UFinAKA: Fingerprint-Based Authentication and Key Agreement With Updatable Blind Credentials

Mei Wang, Jing Chen, *Senior Member, IEEE*, Kun He, *Associate Member, IEEE*, Ruozhou Yu, *Senior Member, IEEE*, Ruiying Du, and Zhihao Qian

***Abstract*— Authentication and key agreement are two basic functionalities to guarantee secure network communications, which are naturally integrated as an *Authentication and Key Agreement (AKA)* protocol. AKAs usually either need a dedicated device to store a cryptographic key or require the user to remember a password. In recent years, AKAs built on biometrics, *e.g.*, human fingerprints, have gained research attention since they avoid these issues. Unlike keys or passwords that can be updated, biometrics are at greater risk that cannot be reused once disclosed. However, existing mechanisms either explicitly expose the biometrics to the server or consume a massive amount of resources. This paper proposes UFinAKA, a privacy-preserving fingerprint-based authentication and key agreement system with updatable blind credentials. UFinAKA explores a fingerprint-based blind credential authentication scheme as a building block such that the server has no access to the fingerprint data hidden within the credential. Furthermore, UFinAKA provides an updatable fingerprint-based credentials AKA protocol, which allows the server to update the blind credentials and guarantees anonymous fingerprint authentication to mitigate further leakage when the server is corrupted. We perform security analysis and experimental evaluation on UFinAKA. The evaluation results show that UFinAKA requires only linear computation overhead for the client, a single round of interaction, and roughly linear computation and storage cost for the server. The running time of UFinAKA is at least 4 times faster than the state-of-the-art solutions, and the storage cost of these solutions is at least 100 times more than UFinAKA.**

***Index Terms*— Fingerprint, authentication and key agreement, updatable, privacy-preserving.**

Manuscript received 4 May 2022; revised 27 May 2023; accepted 20 August 2023; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Zhang. This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFB2700200 and Grant 2021YFA1000600; in part by the National Natural Science Foundation of China under Grant 62302271, Grant 61772383, Grant U1836202, Grant 62076187, and Grant 62172303; in part by the Natural Science Foundation of Shandong Province under Grant ZR2023QF088 and Grant ZR2023MF045; in part by the Natural Science Foundation of Qingdao under Grant 23-2-1-152-zyyd-jch; and in part by the Shandong University Young Scholars Future Program. *(Corresponding authors: Jing Chen; Kun He.)*

Mei Wang is with the School of Cyber Science and Technology and the Key Laboratory of Cryptologic Technology, Shandong University, Qingdao 266237, China, and also with the Quan Cheng Laboratory, Jinan 250103, China (e-mail: wangmeiz@sdu.edu.cn).

Jing Chen, Kun He, and Ruiying Du are with the Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China (e-mail: chenjing@whu.edu.cn; hekun@whu.edu.cn; duraying@whu.edu.cn).

Ruozhou Yu is with the Department of Computer Science, North Carolina State University, Raleigh, NC 27606 USA (e-mail: ryu5@ncsu.edu).

Zhihao Qian is with the School of Computer Science, Wuhan University, Wuhan 430072, China (e-mail: qianzhihao@whu.edu.cn).

Digital Object Identifier 10.1109/TNET.2023.3311130

I. INTRODUCTION

SECURE network communication is a critical requirement of online services, such as online shopping, online healthcare, and finance, protecting the enormous amounts of sensitive user data in such services. As fundamental components of secure network communications, *authentication* provides assurance on the identities of participants involved in the communications, while *key agreement* provides data security during communication sessions against unauthorized parties. Conventionally, authentication and key agreement are achieved using separate protocols, such as in IP Security Internet Key Exchange (IPSec IKE) [1]. In response to the need for both functionalities in most services, integrated *authentication and key agreement (AKA)* protocol that achieves both goals simultaneously has emerged, *e.g.*, in mobile networks [2].

Traditional AKAs usually verify the user identity by validating a secret key stored within a terminal, which may be manipulated by an attacker and is not convenient when the user loses or wants to change the terminal [3]. Password-Authenticated Key Agreement (PAKA) [4], [5], [6], [7], [8] appeared to establish a cryptographic key based on the knowledge of a shared password. However, PAKA is inherently vulnerable to weak passwords, and the users need to put in the effort to remember multiple passwords in daily life. Compared with existing methods, the unique biometrics of human beings provide convenient tools for authenticating human users without storing or remembering their identity credentials. In recent years, biometric-based AKAs emerged and have gained popularity [3], [9], [10], [11], [12], [13], [14], [15]. As the General Data Protection Regulation (GDPR) [16] has classified biometric data as sensitive data to be protected, those biometric AKAs that require the server to store biometric data [9], [10], [11], [12] may be exploited by an adversary once the server is compromised. Erwig et al. [13] proposed Fuzzy Asymmetric Password-Authenticated Key Exchange (fuzzy aPAKE) protocols, but they need frequent interactions between participants, and are only suitable for biometric characteristics in the form of strings, *e.g.*, iris. Wang et al. [3] proposed a one-round Biometrics-Authenticated Key Exchange (BAKE) framework. Nevertheless, BAKE is designed for end-to-end communications where the server plays the role of a forwarder, not for *Client/Server* scenarios where the server is semi-honest but may be compromised.

Our goal is to propose a novel fingerprint AKA system to improve security and efficiency by mitigating the above issues.

We focus on biometric fingerprint because the fingerprint is the most promising biometric for AKAs mainly due to its reliability, ease of use, low-cost capture scanners, and historical employment in border control and law enforcement. However, integrating the fingerprint in AKA is not trivial. Specifically, this paper focuses on addressing the following three critical challenges: 1) **Credential privacy disclosure.** We require that the credential is constructed based on a fingerprint while keeping the fingerprint private. Nevertheless, an adversary may infer information from this credential if the fingerprint is not well-processed [17], [18], [19]. Traditional standard encryption (*e.g.*, AES [20]) may result in low accuracy because two scans of the same fingerprint are rarely identical. Thus, the credential should be blind for the server to preserve fingerprint privacy while guarantee authentication accuracy in the face of noises in realistic environments. 2) **Credential embezzlement risk.** Considering the potentially compromised server and insecure communication channel, an adversary can obtain data from different parts of the AKA. That is, the credential database may be compromised and embezzled to impersonate the roles in the AKA [21], [22]. We require the AKA to prevent further disclosure by updating the credentials and providing anonymous authentication once the corruption is detected. 3) **Expensive operation issue.** Existing research generally utilizes resource-consuming cryptographic techniques, such as *Garbled Circuit (GC)* [23], [24] and *Homomorphic Encryption (HE)* [25], [26] which are not suitable for mobile devices with limited computing power and battery. The AKA should be lightweight in terms of computation overhead, and minimize the interactions between the user and authentication server. In this paper, we propose a privacy-preserving **U**pdatable **Fin**gerprint-based blind credentials **AKA** system, named UFinAKA, to address these challenges by making the following main contributions:

- To prevent the credential privacy disclosure, we propose a Fingerprint-based Blind Credential Authentication (FBCA) scheme as a building block for UFinAKA, where a fingerprint is transformed into a rotation-invariant template hidden in a blind credential. This algorithm guarantees that an adversary cannot infer useful fingerprint information from the blind credential.
- To mitigate the credential embezzlement risk, UFinAKA provides an updatable fingerprint-based blind credentials AKA protocol, which prevents further leakage by updating the compromised credentials and guarantees anonymous authentication. The AKA requires neither the trusted server nor the trusted channel. This enables UFinAKA to self-recover from corruption once the corruption is detected, incurring no additional operations for the client.
- To solve the expensive operation issue, UFinAKA is instantiated with computationally lightweight cryptographic tools, *e.g.*, the garbled Bloom filter. UFinAKA involves only a single round of interaction and linear computation complexity during the process of AKA.
- We analyze the security of UFinAKA in theory. Besides, we implement a prototype and evaluate the performance of UFinAKA with realistic fingerprint databases. The running time/storage cost of UFinAKA is at least 4 times/100 times less than [10], [14], and [13].

TABLE I
COMPARISON WITH THE STATE-OF-THE-ART SOLUTIONS

Scheme	Biometric Privacy	Updatable Credentials	Anonymous Authentication	Single-round Interaction
RFE [10], [14]	✓	×	×	×
MFAKE [12]	✓	×	×	×
fPAKE [11]	×	×	×	×
fuzzy aPAKE [13]	✓	×	×	×
BAKE [3]	✓	×	×	✓
UFinAKA	✓	✓	✓	✓

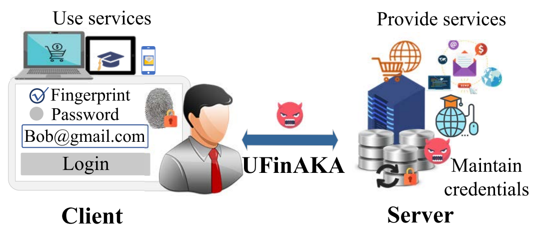


Fig. 1. System model.

TABLE I compares UFinAKA to other state-of-the-art biometric AKA solutions. In the rest of this paper, Section II shows the problem statement, including the system model, threat model, and design goals. We give the basic notation and primary background knowledge in Section III. Section IV describes the technical roadmap, system architecture, and interfaces. We depict the main technical details of UFinAKA in Section V, and give the security analysis and evaluation in Section VI and Section VII, respectively. Section VIII reviews related work, and Section IX concludes this paper.

II. Problem Statement

A. System Model

This work seeks to achieve a two-party integrated authentication and key agreement system. There are two participants in our model (Fig. 1): the *client* and the *server*. The client registers to the server with a blind *credential* in which a fingerprint *template* is hidden. The blind credentials are maintained in a credential database by the server. Sometimes we use the term "user" instead of "client", which refers to a human who possesses the fingerprint and manipulates the client. When the client initiates a session request to the server, they execute a fingerprint AKA protocol to negotiate on a session key, establishing a secure communication channel. If the credential database is compromised, the server can self-recover by updating the credential database independently once the corruption is detected. This model applies to the *Client/Server* scenarios, *e.g.*, online shopping.

B. Threat Model

From the practical perspective, we require the fingerprint template and the agreed session key to be protected in the face of the following security assumptions.

- **Safeguarded Client.** The client honestly processes the users' fingerprint data in a safeguarded environment, strictly abides by the protocol, and does not reveal its fingerprint data to the adversary.

- **Potentially Compromised Server.** The server is vulnerable to an adversary, who tries to learn more information about the client, and can obtain all the credentials and even the secret items stored on the server.
- **Insecure Channel.** We assume the communication channel between the client and server is not secure when the client and server negotiate on a secret session key. All transmitted messages on the insecure channel could be eavesdropped on, tampered with, or forged.

C. Design Goals

UFinAKA aims to provide fingerprint AKA protocol to secure network communications in the Client/Server scenarios with the following properties.

- **Blind Credential**. The fingerprint is processed and protected in the blind credential against the server.
- **Updatable Fingerprint-based Blind Credentials AKA**. The server and the client negotiate a shared secret session key with mutual authentication. The server can update the credential database independently to exclude the lost ones once a credential disclosure attack is detected.
- **Anonymous Authentication**. The server authenticates the user without knowing his identity.
- **Lightweight and Low Interactions**. Both the communication and computation overhead should be low, which is critical for practical applications.

III. Preliminaries

A. Notation

We denote the security parameter as $\lambda \in \mathbb{N}$ where $\mathbb{N}$ is the set of all natural numbers, and denote the number of minutia points that a fingerprint template contains as $n \in \mathbb{N}$. Given a set $\{r_i\}_n$ that is converted from a fingerprint, we refer to its i^{th} element as r_i and denote the set of size n as $\{\}_n$. All cyclic groups $\mathbb{G}$ of prime order q and generators $g \in \mathbb{G}$ mentioned in this paper are generated as referred to Section 9.3.1 in [20]. We write $sk \in_R \mathbb{Z}_q$ to represent an element sk being sampled uniformly at random from $\mathbb{Z}_q$ where $\mathbb{Z}$ is the set of all integers, and q is a prime.

B. Garbled Bloom Filter

Bloom filter [27] is a probabilistic compact data structure for testing set membership. Specifically, a Bloom filter is formed by an array of p bits which represents a set of at most n elements, and a set of m independent uniform hash functions $H = \{h_0, \ldots, h_{m-1}\}$ which map elements to the index numbers uniformly over the range $[0, p-1]$. Dong et al. [28] designed a variant Garbled Bloom Filter (GBF) consisting of two algorithms (GBF.Build, GBF.Query). In a (p, n, m, H)-GBF, each position $GBF[i]$ ($i \in [0, p-1]$) stores a λ-bit string. We slightly devise the GBF such that the server can store a pair set $\{(x_i, y_i)\}_n$ within a GBF. The algorithm GBF.Build$(\{(x_i, y_i)\}_n, p, n, m, H, \lambda) \rightarrow GBF$ satisfies $\bigoplus_{j=0}^{m-1} GBF[h_j(x_i)] = y_i$, and positions that are not mapped by $\{(x_i, y_i)\}_n$ are chosen uniformly. The algorithm GBF.Query$(GBF, x_i, m, H) \rightarrow y_i$ outputs $y_i = \bigoplus_{j=0}^{m-1} GBF[h_j(x_i)]$ ($i \in [1, n]$) to the client.

C. Verifiable Secret Sharing

Shamir's (t, n) secret sharing scheme [29] splits a secret value tok into n shares with the property that any t or more than t shares can recover tok, while any less than t shares reveal no information about tok. We use Feldman Verifiable Secret Sharing (VSS) scheme to provide the above functionality, and further verify the validity of shares. The VSS scheme consists of three algorithms (VSS.Share, VSS.Verify, VSS.Recon). The share generation algorithm SS.ShareGen$(tok, t, n) \rightarrow (\{tok_i\}_n, \{com\}_t)$ splits a secret tok into n shares $\{tok_i\}_n$, and generates t commitments $\{com\}_t$. The verify algorithm VSS.Verify$(\{com\}_t, tok_i) \rightarrow 0/1$ checks whether the share tok_i ($i \in [1, n]$) is valid or not. The reconstruction algorithm SS.ShareRecon$(\{tok_i\}_n, t) \rightarrow tok$ outputs the recovered tok as long as the number of valid shares is not less than the threshold t.

D. Minutiae-Based Fingerprint Representation

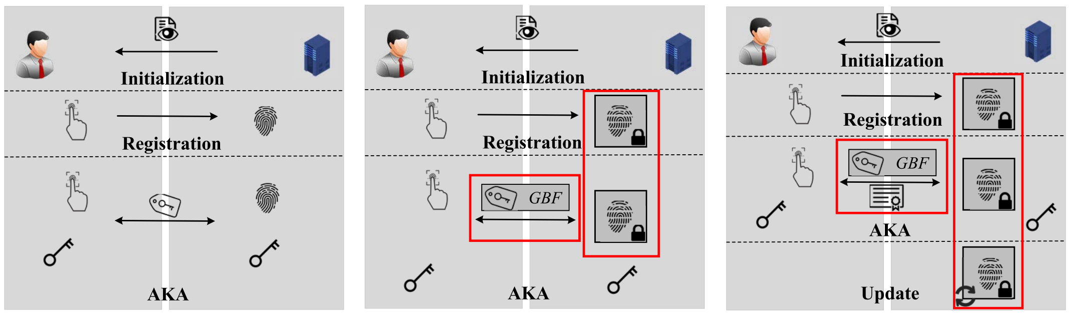
The fingerprint is a unique pattern of ridges and valleys on the surface of an individual finger. Minutiae points [30] are defined as the positions of local discontinuities where the ridge splits or ends. To extract high-accuracy minutiae with varied-quality fingerprint images, the segmentation algorithm first separates the foreground from the noisy background. Then, the original ridge flow pattern is kept with an image enhancement algorithm without introducing false information. Finally, minutiae points are located accurately with binarized minutiae extraction methods. Minutiae points are typically represented as follows: 1) an X-coordinate, 2) a Y-coordinate, 3) an orientation θ, corresponding to the angle between the minutiae ridge and the horizontal line measured in degrees.

IV. An Overview of UFinAKA

A. Technical Roadmap

We introduce the primary considerations and technical line in designing UFinAKA as follows.

1) A Naïve Solution: Plain Fingerprint AKA. Following the research line of [11], we begin with a plain fingerprint authentication and key agreement system that is vulnerable to fingerprint credential privacy disclosure (Fig. 2a). This naïve solution works as follows. In the initialization phase, the server initializes the whole system by producing the public parameters pp. In the registration phase, a client registers with its plain fingerprint template by storing the template as the credential on the server. In the AKA phase, the client requests to the server; the server retrieves the corresponding credential, then binds the template with tok and sends it to the client, where tok is a randomly-chosen secret; the client recovers tok with its correct fingerprint, and a secret session key K is negotiated based on this shared tok between the client and the server. In this solution, the credential directly discloses fingerprint data if it is lost. The breached fingerprint data can no longer be used for authentication because anyone who possesses it can impersonate the client.



(a) A Naïve Solution. (b) Fingerprint-based Blind Credential Authentication. (c) Updatable Fingerprint-based Blind Credentials AKA.

Fig. 2. Technical Roadmap.

2) Fingerprint-Based Blind Credential Authentication: Fingerprint-based Blind Credential Authentication (*FBCA*, Fig. 2b). A natural approach to protecting the fingerprint is to encrypt/hash the template (*e.g.*, AES/SHA-256). However, the fingerprint is not precisely reproducible. Two scans of the same fingerprint are rarely identical, resulting in two different encrypted/hashed templates, which cannot be employed as credentials. Processing then protecting noisy fingerprints becomes a significant challenge that UFinAKA is facing. Current solutions usually depend on heavy cryptographic primitives, *e.g.*, Garbled Circuit [23], which involves heavy overhead. To solve this issue, we propose a new primitive, Fingerprint-based Blind Credential Authentication (FBCA), which shards a fingerprint into a set of strings $\{r_i\}_n$, and hides $\{r_i\}_n$ as a blind credential $\{b_i\}_n$ via encryption. In addition, the authentication functionality is achieved by transmitting a secret tok within a GBF by the server to the client, where tok is retrieved from the GBF only if the client is legal.

3) Updatable Fingerprint-Based Blind Credentials AKA: Updatable Fingerprint-based Blind Credentials AKA (Fig. 2c). The FBCA has guaranteed the privacy of the fingerprint when the blind credential is breached, but still cannot be reused. Consider an adversary who compromises two credential databases of two independent systems. Then the adversary may link the identities by comparing the credentials. UFinAKA addresses this issue in two steps.

- Enabling the server to update blind credentials. The core idea is that the server maintains a secret update factor upd which is stored as $cupd = Enc(sek, upd)$ locally where sek is a standard encryption key. Specifically, when the server detects a corruption, upd is converted to $upd' = upd * \alpha$ and the credential $\{b_i\}_n$ is updated as $b_i' = b_i^\alpha$ where $\alpha \in_R \mathbb{Z}_q$. In addition, the server computes an auxiliary item $w = g^{upd'}$ and sends it to the client, and the client utilizes w to synchronize with $\{b_i'\}_n$ using the new captured fingerprint template. To this end, the lost credential $\{b_i\}_n$ is invalid. Even though the attacker obtains sek and recovers $upd = Dec(sek, cupd)$, he/she still cannot obtain upd' and $\{b_i'\}_n$ because α is randomly and secretly chosen, and sek is also updated to sek' in the update phase. Furthermore, since the update factor is always random, the credentials of the same fingerprint in two independent systems are different.
- Providing anonymous authentication. We require the client generates a different anonymous username every time, with which the server retrieves the same corresponding credential. In detail, the user chooses a random $\varpi \in \mathbb{Z}_q$, then obtains the anonymous username composed of $\mathfrak{ID}_{\mathfrak{C}} = g^{h(ID_C)} \oplus (g^{sk})^{\varpi}$ and g^{ϖ}, where ID_C is the username, $h : \{0,1\}^* \rightarrow \mathbb{Z}_q$ is a hash function, g^{sk} is the public key of the server. Upon receiving $\mathfrak{ID}_{\mathfrak{C}}$, g^{ϖ}, the server computes $\Psi_{ID_C} = \mathfrak{ID}_{\mathfrak{C}} \oplus (g^{\varpi})^{sk}$ where $sk \in_R \mathbb{Z}_q$. The anonymous username $\mathfrak{ID}_{\mathfrak{C}}, g^{\varpi}$ is always different, while the server recovers the same identifier Ψ_{ID_C} with sk, and updates it along with the credential by using upd. Therefore, the adversary cannot link accounts with the compromised identifiers and credentials.

Mitigating server impersonation attacks (Fig. 2c). There is a problem that UFinAKA is still vulnerable to server impersonation attacks where the adversary maliciously plays the role of the server with stolen $\{b_i\}_n$. This issue occurs because we verify the client identity via fingerprint, but have no verification about the server. UFinAKA addresses this issue by attaching a succinct identity proof (digital signature) for the server. In detail, the server possesses a private key $sk \in_R \mathbb{Z}_q$ and a public key g^{sk}, where g is a generator of a cyclic group $\mathbb{G}$ of prime order q; the identity proof $\eta = r_S + l \cdot sk$ where $r_S \in_R \mathbb{Z}_q$, g^{r_S} is public, and l is generated via the key agreement by both the client and server; the client identifies the server by checking whether $g^{\eta} = g^{r_S} \cdot (g^{sk})^l$. Furthermore, if the private key sk is also stolen, the server updates $\{b_i\}_n$, sk and the public g^{sk} to invalidate the lost ones. Thus, the server impersonation attack no longer works.

Low computation overhead and interaction rounds (Fig. 2c). In UFinAKA, we require that an adversary cannot obtain information about the fingerprint template and the agreed session key from the transcripts delivered on the insecure channel. In addition, it should be practical enough for reality. On the one hand, UFinAKA provides a single-round AKA protocol by proposing the single-round primitive, FBCA. It is critical in practice since network latency is a significant bottleneck in secure communications, especially over mobile networks. On the other hand, to strive for excellent user experience, UFinAKA costs minor computation resources by employing lightweight cryptographic techniques (*e.g.*, the garbled Bloom filter) in FBCA construction. The single-round interaction

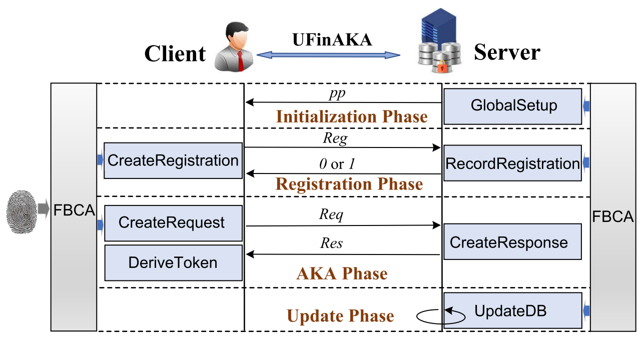


Fig. 3. Architecture of UFinAKA.

and linear computation complexity imply that UFinAKA is suitable for real-world applications.

B. The System Architecture of UFinAKA

UFinAKA intends to provide an updatable fingerprint-based blind credentials AKA system to guarantee secure communications. We propose a new primitive, *Fingerprint-based Blind Credential Authentication* (FBCA) as a building block. As Fig. 3 shows, UFinAKA includes the following four phases. The first two phases act as system preparation, and the remaining two phases present the main process of an updatable fingerprint-based blind credentials AKA.

1) In the *initialization* phase, the server generates the public parameters and the master key.
2) In the *registration* phase, the client registers to the server with its fingerprint-based blind *credential*, and erases the local storage of the terminal. The server receives and stores the credential.
3) In the *AKA* phase, the server and the client negotiate on a session key by running a fingerprint AKA protocol. More specifically, the server authenticates the client with the client's fingerprint, while the server proves the identity with a proof of the possession of a master key.
4) In the *update* phase, the server can update the compromised credentials without raising additional computation overhead for the client.

C. FBCA Interface

The FBCA scheme provides fingerprint privacy preservation, fingerprint error tolerance, and implicit fingerprint authentication, which is defined as follows.

Definition 1 (Fingerprint-Based Blind Credential Authentication): A fingerprint-based blind credential authentication scheme FBCA *consists of five Probabilistic Polynomial Time (PPT) algorithms (*Setup, Shard, Hide, Token, Auth*) that satisfies the correctness property below.*

- Setup$(1^\lambda, \tau) \rightarrow par$*: On input a security parameter* $\lambda \in \mathbb{N}$ *and a threshold* $\tau \in \mathbb{N}$*, this setup algorithm outputs the public parameter* par*, which are implicit inputs to the following algorithms.*
- Shard$(finp) \rightarrow \{r_i\}_n$*: On input the fingerprint image* $finp$*, this fingerprint sharding algorithm outputs a string set* $\{r_i\}_n$ *as the template.*
- Hide$(\{r_i\}_n) \rightarrow \{b_i\}_n$*: On input the template* $\{r_i\}_n$*, this hiding algorithm outputs the blind credential* $\{b_i\}_n$*.*
- Token$(\{b_i\}_n, tok) \rightarrow ht$*: On input the blind credential* $\{b_i\}_n$*, a secret* tok*, this token algorithm outputs a hidden token* ht*.*
- Auth$(ht, \{r_j^*\}_n) \rightarrow tok/\perp$*: On input a hidden token* ht *and the regenerated fingerprint template* $\{r_j^*\}_n$*, this error-tolerant authentication algorithm outputs a token* tok *or* $\perp$*.*

Correctness: *For any* $\lambda \in \mathbb{N}$*, any* $par \leftarrow$ Setup*, any fingerprint templates* $\{r_i\}_n \leftarrow$ Shard$(finp)$ *and* $\{r_j^*\}_n \leftarrow$ Shard$(finp^*)$ *where* $finp$ *and* $finp^*$ *are captured from the same fingerprint, any blind credential* $\{b_i\}_n \leftarrow$ Hide$(\{r_i\}_n)$*, we have* $tok \leftarrow$ Auth$(ht, \{r_j^*\}_n)$ *where* $ht \leftarrow$ Token$(\{b_i\}_n, tok)$*.*

Remark 2 To intuitively aid understanding, we tailor FBCA *for the fingerprint to show the technical intuition. Actually, this method applies to any other noisy factors* (*e.g., environment*) *that can be transformed into a template represented as a set.*

D. UFinAKA Interface

We design UFinAKA to guarantee secure communications in the Client/Server scenarios, which is defined as follows.

*Definition 3 (Updatable Fingerprint-Based Blind Credentials Authentication and Key Agreement): An updatable fingerprint-based blind credentials AKA authentication and key agreement system UFinAKA contains seven PPT algorithms (*GlobalSetup, CreateRegistration, RecordRegistration, CreateRequest, CreateResponse, DeriveToken, UpdateDB*) satisfying the correctness property.*

Initialization Phase:

- GlobalSetup$(1^\lambda) \rightarrow (pp, mk)$*: This algorithm is run by the server, it takes as input a security parameter* λ*, outputs the public parameters* pp *containing all the parameters this system needs and a master key* mk*, where* pp *are implicit inputs to the following algorithms, but* mk *is held secretly by the server.*

Registration Phase:

- CreateRegistration$(finp) \rightarrow Reg$*: This algorithm is run by the client, it takes a fingerprint image* $finp$ *as input, outputs a registration request* Reg *that is transmitted to the server.*
- RecordRegistration$(Reg, sk, DB) \rightarrow \{0,1\}$*: This algorithm is run by the server, it takes as input a registration request* Reg*, a credential database* DB*, outputs* 0 *if* Reg *is not a fresh request, otherwise outputs 1.*

AKA Phase:

- CreateRequest$(finp^*) \rightarrow (LS, Req)$*: This algorithm is run by the client, it takes as input a fingerprint image* $finp^*$*, outputs a local state* LS *that is temporarily stored in a safeguarded environment, and an authentication request* Req *that is transmitted to the server.*
- CreateResponse$(DB, Req) \rightarrow (Res, K_{ID_C})$*: This algorithm is run by the server, it takes as input a credential database* DB*, an authentication request* Req*,*

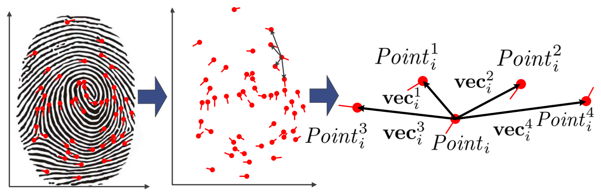


Fig. 4. Diagram of fingerprint sharding.

outputs an authentication response Res, and a session key K_{ID_C}.

- DeriveToken$(LS, Res) \rightarrow K^*_{ID_C} / \perp$: *This algorithm is run by the client, it takes as input a local state LS, and an authentication response Res, outputs a session key* $K^*_{ID_C}$ *if the authentication successes, otherwise outputs* $\perp$.

Update Phase:

- UpdateDB$(DB, mk) \rightarrow (pp', DB', mk')$: *This algorithm is run by the server, it takes as input a credential database DB, a master key mk, outputs updated items* (pp', DB', mk').

Correctness: For any $\lambda \in \mathbb{N}$, *any* $(pp, mk) \leftarrow$ GlobalSetup (1^λ), *any fingerprint images finp, finp* and their templates* $\{r_i\}_n$, $\{r_j^*\}_n$, *any set intersection threshold* $t \leq n$ *contained in pp, any* $Reg \leftarrow$ CreateRegistration$(finp)$ *that is recorded in DB by* RecordRegistration, *we have* $K_{ID_C} = K^*_{ID_C}$ *if the size of* $\{r_i\}_n \cap \{r_i^*\}_n$ *is not less than t, where*

- $(LS, Req) \leftarrow$ CreateRequest$(finp^*)$,
- $(Res, K_{ID_C}) \leftarrow$ CreateResponse(DB, Req),
- $K^*_{ID_C} \leftarrow$ DeriveToken(LS, Res).

Remark 4: For the UpdateDB *algorithm, we require that the above correctness property is still satisfied once the* UpdateDB *algorithm is invoked.*

V. UFinAKA: Updatable Fingerprint-Based Blind Credentials AKA

A. FBCA Construction

We design a concrete FBCA construction as a building block for UFinAKA, and give the technical description and construction as follows. The Setup algorithm outputs the public parameter $par = (\gamma, \tau, \mathbb{G}, q, g, p, n, m, H)$ where γ is the number of minutiae points that a fingerprint string involves, τ is the threshold of Hamming distance between two strings, $\mathbb{G}$ is a cyclic group of prime order q, and g is a generator of $\mathbb{G}$, (p, n, m, H) are parameters for the garbled Bloom filter.

1) Fingerprint Sharding: Most research utilizes a multi-dimensional vector to present a fingerprint [14], [31], *e.g.*, the FingerCode [32], which is a 640-dimensional vector of integers. However, FingerCode always causes redundant overhead in fingerprint authentication, because FingerCode is rotation variant such that a user usually corresponds to several FingerCodes in the database. To mitigate this issue, we turn to the minutiae-based fingerprint representation approach, which converts a fingerprint into a set of minutiae points. In addition, we utilize the relative positions of minutiae points to obtain a rotation-invariant fingerprint template.

Specifically, we first extract n minutiae points $\{Point_i\}_n$ in the *X-Y* coordinate space, and generate a fingerprint template

Algorithm 1 Encode$(finp) \rightarrow \{r_i\}_n$

```
1: Extract $n$ minutiae points $\{Point_i\}_n$ from $finp$
2: while $Point_i \in \{Point_i\}_n$ do
3:     Find the nearest $\gamma$ points to $Point_i$
4:     Construct $\gamma$ vectors $\{\mathbf{vec}_i^j\}_\gamma$
5:     Compute vector lengths $\{d_j\}_\gamma$ for $\{\mathbf{vec}_i^j\}_\gamma$
6:     Compute inter-vector angles $\{\phi_l\}_{\frac{\gamma(\gamma-1)}{2}}$ in $\{\mathbf{vec}_i^j\}_\gamma$
7:     Represent $\{\mathbf{vec}_i^j\}_\gamma$ as string $r_i$ with $\{d_j\}_\gamma$ and $\{\phi_l\}_{\frac{\gamma\cdot(\gamma-1)}{2}}$
8: end while
```

as the Encode algorithm (illustrated in Algorithm. 1) inspired by [30]. Each minutiae point is initialized as the central point for exactly one time. Then the straight-line nearest γ points are chosen to form a structure as shown in Fig. 4. We take $\gamma = 4$ as an example, $Point_i$ $(i \in [1, n])$ is the core point, $Point_i^j$ $(j \in [1, 4])$ is the top γ straight-line nearest points to $Point_i$. We define $\mathbf{vec}_i^j$ $(j \in [1, 4])$ as the vector from $Point_i$ to $Point_i^j$, let d_j denote the length of the vector $\mathbf{vec}_i^j$, and ϕ_l $(l \in [1, 6])$ denote the angles. Then, a string is represented as

$$r_i = d_1 d_2 d_3 d_4 \phi_1 \phi_2 \phi_3 \phi_4 \phi_5 \phi_6$$

concatenated with 10 values. Because this fingerprint sharding algorithm is based on the relative position of minutiae points, rotating fingerprint images does not affect the result of fingerprint representation.

2) Fingerprint Hiding: In reality, to preserve biometric privacy, biometric templates are typically stored secretly in a trusted environment of the terminal. However, this method is inconvenient when the user loses the terminal or wants to change the terminal to a new one. We intend to mitigate this issue by converting a fingerprint into a blind credential with which the client registers on the server. Specifically, to protect the fingerprint template $\{r_i\}_n$ against the compromised server, we generate a blind credential $\{b_i\}_n$ by hiding $\{r_i\}_n$ with cryptographic group operations. Considering the operability of the blind credential, we expect the server could update the blind credential as discussed in Section IV-A.3. Inspired by the Diffie-Hellman Key Exchange, for each string r_i $(i \in [1, n])$ in a fingerprint template $\{r_i\}_n$, we compute

$$b_i = g^{r_i} \pmod q$$

where q is the prime order of a cyclic group $\mathbb{G}$, and g is a generator of $\mathbb{G}$. We have a blind credential $\{b_i\}_n$ in which a fingerprint template $\{r_i\}_n$ is hidden.

3) Fingerprint Token: Generally, when the client wants to communicate with the server securely, it generates a new template $\{r_j^*\}_n$ for the registered fingerprint, then computes the set intersection between $\{g^{r_j^*}\}_n$ and $\{b_i\}_n$ to achieve fingerprint authentication. However, the server is not allowed to expose the credential to the client since the client may be controlled by an adversary, and neither is the client. To solve this problem, we need to compute the set intersection in a secure manner. Since two templates of the same fingerprint are usually different, we combine the Feldman VSS scheme with the garbled bloom filter to depict the fingerprint token

FBCA Construction

- Setup$(1^\lambda, \tau) \rightarrow par$: Output $par = (\gamma, \tau, \mathbb{G}, q, g, p, n, m, H)$.
- Shard$(finp) \rightarrow \{r_i\}_n$: Run Encode$(finp) \rightarrow \{r_i\}_n$, output $\{r_i\}_n$.
- Hide$(\{r_i\}_n) \rightarrow \{b_i\}_n$: For each r_i $(i \in [1, n])$, compute $b_i = g^{r_i} \pmod q$, output $\{b_i\}_n$.
- Token$(\{b_i\}_n, tok) \rightarrow ht$: Run VSS.Share$(tok, t, n) \rightarrow (\{tok_i\}_n, \{com\}_t)$. Choose $\beta \in_R \mathbb{Z}_q$ and compute $v = g^\beta \pmod q$. For each b_i $(i \in [1, n])$, compute $y_i = tok_i \oplus b_i^\beta$. Run $GBF \leftarrow$ GBF.Build$(\{(b_i, y_i)\}_n, p, n, m, H, \lambda)$. Output $ht = (GBF, v)$.
- Auth$(ht, \{r_j^*\}_n) \rightarrow tok/\perp$: Parse ht as (GBF, v). For each string r_j^* $(j \in [1, n])$, generate all strings $\{r_{j^*}^*\}_{n^*}$ that satisfy $ham(r_{j^*}^*, r_j^*) \leq \tau$. Then for each string $r_{j^*}^*$ $(j^* \in [1, n^*])$, compute $g^{r_{j^*}^*} \pmod q$ and run $y_{j^*}^* \leftarrow$ GBF.Query$(GBF, g^{r_{j^*}^*}, m, H)$, compute $tok_{j^*}^* = y_{j^*}^* \oplus v^{r_{j^*}^*}$. Run VSS.Verify$(\{com\}_t, tok_{j^*}^*)$ to check whether $tok_{j^*}^*$ is valid. Output $\perp$ if the number of valid shares is less than t and otherwise the output of VSS.ShareRecon on the t valid shares.

Fig. 5. FBCA construction.

algorithm on the server side. We first generate a secret tok, then split tok into n shares and generate t $(t \leq n)$ commitments by employing the Feldman VSS scheme. To further protect the shares and the credential, we enable the server to build a garbled bloom filter GBF by putting $tok_i \oplus b_i^\beta$ on the positions via hashing b_i. In this case, the client can query GBF to recover tok if the size of the set intersection between $\{g^{r_j^*}\}_n$ and $\{b_i\}_n$ is not less than the threshold t of VSS.

4) Fingerprint Authentication: As considered above, the captured fingerprint is usually noisy, two strings composed with the same minutiae points are not always identical. We have introduced the Feldman VSS scheme in the fingerprint token algorithm. However, computing the set intersection between $\{g^{r_j^*}\}_n$ and $\{b_i\}_n$ still may result in false results, because the minor errors lying in a string r_j^* $(j \in [1, n])$ also may affect the authentication result. To improve the authentication accuracy, we use the Hamming distance metric (*i.e.*, the number of positions that are different between two strings) denoted by $ham(\cdot, \cdot)$ to tolerate the string errors. Specifically, for each string r_j^*, the client finds all strings $\{r_{j^*}^*\}_{n^*}$ that are close to r_j^*, and hides them with the fingerprint hiding algorithm, then queries GBF to obtain and check the validity of tok shares. Finally, the client recovers tok as long as the number of valid shares is not less than the threshold t.

5) FBCA Construction: Putting the above technical details together, we give a concrete instantiation (Fig. 5) of FBCA, where $h : \{0, 1\}^* \rightarrow \mathbb{Z}_q$, and refer to this construction as FBCA from here onwards.

B. UFinAKA Construction

1) Initialization Phase: The initialization phase initializes the whole system and generates the system parameters that all entities need. Specifically, the server runs Algorithm 2, which takes the security parameter λ as input, outputs the public

Algorithm 2 Initialization

```
**Server:** GlobalSetup$(1^\lambda) \rightarrow (pp, mk)$
1: Create the server name $ID_S$
2: $(\tau, \gamma, \mathbb{G}, g, q) \leftarrow$ FBCA.Setup$(1^\lambda)$
3: Generate the key pair $(g^{sk}, sk)$ where $sk \in_R \mathbb{Z}_q$
4: Generate the key $sek$ for symmetric encryption
5: Specify the GBF hash functions $H = (h_0, \ldots, h_{m-1})$
6: Specify the template size $n$
7: Specify the set intersection threshold $t$
8: Specify the hash function $h : \{0, 1\}^* \rightarrow \mathbb{Z}_q$
9: $pp = \{ID_S, \lambda, g^{sk}, (\tau, \gamma, \mathbb{G}, g, q), H, n, t, h\}$
10: $mk = (sk, sek)$
```

system parameters pp and the master key mk. In real-world applications, the server could publish pp on a bulletin board such that all entities in UFinAKA can receive pp, while only the server maintains the mk locally.

2) Registration Phase: The registration phase shown in Algorithm 3 is the only part of UFinAKA that runs over an authenticated channel, which consists of CreateRegistration and RecordRegistration.

Before any activity, the client registers on the server with a registration request Reg created by CreateRegistration, which includes the anonymous username $\mathfrak{ID}_\mathfrak{C}$, g^ϖ and the fingerprint-based blind credential $\{b_i\}_n$. Specifically, the client chooses a random ϖ from $\mathbb{Z}_q$, then computes $\mathfrak{ID}_\mathfrak{C} = g^{h(ID_C)} \oplus (g^{sk})^\varpi$ and g^ϖ, where $h \in pp$ is a hash function. To obtain the fingerprint-based blind credential $\{b_i\}_n$, the client extracts minutiae points $\{Point_i\}_n$ from the user fingerprint, then invokes FBCA.Shard and FBCA.Hide algorithms to convert the fingerprint into $\{b_i\}_n$. Note that the fingerprint is processed in a secure environment (*e.g.*, within the *trusted execution environment*, a secure area built within the central processor), and all the fingerprint data are erased after the registration phase. In other words, the terminal does not need to store any secret items.

When receiving the registration request Reg from the client, the server runs RecordRegistration to record it into the database. The server first needs to determine if $Reg = (\mathfrak{ID}_\mathfrak{C}, g^\varpi, \{b_i\}_n)$ is a fresh request by computing $\Psi_{ID_C} = \mathfrak{ID}_\mathfrak{C} \oplus (g^\varpi)^{sk}$. The server stores $(\mathfrak{ID}_\mathfrak{C}, \{b_i\}_n)$ in the database DB if Ψ_{ID_C} is not registered, and rejects this request immediately, otherwise. We assume Reg is transmitted through an authenticated channel, which means no adversary can modify Reg. The requirement of an authenticated channel is essential for all AKA protocols [33], which can be implemented by the Public Key Infrastructure (PKI).

3) AKA Phase: As a principal part of UFinAKA, the AKA phase shown in Algorithm 4 presents an AKA protocol, which consists of CreateRequest, CreateResponse, and DeriveToken.

When a user needs to communicate with the server securely, the client runs CreateRequest to generate and send an AKA request Req to the server. In detail, Req includes the anonymous username $\mathfrak{ID}_\mathfrak{C}{}^*$, g^{ϖ^*} and a public parameter g^{r_C} $(r_C \in_R \mathbb{Z}_q)$ that assists in negotiating a session key with the server, where the secret r_C and g^{r_C} are stored temporarily as a local state LS. In addition, the client regenerates a fresh

Algorithm 3 Registration

```
Client: CreateRegistration($finp$) → $Reg$
1: Create the username $ID_C$
2: Generate $g^{\varpi}$, $(g^{sk})^{\varpi}$ where $\varpi \in_R \mathbb{Z}_q$
3: Let $\mathfrak{ID}_{\mathfrak{C}} = g^{h(ID_C)} \oplus (g^{sk})^{\varpi}$ where $h : \{0,1\}^* \rightarrow \mathbb{Z}_q$
4: $\{r_i\}_n \leftarrow$ FBCA.Shard($finp$)
5: $\{b_i\}_n \leftarrow$ FBCA.Hide($\{r_i\}_n$)
6: $Reg = (\mathfrak{ID}_{\mathfrak{C}}, g^{\varpi}, \{b_i\}_n)$
Server: RecordRegistration($Reg, sk, DB$) → $0/1$
1: Parse $Reg$ as $(\mathfrak{ID}_{\mathfrak{C}}, g^{\varpi}, \{b_i\}_n)$
2: Let $\Psi_{ID_C} = \mathfrak{ID}_{\mathfrak{C}} \oplus (g^{\varpi})^{sk}$
3: if $\Psi_{ID_C} \notin DB$ then
4:     Record $(\Psi_{ID_C}, \{b_i\}_n)$ in $DB$
5:     return 1
6: else
7:     return 0
8: end if
```

Algorithm 4 AKA

```
Client: CreateRequest($finp^*$) → $(LS, Req)$
1: $\{r_j^*\}_n \leftarrow$ FBCA.Shard($finp^*$)
2: Generate $(g^{r_C}, r_C)$ where $r_C \in_R \mathbb{Z}_q$
3: $LS = (\{r_j^*\}_n, g^{r_C}, r_C)$
4: Generate $g^{\varpi^*}$, $(g^{sk})^{\varpi^*}$ where $\varpi^* \in_R \mathbb{Z}_q$
5: Let $\mathfrak{ID}_{\mathfrak{C}}{}^* = g^{h(ID_C)} \oplus (g^{sk})^{\varpi^*}$ where $h : \{0,1\}^* \rightarrow \mathbb{Z}_q$
6: $Req = (\mathfrak{ID}_{\mathfrak{C}}{}^*, g^{\varpi^*}, g^{r_C})$
Server: CreateResponse($DB, Req$) → $(Res, K_{ID_C})$
1: Parse $Req$ as $(\mathfrak{ID}_{\mathfrak{C}}{}^*, g^{\varpi^*}, g^{r_C})$
2: Recover $\Psi_{ID_C} = \mathfrak{ID}_{\mathfrak{C}}{}^* \oplus (g^{\varpi^*})^{sk}$
3: Retrieve $DB$ for $(\Psi_{ID_C}, \{b_i\}_n)$
4: Generate $tok \in_R \{0,1\}^{\lambda}$
5: $ht \leftarrow$ FBCA.Token($\{b_i\}_n, tok$)
6: Generate $(g^{r_S}, r_S)$ where $r_S \in_R \mathbb{Z}_q$
7: Compute $g^{r_C r_S} = (g^{r_C})^{r_S}$
8: $K_{ID_C} = Hash(tok, ID_S, \Psi_{ID_C}, g^{r_C r_S})$
9: $c = AuthEnc(K_{ID_C}, ID_S)$
10: $l = Hash(g^{r_S}, g^{r_C})$
11: $\eta = r_S + l \cdot sk \pmod q$
12: $Res = (ht, c, g^{r_S}, \eta)$
Client: DeriveToken($LS, Res$) → $K_{ID_C}^* / \perp$
1: Parse $Res$ as $(ht, c, g^{r_S}, \eta)$
2: Parse $LS$ as $(\{r_j^*\}_n, g^{r_C}, r_C)$
3: $tok^* \leftarrow$ FBCA.Auth($ht, \{r_j^*\}_n$)
4: Compute $g^{r_C r_S} = (g^{r_S})^{r_C}$
5: $K_{ID_C}^* = Hash(tok^*, ID_S, g^{h(ID_C)}, g^{r_C r_S})$
6: $ID_S{}^* = AuthDec_{K_{ID_C}^*}(c)$
7: $l^* = Hash(g^{r_S}, g^{r_C})$
8: if $ID_S{}^* = ID_S$ & $g^{\eta} = g^{r_S} \cdot (g^{sk})^{l^*}$ then
9:     return $K_{ID_C}^*$
10: else
11:     return $\perp$
12: end if
```

fingerprint template $\{r_j^*\}_n$ from a newly captured fingerprint image $finp^*$, and temporarily stores $\{r_j^*\}_n$ in LS. In a real-world implementation, for better performance, the client may first transmit the Req, and process the fingerprint in the interval waiting for the response from the server. Note that LS is immediately erased after the AKA phase accomplishes, so the client stores nothing at ordinary times.

Upon receiving Req from the client, the server runs CreateResponse to generate a response Res. The server first recovers $\Psi_{ID_C} = \mathfrak{ID}_{\mathfrak{C}}{}^* \oplus (g^{\varpi^*})^{sk}$ and retrieves $(\Psi_{ID_C}, \{b_i\}_n)$ from DB, then generates a random token $tok \in_R \{0,1\}^{\lambda}$, and hides it in a hidden token ht by invoking the FBCA.Token algorithm, *i.e.*, the server creates a GBF to transmit tok to the client in a secure manner. Afterwards, the client creates the shared session key K_{ID_C} by hashing the items $(tok, ID_S, g^{h(ID_C)}, g^{r_C r_S})$ where $r_S \in_R \mathbb{Z}_q$. In addition, the server computes c by encrypting ID_S using authenticated encryption under K_{ID_C}, and appends c to Res so that the client can verify the validity of the shared session key. Finally, the server proves its identity to the client by generating an identity proof η.

When receiving Res from the server, the client obtains the shared session key $K_{ID_C}^*$ by running DeriveToken. The shared token tok^* is derived by invoking the FBCA.Auth, and we have $tok^* = tok$ if $finp$ and $finp^*$ come from the same fingerprint, and vice versa. The session key $K_{ID_C}^*$ is obtained exactly if both c and η are valid. At this point, the client has established a secure communication channel with the server under the session key $K_{ID_C}^* = K_{ID_C}$.

4) Update Phase: The update phase shown in Algorithm 5 is a novel part of UFinAKA to update the credential database. Considering the case that once the server is compromised, all identifiers Ψ_{ID_C}, blind credentials $\{b_i\}_n$ and even the master key $mk = (sk, sek)$ may be accessed by the attacker. An attacker who holds the lost mk may impersonate the server and break down the system. UFinAKA addresses this issue by enabling the server updates DB to declare the lost credentials invalid when detecting the corruption. We introduce an update factor upd that is initially set as $upd = 1$ for the server, and store two auxiliary parameters $cupd = Enc(sek, upd)$, $w_C = g^{upd}$ in DB, where sek is a standard encryption key. All users' credentials are updated with the same upd, and the client utilizes w_C to synchronize the fingerprint template $\{r_i^*\}_n$ with its blind credentials, *i.e.*, $w_C^{r_i^*} = g^{r_i^* \cdot upd}$. In a nutshell, the server randomly generates a new master key $mk' = (sk', sek')$ to exclude the compromised one, and picks a random $\alpha \in \mathbb{Z}_q$ to update Ψ_{ID_C}, $\{b_i\}_n$, pp, $cupd$, and w_C.

We emphasize that the above operations should be performed only when the attacker is no longer in control of the server. Since the server has made the lost $mk = (sk, sek)$ invalid and generates the new random $mk' = (sk', sek')$, we conclude that even though the attacker has mk, he/she still cannot obtain the new credential $\{b_i'\}_n = \{b_i^{\alpha}\}_n$ unless he/she corrupts the server again and obtains the new standard encryption key sek'. The server could perform update operation periodically or immediately when detecting the DB is compromised. Note that this process involves only the server, and the client is not required to perform additional operations.

Algorithm 5 Update

```
Server: UpdateDB($DB$,$mk$)→($pp'$,$DB'$,$mk'$)
1:  Parse $mk$ as $(sk, sek)$
2:  Generate the new key pair $(g^{sk'}, sk')$ where $sk' \in_R \mathbb{Z}_q$
3:  Generate the new symmetric key $sek' \in_R \mathbb{Z}_q$
4:  while $(\Psi_{ID_C}, \{b_i\}_n) \in DB$ do
5:      Retrieve the corresponding $(cupd, w_C)$ in $DB$
6:      $upd \leftarrow Dec(sek, cupd)$
7:      Pick $\alpha \in_R \mathbb{Z}_q$
8:      while $b_i \in \{b_i\}_n$ do
9:          Compute $b_i' = b_i^{\alpha}$
10:     end while
11:     Compute $upd' = upd * \alpha$, $w_C' = g^{upd'}$
12:     $cupd' = Enc(sek', upd')$
13:     Compute $\Psi_{ID_C}' = \Psi_{ID_C}^{\alpha}$
14:     Store $(\Psi_{ID_C}', \{b_i'\}_n)$, $(cupd', w_C')$ in $DB'$
15: end while
16: $pp' = \{ID_S, \lambda, g^{sk'}, (\tau, \gamma, \mathbb{G}, g, q), H, (t, n)\}$
17: $mk' = (sk', sek')$
```

VI. Security Analysis

Below, we analyze the security of UFinAKA using the simulation paradigm [34]. The simulation paradigm aims at showing that the adversary cannot distinguish the protocol from a random distribution with non-negligible advantage, without breaking certain cryptographic assumptions. The proof allows the adversary to implement an arbitrary side of the protocol, and use previously valid agreements as references, in order to cover a wide range of attacks including man-in-the-middle attacks, dictionary attacks, replay attacks, *etc*. We use $\mathrm{Adv}_{\mathcal{A}}(\lambda)$ to denote the advantage of a PPT adversary $\mathcal{A}$, and define the security of UFinAKA under the threat model with a safeguarded client, a potentially compromised server, and an insecure channel.

Definition 5: UFinAKA is considered to be secure if for every fingerprint dictionary D with size $\|D\|$ and for any PPT $\mathcal{A}$ that makes at most $Q(\lambda)$ online dictionary attacks, it holds that $\mathrm{Adv}_{\mathcal{A}}(\lambda) \leq Q(\lambda)/\|D\| + \mathrm{negl}(\lambda)$.

In UFinAKA, we employ an FBCA scheme that is secure based on the *Decisional Diffie-Hellman* (DDH) assumption. In a nutshell, DDH-assumption implies that no PPT adversary can computationally distinguish the following two distributions (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c), where a, b, and c are randomly and independently chosen from $\mathbb{Z}_q$.

Theorem 6: UFinAKA is secure with the advantage $\mathrm{Adv}_{\mathcal{A}}(\lambda) \leq Q(\lambda)/\|D\| + \mathrm{negl}(\lambda)$ *if* FBCA *is secure and all the hash functions are modelled as random oracles.*

Proof: (Sketch) A simulator $\mathcal{S}$ is constructed to run an adversary $\mathcal{A}$, and it can simulate the view of an honest server in an interaction with an honest client. Concretely, the simulator $\mathcal{S}$ generates the transcripts and handles the following steps:

- $\mathcal{S}$ retrieves (ht, c, g^{r_S}, η) from an honest server. In reality, $ht = (GBF, v)$ where GBF stores $(b_i, tok_i \oplus b_i^{\beta})$, $b_i = g^{r_i}$ $(i \in [1, n])$, and $v = g^{\beta}$ for the random number β.
- $\mathcal{S}$ randomly samples $\{\acute{r}_i\}_n$ and calculates $\{v^{\acute{r}_i}\}_n$, $\{g^{\acute{r}_i}\}_n$.
- $\mathcal{S}$ samples a dummy $\acute{tok}_i$ randomly rather than computing $\{y_i \oplus v^{r_i}\}_n$ in the real case. $\mathcal{S}$ then computes $\{\acute{y}_i = \acute{tok}_i \oplus v^{\acute{r}_i}\}_n$ using the sampled $\{\acute{tok}_i\}_n$ for $i \in [1, n]$.
- $\mathcal{S}$ checks whether $\acute{y}_i = \acute{tok}_i \oplus v^{\acute{r}_i}$ is identical to the dummy y_i in GBF for $i \in [1, n]$ in the real case. If validation is true, $\mathcal{S}$ then recovers b_i by calculating $g^{\acute{r}_i}$.
- $\mathcal{S}$ samples a random $\acute{r}_C$ and computes $g^{\acute{r}_C}$, then $\mathcal{S}$ computes $\acute{K}_{ID_C}$ via $Hash(\{\acute{tok}_i\}_n, ID_S, \Psi_{ID_C}, g^{\acute{r}_C r_S})$.
- $\mathcal{S}$ finally outputs the transcript $\big(g^{\acute{r}_C}, ((GBF(\{b_i, tok_i \oplus b_i^{\beta}\}_n), v), c, g^{r_S}, \eta), tok, \acute{K}_{ID_C}\big)$ with probability $1/M^n$. Indeed, the transcript is $\big(g^{\acute{r}_C}, ((GBF(\{g^{\acute{r}_i}, \acute{tok}_i \oplus v^{\acute{r}_i}\}_n), v), c, g^{r_s}, \eta), \acute{tok}, \acute{K}_{ID_C}\big)$. Otherwise, $\mathcal{S}$ outputs the transcript $\big(g^{\acute{r}_C}, ((GBF, v,)c, g^{r_S}, y), \perp, \perp\big)$ with probability $1-1/M^n$, where we assume the client responses with probability $1/M$ for each fingerprint template.

Below we analyze the security of UFinAKA using the following hybrids:

Hybird_0. Hybird_0 captures the real game, and the advantage of the adversary is $\mathsf{Adv}_{\mathcal{A}}^{\Pi}(\lambda, Q) = \mathsf{Adv}_{\mathcal{A}}^{\mathrm{Hybrid}_0}$ where we use Π to denote UFinAKA.

Hybird_1. Hybird_1 is identical to Hybird_0 with the exception of the fingerprint template. In Hybird_1, the compromised server $\mathcal{A}$ interacts with the client, and $\mathcal{S}$ simulates the transcripts of the client as follows,

- $\mathcal{S}$ sends $(\mathfrak{ID}_{\mathfrak{C}}, g^{r_C})$ to the server where $r_C \in_R \mathbb{Z}_q$.
- $\mathcal{S}$ receives (ht, c, g^{r_S}, η) from the server where $ht = (GBF, v)$.
- $\mathcal{S}$ extracts minutiae points and creates $\{r_i^*\}_n$, then calculates $\{v^{r_i^*}\}_n$ and $\{g^{r^*{}_i}\}_n$.
- $\mathcal{S}$ recovers y_i by reading $g^{r_i^*}$ from GBF, then attempts to recover tok_i by computing $y_i \oplus v^{r_i^*}$.
- $\mathcal{S}$ recovers tok and obtains the session key by computing $K_{ID_C}^* \leftarrow Hash(tok, ID_S, g^{h(ID_C)}, g^{r_S \cdot r_C})$.
- $\mathcal{S}$ aborts if it receives a different (ht, c, g^{r_S}, η) with probability $1 - 1/M^n$. Otherwise, $\mathcal{S}$ sends $\big(\mathfrak{ID}_{\mathfrak{C}}, g^{r_C}, ((GBF(\{b_i, tok_i \oplus b_i^{\beta}\}_n), v), c, g^{r_S}, \eta), \{tok_i\}, K_{ID_C}^*\big)$, where $tok_i := y_i \oplus v^{r_i^*}$.

We note that, the transcript $\big(\mathfrak{ID}_{\mathfrak{C}}, g^{r_C}, ((GBF(\{b_i, y_i \oplus b_i^{\beta}\}_n), v), c, g^{r_S}, \eta), tok, K_{ID_C}^*\big)$ outputted by the honest server is determined by the inner randomness. In this case, the transcript $\big(\mathfrak{ID}_{\mathfrak{C}}, g^{r_C}, ((GBF(\{b_i, y_i \oplus b_i^{\beta}\}_n), v), c, g^{r_S}, \eta), tok, K_{ID_C}^*\big)$ is from the real one. Hence, no adversary $\mathcal{A}$ can distinguish the two games with overwhelming probability. Namely, the view of the simulator $\mathcal{S}$ with the server is indistinguishable from the real transcript between the client and the server.

Hybird_2. Hybird_2 is identical to Hybird_1 except that the client modifies the approach to generate g^{r_C}. As we know that we regard g^{r_C} as a commitment. Hence, Hybird_2 and Hybird_1 are statistically indistinguishable via a straightforward reduction.

Hybird_3. Hybird_3 is identical to Hybird_2 with exception of the generation of tok. In this hybrid, the client simulates tok randomly from the uniform distribution with probability $1/\lambda$. Hence, Hybird_3 and Hybird_2 are statistically indistinguishable via a straightforward reduction.

Hybird_4. Hybird_4 is identical to Hybird_3 with the exception of y_i generation. In particular, we modify the approach to generate y_i by computing $y_i = tok_i \oplus v^{r^*{}_i}$ rather than $y_i := tok_i \oplus b_i^{\beta}$ for $i \in [1, n]$, and each tok_i is sampled

TABLE II
ASYMPTOTIC COMPARISON RESULTS

Scheme		Techniques	Rounds	Multiplications	Exponentiations	Hashes	Symmetric Encryption	Secret Sharing
RFE [10], [14]	server	DL	1	–	–	–	$\bar{n}^{\zeta}$	–
	client			–	–	–	$\bar{n}^{\zeta}$	–
PassBio [31]	server	TPE	1	$6\bar{n}^3+\zeta$	–	–	–	–
	client			$\bar{n}^2+\zeta$	–	–	–	–
fPAKE [11]-1	server	GC	5	–	$3\widetilde{n}+\zeta$	$4\widetilde{n}+\zeta$	–	–
	client			–	$3\widetilde{n}+\zeta$	$4\widetilde{n}+\zeta$	–	–
fPAKE [11]-2	server	PAKA +SS	2	–	$2\widetilde{n}+\zeta$	$\widetilde{n}$	–	ζ
	client			–	$2\widetilde{n}+\zeta$	$\widetilde{n}$	–	ζ
fuzzy aPAKE [13]-1	server	SS +OT	2	$\widetilde{n}+\zeta$	$4\widetilde{n}+\zeta$	$2\widetilde{n}+\zeta$	–	$2\widetilde{n}+\zeta$
	client			$2\widetilde{n}+\zeta$	$4\widetilde{n}+\zeta$	$3\widetilde{n}+\zeta$	1	$2\widetilde{n}+\zeta$
fuzzy aPAKE [13]-2	server	aPAKE	2	$2\widetilde{n}+\zeta$	$5\widetilde{n}+\zeta$	$2\widetilde{n}+\zeta$	–	$\widetilde{n}$
	client			$4\widetilde{n}+\zeta$	$5\widetilde{n}+\zeta$	$4\widetilde{n}+\zeta$	–	$2\widetilde{n}$
UFinAKA	server	FBCA	1	–	$n+\zeta$	$mn+\zeta$	–	-
	client			–	$2n+\zeta$	$mn+\zeta$	–	-

randomly. In this case, the simulator $\mathcal{S}$ simulates the fingerprint strings $\{r_i\}_n$ from uniform distribution with probability $1/\lambda^n$, and $\mathcal{S}$ is allowed to query the server at most $Q(\lambda)$ times. Thus, $\mathsf{Adv}_{\mathcal{A}}^{\text{Hybrid}_4}(\lambda) \leq Q(\lambda)/\lambda^n + \text{negl}(\lambda)$. In addition, $tok_i \oplus v^{r^*{}_i}$ is indistinguishable from $tok_i \oplus b_i^{\beta}$ because $v^{r^*{}_i}$ is indistinguishable from b_i^{β}.

Therefore, the advantage of the adversary is bounded $Q(\lambda)/\lambda^n + \text{negl}(\lambda)$, and the Theorem 6 is proved. ■

VII. EVALUATION

A. Asymptotic Comparison

We give the asymptotic comparison with the state-of-the-art in TABLE II. We use two rows to describe the server's and the client's theoretical performance, respectively, in terms of Techniques, Rounds, Multiplications, Exponentiations, Hashes, Symmetric Encryption, and Secret Sharing. Each round is defined as a bi-directional interaction between the client and the server. Because these representative schemes use different types of templates, we utilize three distinct parameters ($\bar{n}$, $\widetilde{n}$, n) to analyze the complexity, where $\bar{n}$ denotes the dimension of a vector template, $\widetilde{n}$ denotes the bit-length of a string template, n denotes the size of a multi-string template, and ζ is a constant.

An $\bar{n}$-dimensional vector template is employed in RFE [10], [14], *e.g.*, FingerCode is a vector of integers with dimension 640. The RFE [10], [14] is efficient with single-round interaction that is the same as UFinAKA, but the server needs to store a string, which may be attacked. Similar to RFE [10], [14], PassBio [31] based on Threshold Predicate Encryption (TPE) has single-round interaction, whose performance bottleneck lies in matrix multiplication and matrix inversion, which are transformed to the number of multiplications. However, PassBio only achieves privacy-preserving fingerprint authentication functionality, the number of rounds increases for PassBio to negotiate a session key. We present PassBio in the comparison because PassBio provides a representative fingerprint authentication approach without using RFE. Another type of template is represented by an $\widetilde{n}$-bit string in fPAKE [11] (including two constructions named fPAKE-1 and fPAKE-2) and fuzzy aPAKE [13](including two constructions named fuzzy aPAKE-1 and fuzzy a PAKE-2), *e.g.*, a 2048 bit IrisCode.

We note that fPAKE [11]-1 needs both frequent interactions and heavy computation, fPAKE [11]-2 is more efficient than fPAKE [11]-1 in terms of the number of rounds and computation complexity. However, both these two constructions need a trusted server that is responsible for the security of fingerprint templates. We also notice that both fuzzy aPAKE [13]-1 and fuzzy aPAKE [13]-2 protect the biometric template from the server, but they still need two interactions and heavy computation. In addition, they work only for the biometric template in the form of the bit-string, *e.g.*, the IrisCode. In contrast, UFinAKA solves all the above issues and needs only single-round interaction, approximately linear exponentiations (n is the number of fingerprint strings, the experimental average n is 95, m is generally set as 3). In addition, UFinAKA supports anonymous authentication and updatable blind credentials.

B. Implementation

To measure the performance of UFinAKA, we implement a prototype in Python using two laptops as the server and the client, with the Intel (R) Core (TM) i5-8300H CPU @ 2.30 GHz, 8 GB memory, and Intel (R) Core (TM) i7-8500U @ 1.80GHz, 8 GB memory, respectively. The FBCA is instantiated with Curve25519 and the hash functions with SHA-256. We conduct experiments on four databases from the Third International Fingerprint Verification Competition (FVC2004) [35], fingerprint images are preprocessed with the OpenCV library, and minutiae points are extracted as coordinate values. For Encode, we experimentally choose $\gamma = 4$ to present a fingerprint string using ten 4-bit values.

C. Experimental Results

1) Running Time: We investigate the UFinAKA computation performance with the running time of main consuming algorithms. Measurements are given as the average over 100 tests on different databases (DB1, DB2, DB3, DB4 in

TABLE III
RUNNING TIME (S) OF ALGORITHMS IN FBCA

	n	FBCA.Shard	FBCA.Blind	FBCA.Token	FBCA.Auth
DB1	95	0.094	0.029	0.021	0.5542
DB2	91	0.100	0.028	0.020	0.5362
DB3	138	0.130	0.043	0.031	0.864
DB4	150	0.145	0.045	0.039	0.963

FVC2004). Note DB1 and DB2 are natural human fingerprints, while the distorted DB3 and synthetic DB4 involve many noisy points. The results are presented in TABLE III.

In the registration phase, the client needs to perform FBCA.Shard and FBCA.Hide algorithms. The results show that both FBCA.Shard and FBCA.Hide are very efficient, costs less than 0.2 seconds in all. In the AKA phase, the client invokes FBCA.Shard and FBCA.Auth algorithms. We note that the computation performance is roughly linear with the number of minutiae points. The client costs less than 1 second, even for low-quality fingerprint images in DB4, which is acceptable in practice. FBCA.Auth is the most consuming operation for the client, mainly because it involves lots of exponentiation operations. Fortunately, the number of exponentiations depends on n, only $40 \sim 100$ in general. In addition, we can execute FBCA.Shard algorithm at the same time as sending the Req to shorten the running time. The server runs only FBCA.Token algorithm in the AKA phase, which mainly involves a large number of lightweight operations, *e.g.*, hash functions. The running time of FBCA.Shard is less than 0.04 seconds, even for the noisiest DB4. Therefore, UFinAKA is efficient for both the client and the server.

2) Communication Time: We measured the communication performance in various network environments. For the transcripts Reg, Req, and Res, we set six different network bandwidths (1 Mbps to 300 Mbps) on the Local Area Network (LAN) and the Internet, respectively. We find that different databases in FVC2004 have little effect on the communication performance, possibly because of the network fluctuation, so we only show the representative results on DB1. As shown in TABLE IV, Res costs the most communication resources. The transmission time is about 465 milliseconds on 1 Mbps Internet, which is acceptable for practical applications. Besides, the results show irregular fluctuations occasionally, especially for the Req on the Internet. We consider the main reason is the network instability, and the transmitted items are not heavy enough to show the regular results, which explains the practicability from another side. In addition, UFinAKA needs only single-round interaction in the AKA phase, which signally contributes to its communication advancement.

3) Server Concurrent Experiment: We measure UFinAKA for the server with multiple clients concurrently since the service provider often serves multiple clients simultaneously. Our instantiated FBCA.Token mainly involves VSS.ShareGen and GBF.Build, we briefly test the running time of them. In addition, the server may update its database regularly, and store the blind credentials for multiple clients. We test

TABLE IV
COMMUNICATION TIME (ms) IN VARIOUS NETWORKS

		1 Mbps	10 Mbps	50 Mbps	100 Mbps	200 Mbps	300 Mbps
LAN	Reg	7.2	9.5	9.2	5.4	4.8	4.1
	Req	2.9	2.6	2.1	1.5	1.6	1.4
	Res	7.3	8.7	7.2	7.6	6.8	6.1
Internet	Reg	294.3	209.5	210.6	188.8	187.3	170.8
	Req	3.5	4	2.2	2.2	2	2
	Res	465	309.5	298.7	292.5	289.9	289

Reg is the registration request, Req is the AKA request, Res is the AKA response.

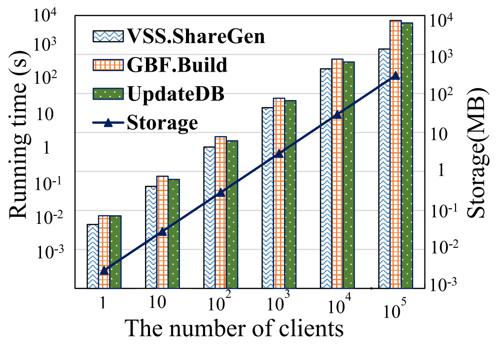


Fig. 6. Running time and storage cost at server side.

the running time of VSS.ShareGen, GBF.Build, UpdateDB and the storage cost to show the practical concurrent performance of the server. As shown in Fig. 6, both the running time and the storage cost roughly increase linearly with the number of clients. In the case of one client, the running time of VSS.ShareGen, GBF.Build, UpdateDB are all less than 0.01 seconds, showing the practicability of UFinAKA. Besides, in the update phase, once the credential database is compromised, the server updates the database by invoking UpdateDB independently. Since the client can synchronize with the updated credential database by utilizing the auxiliary items depicted in the update phase, the update operations bring no additional overhead to the AKA protocol.

4) Comparisons: We compare the experiential performance of UFinAKA with the state-of-the-art solutions to show the practicability of UFinAKA. As concluded in Section VII-C.2, the transcripts in UFinAKA are of small size, and the experimental communication time is affected by the network environment. However, the communication performance is partly reflected by the server storage overhead. So we compare the running time and server storage overhead with RFE [10], [14], fuzzy aPAKE-1 and fuzzy aPAKE-2 [13]. To make the comparison more convincing, we implement RFE as required in [10] and [14], and instantiate fuzzy aPAKE-1 and fuzzy aPAKE-2 as recommended in [13]. Both RFE and fuzzy aPAKE [13] employ 5 FingerCodes as Res since FingerCodes are rotation variant as recommended in [31]. Note that we set the hamming error threshold is 2 in both [13] and [31] to show the upper bounds of the performance. The comparison results are shown in Fig 7. The running time of RFE, fuzzy aPAKE-1, fuzzy aPAKE-2 is more than 4, 1300, 1400 times cost of UFinAKA, respectively. The storage overhead of RFE, fuzzy aPAKE-1, fuzzy aPAKE-2 is more than 400, 100, 8×10^5 times

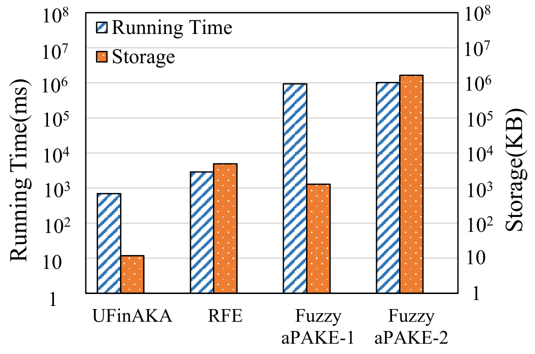


Fig. 7. Comparison of running time (ms) and storage (KB).

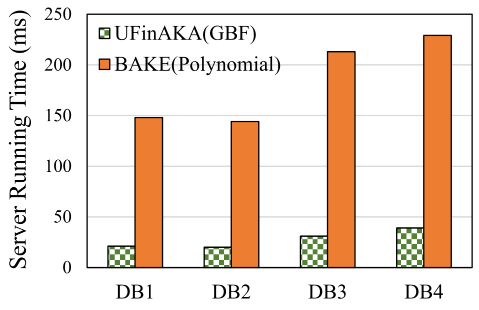


Fig. 8. Comparison of server running time.

cost of UFinAKA, respectively. Therefore, UFinAKA has significant advantage over RFE and fuzzy aPAKE in terms of both computation and server storage.

In addition, since the server needs to interact with multiple clients concurrently, UFinAKA employs the garbled Bloom filter instead of polynomial interpolation in BAKE. The comparison results are shown in Fig 8. The running time of BAKE is about 7 times cost of UFinAKA in four databases. Therefore, UFinAKA shows better performance than BAKE in the Client/Server mode.

VIII. Related Work

Depending on the underlying techniques, we divide AKAs into two categories (Fig. 9): *Key-based* and *Human-based.*

Key-based AKAs are widely utilized in real-world applications, storing a secret key within the terminal and verifying the identity by checking the key. The most common key-based AKA is typically built on a powerful PKI, where a Certificate Authority issues an identity certificate and a secret key for each party. A simple example of PKI-based AKAs is the Transport Layer Security (TLS) handshake [36], which depends on a trusted certificate authority to sign the document associating the public key. Besides, the PKI is well known as resource-consuming, which is not suitable for some limited environments. Another representative key-based AKA is designed based on a shared symmetric key. Avoine et al. proposed SAKE [37], a lightweight AKA protocol solely based on *Symmetric Key (SK)*, utilizing a resynchronization technique, and Boydet al. [38] improved SAKE with synchronization robustness. These solutions are employed in resource-limited cases, *e.g.*, the Industrial Internet of Things [39], but require the two participants to share a secret only accessed by

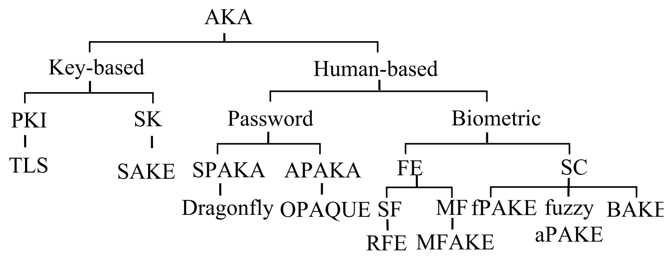


Fig. 9. Categories of AKAs.

themselves in advance, which is not convenient in practice. Furthermore, these key-based AKAs verify the possession of the key rather than the participant, and need an additional cryptographic device to store a high-entropy secret key.

Human-based AKAs generally verify the user identity based on personal authentication factors that are not stored within the terminal. The most common human-based AKA is the password-based AKA [4], [40], [41], [42], [43], [44], [45], which converts a low-entropy secret password into a random-looking session key. The *Symmetric Password-based AKA (SPAKA)* (*e.g.*, SPAKE2 [41], Dragonfly [44]) considers two participants that share the same password to generate a shared session key, but sharing the password means an underlying risk that the server discloses passwords. Thus, the *Asymmetric PAKA (APAKA)* [42], [43], [45] was proposed to resist the server compromise. Jarecki et al. [42] proposed a Universal Composable (UC) strong APAKA, named QPAQUE, to prevent pre-computation attacks. Bradley et al. [43] gave further efficiency improvements and weaker security assumptions compared with QPAQUE. However, the password-based approaches have trouble with weak passwords and password forgetting.

Biometric AKAs [3], [10], [12], [13], [14], [46] provide more user-friendly AKA functionality via utilizing unique biometric features, *e.g.*, the human fingerprint. The representative biometric AKA schemes [10], [12], [14], [46] are generally constructed based on the *Fuzzy Extractor (FE)* [47], which directly generates a secret key from the repeated noisy biometrics. The research on FE-based AKAs is classified into the Single-Factor (SF) solution (*e.g.*, the RFE [10] that utilizes biometrics to provide authentication) and the Multi-Factor (MF) solution (*e.g.*, MFAKA [12] and GAKA [48] that combine biometrics with other factors to provide authentication). FEs have strong transportability, but require the server to store a secret, which may be exploited by an adversary. Recently, the *Secure Computation (SC)* provides more appropriate candidates for biometric AKAs [3], [11], [13], [15]. However, the *fuzzy Password Authenticated Key Exchange (fPAKE)* [11] exposed the template to the server, the fuzzy aPAKE [13] consumed heavy computation overhead and frequent interactions, and the BAKE [3] and FAKE [15] is not designed for the Client/Server model. In contrast, UFinAKA solves all the above issues for the first time by designing an efficient FBCA scheme with computationally lightweight GBF and VSS. The server can self-recover from corruption by updating credentials. In addition, although UFinAKA is designed specifically for the human fingerprint, we reasonably believe that this approach may be feasible for other biometric characteristics (*e.g.*, the iris and face) which could be converted into multiple strings or minutiae points.

IX. Conclusion

To establish secure network communications in online services, we presented a privacy-preserving updatable fingerprint-based blind credentials authentication and key agreement system, UFinAKA, which supports updatable fingerprint-based blind credentials and anonymous authentication. Specially, we proposed an FBCA primitive as the building block to preserve fingerprint privacy for UFinAKA. We also designed a fingerprint-based AKA protocol with updatable blind credentials based on an efficient FBCA construction, involving linear computation complexity and single-round interaction. The security proof showed that UFinAKA is secure in the practical environment. The experimental evaluation results showed that the running time of UFinAKA is less than 1 second and the storage cost is less than 1 MB, which is appropriate in practice. The server's running time and storage cost are small and roughly increases linearly with the number of clients, which is suitable for the Client/Server mode in reality.

References

[1] D. Felsch, M. Grothe, J. Schwenk, A. Czubak, and M. Szymanek, "The dangers of key reuse: Practical attacks on IPsec IKE," in *Proc. 27th USENIX Secur. Symp.* Berkeley, CA, USA: USENIX Association, Aug. 2018, pp. 1–18.

[2] R. Borgaonkar, L. Hirschi, S. Park, and A. Shaik, "New privacy threat on 3G, 4G, and upcoming 5G AKA protocols," *Proc. Privacy Enhancing Technol.*, vol. 2019, no. 3, pp. 108–127, Jul. 2019.

[3] M. Wang, K. He, J. Chen, Z. Li, W. Zhao, and R. Du, "Biometrics-authenticated key exchange for secure messaging," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 2618–2631.

[4] S. M. Bellovin and M. Merritt, "Encrypted key exchange: Password-based protocols secure against dictionary attacks," in *Proc. IEEE Comput. Soc. Symp. Res. Secur. Privacy*, May 1992, pp. 72–84.

[5] D. Pointcheval, "Password-based authenticated key exchange," in *Proc. Int. Workshop Public Key Cryptogr.* Berlin, Germany: Springer, 2012, pp. 390–397, doi: 10.1007/978-3-642-30057-8_23.

[6] M. Abdalla, F. Benhamouda, and P. MacKenzie, "Security of the J-PAKE password-authenticated key exchange protocol," in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 571–587.

[7] M. Abdalla, T. Eisenhofer, E. Kiltz, S. Kunzweiler, and D. Riepel, "Password-authenticated key exchange from group actions," in *Proc. Annu. Int. Cryptol. Conf. (CRYPTO)*. Cham, Switzerland: Springer, 2022, pp. 699–728.

[8] F. Hao and P. C. van Oorschot, "SoK: Password-authenticated key exchange—Theory, practice, standardization and real-world lessons," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, May 2022, pp. 1–15.

[9] D. Pointcheval and S. Zimmer, "Multi-factor authenticated key exchange," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.* Berlin, Germany: Springer, 2008, pp. 277–295.

[10] R. Canetti, B. Fuller, O. Paneth, L. Reyzin, and A. Smith, "Reusable fuzzy extractors for low-entropy distributions," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, Apr. 2016, pp. 117–146.

[11] P. Dupont, J. Hesse, D. Pointcheval, L. Reyzin, and S. Yakoubov, "Fuzzy password-authenticated key exchange," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Heidelberg, Germany: Springer, Mar. 2018, pp. 393–424.

[12] R. Zhang, Y. Xiao, S. Sun, and H. Ma, "Efficient multi-factor authenticated key exchange scheme for mobile communications," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 4, pp. 625–634, Jul. 2019.

[13] A. Erwig, J. Hesse, M. Orlt, and S. Riahi, "Fuzzy asymmetric password-authenticated key exchange," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Cham, Switzerland: Springer, Dec. 2020, pp. 761–784.

[14] R. Canetti, B. Fuller, O. Paneth, L. Reyzin, and A. Smith, "Reusable fuzzy extractors for low-entropy distributions," *J. Cryptol.*, vol. 34, no. 1, p. 2, Jan. 2021.

[15] M. Jiang, S. Liu, S. Han, and D. Gu, "Fuzzy authenticated key exchange with tight security," in *Proc. Eur. Symp. Res. Comput. Secur.* Cham, Switzerland: Springer, Sep. 2022, pp. 337–360.

[16] *General Data Protection Regulation*. Accessed: Sep. 6, 2023. [Online]. Available: https://gdpr-info.eu/

[17] A. K. Jindal, I. Shaik, V. Vasudha, S. R. Chalamala, R. Ma, and S. Lodha, "Secure and privacy preserving method for biometric template protection using fully homomorphic encryption," in *Proc. IEEE 19th Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom)*, Dec. 2020, pp. 1127–1134.

[18] Y. Lai, Z. Jin, K. Wong, and M. Tistarelli, "Efficient known-sample attack for distance-preserving hashing biometric template protection schemes," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 3170–3185, 2021.

[19] V. K. Hahn and S. Marcel, "Biometric template protection for neural-network-based face recognition systems: A survey of methods and evaluation techniques," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 639–666, 2023.

[20] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 3rd ed. Boca Raton, FL, USA: CRC Press, 2021.

[21] A. M. Johnston and P. S. Gemmell, "Authenticated key exchange provably secure against the man-in-the-middle attack," *J. Cryptol.*, vol. 15, no. 2, pp. 139–148, Jan. 2002.

[22] Z. Ma and J. He, "Outsider key compromise impersonation attack on a multi-factor authenticated key exchange protocol," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.* Heidelberg, Germany: Springer, Sep. 2022, pp. 320–327.

[23] A. C.-C. Yao, "How to generate and exchange secrets," in *Proc. 27th Annu. Symp. Found. Comput. Sci.* Washington, DC, USA: IEEE Computer Society, Oct. 1986, pp. 162–167.

[24] Y. Huang, L. Malka, D. Evans, and J. Katz, "Efficient privacy-preserving biometric identification," in *Proc. 17th Conf. Netw. Distrib. Syst. Secur. Symp.*, 2011, pp. 90–98.

[25] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 45th Annu. ACM Symp. Theory Comput.*, May 2009, pp. 169–178.

[26] G. Pradel and C. Mitchell, "Privacy-preserving biometric matching using homomorphic encryption," in *Proc. IEEE 20th Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom)*, Oct. 2021, pp. 494–505.

[27] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[28] C. Dong, L. Chen, and Z. Wen, "When private set intersection meets big data: An efficient and scalable protocol," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Nov. 2013, pp. 789–800.

[29] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[30] J. Hartloff, J. Dobler, S. Tulyakov, A. Rudra, and V. Govindaraju, "Towards fingerprints as strings: Secure indexing for fingerprint matching," in *Proc. Int. Conf. Biometrics (ICB)*, Jun. 2013, pp. 1–6.

[31] K. Zhou and J. Ren, "PassBio: Privacy-preserving user-centric biometric authentication," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 12, pp. 3050–3063, Dec. 2018.

[32] M. Barni et al., "Privacy-preserving fingercode authentication," in *Proc. 12th ACM workshop Multimedia Secur.*, Sep. 2010, pp. 231–240.

[33] M. Bellare, R. Canetti, and H. Krawczyk, "A modular approach to the design and analysis of authentication and key exchange protocols," in *Proc. 30th Annu. ACM Symp. Theory Comput. (STOC)*, May 1998, pp. 419–428.

[34] Y. Lindell, "How to simulate it—A tutorial on the simulation proof technique," in *Tutorials on the Foundations of Cryptography*. Cham, Switzerland: Springer, 2017, pp. 277–346.

[35] *Fvc2004*. Accessed: Sep. 6, 2023. [Online]. Available: http://bias.csr.unibo.it/fvc2004/

[36] D. Sikeridis, P. Kampanakis, and M. Devetsikiotis, "Post-quantum authentication in TLS 1.3: A performance study," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–16.

[37] G. Avoine, S. Canard, and L. Ferreira, "Symmetric-key authenticated key exchange (SAKE) with perfect forward secrecy," in *Proc. Cryptographers' Track RSA Conf.* Heidelberg, Germany: Springer, Feb. 2020, pp. 199–224.

[38] C. Boyd, G. T. Davies, B. de Kock, K. Gellert, T. Jager, and L. Millerjord, "Symmetric key exchange with full forward security and robust synchronization," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Cham, Switzerland: Springer, Dec. 2021, pp. 681–710.

[39] Q. Fan, J. Chen, M. Shojafar, S. Kumari, and D. He, "SAKE: A symmetric authenticated key exchange protocol with perfect forward secrecy for industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 18, no. 9, pp. 6424–6434, Sep. 2022.

[40] R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie, "Universally composable password-based key exchange," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Heidelberg, Germany: Springer, 2005, pp. 404–421.

[41] J. Becerra, D. Ostrev, and M. Škrobot, "Forward secrecy of SPAKE2," in *Proc. Int. Conf. Provable Secur.* Jeju-do, South Korea: Springer, Oct. 2018, pp. 366–384.

[42] S. Jarecki, H. Krawczyk, and J. Xu, "OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Heidelberg, Germany: Springer, 2018, pp. 456–486.

[43] T. Bradley, S. Jarecki, and J. Xu, "Strong asymmetric PAKE based on trapdoor CKEM," in *Proc. CRYPTO*. Heidelberg, Germany: Springer, Aug. 2019, pp. 798–825.

[44] M. Vanhoef and E. Ronen, "Dragonblood: Analyzing the dragonfly handshake of WPA3 and EAP-pwd," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 517–533.

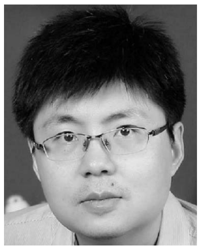
[45] M. Abdalla, M. Barbosa, T. Bradley, S. Jarecki, J. Katz, and J. Xu, "Universally composable relaxed password authenticated key exchange," in *Proc. Annu. Int. Cryptol. Conf.* Cham, Switzerland: Springer, Aug. 2020, pp. 278–307.

[46] X. Boyen, Y. Dodis, J. Katz, R. Ostrovsky, and A. Smith, "Secure remote authentication using biometric data," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Heidelberg, Germany: Springer, 2005, pp. 147–163.

[47] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.* Heidelberg, Germany: Springer, 2004, pp. 523–540.

[48] Z. Li, M. Wang, V. Sharma, and P. Gope, "Sustainable and round-optimized group authenticated key exchange in vehicle communication," *IEEE Trans. Intell. Transp. Syst.*, early access, Dec. 28, 2022, doi: 10.1109/TITS.2022.3169182.

Mei Wang received the M.S. degree in cryptography from Xidian University, Xi'an, China, in 2016, and the Ph.D. degree in cyberspace security from Wuhan University, Wuhan, China, in 2022. She is currently an Assistant Researcher with the School of Cyber Science and Technology, Shandong University. She has published research papers in international journals and conferences, such as the IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, the *Future Generation Computer Systems*, and ACM CCS. Her research interests include applied cryptography, cloud computing, mobile computing, and privacy protection.

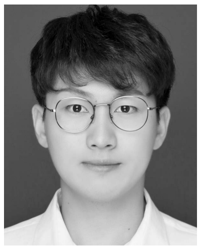


Jing Chen (Senior Member, IEEE) received the Ph.D. degree in computer science from the Huazhong University of Science and Technology, Wuhan. He is currently a Professor with the Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan. He has published over 90 research articles in many international journals and conferences, such as the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, the IEEE TRANSACTIONS ON MOBILE COMPUTING, the USENIX Security, ACM CCS, and INFOCOM. His research interests are in the areas of network security and cloud security.

Kun He (Associate Member, IEEE) received the Ph.D. degree in computer science from Wuhan University, Wuhan. He has published research articles in the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, the IEEE TRANSACTIONS ON MOBILE COMPUTING, the USENIX Security, ACM CCS, and INFOCOM. His research interests include cryptography, network security, mobile computing, and cloud computing.

Ruozhou Yu (Senior Member, IEEE) received the Ph.D. degree in computer science from Arizona State University, Tempe, AZ, USA, in 2019. He is currently an Assistant Professor of computer science with North Carolina State University. His research interests include the Internet of Things, cloud/edge computing, smart networking, algorithms and optimization, security and privacy, and blockchain.

Ruiying Du received the B.S., M.S., and Ph.D. degrees in computer science from Wuhan University, Wuhan, China, in 1987, 1994, and 2008, respectively. She is currently a Professor with the School of Cyber Science and Engineering, Wuhan University. She has published over 80 research articles in many international journals and conferences, such as the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, the USENIX Security, ACM CCS, INFOCOM, SECON, and TrustCom. Her research interests include network security, wireless networks, and mobile computing.



Zhihao Qian is currently pursuing the bachelor's degree with the School of Cyber Science and Engineering, Wuhan University. He is excellent in computer programming. His research interests include network security and system security.