_\alpha = AID \oplus TK$, $y_2 = \hat{e}(K_{\beta_t}, H_1(ID_\alpha \| \| MPK_t))$. B

then asks the $\boldsymbol{Hash_{h2}}$ query to obtain $v_\beta = h_2(ID_\alpha || ID_\beta || MPK_t || y_1 || y_2)$ and returns $<N_\beta, v_\beta>$.

4. ***Execute*** (Π_α^m, Π_β^n): B asks the corresponding ***Send***query to obtain the result and return it.

B answers with $g = \hat{e}(P, P)^{xyz}$ to the DBDH problem if A answers with $b = 1$ to the $Test_{SK}$ query and answers with $g \neq \hat{e}(P, P)^{xyz}$ if A answers with $b = 0$. Whether B can successfully solve the DBDH problem depends on whether A asks the *Send* and $Test_{SK}$ queries on members U and V. The probability of A asking the *Send* and $Test_{SK}$ queries on member U, whose partner is V, or on member V (whose partner is U) is $2/(m + n + d)$. If A successfully guesses b in the $Test_{SK}$ query with advantage δ, which is non-negligible, then B can solve the ECCDH problem with advantage $2\delta/(m + n + d)$, which is also non-negligible. It contradicts the DBDH assumption, and thus the proposed protocols achieve existential unforgeability. □

Proof of Theorem 3. Suppose that there are one trusted third-party registration center (*RC*), m Server, n natural persons (*NP*), and d IoT devices in the smart city system, and each entity has a unique identity. Suppose that there is an adversary A, who has the ability to break the anonymity of *NP* in the proposed ACAKE protocol with a non-negligible advantage in probabilistic polynomial time. Then, we can construct a Challenge Algorithm B to solve the DBDH problem with a non-negligible advantage by using A's ability of forging a member. An instance (G_1, G_2, P, $\hat{e}$, q, $X = xP$, $Y = yP$, $Z = zP$, g) of the DBDH problem is given to B, and B's goal is to determine if $g = \hat{e}(P, P)^{xyz}$. B guesses that A tends to break the anonymity of a natural person, who makes an ACAKE with entity U. B permeates the DBDH problem into the queries. B manages one time seed list L_{seed} and four hash lists L_{H1}, L_{H2}, L_{h1}, and L_h, which are empty initially. For each t-time period, B randomly generates a_t, $r_t \in_R Z_q^*$, computes $MPK_t = a_t \cdot X$, appends (t, a_t, MPK_t) to list L_{seed}, and appends (($ID_U || MPK_t$), r_t, $r_t \cdot Y$) to list L_{H1}, i.e., let $H_1(ID_U || MPK_t) = r_t \cdot Y$. Note that $MRK_t = a_t \cdot x$ is unknown here. B sets public parameters Pub = {q, G_1, G_2, P, $\hat{e}$} and simulates the oracle queries:

1. The ***Hash***, ***Reveal***, ***Corrupt***, and ***Test*** queries are identical to those in the proof of Theorem 1, and the ***Extract*** query is identical to that in the proof of Theorem 2.
2. ***Send*** (Π_β^n, M): There are three different types of ***Send*** query in the proposed protocols: Update, CAKE, and ACAKE. Assume that the current time is in t-time period. B checks list L_{seed} to obtain (t, a_t, MPK_t), asks the $\boldsymbol{Hash_H}$ query to obtain $H_1(ID_\beta || MPK_t)$, finds (($ID_\beta || MPK_t$), r_β, $H_1(ID_\beta || MPK_t)$) in L_{H1} to obtain r_β, and responds according to the type of ***Send*** query.
   a. ***Send*** (RC, $<ID_\alpha, t^{OLD}, N_1, N_2$, "Update">): B checks list L_{seed} to obtain (t^{OLD}, $a_{t\ OLD}$, ${MPK_t}^{OLD}$), asks the $\boldsymbol{Hash_H}$ query for ($ID_\alpha || MPK_t$), and finds (($ID_\alpha || MPK_t$), r_t, $H_1(ID_\alpha || MPK_t)$) and (($ID_\alpha || {MPK_t}^{OLD}$), r_{tOLD}, $H_1(ID_\alpha || {MPK_t}^{OLD})$) in L_{H1} to obtain r_t and r_{tOLD}. If $\alpha = U$, then B finds (($ID_A || t^{OLD} || N_1 || R || K_{\alpha_(t\mathrm{OLD})}$), N_2) in list L_{h2} to obtain R and $K_{\alpha_(t\mathrm{OLD})}$ and computes $K_{\alpha_t} = a_t \cdot {a_{tOLD}}^{-1} \cdot r_t \cdot {r_{tOLD}}^{-1} \cdot K_{\alpha_(t\mathrm{OLD})}$. If $\alpha \neq U$, then B computes $K_{\alpha_(t\mathrm{OLD})} = a_t \cdot r_{t\mathrm{OLD}} \cdot X$ and $K_{\alpha_t} = a_t \cdot r_t \cdot X$. B then asks the $\boldsymbol{Hash}_{H2}$ query to obtain $H_2(K_{\alpha_(t\mathrm{OLD})} || R)$, asks the $\boldsymbol{Hash}_{h2}$ query to obtain $d_{\alpha_t} = h_2(MPK_t || K_{\alpha_t} || K_{\alpha_(t\mathrm{OLD})} || R)$, computes $U_{\alpha_t} = K_{\alpha_t} + H_2(K_{\alpha_(t\mathrm{OLD})} || R)$ and $V_{\alpha_t} = MPK_t + H_2(K_{\alpha_(t\ \mathrm{OLD})} || N_1)$, and returns $<ID_\alpha, t, U_{\alpha_t}, V_{\alpha_t}, d_{\alpha_t}>$.
   b. ***Send*** (Π_β^n, $<ID_\alpha, N_\alpha$, "CAKE">): B randomly generates $n_\beta \in_R Z_q^*$, computes $N_\beta = n_\beta \cdot P$ and $y_1 = n_\beta \cdot N_\alpha$, and asks the $\boldsymbol{Hash_H}$ query to obtain $H(ID_\alpha || MPK_t)$. If $\beta \neq U$, then B computes $K_{\beta_t} = a_t \cdot r_\beta \cdot X$ and $y_2 = \hat{e}(K_{\beta_t}, H_1(ID_\alpha || MPK_t))$. If $\beta = U$, then B finds (($ID_\alpha || MPK_t$), r, $H_1(ID_\alpha || MPK_t)$) in L_{H1} to obtain r and computes $K_{\alpha_t} = a_t \cdot r \cdot X$ and $y_2 = \hat{e}(K_{\alpha_t}, H_1(ID_\beta || MPK_t))$. B then asks the $\boldsymbol{Hash_{h2}}$ query to obtain $v_\beta = h_2(ID_\alpha || ID_\beta || MPK_t || y_1 || y_2)$ and returns $<ID_\beta, N_\beta, v_\beta>$.
   c. ***Send*** (Π_β^n, $<AID, N_\alpha$, "ACAKE">): B randomly generates $n_\beta \in_R Z_q^*$ and computes $N_\beta = n_\beta \cdot P$ and $y_1 = n_\beta \cdot N_\alpha$. If $\beta = U$, then B randomly chooses v_β

$\in_R \{0, 1\}^l$. If $\beta \neq U$, then B computes $K_{\beta_t} = a_t \cdot r_\beta \cdot X$, $TK = \hat{e}(N_\alpha, K_{\beta_t})$, and $ID_\alpha = AID \oplus TK$, asks the $\boldsymbol{Hash_{H1}}$ query to obtain $H_1(ID_\alpha \| MPK_t)$, computes $y_2 = \hat{e}(K_{\beta_t}, H_1(ID_\alpha \| MPK_t))$, and asks the $\boldsymbol{Hash_{h2}}$ query to obtain $v_\beta = h_2(ID_\alpha \| ID_\beta \| MPK_t \| y_1 \| y_2)$. B then returns $<N_\beta, v_\beta>$.

3. ***Execute*** $(\Pi_\alpha^m, \Pi_\beta^n)$: Suppose that the current time is in t-time period. B checks list L_{seed} to obtain (t, a_t, MPK_t). B asks the $\boldsymbol{Hash_{H1}}$ query to obtain $H_1(ID_\alpha \| MPK_t)$ and $H_1(ID_\beta \| MPK_t)$ and finds $((ID_\alpha \| MPK_t), H_1(ID_\alpha \| MPK_t), r_\alpha)$ and $((ID_\beta \| MPK_t), H_1(ID_\beta \| MPK_t), r_\beta)$ in L_{H1} to obtain r_α and r_β.
   a. $\boldsymbol{Execute_{UD}}$ (α, RC): B chooses t^{OLD}, checks list L_{seed} to obtain $(t^{OLD}, a_{tOLD}, MPK_t{}^{OLD})$, randomly generates $n \in_R Z_q{}^*$, and computes $N_1 = n \cdot P$ and $R = n \cdot MPK_t{}^{OLD}$. If $\beta = U$, then B randomly chooses $N_2 \in_R \{0, 1\}^l$. If $\beta \neq U$, then B asks the $\boldsymbol{Hash_{h2}}$ query to obtain $N_2 = h_2(ID_A \| t^{OLD} \| N_1 \| R \| a_{tOLD} \cdot r_\alpha \cdot X)$. B asks the ***Send*** $(RC, <ID_\alpha, t^{OLD}, N_1, N_2, \text{“Update”}>)$ query to obtain $<ID_\alpha, t, U_{\alpha_t}, V_{\alpha_t}, d_t>$. B then returns $<ID_\alpha, t^{OLD}, N_\alpha, \text{“Update”}>$ and $<ID_\alpha, t, U_{\alpha_t}, V_{\alpha_t}, d_{\alpha_t}>$.
   b. $\boldsymbol{Execute_{CAKE}}$ $(\Pi_\alpha^m, \Pi_\beta^n)$: There must be one out of $\alpha \neq U$ and $\beta \neq U$ that is true. If $\alpha \neq U$, then B computes $K_{\alpha_t} = a_t \cdot r_\alpha \cdot X$, asks the $\boldsymbol{Hash_{H1}}$ query to obtain $H_1(ID_\beta \| MPK_t)$, and computes $y_2 = \hat{e}(K_{\alpha_t}, H_1(ID_\beta \| MPK_t))$. If $\beta \neq U$, then B computes $K_{\beta_t} = a_t \cdot r_\beta \cdot X$, asks the $\boldsymbol{Hash_H}$ query to obtain $H_1(ID_\alpha \| MPK_t)$, and computes $y_2 = \hat{e}(K_{\beta_t}, H_1(ID_\alpha \| MPK_t))$. B randomly generates $n_\alpha, n_\beta \in_R Z_q{}^*$ and computes $N_\alpha = n_\alpha \cdot P$, $N_\beta = n_\beta \cdot P$, and $y_1 = n_\alpha \cdot n_\beta \cdot P$. B asks the $\boldsymbol{Hash_{h1}}$ query to obtain $SK = h_1(ID_\alpha \| ID_\beta \| N_\alpha \| N_\beta \| y_1 \| y_2)$ and asks the $\boldsymbol{Hash_{h2}}$ query to obtain $h_2(SK)$, $v_\alpha = h_2(ID_\alpha \| ID_\beta \| h_2(SK))$, $v_\beta = h_2(ID_\alpha \| ID_\beta \| MPK_t \| y_1 \| y_2)$.
   c. $\boldsymbol{Execute_{ACAKE}}$ $(\Pi_\alpha^m, \Pi_\beta^n)$: If $\beta = U$, then B lets $N_\alpha = Z$, $TK = g$. If $\beta \neq U$, then B randomly generates $n_\alpha \in_R Z_q{}^*$ and computes $N_\alpha = n_\alpha \cdot P$ and $TK = \hat{e}(N_\alpha, a_t \cdot r_\beta \cdot X)$. B computes $AID = TK \oplus ID_U$, randomly generates $n_\beta \in_R Z_q{}^*$ and computes $N_\beta = n_\beta \cdot P$, $y_1 = n_\beta \cdot N_U$, and $y_2 = \hat{e}(a_t \cdot r_\alpha \cdot X, H_1(ID_\beta \| MPK_t))$. B asks the $\boldsymbol{Hash_{h2}}$ query to obtain $v_\beta = h_2(ID_\alpha \| ID_\beta \| MPK_t \| y_1 \| y_2)$ and asks the $\boldsymbol{Hash_{h1}}$ query to obtain $SK = h_1(ID_\alpha \| ID_\beta \| N_U \| N_\beta \| y_1 \| y_2)$. B then asks the $\boldsymbol{Hash_{h2}}$ query to obtain $v_\alpha = h_2(ID_\alpha \| ID_\beta \| h_2(SK))$ and returns $\{<AID, N_U, \text{“ACAKE”}>, <N_\beta, v_\beta>, <v_\alpha>\}$.

B answers with $g = \hat{e}(P, P)^{xyz}$ to the DBDH problem if A answers with $b = 1$ to the $Test_{ID}$ query and answers with $g \neq \hat{e}(P, P)^{xyz}$ if A answers with $b = 0$. Whether B can successfully solve the DBDH problem depends on whether A asks the *Send* and *Test* queries on members U and V. The probability of A asking the $Send_{ACAKE}$ and $Test_{ID}$ queries on any one of the natural members, whose partner is U, is $n/(m + n + d)$. If A successfully guesses b in the *Test* query with advantage δ, which is non-negligible, then B can solve the ECCDH problem with advantage $\delta n/(m + n + d)$, which is also non-negligible. It contradicts the DBDH assumption; hence, the proposed protocol achieves existential unforgeability. □

References

1. Alam, T. A reliable communication framework and its use in Internet of Things (IoT). *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.* **2018**, *2*, 450–456.
2. Agiwal, M.; Roy, A. Next generation 5G wireless networks: A comprehensive survey. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 1617–1655. [CrossRef]
3. Borkar, S.; Pande, H. Application of 5G next generation network to Internet of Things. In Proceedings of the IoT and Applications (IOTA), Pune, India, 22–24 January 2016; pp. 443–447.
4. Raol, S.K.; Prasad, R. Impact of 5G technologies on smart city implementation. *Wirel. Pers. Commun.* **2018**, *100*, 161–176.
5. Cia, M.D.; Mason, F.; Peron, D.; Chiariotti, F. Using smart city data in 5G self-organizing networks. *IEEE IoT* **2019**, *5*, 645–654.
6. Tiburski, R.T.; Amaral, L.A.; Hessel, F. Security challenges in 5G-Based IoT middleware systems. In *Internet of Things (IoT) in 5G Mobile Technologies*; Springer: Cham, Switzerland, 2016; pp. 399–418.

7. Chuang, Y.H.; Lei, C.L.; Shiu, H.J. Cryptanalysis of four biometric based authentication schemes with privacy-preserving for multi-server environment and design guideline. In Proceedings of the 15th Asia Joint Conference on Information Security (AsiaJCIS), Taipei, Taiwan, 20–21 August 2020; pp. 66–73.
8. Lamport, L. Password authentication with insecure communication. *Commun. ACM* **1981**, *24*, 770–772. [CrossRef]
9. Wang, W.; Chen, Y.; Zhang, Q. Privacy-preserving location authentication in Wi-Fi networks using fine-grained physical layer signature. *IEEE Trans. Wirel. Commun.* **2016**, *15*, 1218–1225. [CrossRef]
10. Chuang, Y.H.; Lei, C.L. An independent three-factor mutual authentication and key agreement scheme with privacy preserving for multiserver environment and a survey. *Int. J. Commun. Syst.* **2021**, *34*, e4660.
11. Li, X.; Sangaiah, A.K.; Kumari, S.; Wu, F.; Shen, J.; Khan, M.K. An efficient authentication and key agreement scheme with user anonymity for roaming service in smart city. *Pers. Ubiquit Comput.* **2017**, *21*, 791–805. [CrossRef]
12. Li, J.L.; Zhang, W.G.; Dabra, V.; Choo, K.K.R.; Kumari, S.; Hogrefe, D. AEP-PPA: An anonymous, efficient and provably-secure privacy- preserving authentication protocol for mobile services in smart cities. *J. Netw. Comput. Appl.* **2019**, *134*, 52–61. [CrossRef]
13. Reddy, A.G.; Suresh, D.; Phaneendra, K.; Shin, J.S.; Odelude, V. Provably secure pseudo-identity based device authentication for smart cities environment. *Sustain. Cities Soc.* **2018**, *41*, 878–885. [CrossRef]
14. Xie, Q.; Hwang, L. Security enhancement of an anonymous roaming authentication scheme with two-factor security in smart city. *Neurocomputing* **2019**, *347*, 131–138. [CrossRef]
15. Jegadeesan, S.; Azees, M.; Kumar, P.M.; Manogaran, G.; Chilamkurti, N.; Varatharajan, R.; Hsu, C.H. An efficient anonymous mutual authentication technique for providing secure communication in mobile cloud computing for smart city applications. *Sustain. Cities Soc.* **2019**, *49*, 101522. [CrossRef]
16. Alzahrani, B.A.; Chaudhry, S.A.; Barnawi, A.; Al-Barakati, A.; Alsharif, M.H. A privacy preserving authentication scheme for roaming in IoT-based wireless mobile networks. *Symmetry* **2020**, *12*, 287. [CrossRef]
17. Chuang, Y.H.; Tseng, Y.M.; Lei, C.L. Efficient mutual authentication and key agreement with user anonymity for roaming services in global mobility networks. *Int. J. Innov. Comput. Inf. Control* **2012**, *8*, 6415–6428.
18. Tseng, Y.M.; Huang, S.S.; You, M.L. Strongly secure ID-based authenticated key agreement protocol for mobile multi-server environments. *Int. J. Commun. Syst.* **2017**, *30*, e3251. [CrossRef]
19. Yu, Y.; Hu, L.; Chu, J. A secure authentication and key agreement scheme for IoT-based cloud computing environment. *Symmetry* **2020**, *12*, 150. [CrossRef]
20. Chuang, Y.H.; Tseng, Y.M. Towards generalized ID-based user authentication for mobile multi-server environment. *Int. J. Commun. Syst.* **2012**, *25*, 447–460. [CrossRef]
21. Tseng, Y.M.; Huang, S.S.; Tsai, T.T.; Ke, J.H. List-free ID-based mutual authentication and key agreement protocol for multi-server architectures. *IEEE Trans. Emerg. Top. Comput.* **2016**, *4*, 102–122. [CrossRef]
22. Chuang, Y.H.; Lei, C.L. Privacy protection for telecare medicine information systems with multiple servers using a biometric-based authenticated key agreement scheme. *IEEE Access* **2019**, *7*, 186480–186490.
23. Odelu, V.; Das, A.K.; Goswami, A. A secure biometrics-based multi-server authentication protocol using smart cards. *IEEE Trans. Inf. Forensics Secur.* **2015**, *10*, 1953–1966. [CrossRef]
24. Park, Y.; Park, Y. Three-Factor user authentication and key agreement using elliptic curve cryptosystem in wireless sensor networks. *Sensors* **2016**, *16*, 2123. [CrossRef]
25. Qi, M.; Chen, Y. A secure biometrics-based authentication key exchange protocol for multi-server TMIS using ECC. *Comput. Meth. Prog. Bio.* **2018**, *164*, 101–109. [CrossRef]
26. Mukhopadhyay, D. PUFs as promising tools for security in internet of things. *IEEE Des. Test.* **2016**, *33*, 103–115. [CrossRef]
27. Tuyls, P.; Skoric, B. Secret key generation from classical physics: Physical uncloneable functions. *Amiware Hardw. Technol. Driv. Ambient Intell.* **2006**, *5*, 421–447.
28. Herder, C.; Yu, M.D.; Koushanfar, F.; Devadas, S. Physical unclonable functions and applications: A tutorial. *Proc. IEEE* **2014**, *8*, 1126–1141. [CrossRef]
29. Dodis, Y.; Ostrovsky, R.; Reyzin, L.; Smith, A. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *Siam J. Comput.* **2008**, *38*, 97–139. [CrossRef]
30. Kang, H.; Hori, Y.; Katashita, T.; Hagiwara, M.; Iwamura, K. Cryptographie key generation from PUF data using efficient fuzzy extractors. In Proceedings of the 16th International Conference on Advanced Communication Technology (ICACT), Pyeongchang, Korea, 16–19 February 2014; pp. 23–26.
31. Delvaux, J.; Gu, D.; Verbauwhede, I.; Hiller, M.; Yu, M.M. Efficient fuzzy extraction of PUF-induced secrets: Theory and applications. In Proceedings of the 18th International Conference on Cryptographic Hardware and Embedded Systems (CHES), Santa Barbara, CA, USA, 17–19 August 2016; pp. 412–431.
32. Chatterjee, U.; Govindan, V.; Sadhukhan, R.; Mukhopadhyay, D.; Chakraborty, R.S.; Mahata, D.; Prabhu, M.M. Building PUF based authentication and key exchange protocol for IoT without explicit CRPs in verifier database. *IEEE Trans. Dependable Secur. Comput.* **2019**, *16*, 424–437. [CrossRef]
33. Choi, K.Y.; Hwang, J.Y.; Lee, D.H.; Seo, I.S. ID-based authenticated key agreement for low-power mobile devices. In Proceedings of the 10th Information Security and Privacy (ACISP), Brisbane, Australia, 4–6 July 2005; Volume 3574, pp. 494–505.
34. Boneh, D.; Franklin, M. Identity-based encryption from the Weil pairing. In Proceedings of the 21th Annual International Cryptology Conference (CRYPTO), Santa Barbara, CA, USA, 14–16 August 2001; Volume 2139, pp. 213–229.

35. Boneh, D.; Franklin, M. Identity-based encryption from the Weil pairing. *SIAM J. Comput.* **2003**, *32*, 586–615. [CrossRef]
36. Choi, K.Y.; Hwang, J.Y.; Lee, D.H. Efficient ID-based group key agreement with bilinear maps. In Proceedings of the Public Key Cryptography (PKC), Singapore, 1–4 March 2004; Volume 2947, pp. 130–144.
37. Wang, D.; Wang, P. Two birds with one stone: Two-factor authentication with security beyond conventional bound. *IEEE Trans. Dependable Secur. Comput.* **2018**, *15*, 708–722. [CrossRef]
38. Kocher, P.; Jaffe, J.; Jun, B. Differential power analysis. In Proceedings of the 19th Annual International Cryptology Conference (CRYPTO), Santa Barbara, CA, USA, 19–23 August 1999; pp. 388–397.
39. Messerges, T.S.; Dabbish, E.A.; Sloan, R.H. Examining smart-card security under the threat of power analysis attacks. *IEEE Trans. Comput.* **2002**, *51*, 541–552. [CrossRef]
40. Rane, S.; Wang, Y.; Draper, S.C.; Ishwar, P. Secure biometrics: Concepts, authentication architectures, and challenges. *IEEE Signal. Process. Mag.* **2013**, *30*, 51–64. [CrossRef]
41. Rührmair, U.; Sölter, J.; Sehnke, F.; Xu, X.; Mahmoud, A.; Stoyanova, V.; Dror, G.; Schmidhuber, J.; Burleson, W.; Devadas, S. PUF modeling attacks on simulated and silicon data. *IEEE Trans. Inf. Forensics Secur.* **2013**, *8*, 1876–1891. [CrossRef]
42. Bellare, M.; Rogaway, P. Random oracles are practical: A paradigm for designing efficient protocols. In Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS), Fairfax, VA, USA, 3–5 November 1993; pp. 62–73.
43. Scott, M.; Costigan, N.; Abdulwahab, W. Implementing cryptographic pairings on smartcards. In Proceedings of the Cryptographic Hardware and Embedded Systems (CHES), Yokohama, Japan, 10–13 October 2006; Volume 4249, pp. 134–147.
44. Cao, X.; Zeng, X.; Kou, W.; Hu, L. Identity-based anonymous remote authentication for value-added services in mobile networks. *IEEE Trans. Veh. Technol.* **2009**, *58*, 3508–3517. [CrossRef]
45. Xiong, H.; Qin, Z. Revocable and scalable certificateless remote authentication protocol with anonymity for wireless body area networks. *IEEE Trans. Inf. Forensics Secur.* **2015**, *10*, 1442–1455. [CrossRef]
46. Cavalieri, S.; Cutuli, G. Implementing encryption and authentication in KNX using Diffie-Hellman and AES algorithms. In Proceedings of the 35th Annual Conference of IEEE Industrial Electronics, Porto, Portugal, 3–5 November 2009; pp. 2459–2464.