User-autonomous Multi-Factor Authentication Supporting Arbitrary Factor Configurations

Li, Wenting; Cheng, Haibo ; Liang, Kaitai

DOI
10.1109/TIFS.2025.3622084

Publication date
2025

Document Version
Final published version

Published in
IEEE Transactions on Information Forensics and Security

User-Autonomous Multi-Factor Authentication Supporting Arbitrary Factor Configurations

Wenting Li, Haibo Cheng, and Kaitai Liang, *Member, IEEE*

***Abstract*—Multi-factor authentication (MFA) is widely used to secure high-value digital assets in web applications. Traditional t-factor authentication (t-FA) enhances security by requiring users to present t factors, which often becomes inconvenient as the number of required factors increases. Threshold (t, n)-MFA (T-MFA) improves usability by allowing users to authenticate with any t factors from a set of n. However, T-MFA treats all factors as equal, ignoring the varying security strengths of different factors. For instance, passwords are generally less secure than smart cards, yet T-MFA fails to account for these differences. This restricts its ability to balance security and usability effectively. To overcome this, we propose *AS-MFA*, a new primitive allowing users to configure factor combinations based on the security strength of each factor. Our scheme employs secret sharing for general access structures, ensuring that authentication is granted only when a valid combination of factors is presented. Unlike T-MFA limited to threshold configurations, AS-MFA supports arbitrary factor combinations, offering *greater user autonomy*. We formally define the security of AS-MFA and prove the security of our design. In terms of performance, the protocol requires *only two* communication rounds and achieves computational efficiency, involving t_2 fuzzy extractor operations, $2 + 3t_1 + 3t_2$ exponentiations, and 2 multi-exponentiations for a factor combination consisting of t_1 passwords, t_2 biometrics, and t_3 devices. For threshold configurations, AS-MFA outperforms Li et al.'s T-MFA by requiring fewer exponentiation operations, offering a *constant* and lower computation cost compared to the *linear* cost in t of T-MFA.**

***Index Terms*—Access structure, multi-factor authentication, key exchange, password, secret sharing.**

I. Introduction

MULTI-FACTOR authentication (MFA) is widely used in web applications, such as Apple ID, GitHub, and online banking services. Traditional t-factor authentication (t-FA) improves security by requiring users to present t factors simultaneously, making it harder for adversaries to impersonate users. However, as t increases, usability decreases because users are more likely to forget or lose some factors. As a result, in such static MFA, t is typically limited to 2 or 3 in practice, limiting the potential for higher security improvements.

To address this, Li et al. proposed threshold MFA (T-MFA) [1], which improves usability by allowing users to authenticate with any t factors from a set of n. As shown in Fig. 1, if a user forgets or loses one factor (e.g., a smartcard), they can use another factor (e.g., a smartwatch) instead. T-MFA also allows non-portable factors, e.g., a home smart speaker or an office intranet server, to be included, increasing t without affecting usability. Further, T-MFA naturally supports backup factors, allowing users to revoke lost factors while maintaining access.

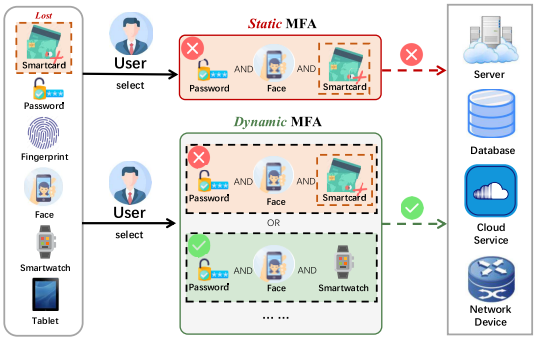


Fig. 1. Difference between static and dynamic MFA, if some factor is lost.

A. Motivation

While T-MFA improves usability, it treats all factors as equally secure, ignoring differences in their strengths. For example, passwords are generally less secure than smart cards, and even passwords vary in strength [2], [3]. Similarly, biometric factors depend on the quality of recognition devices, for instance, structured light on high-end smartphones is more secure than front cameras on lower-end devices [4], [5].

As shown in Fig. 2, T-MFA only supports threshold factor configurations, meaning the overall security depends on the weakest t factors. To improve security, one could raise t or exclude weaker factors, but T-MFA lacks the flexibility to adjust factor combinations based on individual factor strengths or specific user needs. For example, it cannot require additional factors to offset weaker ones like passwords, limiting its ability to balance security and usability.

Received 15 January 2025; revised 22 June 2025 and 3 September 2025; accepted 12 October 2025. Date of publication 15 October 2025; date of current version 30 October 2025. This work was supported in part by the Beijing Institute of Graphic Communication (BIGC) Project under Grant Ea202515, in part by Beijing Science and Technology Plan under Grant Z241100007624008, in part by the Publishing Think Tank Platform Development Project under Grant KYCPT202514, and in part by the National Natural Science Foundation of China under Grant 62202012. The associate editor coordinating the review of this article and approving it for publication was Prof. Muhammad Khurram Khan. *(Corresponding author: Haibo Cheng.)*

Wenting Li is with the School of Information Engineering, Beijing Institute of Graphic Communication, Beijing 102699, China (e-mail: wentingli@pku.edu.cn).

Haibo Cheng is with the National Engineering Research Center for Software Engineering, Peking University, Beijing 100871, China, and also with Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China (e-mail: hbcheng@pku.edu.cn).

Kaitai Liang is with the Faculty of Technology, University of Turku, 20014 Turku, Finland, and also with the Department of Intelligent Systems, Delft University of Technology, 2628 CD Delft, The Netherlands (e-mail: kaitai.liang@utu.fi).

Digital Object Identifier 10.1109/TIFS.2025.3622084

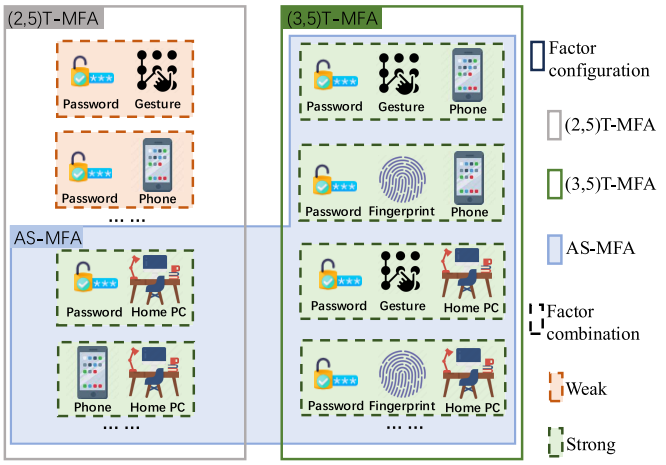


Fig. 2. Difference in supported factor configurations between T-MFA and AS-MFA. AS-MFA supports arbitrarily strong combinations of authentication factors, whereas T-MFA is limited to combinations with a fixed number (e.g., 2 or 3) of factors.

B. As-MFA

To address these limitations, we propose *AS-MFA*, a new MFA primitive that supports arbitrary configurations of factor combinations. Using concepts from secret sharing [6], [7], we formally define AS-MFA. Let $\mathcal{F}$ represent a user's set of authentication factors. A subset $FC \subseteq \mathcal{F}$ is called *qualified* if it can successfully authenticate the user. The family of all such subsets is called the *access structure*, denoted $\mathcal{AS}$. The access structure is monotone if $FC \in \mathcal{AS}$ and $FC \subseteq FC'$ imply $FC' \in \mathcal{AS}$, as possessing more factors cannot reduce authentication capability. AS-MFA supports any monotone $\mathcal{AS}$, enabling maximum flexibility in factor configurations.

C. User Autonomy

AS-MFA provides users with greater autonomy to tailor their factor configurations based on the security strengths of their factors and their individual needs, as illustrated in Fig. 2. Authentication factors vary in security, and so do user requirements; for example, a public figure's social media account demands higher security than that of an average user. Unlike existing MFA, including T-MFA, AS-MFA enables user-tailored configurations, achieving an optimal balance between usability and security.

D. Design Challenges

There are two main design approaches to implementing AS-MFA, each presenting different challenges: some suffer from inherent vulnerabilities, while others struggle to support arbitrary factor configurations.

The first approach verifies each factor individually and then checks whether the verified factor set is qualified. This approach extends industrial MFA systems [8], [9], where each factor—such as a password or one-time password (OTP)—is verified independently. Whether the factor set is qualified can be enforced by simple server-side logic. However, this design inherits the vulnerabilities of industrial MFA schemes. Because factors are verified and stored separately, sensitive information is often inadequately protected: for instance, passwords are hashed, but OTP secrets are typically stored in plaintext. If the server is compromised, attackers can steal this data and compromise user accounts [10], [11], [12]. Moreover, because of factor reuse across services, attackers may compromise additional accounts on other platforms by reusing the same factors. For example, in 2022, Twilio suffered a breach in which attackers gained unauthorized access to internal systems via social engineering, resulting in the exposure of sensitive data and customer account compromise [13], [14]. In summary, this approach offers ease of deployment, but has inherent vulnerabilities in the event of server compromise.

The second approach leverages cryptographic techniques to mitigate the risks of server compromise, as seen in many academic MFA schemes [1], [15], [16]. These schemes are typically based on authenticated key exchange (AKE) protocols that derive a session key through the joint contribution of authentication factors, rather than verifying each one individually. While this approach significantly improves security, current designs are rigid and tailored to specific factor combinations. For example, OpTFA [17] only supports one password and one device, while Han et al.'s protocol [15] requires one password and one biometric. These designs do not generalize easily to other factor sets. Li et al.'s T-MFA scheme [1], which is the closest to AS-MFA, supports a threshold-based factor configuration but is restricted to one password and $t-1$ devices. This limitation stems from the use of threshold oblivious pseudorandom functions (TOPRF) for password protection, which are not easily extendable to support multiple passwords or other types of factors. In summary, this approach addresses server-side vulnerabilities, but faces major obstacles in supporting arbitrary factor configurations.

E. Our Design

To implement AS-MFA and support arbitrary factor combinations, we propose a novel scheme that integrates authentication and key exchange (KE), forming the AS-MFAKE protocol. The core design leverages secret sharing schemes for general access structures [6], [7]. To the best of our knowledge, this is the first work to incorporate such secret sharing techniques into the design of MFA protocols. The authentication key is shared based on the access structure, allowing only qualified factor combinations to reconstruct it.

Given the varying properties of authentication factors, our scheme employs tailored methods for share assignment. Device factors store shares directly on the devices. For low-entropy factors, shares are generated using an oblivious pseudorandom function (OPRF) [18]. Biometric factors use fuzzy extractors [19], [20] to generate shares. This approach ensures compatibility with general access structures and supports a wide range of practical authentication factors.

F. Security and Efficiency Analysis

We formally model the security of AS-MFAKE using a game-based definition to capture the inevitable attacks and model the highest attainable security under given factor configurations. Specifically, such security of AS-MFAKE depends on the guessability of factors in the weakest qualified combination. We prove the security of our design within this model.

To evaluate its practicality, we implement our protocol in a real-world environment and demonstrate its efficiency in both communication and computation. In terms of communication, the protocol requires only 2 rounds, matching the efficiency of Li et al.'s T-MFA. For computation, given a factor combination FC containing t_1 passwords, t_2 biometric factors, and t_s devices, the protocol requires t_2 fuzzy extractor operations, $2+3t_1+3t_2$ exponentiations, and 2 multi-exponentiations.

For threshold configurations, such as one password and $t-1$ devices (the limit of Li et al.'s T-MFA), AS-MFAKE achieves lower and *constant* computation costs, requiring only 5 exponentiations and 2 multi-exponentiations, compared to T-MFA's *linear* cost of $t+4$ exponentiations and 2 multi-exponentiations. Even for simple configurations like one password and one device, our protocol demonstrates significant efficiency, requiring only 16.66 ms, *61.46%* faster than Li et al.'s T-MFA, which takes 26.86 ms.

G. Compatibility of Our Design

Our design is compatible with existing public-key AKE protocols, such as TLS, and requires no modifications to already deployed infrastructures. On the server side, only the OPRF component needs to be additionally implemented, or alternatively the three-party OPRF can be employed, while the rest of the authentication process remains unchanged. The secret sharing operations are performed solely on the user side and are fully transparent to the server, allowing flexible use of any secret sharing implementation across various devices. In summary, our client-side design is transparent to the server, while the server-side design preserves compatibility with existing AKE protocols such as TLS, thereby ensuring strong deployability of our scheme.

H. Summary

Our main contributions are as follows:

1) We introduce a novel MFA primitive, AS-MFA, allowing users to autonomously configure arbitrary factor combinations for authentication.
2) We design an AS-MFA scheme using secret sharing for general access structures, enabling support for arbitrary factor configurations.
3) We define a game-based security model for AS-MFA and formally prove the security of our scheme within the model.
4) We implement our protocol in a real-world environment, demonstrating its efficiency in communication and computation.

II. Related Works

We provide a brief overview of several typical MFA protocols, encompassing both industrial and academic approaches.

A. MFA Schemes in Industry

In recent years, MFA has been widely adopted by various industries, including technical services (e.g., GitHub [24]), university online platforms (e.g., Carnegie Mellon University [25]), and financial services (e.g., Bank of America [26]). Prominent MFA implementations include Google Authenticator [9], FIDO U2F [8], and Duo 2FA [21].

However, these schemes typically authenticate factors independently. While this design simplifies implementation, it introduces significant security risks if the authentication server is compromised. In these schemes, the client establishes a secure channel (e.g., via TLS) and transmits factors or related information to the server. The server then verifies each factor separately based on its stored data. This architecture makes server-side vulnerabilities critical, as they may expose *all* authentication factors. Below are some common risks:

1) *Malicious servers.* Attackers may set up malicious servers (e.g., pirated websites) to lure users into registering, subsequently stealing their authentication factors.
2) *Insider threats.* Authorized insiders with access permissions can directly extract stored authentication factors.
3) *Software bugs.* For instance, GitHub [27] and Google [28] have inadvertently logged plaintext passwords due to software bugs, potentially exposing them through logs.
4) *Server compromise.* If the server is breached, the attacker can recover authentication factors stored separately. For example, password hashes stored for verification can be cracked using efficient password guessing algorithms [29], [30], [31], [32], revealing plaintext passwords.
5) *PKI failures.* Issues such as software bugs, poor user practices, rogue certificate authorities (CAs), or server mismanagement can lead to PKI failures. For instance, software may fail to verify certificates properly, users may accept malicious certificates, rogue CAs may issue fake certificates, or servers may share private keys with CDN providers. Such failures compromise the secure TLS channel, exposing transmitted authentication factors.

More specifically, Google Authenticator [9] employs the time-based one-time password (TOTP) algorithm [33] for second-factor authentication. In the above cases, the secret stored on the server is leaked, and then an attacker can directly compromise user accounts. FIDO U2F [8] utilizes public-key cryptography for second-factor authentication. This design mitigates the risk of second-factor compromise but does not offer additional protection for passwords. When traditional password authentication over TLS is used, passwords remain susceptible to leakage in the aforementioned scenarios.

B. Academic Studies on MFA

MFA has garnered significant attention in academia, with numerous studies focusing on both the security analysis and the design of MFA protocols [15], [16], [17], [22]. Below, we provide a brief overview of these works.

1) Security Analysis Methods: Many studies employ heuristic attacks, BAN logic, or automated tools [34], [35], [36] for security analysis. However, these methods often fail to provide cryptographically sound security. The widely accepted approach is to use game-based models, such as the BPR or ROR models [37], [38], to establish provable security guarantees [39], [40].

For 2FA, Shirvanian et al. [22] introduced security requirements based on various compromise combinations and provided specific security bounds for each scenario. However,

their model did not address server corruption. Later, Jarecki et al. [17] extended the model by considering additional compromise cases, capturing the highest achievable security for 2FA. Building on this, Li et al. [1] further extended the model to define the highest attainable security for T-MFA.

2) Academic MFA Schemes: We review both typical and recent schemes.

Using their security model, Shirvanian et al. [22] proposed a 2FA scheme combining passwords and devices, inspired by industrial 2FA designs. The scheme enhances security against server compromise. However, it has similar issues as industrial schemes, i.e., failing to prevent the server from accessing plaintext passwords. To address these limitations, Jarecki et al. [17] proposed an improved 2FA scheme, *OpTFA*, which mitigates the vulnerabilities. However, OpTFA suffers from significant complexity, requiring 12 exponentiations, 2 multi-exponentiations, and 10 communication rounds. Further, Li et al. [1] proposed T-MFA and a specific scheme, provide high usability, security and efficiency. As discussed before, their scheme only supports threshold factor configurations, and fails to achieve the optimal balance between security and usability.

Recently, Han et al. [15] proposed a 2FA scheme supporting the combination of one password and one biometric factor. However, their protocol requires 6 communication rounds and, more critically, is vulnerable to offline guessing attacks even without server compromise. Specifically, an attacker can impersonate the server to interact with the client by using trivial values for random variables (e.g., $y = 0$ and $k_1 = 1$), obtaining the verified value $\sigma_1 = H(ID_U, ID_S, X, 1, X, H_0(pw, R))$. By monitoring P from an honest session between the client and server—where P is generated from the biometric factor—the attacker can guess both the password and biometric factor. Using a guessed biometric factor bio' to derive R' with P, the attacker can compute σ'_1 using a guessed password pw' and verify if $\sigma'_1 = \sigma_1$. Jiang et al. [16] proposed another 2FA scheme supporting the same combination of factors as Han et al. However, their protocol fails to provide server authentication. Similar to the attack on Han et al.'s scheme, an attacker can impersonate the server, use trivial values for random variables (e.g., $a_{b,j} = 1$, $k' = 0$, and P as the maximum), and successfully deceive the client during authentication. This results in the attacker obtaining the same session key $PRG(1)$ as the client.

III. Preliminaries

We review the main building blocks used in our design of AS-MFA, including authenticated key exchange, secret sharing, oblivious pseudorandom function and fuzzy extractor.

A. Authenticated Key Exchange

Authenticated key exchange (AKE) enables two parties to authenticate each other and establish a shared session key for secure communication. Each party possesses a long-term cryptographic key to execute the protocol. Over the past decades, extensive research on AKE has led to the development of numerous protocols, such as [41], [42], and [43]. Some AKE protocols rely on shared cryptographic keys between the two parties. However, these protocols are vulnerable to key-compromise impersonation (KCI) attacks. Specifically, if one party, denoted as P_1, is compromised, an attacker can use the shared key to impersonate the other party, P_2, and interact with P_1. To mitigate KCI attacks, protocols leveraging public-key cryptography have been proposed. In these protocols, each party holds a private key and the other party's public key. Thus, even if P_1 is compromised, the attacker cannot impersonate P_2 to interact with P_1, as the attacker lacks P_2's private key.

Our AS-MFA requires strong security against KCI attacks. To achieve this, we adopt HMQV [44], one of the most efficient AKE protocols with built-in KCI resistance. As illustrated in Figs. 3 and 4, HMQV is highly efficient:

- *Communication cost:* HMQV requires only two rounds.
- *Computational cost:* Each party performs one exponentiation and one multi-exponentiation during authentication.

HMQV provides implicit authentication, meaning the two parties do not explicitly authenticate each other until the session key is used with an authentication mechanism, such as authenticated encryption. Explicit authentication can be achieved by adding one additional round, as described by Bellare et al. [37]. HMQV has been formally proven AKE-secure and KCI-secure in the random oracle model under the Computational Diffie-Hellman (CDH) assumption [44].

Parameters

- G is a cyclic group generated by g with a prime order m.
- H and H' are cryptographic hash functions, where $H : G \to \mathbb{Z}_m$ and $H' : G \times G \to \{0,1\}^l$. The parameter l represents the length of the session key.

Initialization

- Each participant A generates their private key $k_A \in \mathbb{Z}_m$ and the corresponding public key $K_A \leftarrow g^{k_A}$. It is assumed that each participant securely receives the public key of the other party (in our AS-MFA protocol, public keys are exchanged over a secure channel during the registration phase).

Authentication

- Participant A selects a random value $x \in \mathbb{Z}_m$, computes $X \leftarrow g^x$, and sends (A, X) to participant B.
- Upon receiving (A, X), participant B selects a random value $y \in \mathbb{Z}_m$, computes $Y \leftarrow g^y$, and sends Y back to A. Then, B computes the session key as
$$SK \leftarrow SK(k_B, K_A, y, X),$$
where
$$SK(k_B, K_A, y, X) = H((XK_A^{H'(X, g^{k_B})})^{y + H'(g^y, K_A)k_B}).$$
- Receiving Y, participant A computes the session key as
$$SK \leftarrow SK(k_A, K_B, x, Y).$$

Fig. 3. 2-round HMQV with implicit authentication. Adding one round [37] enables explicit authentication by including the messages $H''(SK, 1)$ and $H''(SK, 2)$ in the final two rounds, where H'' is a hash function with range $\{0,1\}^l$.

B. Secret Sharing

Secret sharing, introduced by Shamir [45] and Blakley [46], distributes a secret among parties, allowing only specific subsets to reconstruct it.

Participant A		**Participant** B
Store: k_A, K_B		Store: k_B, K_A
Pick $x \leftarrow\$ \mathbb{Z}_m, X \leftarrow g^x$	$\xrightarrow{X}$	Pick $y \leftarrow\$ \mathbb{Z}_m, Y \leftarrow g^y$
$SK \leftarrow SK(x, k_A, Y, K_B)$	$\xleftarrow{Y}$	$SK \leftarrow SK(k_B, K_A, y, X)$

Fig. 4. Authentication phase of HMQV: Each participant holds their own private key and the other's public key; after authentication, both derive a shared session key.

The simplest setting involves n parties $\{P_i\}_{i=1}^n$ sharing a secret K, where all n parties are required to reconstruct K, and any $n-1$ parties cannot recover it. A more common setting is the (t,n) threshold scheme, where K is shared among n parties, and any subset of t or more parties can reconstruct K. The most general setting supports arbitrary qualified subsets of parties, represented by an access structure $\mathcal{AS}$, where parties may have differing reconstruction powers.

Formally, given a party set $\mathcal{P} = \{P_i\}_{i=1}^n$, the access structure $\mathcal{AS}$ is a subset of $2^{\mathcal{P}}$ such that a subset $\mathcal{P}' \subseteq \mathcal{P}$ can reconstruct the secret if and only if $\mathcal{P}' \in \mathcal{AS}$. For the simplest setting, $\mathcal{AS} = \{\mathcal{P}\}$. In the (t,n) threshold setting, $\mathcal{AS} = \{\mathcal{P}' \subseteq \mathcal{P} \mid |\mathcal{P}'| \geq t\}$. In the most general case, $\mathcal{AS}$ is an arbitrary monotone subset of $2^{\mathcal{P}}$, where $\mathcal{AS}$ is monotone if $\mathcal{P}_1 \in \mathcal{AS}$ and $\mathcal{P}_1 \subseteq \mathcal{P}_2$ imply $\mathcal{P}_2 \in \mathcal{AS}$.

1) Simplest Setting: For this setting, the secret K can be split into n shares $\{S_i\}_{i=1}^n$, where S_i $(1 \leq i \leq n-1)$ is randomly generated, and $S_n = K \oplus S_1 \oplus \ldots \oplus S_{n-1}$. Reconstruction requires all n shares: $K = S_1 \oplus S_2 \oplus \ldots \oplus S_n$. Fewer than n shares reveal no information about K.

2) ***(t, n)*** *Threshold Setting:* In this setting, Shamir's scheme [45] is widely used. It employs a polynomial of degree $t-1$:

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1},$$

where $a_0 = K$ and $a_1, a_2, \ldots, a_{t-1}$ are random coefficients. Each share S_i is defined as $f(i)$. Reconstruction from any t shares $\{S_i\}_{i \in I}$ uses Lagrange interpolation:

$$K = \sum_{i \in I} S_i \lambda_i, \quad \lambda_i = \prod_{j \in I, j \neq i} \frac{-j}{i-j}.$$

With fewer than t shares, the polynomial is underdetermined, and K remains hidden.

3) General Access Structure: For arbitrary access structures $\mathcal{AS}$, Ito et al. [6] extended Shamir's scheme by assigning multiple shares to certain parties. The access structure $\mathcal{AS}$ is represented as a Boolean function f_b over $2^{\mathcal{P}}$, where $f_b(A) = 1$ if and only if $A \in \mathcal{AS}$. This can be expressed in conjunctive normal form (CNF):

$$f_b(A) = \bigwedge_{t=1}^{s} \bigvee_{i \in I_t} P_i(A) = \bigwedge_{t=1}^{s} \bigvee_{i \in I_t} 1_{P_i \in A}.$$

Here, $1_{P_i \in A}$ is a Boolean indicator for P_i being in A. A polynomial f is then constructed, and shares $f(t)$ are distributed to parties $P_i \in I_t$. This ensures that only subsets $A \in \mathcal{AS}$ can reconstruct the secret. Several works [7], [47], [48], [49] have proposed optimizations to improve the efficiency of secret sharing schemes for general access structures.

Parameters

- G is a cyclic group generated by g with a prime order m.
- H, H' are two hash functions, where $H : \{0,1\}^* \to \{0,1\}^l$ and $H' : \{0,1\}^* \to G$. The pseudorandom function (PRF) $F_k(x)$ is defined as $F_k(x) = H(x, H'(x)^k)$.

Initialization

- The server S selects a secret key $k \leftarrow\$ \{0,1\}^l$ at random.

Computation

- Upon receiving input x, the user U randomly picks $t \leftarrow\$ \mathbb{Z}_m$, computes $\alpha \leftarrow H'(x)^t$, and sends α to S.
- Upon receiving α, the server S computes $\beta \leftarrow \alpha^k$ and sends β back to U.
- Upon receiving β, the user U computes $H(x, \beta^{1/t})$ and outputs the result.

Fig. 5. 2HashDH.

User U		**Server** S
Input: x		Store: k
Pick $t \leftarrow\$ \mathbb{Z}_m$		
$\alpha \leftarrow H'(x)^t$	$\xrightarrow{\alpha}$	
	$\xleftarrow{\beta}$	$\beta \leftarrow \alpha^k$
Output $H(x, \beta^{1/t})$		

Fig. 6. Computation phase of 2HashDH: User U obtains the OPRF output $F_k(x) = H(x, H'(x)^k)$ on input x, while the server S, holding the key k, learns nothing about x.

C. Oblivious Pseudorandom Function

An Oblivious Pseudorandom Function (OPRF) consists of a pseudorandom function (PRF) F and a client-server protocol. In this protocol, the server holds a secret key k, and the client inputs x to compute $F_k(x)$ without learning k. Simultaneously, the server learns nothing about the client's input x. OPRFs are particularly useful for enhancing low-entropy passwords pw by transforming them into cryptographic keys $F_k(pw)$.

In our protocol, we utilize the OPRF scheme *2HashDH* [18], whose structure is shown in Fig. 5. The design of 2HashDH combines two hash functions with a Diffie-Hellman-based framework. As shown in Fig. 6, the client U uses a random value t to mask the input x from the server S, while S performs an exponentiation operation without exposing the secret key k.

The 2HashDH scheme is proven secure in the random oracle model under the One-More Diffie-Hellman (OMDH) assumption, ensuring both correctness and privacy.

D. Fuzzy Extractor

Fuzzy extractors [19] enable the generation of cryptographic keys from biometric data, addressing the variability and noise inherent in such inputs. Biometric traits, such as fingerprints and facial features, are widely used in modern authentication systems due to their uniqueness and resistance to replication. However, biometric data is inherently noisy, making it unsuitable for direct cryptographic key generation, as repeated readings often vary slightly. Unlike passwords or tokens,

biometric inputs require mechanisms to handle this variability while ensuring secure and consistent key generation.

A fuzzy extractor generates a cryptographic key r and a helper string h during an enrollment phase:

$$(r, h) \leftarrow \text{Gen}(w),$$

where w is the noisy biometric input. The helper string h is stored for later use. During authentication, when a slightly different biometric input w' is provided, the fuzzy extractor reconstructs the key r using h, provided the noise between w and w' is within a predefined threshold t:

$$\text{Rep}(w', h) = r \quad \text{if} \quad \text{dis}(w', w) \leq t.$$

Crucially, the helper string h reveals no information about the biometric input w or the cryptographic key r, ensuring both security and privacy.

Since their introduction [19], fuzzy extractors have been the subject of extensive research aimed at improving reusability [50], computational efficiency [51], and quantum resistance [52]. The most recent reusable fuzzy extractor scheme [20] has a Python implementation available at https://github.com/carter-yagemann/python-fuzzy-extractor.

IV. Security Model for AS-MFAKE

To formally define the security of AS-MFAKE with support for general access structures, we extend Li et al.'s game-based model for T-MFA [1]. This model draws inspiration from the CK-adversary model for AKE [53], [54] and Jarecki et al.'s security model for 2FA [17].

A. Protocol Participants and Authentication Factors

The participants in AS-MFAKE include a user U, a client C, a set of devices $\{D_i\}_{i\in I}$ (where I is the index set), and a server S. Like T-MFA, AS-MFA allows U to use multiple devices $\{D_i\}_{i\in I}$ as authentication factors.

However, AS-MFA offers greater flexibility compared to T-MFA. It permits U to use zero, one, or multiple passwords as factors, rather than restricting them to a single password. For example, U could use a 6-digit PIN on a smartphone and an 8-character password on a desktop. Additionally, AS-MFA supports the use of multiple biometric factors, rather than being limited to just one.

To describe U's configuration of factor combinations, we adopt the concept of an access structure from secret sharing. The access structure $\mathcal{AS}_U$ specifies the qualified subsets of factors. Specifically, U with a combination (or set) FC of factors can pass authentication if and only if $FC \in \mathcal{AS}_U$.

B. Flexible Client Usage

AS-MFAKE permits users to authenticate using any client, providing flexibility for logging in from public or new devices. This approach enhances usability by avoiding reliance on a fixed client.

C. Registration

During the registration phase of an AS-MFAKE protocol Π, U selects n authentication factors and autonomously defines an access structure $\mathcal{AS}_U$ based on their security requirements and the strength of the factors, which is then registered with the server S.

Both S and U generate and store long-lived secrets, with storage methods varying by factor type:

- For device-based factors, secrets are randomly generated and stored on the device.
- For biometric factors, secrets are derived from biometric features and do not require storage.
- For passwords, secrets are stored in the user's memory.

The client C cannot store long-lived secrets, as users may log in from different clients.

As in standard authentication studies, the registration process is assumed to be secure. For high-value accounts, such as bank accounts, registration may involve in-person verification at a bank. In most real-world applications, registration is conducted over an S-authenticated TLS channel.

D. Authentication

In the authentication phase of Π, the user U uses a set FC of factors and runs Π on a client C with the server S over a public channel.

Authentication can be categorized into two types:

1) *Explicit authentication:* Both U and S are aware of whether authentication succeeds or fails. Upon success, S and C mutually accept each other's identity, and a session key SK is securely established between them. This key, unknown to the devices or attackers, is typically used to create a secure communication between C and S.
2) *Implicit authentication:* Neither U nor S immediately knows if authentication succeeded. Even after a failure, both S and C may compute session keys, but these will differ. The validity of the session key is verified in subsequent communications.

E. Communication Between Client and Authentication Factors

Unlike the remote communication between C and S, communication between C and authentication factors occurs over a local channel, where the client and factors are in physical proximity. For example:

1) When a smartphone is used as a factor, the local channel can be established via Bluetooth or other methods, such as QR codes combined with Wi-Fi [22].
2) For biometric factors, biometric data is obtained locally from the user.

F. Parallel Instances and Partnering

Our model supports parallel protocol instances, capturing scenarios requiring concurrent execution for security analysis. For example, this includes attacks like Lowe's parallel session attack [55] on the Needham-Schroeder AKE protocol [56]. We denote the i-th instance of a party P as P^i.

To define how instances pair, we use session IDs (SIDs). Upon successful acceptance, each instance of C or S generates a SID, a partner ID (PID), and a session key (SK).

Definition 1 (Partnering) Instances C^i and S^j are considered partners if they accept with SIDs sid and sid' and PIDs pid and pid', satisfying:

1) $sid = sid'$, indicating that their interaction histories match and no active attacks occurred.
2) C^i outputs $pid = S$ and S^j outputs $pid' = C$, confirming mutual acceptance.

G. Attacker Ability

As in the Dolev–Yao model, our framework allows *the attacker to overhear, intercept, and synthesize any message on a public channel*. Thus, during the authentication phase, the attacker has full control over the communication between the client and the server.

Our model also accounts for security under compromised participants or authentication factors. To this end, it permits *the attacker to corrupt any participants and authentication factors* (except the user) in the protocol, gaining full control over them. We consider two types of corruption:

1) *Strong corruption*: The attacker obtains both long-lived secrets and internal states (e.g., random numbers not yet erased). This is denoted as "**Corrupt**".
2) *Weak corruption*: The attacker only acquires long-lived secrets without access to internal states. This is denoted as "**Compromise**".

For device-based authentication factors, attackers can both corrupt and compromise them as participants. However, for non-device factors such as passwords and biometrics, attackers can only compromise these factors by stealing their long-lived secrets. These factors cannot be corrupted, as they lack internal states. This distinction models scenarios where such factors are leaked, e.g., when an attacker gains access to biometric data like facial features.

For simplicity, we refer to factors that are neither compromised nor corrupted as *honest*.

H. Security Game

To model the interaction between an attacker and real-world participants, we define a security game between the attacker and the challenger. The challenger simulates the protocol Π, while the attacker issues queries to interact with the challenger, similar to real-world protocol execution. The challenger responds with relevant information, such as long-lived secrets for **Compromise** queries.

The attacker's capabilities are formally defined through these queries and the challenger's responses. The attacker aims to "win" the game by breaking the protocol. *A protocol is considered secure if no attacker can win the game with an advantage (or probability) exceeding a specified bound.*

For an AKE protocol Π, including AS-MFAKE, the attacker's objective is to compromise the session keys. To quantify the information gained, the security game challenges the attacker to distinguish between the real session key and a randomly generated key of the same length.

If the attacker cannot distinguish between the two, she has no knowledge of the real session key. However, if she can differentiate them, she has gained information about the session key, thus breaking the protocol Π.

We assume registration is completed before the game and disallow further registrations, simplifying analysis without affecting the definition.

During the game, the attacker can issue the following queries, and the challenger responds accordingly:

1) **Send**(P, i, Q, M): Simulates protocol Π by running instance P^i of party P. Instance P^i processes message M from Q and returns its response to the attacker. If $M =$ **Init** and $Q = \perp$, P^i is initialized, and its first message(s) are returned.
   This query models legitimate protocol execution and message exchanges. The special message **Init** allows the creation of new instances and sessions for P. If the attacker forges a message instead of relaying it honestly, we refer to the query as *rogue*, modeling an active attack.
2) **Reveal**(P, i): If P^i has accepted, return its session key; otherwise, return $\perp$.
   This query models forward security by ensuring that leaking an old session key does not affect the security of new ones.
3) **ESReveal**(P, i): Returns the ephemeral secret associated with instance P^i.
   This query tests the protocol's resilience to session key compromise when ephemeral secrets are leaked.
4) **Corrupt**(P): Grants the attacker full control over party P, including its long-lived secrets and internal states.
5) **Compromise**(P, U) (for $P = S$, or some factor): Reveals the long-lived secrets of P related to user U, or directly exposes U's authentication factors.
6) **Test**(P, i): If P^i has accepted, flip a coin b:
   a) If $b = 1$, return the real session key of P^i.
   b) If $b = 0$, return a random value of the same length as the session key.
   This query challenges the attacker to distinguish between the real session key and a random one. *The* **Test**$(\cdot, \cdot)$ *query can only be issued once.*

Using these queries, the attacker $\mathcal{A}$ must guess whether the session key in the **Test** query is real or random. Formally, $\mathcal{A}$ makes a guess b' for the coin b, where $b' = 1$ indicates the real key and $b' = 0$ indicates the random key. The attacker wins the game if and only if $b' = b$.

I. Attacker's Advantage

The advantage of $\mathcal{A}$ is defined as:

$$\mathsf{Adv}^{\text{as-mfake}}_{\Pi,\mathcal{AS}}(\mathcal{A}) = |2\Pr[b' = b] - 1|,$$

with the maximum advantage given by:

$$\mathsf{Adv}^{\text{as-mfake}}_{\Pi,\mathcal{AS}} = \max_{\mathcal{A}} \mathsf{Adv}^{\text{as-mfake}}_{\Pi,\mathcal{AS}}(\mathcal{A}).$$

Here, $|2\Pr[b' = b] - 1|$ ensures the advantage lies in $[0, 1]$, as a trivial attacker would have a winning probability of $\frac{1}{2}$. An advantage of 0 implies the attacker cannot distinguish the real session key from a random one, whereas an advantage of 1 indicates the attacker can always identify the real session key.

J. Fresh Session Keys

In the game, the attacker can trivially obtain session keys for some instances by issuing **Reveal** or **Corrupt** queries. Protecting these keys is impossible for any protocol. Therefore, the attacker is restricted to performing the **Test** query only

on *fresh* instances, which are expected to remain secure. Freshness is defined as follows:

Definition 2 (Freshness) An instance P^i is *fresh* if neither **Reveal**(P, i) nor **Reveal**(Q, j) has been invoked, where Q^j is the partner instance of P^i (if such a partner exists), and one of the following conditions holds:

1) None of **Compromise**(S, U), **Corrupt**(C), or **Corrupt**(S) has been invoked, and at least one of the user's authentication factors remains intact (i.e., neither compromised nor corrupted).
2) The internal states of P^i and its partner Q^j (if any) have not been leaked. Specifically, neither **ESReveal**(P, i) nor **ESReveal**(Q, j) has been invoked.

K. Security Definitions for as-MFAKE

The security of AS-MFAKE relies on the user-defined access structure $\mathcal{AS}$. For example, if $\{pw\} \in \mathcal{AS}$, allowing authentication with only a password, the security would be weak, as the attacker could carry out guessing attacks on the password. To address this, we first examine the basic case with only device-based factors, provide a formal definition for this scenario, and then extend the definition to include passwords and biometric factors.

Definition 3: [AS-MFAKE with Device-Based Factors]

An *AS-MFAKE* protocol Π is secure if, for any probabilistic polynomial-time (PPT) attacker $\mathcal{A}$, the following holds:

1) If for every $FC \in \mathcal{AS}$, $\mathcal{A}$ does not compromise or corrupt all factors in FC, then

$$\mathsf{Adv}^{\text{as-mfake}}_{\Pi,\mathcal{AS}}(\mathcal{A}) \leq \mathsf{negl}(\kappa),$$

where κ is the security parameter, and $\mathsf{negl}(\kappa)$ denotes a negligible value in κ.

For device-based factors, the bound is straightforward: if all factors in FC are compromised, the attacker can impersonate the user; otherwise, the attacker cannot achieve a non-negligible advantage.

L. Extension to Guessable Factors

When passwords and biometric factors are included, security analysis becomes more complex, as these factors suffer from online and offline guessing. We classify factors as:

1) *Guessable Factors*: Including passwords and biometrics, susceptible to guessing.
2) *Unguessable Factors*: Including devices storing cryptographic keys, resistant to guessing attacks.

Definition 4: [AS-MFAKE with Guessable Factors]

An *AS-MFAKE* protocol Π is secure if, for any PPT attacker $\mathcal{A}$ and security parameter κ, the advantage of $\mathcal{A}$ is bounded as:

$$\mathsf{Adv}^{\text{as-mfake}}_{\Pi,\mathcal{AS}}(\mathcal{A}) \leq \max_{FC \in \mathcal{AS}} \mathsf{Adv}^{\text{as-mfake}}_{\Pi,FC}(\mathcal{A}) + \mathsf{negl}(\kappa). \quad (1)$$

For each FC, let FC' denote the subset of FC consisting of honest factors, and $f_{FC'}(i)$ denote the maximum cracking rate with i guesses for the combination FC'. The advantage is bounded as:

1) If FC' contains at least one unguessable factor:

$$\mathsf{Adv}^{\text{as-mfake}}_{\Pi,FC}(\mathcal{A}) \leq \mathsf{negl}(\kappa). \quad (2)$$

2) If all factors in FC' are guessable:
   a) If S is uncompromised:

$$\mathsf{Adv}^{\text{as-mfake}}_{\Pi,FC}(\mathcal{A}) \leq f_{FC'}(q) + \mathsf{negl}(\kappa), \quad (3)$$

   where q is the number of rogue **Send** queries.
   b) If S is compromised:

$$\mathsf{Adv}^{\text{as-mfake}}_{\Pi,FC}(\mathcal{A}) \leq f_{FC'}(q') + \mathsf{negl}(\kappa), \quad (4)$$

   where q' is the number of offline operations on S's long-lived secrets.

M. Inevitable Attacks and Explanation of Bounds

Eq. (1) shows that the security of AS-MFAKE is determined by the weakest factor combination in $\mathcal{AS}$ under inevitable guessing attacks. For each factor combination FC':

1) If FC' includes at least one unguessable factor, the attacker gains only negligible advantage, as shown in Eq. (2).
2) If all factors in FC' are guessable:
   a) If the server S is uncompromised, online guessing attacks are inevitable. In this case, the attacker can interact with the server using candidate guesses for the target factors. A successful guess yields the session key, effectively winning the game. Such attacks require at least one **Send** query per attempt, restricting the attacker to q online queries, as shown in Eq. (3).
   b) If the server S is compromised, offline guessing attacks become inevitable. The attacker can test guess candidates for the target factors against the compromised server-side data. These attacks exploit the long-lived secrets obtained from S, allowing up to q' offline guesses per simulation, as shown in Eq. (4).

For each (online or offline simulated) session, the attacker can guess only one combination FC' with a single candidate FC'_i. Upon authentication failure, the attacker knows that at least one factor in FC'_i is incorrect but cannot identify which one. The optimal strategy is to guess factor combinations in descending order of probability $\Pr(FC'_i)$. For independent factors, $\Pr(FC'_i) = \prod_{P \in FC'_i} \Pr(P)$. For dependent factors, such as passwords and biometric factors related to the user, the probability is determined by their joint distribution.

N. Insecurity of Approaches With Individual Factor Authentication

Given the formal definition of AS-MFAKE, the insecurity of approaches that authenticate each factor independently becomes evident. If the server S is compromised, the attacker can perform offline guessing for all guessable factors individually.

In contrast, AS-MFAKE offers stronger guarantees:

1) If every factor combination FC contains at least one honest and unguessable factor, the attacker cannot conduct offline guessing attacks at all.
2) If all factors in each FC are neither guessable nor compromised, then even with access to server-side data, the attacker can only verify the correctness of an entire factor combination offline, not the correctness of individual factors separately.

V. Our AS-MFAKE Protocol

We first present a basic AS-MFAKE protocol using device-based factors, leveraging secret sharing for general access structures. We then extend the protocol to include passwords and biometric factors.

A. Requirement for Factor Configurations

As discussed in Section IV, the overall security of AS-MFAKE is determined by the weakest factor combination within the user-defined factor configuration. Therefore, to ensure the desired level of protection, users must be guided to construct robust configurations that resist inevitable guessing attacks.

We formalize two acceptable standards for factor combinations:

- **Ideal Configuration:** Each factor combination must include at least one *unguessable* factor—such as a smartphone authenticator or a hardware cryptographic token—which provides inherent resistance against guessing attacks.
- **Acceptable Configuration:** Each factor combination must possess sufficient entropy to meet security requirements (e.g., resisting targeted guessing attacks for at least one year). This is suitable when unguessable factors are unavailable.

Since authentication factors vary in type and security characteristics, we abstract away from specific setups and instead propose a high-level evaluation framework based on entropy and guessability.

To support this framework, we rely on prior work quantifying the guessability of common factors:

- *Passwords* have been widely analyzed. Transformer-based models [57] accurately estimate password distributions and suggest a minimum entropy of approximately 7 bits [58], [59].
- *Biometrics* such as fingerprints have also been evaluated. Recent work constructed a dictionary of 200,000 fingerprints, breaking Android smartphone fingerprint authentication with an average of 21,739 attempts, implying around 14.4 bits of entropy [5].
- For *multiple passwords* from the same user with structural similarities, Transformer-based models capture these patterns and yield accurate guessability estimates [57].

When combining independent factors, the total entropy of the combination is approximately the sum of the entropies of the individual factors. Similarly, their joint guessing probability is the product of individual guessing probabilities. As formalized in Definition 4, the effective security of a combination directly depends on its overall guessability.

Therefore, to enforce a security baseline, systems should (1) estimate the entropy of each factor, (2) evaluate the entropy of factor combinations, and (3) ensure that the weakest combination satisfies the configured security threshold.

B. Our Basic as-MFAKE Protocol

The core idea of our design is to distribute the user's authentication key among devices based on the user's access structure $\mathcal{AS}$ using secret sharing (SS). This enables the user to reconstruct the authentication key with any factor combination $FC \in \mathcal{AS}$, which can then be used for authenticated key exchange (AKE).

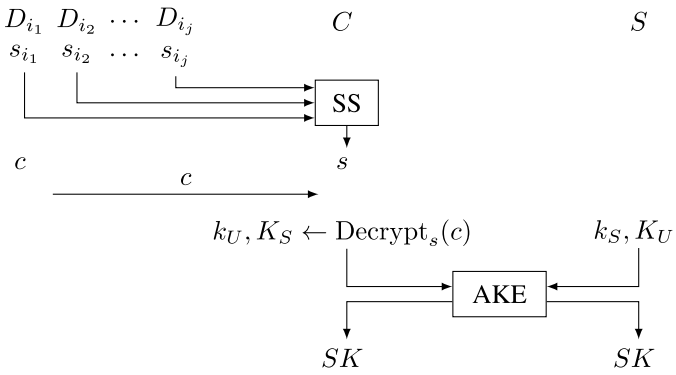


Fig. 7. Design of our AS-MFAKE protocol with the factor combination $FC = \{D_{i_1}, D_{i_2}, \ldots, D_{i_j}\} \in \mathcal{AS}$.

To resist key-compromise impersonation (KCI) attacks, the AKE must be asymmetric, requiring the user to store the server's public key K_S. A straightforward solution is to rely on a Public Key Infrastructure (PKI), but this introduces vulnerabilities to PKI failures. Alternatively, K_S could be stored directly on the user's devices. However, if a device is compromised, the attacker could forge K_S and impersonate the server, enabling offline guessing attacks without compromising or corrupting K_S. This violates the security requirements outlined in Definition 4 and significantly weakens the protocol's security. Offline guessing is particularly problematic, as it is far easier to execute than online guessing, which can be mitigated by server-side measures like rate limiting (e.g., locking accounts after multiple failed login attempts).

To mitigate these risks, we avoid using the shared secret s directly as the authentication key. Instead, s is used to encrypt the user's private key k_U and the server's public key K_S, producing a ciphertext c that is stored on the device(s). Using authenticated encryption (AE) ensures that the public key is authenticated and cannot be forged, addressing the aforementioned vulnerabilities.

We outline our AS-MFAKE protocol, as shown in Figs. 7 and 8:

1) *Registration Phase:*
   a) The user generates a secret s and shares it among devices using secret sharing such that s can be reconstructed from a factor combination FC if and only if $FC \in \mathcal{AS}$.
   b) The user generates private and public keys k_U and K_U for authentication, sends K_U to the server, and obtains the server's public key K_S. The user encrypts k_U and K_S using s and stores the resulting ciphertext c on each device.
   c) The server stores the user's public key K_U.
2) *Authentication Phase:* As illustrated in Fig. 7,
   a) The user reconstructs the secret s from a factor combination $FC \in \mathcal{AS}$.
   b) The user retrieves the ciphertext c from the devices and decrypts it using s to recover k_U and K_S. If some devices are compromised and their

Device D_i		**Client** C		**Server** S
Reg:		Generate key $s \leftarrow\$ \{0,1\}^l$		Maintain private/public keys k_S, K_S
		Generate share s_i of s under $\mathcal{AS}$		
		Generate private/public keys k_U, K_U	$\xleftarrow{K_S}$	
Store s_i, c	$\xleftarrow{s_i, c}$	$c \leftarrow \text{Decrypt}_s(k_U, K_S)$	$\xrightarrow{K_U}$	Store K_U
Auth:				
	$\xrightarrow{s_i, c}$	$x \leftarrow\$ \mathbb{Z}_{m_1}, X \leftarrow g_1^x$	$\xrightarrow{X}$	
		Reconstruct s from $\{s_{i_j}\}_j$, when $\{D_{i_j}\}_j \in \mathcal{AS}$	$\xleftarrow{Y}$	$y \leftarrow\$ \mathbb{Z}_{m_1}, Y \leftarrow g_1^y$
		$k_U, K_S \leftarrow \text{Decrypt}_s(c)$		
		$SK \leftarrow H_1((YK_S^{H_2'(Y,K_U)})^{x+H_2'(X,K_S)k_U})$		$SK \leftarrow H_1((XK_U^{H_2'(X,K_S)})^{y+H_2'(Y,K_U)k_S})$

Fig. 8. Our AS-MFAKE protocol with device factors and access structure $\mathcal{AS}$.

corresponding c is forged, the user can detect this tampering and reset the devices to restore security.

c) The user uses k_U and K_S to execute the AKE protocol with the server.

Theorem 1: Based on a SS scheme Π_{ss}, an AE scheme Π_{ae}, and a KCI-secure AKE scheme Π_{ake}, our AS-MFAKE protocol is secure as defined in Definition 3. Specifically,

$$\mathsf{Adv}^{\text{as-mfake}}_{\Pi,\mathcal{AS}} \leq \mathsf{Adv}^{\text{ss}}_{\Pi_{ss}} + \mathsf{Adv}^{\text{ae}}_{\Pi_{ae}} + \mathsf{Adv}^{\text{ake-kci}}_{\Pi_{ake}} \leq \mathsf{negl}(\kappa),$$

where $\mathsf{Adv}^{\text{ss}}_{\Pi_{ss}}$, $\mathsf{Adv}^{\text{ae}}_{\Pi_{ae}}$, *and* $\mathsf{Adv}^{\text{ake-kci}}_{\Pi_{ake}}$ *represent the maximum advantage of breaking* Π_{ss}, Π_{ae}, *and* Π_{ake}, *respectively.*

The bound is intuitive, as illustrated by the following attack chains. As shown in Fig. 7, an attacker seeking to obtain SK needs to either compromise k_U or break the AKE. To acquire k_U, the attacker needs to either compromise s or break the encryption scheme. Finally, obtaining s requires either compromising the necessary shares of s or breaking the SS.

Proof: As stated in Definition 3, we consider the case where, for every factor combination $FC \in \mathcal{AS}$, at least one device remains honest. We do not account for scenarios where the server is compromised and the attacker impersonates it during the session. However, we do consider the scenario where the server is compromised but does not impersonate the server, instead honestly forwarding messages between legitimate parties. This scenario is referred to as KCI security, which our AS-MFAKE protocol inherits from the AKE protocol.

To prove the theorem, we employ a hybrid argument with the following games:

1) G_1: The game for AKE security, involving only C and S running Π_{ake} for key exchange.
2) G_2: Identical to G_1, except the attacker is also given the ciphertext c.
3) G_3: The game for AS-MFAKE, which is the same as G_2, but the attacker additionally learns shares of s that do not suffice to reconstruct the secret under $\mathcal{AS}$.

In G_1, no attacker can achieve a non-negligible advantage due to the KCI security of Π_{ake}:

$$\mathsf{Adv}^{G_1} = \mathsf{Adv}^{\text{ake-kci}}_{\Pi_{ake}}.$$

The difference between G_1 and G_2 is negligible; otherwise, the attacker could distinguish a random string from a ciphertext without the key, breaking the ciphertext indistinguishability of Π_{ae}. Therefore:

$$\mathsf{Adv}^{G_2} \leq \mathsf{Adv}^{G_1} + \mathsf{Adv}^{\text{ae}}_{\Pi_{ae}} \leq \mathsf{Adv}^{\text{ake-kci}}_{\Pi_{ake}} + \mathsf{Adv}^{\text{ae}}_{\Pi_{ae}}.$$

The difference between G_2 and G_3 is also negligible, as the attacker cannot infer s without sufficient shares. Thus:

$$\mathsf{Adv}^{G_3} \leq \mathsf{Adv}^{G_2} + \mathsf{Adv}^{\text{ss}}_{\Pi_{ss}} \leq \mathsf{Adv}^{\text{ake-kci}}_{\Pi_{ake}} + \mathsf{Adv}^{\text{ae}}_{\Pi_{ae}} + \mathsf{Adv}^{\text{ss}}_{\Pi_{ss}}.$$

This concludes the proof.□

As demonstrated in the proof, our AS-MFAKE protocol is modular. The security proof is independent of the specific constructions of Π_{ss}, Π_{ae}, and Π_{ake}. Hence, any secure instantiations of these schemes can be employed.

C. Our as-MFAKE Protocol With Guessable Factors

Passwords and biometric factors cannot directly store the shares of the secret s, unlike device-based factors. As shown in Fig. 9, to integrate these factors into the secret sharing scheme, we utilize OPRF and fuzzy extractors to bind passwords and biometric factors to their respective shares.

1) *Registration Phase:* This follows the same steps as the basic AS-MFAKE protocol, with additional processing for passwords and biometric factors:
   a) For each password pw_i, the server generates a key k_i for the OPRF F. The user runs the OPRF with the server to compute $s'_{pw_i} = F_{k_i}(pw_i)$. The user then uses s'_{pw_i} to encrypt the share s_{pw_i} for pw_i, producing ciphertext c_{pw_i}. The user sends c_{pw_i} to the server, which stores it.
   b) For biometric factors bio_i, the user generates a key r_i and helper string h_i from the noisy biometric input w_i using a fuzzy extractor, i.e., $(r_i, h_i) \leftarrow \text{Gen}(w_i)$. The user stores h_i on the devices and processes r_i in the same way as passwords. Biometric features are stored on devices instead of servers to mitigate lifelong risks from server-side leakage.
2) *Authentication Phase:* The steps are similar to the basic protocol but include retrieving shares for guessable factors:
   a) For passwords pw_i, the user runs the OPRF with the server to regenerate s'_{pw_i}. The user retrieves c_{pw_i} from the server and decrypts it using s'_{pw_i} to obtain s_{pw_i}.
   b) For biometric factors bio_i, the user retrieves h_i from the devices and regenerates r_i using the noisy biometric input w'_i, i.e., $r_i \leftarrow \text{Rep}(w'_i, h_i)$. The regenerated r_i is then processed as described for passwords.

Device D_i		**Client** C		**Server** S
Reg:		input pw_i and w_i		
		$t_{pw_i} \leftarrow\$ \mathbb{Z}_{m_2}$, $\alpha_{pw_i} \leftarrow H_2'(pw_i)^{t_{pw_i}}$		
		$(r_i, h_i) \leftarrow \text{Gen}(w_i)$, $t_{bio_i} \leftarrow\$ \mathbb{Z}_{m_2}$, $\alpha_{bio_i} \leftarrow H_2'(r_i)^{t_{bio_i}}$	$\xrightarrow{\alpha_{pw_i}, \alpha_{bio_i}}$	Generate $k_{pw_i}, k_{bio_i} \leftarrow\$ \mathbb{Z}_{m_2}$
Store		$s'_{pw_i} \leftarrow H_2(pw_i, \beta_{pw_i}^{1/t_{pw_i}})$, $s'_{bio_i} \leftarrow H_2(r_i, \beta_{bio_i}^{1/t_{bio_i}})$	$\xleftarrow{\beta_{pw_i}, \beta_{bio_i}}$	$\beta_{pw_i} = \alpha_{pw_i}^{k_{pw_i}}, \beta_{bio_i} = \alpha_{bio_i}^{k_{bio_i}}$
c_{pw_i}, c_{bio_i}, h_i	$\xleftarrow{c_{pw_i}, c_{bio_i}, h_i}$	$c_{pw_i} \leftarrow \text{Encrypt}_{s'_{pw_i}}(s_{pw_i}), c_{bio_i} \leftarrow \text{Encrypt}_{s'_{bio_i}}(s_{bio_i})$		Store k_{pw_i}, k_{bio_i}
Auth:		input pw_i, w_i'		
	$\xrightarrow{c_{pw_i}, c_{bio_i}, h_i}$	$t_{pw_i} \leftarrow\$ \mathbb{Z}_{m_2}$, $\alpha_{pw_i} \leftarrow H_2'(pw_i)^{t_{pw_i}}$		
		$r_i \leftarrow \text{Rep}(w_i', h_i)$, $t_{bio_i} \leftarrow\$ \mathbb{Z}_{m_2}$, $\alpha_{bio_i} \leftarrow H_2'(r_i)^{t_{bio_i}}$	$\xrightarrow{\alpha_{pw_i}, \alpha_{bio_i}}$	
		$s'_{pw_i} \leftarrow H_2(pw_i, \beta_{pw_i}^{1/t_{pw_i}})$, $s'_{bio_i} \leftarrow H_2(r_i, \beta_{bio_i}^{1/t_{bio_i}})$	$\xleftarrow{\beta_{pw_i}, \beta_{bio_i}}$	$\beta_{pw_i} = \alpha_{pw_i}^{k_{pw_i}}, \beta_{bio_i} = \alpha_{bio_i}^{k_{bio_i}}$
		$s_{pw_i} \leftarrow \text{Decrypt}_{s'_{pw_i}}(c_{pw_i}), s_{bio_i} \leftarrow \text{Decrypt}_{s'_{bio_i}}(c_{bio_i})$		

Fig. 9. Extension to our AS-MFAKE protocol to support password (pw_i) and biometric (bio_i) factors through processing of their shares s_{pw_i} and s_{bio_i}.

1) Security Requirements for Authenticated Encryption: To meet the security guarantees outlined in Definition 4, the AE scheme used in our protocol must satisfy the following properties, as in Li et al.'s T-MFAKE design:

1) *Authentication:* The AE scheme must ensure security against chosen ciphertext attacks, providing message integrity and confidentiality. This can be achieved by combining an encryption scheme with a message authentication code (MAC).
2) *Random-Key Robustness:* This property ensures that it is infeasible to construct a ciphertext c that is decryptable for two random given keys k_1 and k_2. Formally, for any PPT attacker $\mathcal{A}$,

$$\Pr_{k_1,k_2 \leftarrow \$\{0,1\}^l}[c \leftarrow \mathcal{A}(k_1,k_2) \text{s.t. } \text{Dec}_{k_i}(c) \neq \perp, i = 1,2]$$

is negligible.
3) *Equivocability:* This property ensures that the encryption of a message can be simulated without knowing the plaintext initially. Formally, the outputs of the following two games are indistinguishable to any adversary.
   a) Real game: $\mathcal{A}$ provides a message m, generates (k, c) using $k \leftarrow\$ \{0,1\}^l$ and $c \leftarrow \text{Enc}_k(m)$.
   b) Simulated game: $\mathcal{A}$ provides m, and a simulator $\mathcal{S}$ generates (c, k) such that $c \leftarrow \mathcal{S}(|m|)$ and $k \leftarrow \mathcal{S}(m)$.

Theorem 2: Our AS-MFAKE protocol with guessable factors is secure as defined in Definition 4.

The attack chain is similar to that in Theorem 1, except that the adversary is additionally able to guess certain factors to recover s, and subsequently derive k_U and SK.

Proof: We consider the simple case where, for a factor combination FC, the combination FC' of honest factors in FC contains only one guessable factor. For cases where FC' contains multiple guessable factors, the proof can be easily extended. We treat the guessable factor as a password for analysis. The security of biometric factors follows from the security of the fuzzy extractor.

To prove the bound for this simple case, we reduce the security of our AS-MFAKE protocol to that of OPAQUE [60], an asymmetric PAKE being standardized by the IETF. This reduction avoids addressing details already covered in the proof of OPAQUE's security.

Formally, given an attacker $\mathcal{A}$ that breaks our AS-MFAKE protocol, we construct an attacker $\mathcal{B}$ to break OPAQUE. To clarify, we briefly review the OPAQUE game G_{OPAQUE} and compare it to the game for our AS-MFAKE protocol:

1) G_{OPAQUE}: The game for OPAQUE involves a client C and a server S. The client runs OPRF F using a password pw to compute $rw = F_k(pw)$, decrypts the ciphertext c containing k_U, K_S using rw, and executes AKE with k_U, K_S.
2) $G_{\text{AS-MFAKE}}$: The user (client) computes $s_{pw} = F_k(pw)$ via OPRF, reconstructs s using s_{pw} and other compromised shares (corresponding to factors in FC), decrypts the ciphertext c of k_U, K_S using s, and runs AKE with k_U, K_S.
3) G'_{OPAQUE}: A modified version of G_{OPAQUE}, where $F_k(pw)$ is replaced with s, reconstructed as in $G_{\text{AS-MFAKE}}$.

If the attacker knows only compromised shares and not s_{pw}, the security of Π_{ss} ensures that s appears as a random string. Thus, the computation of s from pw is equivalent to a pseudorandom function.

The advantage in G'_{OPAQUE} is bounded as in G_{OPAQUE}:

1) If S is neither corrupted nor compromised,

$$\mathsf{Adv}^{\text{apake}}_{\Pi_{opaque}} \leq f_{pw}(q) + \mathsf{negl}(\kappa),$$

where q is the number of rogue **Send** queries, and $f_{pw}(i)$ is the maximum cracking rate for passwords with i guesses.
2) If S is corrupted or compromised,

$$\mathsf{Adv}^{\text{apake}}_{\Pi_{opaque}} \leq f_{pw}(q') + \mathsf{negl}(\kappa),$$

where q' is the number of offline operations on S's long-lived secrets.

Using attacker $\mathcal{A}$ for $G_{\text{AS-MFAKE}}$, we construct $\mathcal{B}$ to break G'_{OPAQUE}, giving:

$$\mathsf{Adv}^{\text{as-mfake}}_{\Pi,FC}(\mathcal{A}) \leq \mathsf{Adv}^{\text{apake}}_{\Pi_{opaque}} + \mathsf{negl}(\kappa).$$

Thus, the advantage of breaking our AS-MFAKE protocol is bounded by $\mathsf{Adv}^{\text{apake}}_{\Pi_{opaque}}$.□

TABLE I
SUMMARY AND COMPARISON OF EXISTING MULTI-FACTOR AUTHENTICATION SCHEMES

Schemes	Type	Techniques	Advantages	Limitations
Industrial schemes [8], [9], [21]	Static	TLS + Separate factor authentication	Deployability.	Suffer from PKI failures and server compromise.
Shirvanian et al. [22]	Static	Salt reconstruction	Resist server compromise.	Suffer from server corruption.
Jarecki et al. [17]	Static	KE + SAS-MA + PTR + aPAKE	Highest attainable security.	High computational and communication costs.
Han et al. [15]	Static	DH KE + fuzzy extractor	Support the combination of a password and a biometric factor.	Suffer from offline guessing even without server compromise.
Jiang et al. [16]	Static	RSS + OT + PRG	Support the combination of a password and a biometric factor.	Fail to provide server authentication and incur high computational and communication costs.
Li et al.'s T-MFAKE [23]	Dynamic	TOPRF + SaAKE (+ fuzzy extractor)	Security (highest attainable) and usability.	Only support threshold factor configurations.
Our AS-MFAKE	Dynamic	SS + OPRF + AKE (+ fuzzy extractor)	Security (highest attainable), usability (highest attainable), and efficiency.	No above weaknesses.

TABLE II
PERFORMANCE COMPARISON

Protocol	**Factor combination**	**Computation cost**			**Communication cost**		**Storage cost**	
		Server	Client	Device	Client-Server	Client-Device	Server	Device
OpTFA [17]	1 PW and 1 device	4 exp + 1 mexp	9 exp + 1 mexp	3 exp	5 rounds	5 rounds	1 hash + 2 kPRF + 2 priv + 4 pub	2 kPRF
		5.74 ms	30.82 ms	30.42 ms	317.48 ms	2.23 ms	420 B	64 B
Li et al.'s T-MFAKE [1]	1 PW and 1 device	2 exp + 1 mexp	3 exp + 1 mexp	1 exp	2 rounds	2 rounds	1 kPRF + 1 priv + 2 pub	1 hash + 1 kPRF + 1 priv + 2 pub
		3.33 ms	13.28 ms	10.25 ms	123.26 ms	0.97 ms	194 B	226 B
	1 PW and t_2 devices	2 exp + 1 mexp	3 exp + 1 mexp	t_2 exp	2 rounds	2 rounds	1 kPRF + 1 priv + 2 pub	t_2 hash + t_2 kPRF + t_2 priv + $2t_2$ pub
	1 PW, t_2 devices, and 1 Bio. factor	2 exp + 1 mexp	3 exp + 1 mexp	$t_2 + 1$ exp + 1 fe	2 rounds	2 rounds	1 kPRF + 1 priv + 2 pub	t_2 hash + t_2 kPRF + t_2 priv + $2t_2$ pub + t_2 fe
Our AS-MFAKE	1 PW and 1 device	2 exp + 1 mexp	3 exp + 1 mexp	0	2 rounds	2 rounds	1 kPRF + 1 priv + 2 pub	1 hash + 1 kPRF + 1 priv + 2 pub
		3.34 ms	13.32 ms	0 ms	121.74 ms	0.99 ms	194 B	226 B
	1 PW and t_2 devices	2 exp + 1 mexp	3 exp + 1 mexp	0	2 rounds	2 rounds	1 kPRF + 1 priv + 2 pub	t_2 hash + t_2 kPRF + t_2 priv + $2t_2$ pub
	1 PW, t_2 devices and 1 Bio. factor	3 exp + 1 mexp	5 exp + 1 mexp	1 fe	2 rounds	2 rounds	2 kPRF + 1 priv + 2 pub	t_2 hash + t_2 kPRF + t_2 priv + $2t_2$ pub + t_2 fe
	t_1 PWs, t_2 devices, and t_3 Bio. factors	$1+t_1+t_3$ exp + 1 mexp	$1+2t_1+2t_3$ exp + 1 mexp	t_3 fe	2 rounds	2 rounds	$t_1 + t_3$ kPRF + 1 priv + 2 pub	t_2 hash + t_2 kPRF + t_2 priv + $2t_2$ pub + t_2t_3 fe

Computation cost: exp: number of exponentiation operations; mexp: number of multi-exponentiation operations; fe: number of fuzzy extractor operations.
Storage cost: kPRF: length of keys for PRF; priv: length of private keys; pub: length of public keys; hash: length of hash values; fe: length of the helper string for the fuzzy extractor.
The storage requirements of our AS-MFAKE depend on the access structure $\mathcal{AS}$. For storage cost, we adopt a threshold access structure in AS-MFAKE to enable a fair comparison with Li et al.'s T-MFAKE [1].

While AS-MFAKE supports multiple biometric traits as independent factors, it is recommended to combine them into a single factor to reduce the risk of false acceptance caused by noise, as suggested in [1].

D. Communication and Computation Costs

For a fair comparison with Li et al.'s T-MFAKE, we implement our AS-MFAKE protocol using the same AKE and OPRF components: HMQV and 2HashDH. The details of implemented protocol is illustrated in Fig. 10.

1) Communication Cost: Using HMQV and 2HashDH, as shown in Fig. 11, our AS-MFAKE requires only two communication rounds, matching Li et al.'s T-MFAKE, since HMQV, 2HashDH, and SS can execute in parallel. However, if alternative AKE or OPRF protocols are used that cannot run concurrently, AS-MFAKE may require additional rounds.

2) Computation Cost: In AS-MFAKE, the main computational operations include exponentiation, multi-exponentiation, fuzzy extractor, symmetric encryption, and multiplication. Since the overhead of the last two operations is negligible compared to other operations, we exclude these from the analysis. We focus on the authentication phase, as registration occurs only once, while authentication happens for every login.

The computational cost depends on the factor combination used during login. For a combination FC with t_1 passwords, t_2 biometric factors and t_3 devices, AS-MFAKE requires t_2 fuzzy extractor executions, $t_1 + t_2$ OPRF executions, and one AKE execution. This results in t_2 fuzzy extractor operations, $2 + 3t_1 + 3t_2$ exponentiation operations, and 2 multi-exponentiation operations. See Table II for details.

E. Extensions: Refreshment and Anonymity

1) Refreshment: Similar to T-MFAKE, our AS-MFAKE can periodically refresh the shares in SS, enhancing both security and usability. Regular refreshment ensures that attackers must compromise enough shares within a single period, increasing

Parameters: The subscripts 1 and 2 distinguish the parameters of HMQV and 2HashDH:

- G_1 (G_2) is a cyclic group generated by g_1 (g_2) with a prime order m_1 (m_2).
- H_1, H_1', H_2, and H_2' are hash functions: $H_1 : G_1 \to \mathbb{Z}_{m_1}$, $H_1' : G_1 \times G_1 \to \{0,1\}^l$, $H_2 : \{0,1\}^* \to \{0,1\}^l$, $H_2' : \{0,1\}^* \to G_2$. The PRF $F_k(x)$ is defined as $F_k(x) = H_2(x, H_2'(x)^k)$.

Registration:

- The user U on the client C generates $s \leftarrow\$ \{0,1\}^l$, and splits s into shares s_P for each factor P based on the access structure $\mathcal{AS}$. For a device-based factor D_i, s_{D_i} is stored on D_i. For each password pw_i, C initiates an OPRF with the server, computes $s'_{pw_i} \leftarrow F_{k_{pw_i}}(pw_i) = H_2(pw_i, H_2'(pw_i)^{k_{pw_i}})$, encrypts s_{pw_i} with s'_{pw_i}, and sends the ciphertext c_{pw_i} to the server S. For a biometric factor bio_i with noisy input w_i, C extracts $(r_i, h_i) \leftarrow \mathrm{Gen}(w_i)$, initiates an OPRF to compute $s'_{bio_i} \leftarrow F_{k_{bio_i}}(r_i)$, encrypts s_{bio_i} with s'_{bio_i}, sends c_{bio_i} to S, and stores h_i on devices.
- C generates private and public keys $k_U \leftarrow\$ \mathbb{Z}_{m_1}$ and $K_U \leftarrow g_1^{k_U}$. K_U is sent to S, and K_S is obtained. C encrypts k_U and K_S with s, storing the ciphertext c on devices.
- The server S stores K_U and, for each guessable factor P, generates and stores k_P and c_P.

Authentication:

- U on C selects $x \leftarrow\$ \mathbb{Z}_{m_1}$, computes $X = g_1^x$, and sends (U, X) to the server S.
- At the same time, U chooses a factor combination FC to retrieve the shares for factors in FC. For devices, shares are retrieved directly. For each password pw_i, the user selects $t_{pw_i} \leftarrow\$ \mathbb{Z}_{m_2}$, computes $\alpha_{pw_i} \leftarrow H_2'(pw_i)^{t_{pw_i}}$, and sends α_{pw_i} to S. For each biometric factor bio_i with noisy input w_i', h_i is retrieved, and $r_i \leftarrow \mathrm{Rep}(w_i', h_i)$ is computed. C picks $t_{bio_i} \leftarrow\$ \mathbb{Z}_{m_2}$, computes $\alpha_{bio_i} \leftarrow H_2'(R)^{t_{bio_i}}$, and sends α_{bio_i} to S.
- Receiving (U, X) and α_{P_i} for guessable factors P_i, S selects $y \leftarrow\$ \mathbb{Z}_{m_1}$, computes $Y \leftarrow g_1^y$, $\beta_{P_i} \leftarrow \alpha_{P_i}^{k_{P_i}}$, and sends $(Y, \{\beta_{P_i}, c_{P_i}\}_{P_i})$ back to C. The session key is:
$$SK \leftarrow H_1((XK_U^{H_2'(X,K_S)})^{y+H_2'(Y,K_U)k_S}).$$
- Receiving $(Y, \{\beta_{P_i}, c_{P_i}\}_{P_i})$, C computes s'_{P_i}: for passwords, $s'_{pw_i} \leftarrow H_2(pw_i, \beta_{pw_i}^{1/t_{pw_i}})$; for biometrics, $s'_{bio_i} \leftarrow H_2(r_i, \beta_{bio_i}^{1/t_{bio_i}})$. U decrypts s_{P_i} from c_{P_i} using s'_{P_i}, retrieves s_{D_i} from devices, reconstructs s, decrypts k_U and K_S from c using s, and computes:
$$SK \leftarrow H_1((YK_S^{H_2'(Y,K_U)})^{x+H_2'(X,K_S)k_U}).$$

Fig. 10. AS-MFAKE protocol with support for guessable factors, using HMQV and 2HashDH for AKE and OPRF.

their difficulty. Additionally, if a factor is lost, its corresponding share can be revoked during the refreshment process.

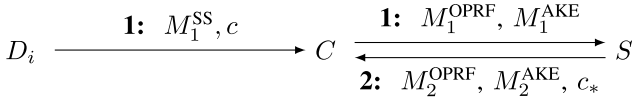


Fig. 11. Message flows of our AS-MFAKE within *only two* rounds.

2) Anonymity: AS-MFAKE can integrate anonymity mechanisms, similar to T-MFAKE, to protect user privacy. This ensures that 1) attackers cannot link a session to a specific user ID, and 2) they cannot determine whether two sessions belong to the same user. The integration of AS-MFAKE with anonymity mechanisms is straightforward, and specific implementation details are omitted here.

F. Comparison Between as-MFAKE and Li Et Al.'s T-MFAKE

AS-MFAKE provides greater flexibility in factor configurations compared to T-MFAKE, which is limited to a threshold access structure including all t-size subsets of $\mathcal{F}$. Even for threshold access structures, our AS-MFAKE design offers distinct advantages:

1) *Password Flexibility:* Li et al.'s T-MFAKE supports only one password and always requires it for authentication due to the construction of their threshold OPRF. AS-MFAKE removes this restriction, allowing users to select zero, one, or multiple passwords as factors.
2) *Modular Design:* T-MFAKE depends on the specific threshold OPRF scheme, 2HashTDH, for security reduction. AS-MFAKE, on the other hand, is modular and supports any secure OPRF, as well as various SS, AE, and AKE schemes, offering greater flexibility.
3) *Efficiency:* As detailed in Section V-D, AS-MFAKE achieves a constant computational cost regardless of the number of devices, while the cost of T-MFAKE increases linearly with the number of devices.

G. Deployment Consideration

1) Storage on Server Side: The server must securely store its own long-term private key k_S, as well as user-related authentication data, including each user's public key k_U and associated OPRF keys.

The server's private key k_S demands the highest level of protection, as its compromise would allow an adversary to impersonate the server and potentially compromise all client-server interactions.

User authentication data requires a slightly lower level of protection. If leaked, it may enable offline guessing attacks against user factors, when the weakest combination consists solely of guessable factors (e.g., passwords). However, if every valid factor combination always includes at least one unguessable and honest factor, e.g., a secure hardware token, then the exposure of this data does not lead to a successful impersonation.

Importantly, the server does not store sensitive user secrets such as password hashes or raw biometric templates. This design choice mitigates the risk of long-term user harm in the event of server compromise.

In cloud-based deployments, additional protections must be enforced. The server's private key k_S and sensitive user

authentication data should be stored using hardware-backed key management systems such as Hardware Security Modules (HSMs) or cloud-native Key Management Services (KMS). To prevent unauthorized access, authentication data should be encrypted at rest and in transit, and processed within confidential computing environments (e.g., Intel SGX, AMD SEV) whenever possible. Secure tenant isolation mechanisms must also be employed to protect against cross-VM or side-channel attacks in multi-tenant environments.

2) Storage on Device Side: Each user device is responsible for securely storing the local secrets required to complete the authentication process. These include the share s_i of s, as well as biometric templates.

The secret share is meaningless on its own, but biometric templates are highly sensitive and must remain local. All biometric matching should be performed on-device to preserve user privacy and to minimize the risk of template leakage. These templates should never be transmitted to the server or any third party.

To ensure confidentiality and integrity, all sensitive data on the device should be protected using platform-native secure storage mechanisms, such as Secure Enclave (iOS), Trusted Execution Environment (TEE), or Android Keystore. These environments provide hardware-level protection against unauthorized access, even in the presence of compromised operating systems.

Devices should also implement runtime protections, such as secure boot, device attestation, anti-tampering mechanisms, and detection of rooting or jailbreaking. In the event of device loss or compromise, the system should support secure recovery or revocation mechanisms that prevent adversaries from using leaked device shares to authenticate.

3) Compatibility: On the server side, our protocol involves only the AKE and OPRF operations. Hence, our design is compatible with existing AKE protocols, such as TLS, and requires no modifications to already deployed infrastructures. The only additional requirement is the implementation of the OPRF component, or alternatively, the use of a three-party OPRF. On the user side, our design introduces secret sharing, but this operation is fully transparent to the server, allowing flexible use of any secret sharing implementation across various devices. Therefore, our design preserves compatibility with existing AKE protocols such as TLS, ensuring strong deployability of our scheme.

VI. Implementation and Performance

We implement our AS-MFAKE protocol in a real-world environment, demonstrating its efficiency and practicality.

A. Implementation

1) Participants:

1) The server S runs on an Alibaba Cloud Light Application Server with 2 CPUs (Intel Xeon Platinum 8269CY @ 2.50GHz) and 2GB of memory.
2) The client C runs Chrome on a Windows 11 PC equipped with an Intel Core i5-7300HQ CPU, 16GB of memory, and a Bluetooth 5.0 adapter. A WebAssembly application is employed to handle authentication factors.
3) The device D is a Redmi Note 12 smartphone with a Qualcomm Snapdragon 4 platform, Bluetooth 5.1, and MIUI 13. An app is used to support protocol operations.

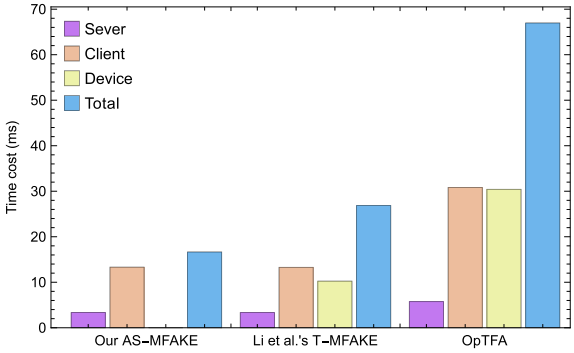


Fig. 12. Time cost of AS-MFAKE, Li et al.'s T-MFAKE, and OpTFA with 1 password and 1 device.

2) Communication: The client communicates with the server over the Internet and with the device via Bluetooth. The round-trip times are approximately 100 ms (client-server) and 1 ms (client-device).

3) Cryptographic Algorithms: As in [1], we use the following:

- SHA-256 for hashing and HMAC-SHA256 for HMAC.
- AES in CTR mode for symmetric encryption.
- Encrypt-then-MAC (AES and HMAC-SHA256) for authenticated encryption.
- NIST P-256 (secp256r1) for elliptic curve exponentiation. The public key is stored in uncompressed form with the prefix 0×04.

4) Repeated Experiments: To ensure the stability of the results, we repeat each test 10,000 times and report the average result.

B. Performance Analysis

1) Theoretical Analysis: Compared to Li et al.'s T-MFAKE, AS-MFAKE incurs lower costs for threshold access structures, as shown in Table II:

1) *Basic case:* 1 password and t devices. AS-MFAKE requires 5 exponentiation and 2 multi-exponentiation operations, compared to $5+t$ exponentiation and 2 multi-exponentiation operations for T-MFAKE.
2) *General case:* 1 password, 1 biometric factor, and t devices. AS-MFAKE requires 8 exponentiation and 2 multi-exponentiation operations, while T-MFAKE needs $6+t$ exponentiation, 2 multi-exponentiation, and 1 fuzzy extractor operation.

T-MFAKE incurs higher costs because it requires one exponentiation per device in its threshold OPRF, whereas AS-MFAKE uses only multiplications in SS for devices.

2) Experimental Results: We measure the authentication time for AS-MFAKE, T-MFAKE [1], and OpTFA [17]:

- *1 password and 1 device:* Fig. 12 shows AS-MFAKE achieves an average time of 16.66 ms, compared to 26.86 ms for T-MFAKE and 66.98 ms for OpTFA. AS-MFAKE improves efficiency by *61.46%* over T-MFAKE.

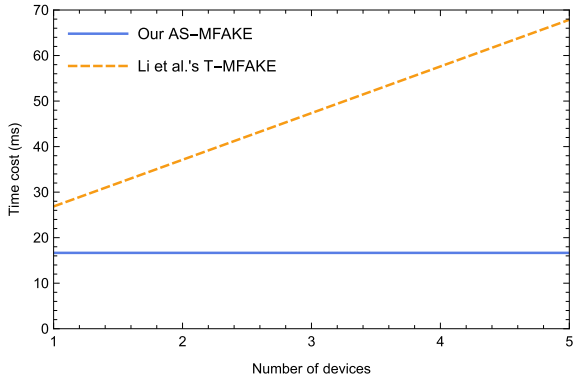


Fig. 13. Time cost of AS-MFAKE and Li et al.'s T-MFAKE with 1 password and t devices.

- *1 password and t devices:* Fig. 13 shows AS-MFAKE maintains a *constant* time cost, while T-MFAKE scales *linearly* with t.

VII. Conclusion

We propose AS-MFA, a novel MFA primitive that allows users to configure factor combinations autonomously, based on their specific security needs and the strength of each factor. This approach strikes an optimal balance between security and usability by utilizing secret sharing for general access structures, enabling support for arbitrary factor combinations.

The security of AS-MFA is formally defined through a game-based model, and we provide a rigorous proof of its security. The protocol demonstrates efficiency in both communication and computation. It requires only two communication rounds, and for a factor combination with t_1 passwords, t_2 biometric factors, and t_3 devices, it performs t_2 fuzzy extractor operations, $2 + 3t_1 + 3t_2$ exponentiations, and 2 multi-exponentiations. These results highlight AS-MFA as a practical and secure solution for modern MFA systems.

References

[1] W. Li, H. Cheng, P. Wang, and K. Liang, "Practical threshold multi-factor authentication," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 3573–3588, 2021.

[2] D. Pasquini, A. Gangwal, G. Ateniese, M. Bernaschi, and M. Conti, "Improving password guessing via representation learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 1382–1399.

[3] D. Pasquini, G. Ateniese, and C. Troncoso, "Universal neural-cracking-machines: Self-configurable password models from auxiliary data," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2024, pp. 1365–1384.

[4] Z. Wu, Y. Cheng, J. Yang, X. Ji, and W. Xu, "DepthFake: Spoofing 3D face authentication with a 2D photo," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2023, pp. 917–933.

[5] Y. Chen, Y. Yu, and L. Zhai, "InfinityGauntlet: Brute-force attack on smartphone fingerprint authentication," in *Proc. USENIX Secur.*, 2023, pp. 2027–2041.

[6] M. Ito, A. Saito, and T. Nishizeki, "Secret sharing scheme realizing general access structure," *Electron. Commun. Jpn. (III, Fundam. Electron. Sci.)*, vol. 72, no. 9, pp. 56–64, Jan. 1989.

[7] B. Applebaum, A. Beimel, O. Farràs, O. Nir, and N. Peter, "Secret-sharing schemes for general and uniform access structures," in *EUROCRYPT*. Cham, Switzerland: Springer, 2019, pp. 441–471.

[8] FIDO Alliance.(2017). *Universal 2nd Factor (U2F) Overview*. [Online]. Available: https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-overview-v1.2-ps-20170411.html

[9] Google.(2020). *GitHub-Google/Google-Authenticator: Open Source Version of Google Authenticator (Except the Android App)*. [Online]. Available: https://github.com/google/google-authenticator

[10] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang, "The tangled web of password reuse," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 23–26.

[11] B. Pal, T. Daniel, R. Chatterjee, and T. Ristenpart, "Beyond credential stuffing: Password similarity models using neural networks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 417–434.

[12] F. Huang, J. Yang, H. Cheng, W. Li, and P. Wang, "Personalized password guessing via modeling multiple leaked credentials of the same user," in *Proc. ESORICS*, 2025, pp. 224–242.

[13] Twilio.(2022). *Incident Report: Employee and Customer Account Compromise*. Accessed: Jun. 8, 2025. [Online]. Available: https://www.twilio.com/en-us/blog/august-2022-social-engineering-attack

[14] S. Foundation. (2022). *Twilio Incident: What Signal Users Need to Know*. Accessed: Jun. 8, 2025. [Online]. Available: https://support.signal.org/hc/en-us/articles/4850133017242-Twilio-Incident-What-Signal-Users-Need-to-Know

[15] Y. Han, C. Xu, C. Jiang, and K. Chen, "A secure two-factor authentication key exchange scheme," *IEEE Trans. Depend. Secure Comput.*, vol. 21, no. 6, pp. 5681–5693, Nov. 2024.

[16] C. Jiang, C. Xu, Y. Han, Z. Zhang, and K. Chen, "Two-factor authenticated key exchange from biometrics with low entropy rates," *IEEE Trans. Inf. Forensics Security*, vol. 19, pp. 3844–3856, 2024.

[17] S. Jarecki, H. Krawczyk, M. Shirvanian, and N. Saxena, "Two-factor authentication with end-to-end password security," in *Proc. PKC*. Cham, Switzerland: Springer, 2018, pp. 431–461.

[18] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu, "Highly-efficient and composable password-protected secret sharing (or: How to protect your Bitcoin wallet online)," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Mar. 2016, pp. 276–291.

[19] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *EUROCRYPT*. Cham, Switzerland: Springer, 2004, pp. 523–540.

[20] R. Canetti, B. Fuller, O. Paneth, L. Reyzin, and A. Smith, "Reusable fuzzy extractors for low-entropy distributions," *J. Cryptol.*, vol. 34, no. 1, pp. 1–33, Jan. 2021.

[21] CISCO.(2021). *Duo Security Two-Factor Authentication*. [Online]. Available: https://goo.gl/wT3ur9

[22] M. Shirvanian, S. Jarecki, N. Saxena, and N. Nathan, "Two-factor authentication resilient to server compromise using mix-bandwidth devices," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 1–16.

[23] M. Wazid, A. K. Das, V. Odelu, N. Kumar, and W. Susilo, "Secure remote user authenticated key establishment protocol for smart home environment," *IEEE Trans. Depend. Secure Comput.*, vol. 17, no. 2, pp. 391–406, Mar. 2020.

[24] GitHub.*Securing Your Account With Two-Factor Authentication (2FA)*. Accessed: Jan. 15, 2025. [Online]. Available: https://docs.github.com/en/authentication/securing-your-account-with-two-factor-authentication-2fa

[25] C. M. University. *How to Use Two-Factor Authentication*. Accessed: Jan. 15, 2025. [Online]. Available: https://www.cmu.edu/computing/services/security/identity-access/authentication/how-to/index.html

[26] B. of America.*Take Your Security to the Next Level*. Accessed: Jan. 15, 2025. [Online]. Available: https://www.bankofamerica.com/security-center/online-banking/

[27] Z. Whittaker. (2018). *Github Says Bug Exposed Some Plaintext Passwords*. [Online]. Available: https://www.zdnet.com/article/github-says-bug-exposed-account-passwords/

[28] L. H. Newman. (2019). *Google Has Stored Some Passwords in Plaintext Since 2005*. [Online]. Available: https://www.wired.com/story/google-stored-gsuite-passwords-plaintext/

[29] M. Weir, S. Aggarwal, B. d. Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *Proc. 30th IEEE Symp. Secur. Privacy*, May 2009, pp. 391–405.

[30] J. Ma, W. Yang, M. Luo, and N. Li, "A study of probabilistic password models," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 538–552.

[31] J. Yang, W. Li, H. Cheng, and P. Wang, "Targeted password guessing using neural language models," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2025, pp. 1–5.

[32] J. Yang, W. Li, H. Cheng, and P. Wang, "Username-password models beyond traditional password guessability assessment," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2025, pp. 1–5.

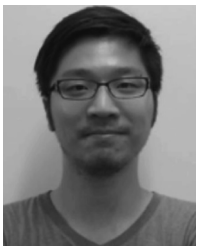
[33] D. M'Raihi, J. Rydell, M. Pei, and S. Machani, *TOTP: Time-Based One-Time Password Algorithm*, document RFC 6238, May 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6238

[34] Y. Xiong, C. Su, W. Huang, F. Miao, W. Wang, and H. Ouyang, "SmartVerif: Push the limit of automation capability of verifying security protocols by dynamic strategies," in *Proc. USENIX*, 2020, pp. 253–270.

[35] M. Barbosa et al., "SoK: Computer-aided cryptography," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 777–795.
[36] J. Ni, K. Zhang, and A. V. Vasilakos, "Security and privacy for mobile edge caching: Challenges and solutions," *IEEE Wireless Commun.*, vol. 28, no. 3, pp. 77–83, Jun. 2021, doi: 10.1109/MWC.001.2000329.
[37] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *EUROCRYPT*. Cham, Switzerland: Springer, 2000, pp. 139–155.
[38] M. Abdalla, P.-A. Fouque, and D. Pointcheval, "Password-based authenticated key exchange in the three-party setting," in *Proc. PKC*, vol. 153. Cham, Switzerland: Springer, 2006, p. 27.
[39] Y. Guo, Y. Guo, P. Xiong, F. Yang, and C. Zhang, "Deeper insight into why authentication schemes in IoT environments fail to achieve the desired security," *IEEE Trans. Inf. Forensics Security*, vol. 19, pp. 4615–4627, 2024.
[40] W. Li, P. Wang, and K. Liang, "HPAKE: Honey password-authenticated key exchange for fast and safer online authentication," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 1596–1609, 2023.
[41] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *CRYPTO*. Cham, Switzerland: Springer, 2007, pp. 232–249.
[42] T. Jager, E. Kiltz, D. Riepel, and S. Schäge, "Tightly-secure authenticated key exchange, revisited," in *Proc. EUROCRYPT*. Cham, Switzerland: Springer, 2021, pp. 117–146.
[43] J. Pan, B. Wagner, and R. Zeng, "Lattice-based authenticated key exchange with tight security," in *Proc. CRYPTO*. Cham, Switzerland: Springer, 2023, pp. 616–647.
[44] H. Krawczyk, "HMQV: A high-performance secure Diffie–Hellman protocol," in *Proc. CRYPTO*, 2005, pp. 546–566.
[45] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.
[46] G. R. BLAKLEY, "Safeguarding cryptographic keys," in *Proc. Int. Workshop Manag. Requirements Knowl. (MARK)*, Jun. 1979, pp. 313–318.
[47] M. Karchmer and A. Wigderson, "On span programs," in *Proc. 8th Annu. Struct. Complex. Theory Conf.*, 1993, pp. 102–111.
[48] J. Benaloh and J. Leichter, "Generalized secret sharing and monotone functions," in *Proc. CRYPTO*. Cham, Switzerland: Springer, 1990, pp. 27–35.
[49] Z. Liu and V. Vaikuntanathan, "Breaking the o(2^n) barrier: Faster secret sharing for general access structures," in *Proc. STOC*, pp. 997–1009.
[50] X. Boyen, "Reusable cryptographic fuzzy extractors," in *Proc. 11th ACM Conf. Comput. Commun. Secur.*, Oct. 2004, pp. 82–91.
[51] B. Fuller, L. Reyzin, and A. Smith, "When are fuzzy extractors possible?," in *Proc. ASIACRYPT*. Cham, Switzerland: Springer, 2016, pp. 277–306.
[52] G. Barthe, B. Portela, F.-X. Standaert, and F. Vercauteren, "Robust and reusable fuzzy extractors from lattices," in *Proc. CRYPTO*. Cham, Switzerland: Springer, 2018, pp. 432–453.
[53] R. Canetti and H. Krawczyk, "Universally composable notions of key exchange and secure channels," in *Proc. EUROCRYPT*, 2002, pp. 337–351.
[54] J. Srinivas, A. K. Das, M. Wazid, and N. Kumar, "Anonymous lightweight chaotic map-based authenticated key agreement protocol for industrial Internet of Things," *IEEE Trans. Depend. Secure Comput.*, vol. 17, no. 6, pp. 1133–1146, Nov. 2020.
[55] G. Lowe, "A hierarchy of authentication specifications," in *Proc. 10th Comput. Secur. Found. Workshop*, Jun. 1997, pp. 31–43.
[56] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Commun. ACM*, vol. 21, no. 12, pp. 993–999, Dec. 1978.
[57] H. Cheng, F. Huang, J. Yang, W. Li, and P. Wang, "Practically secure honey password vaults: New design and new evaluation against online guessing," in *Proc. USENIX Secur.*, 2025, pp. 7781–7798.
[58] J. Bonneau, "The science of guessing: Analyzing an anonymized corpus of 70 million passwords," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 538–552.
[59] A. Juels and T. Ristenpart, "Honey encryption: Security beyond the brute-force bound," in *Proc. EUROCRYPT*. Cham, Switzerland: Springer, 2014, pp. 293–310.
[60] S. Jarecki, H. Krawczyk, and J. Xu, "OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks," in *Proc. EUROCRYPT*. Cham, Switzerland: Springer, 2018, pp. 456–486.

Wenting Li received the Ph.D. degree from Peking University in 2021. She is currently a Lecturer with the School of Information Engineering, Beijing Institute of Graphic Communication, China. She has authored more than 20 papers in journals and conference proceedings, including IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, USENIX Security, and ESORICS. Her research interests include authentication protocols, password security, and copyright protection.

Haibo Cheng received the B.S. degree in pure mathematics from Nankai University, China, in 2011, and the M.S. degree in pure mathematics and the Ph.D. degree in information security from Peking University, China, in 2015 and 2021, respectively. He is currently an Assistant Researcher with the National Engineering Research Center for Software Engineering and the Key Laboratory of High Confidence Software Technologies, Ministry of Education, Peking University. His work has appeared in venues, such as USENIX Security, NDSS, ESORICS, IFIP DSN, and ACM ASIACCS. His research interests include honey encryption and password security.



Kaitai Liang (Member, IEEE) received the Ph.D. degree in computer science, specializing in applied cryptography from the City University of Hong Kong. He is currently an Associate Professor with the Faculty of Technology, University of Turku, Finland, and the Department of Intelligent Systems, Delft University of Technology, The Netherlands. His research interests include applied cryptography, data security, user privacy, cybersecurity, blockchain security, and privacy-enhancing technologies.