AsymSAE: Verifier-Based Asymmetric SAE Protocol for Personal Wi-Fi Networks

Zhiqiang Zhao, Jingwei Jiang, Xuexian Hu, Wei Guo, Jiahui Gao, and Yining Liu, *Senior Member, IEEE*

***Abstract*—Password-authenticated key exchange (PAKE) remains central to WPA3-Personal, the personal mode of the latest Wi-Fi security standard, where the Simultaneous Authentication of Equals (SAE) protocol enables secure Wi-Fi access via a low-entropy password. However, SAE's symmetric password authentication makes all associated stations (STAs) vulnerable once the network-edge access point (AP) is compromised. Although SAE+ (IEEE TIFS 2024) addresses this issue in the client–server model with verifier-based asymmetric authentication, its weak binding of the password-derived verifier, ephemeral randomness, and protocol transcripts still enables key-compromise impersonation (KCI) attacks and offline dictionary attacks. Therefore, designing a verifier-based asymmetric SAE protocol that preserves the SAE workflow while resisting these attacks remains challenging. To address this challenge, we propose AsymSAE, a verifier-based asymmetric SAE protocol tailored for personal Wi-Fi networks. Following the SAE workflow in IEEE Std 802.11-2024, AsymSAE realizes asymmetric authentication by allowing the STA and AP to use the user password and the password-derived verifier, respectively. We further formulate a verifier-based security model to capture the asymmetric structure, and prove the security of AsymSAE in this model. Heuristic security analysis and ProVerif verification demonstrate that AsymSAE provides forward secrecy, transcript-based offline dictionary attack resistance, AP-side KCI resistance, session-key confidentiality, and mutual authentication. Our performance evaluation, including ns-3-based network simulation, demonstrates that AsymSAE can achieve significant verifier-based security enhancement without incurring extra communication overhead, while introducing minimal additional computational overhead and authentication latency.**

***Index Terms*—WPA3-Personal, password-authenticated key exchange, asymmetric SAE, offline dictionary attacks, KCI attacks.**

I. Introduction

WIRELESS local area networks have become one of the most widely deployed access technologies in personal, enterprise, and public communication environments [1]. In personal Wi-Fi networks, users typically authenticate to an access point (AP) using a human-memorable password. The security of such password-based Wi-Fi access is standardized by the Wi-Fi Alliance through the Wi-Fi Protected Access (WPA) family of standards [2]. WPA3, the latest Wi-Fi security standard, was designed to address several known weaknesses in WPA2-Personal, including offline dictionary attacks from captured authentication handshakes and key reinstallation attacks [3]. To improve the security of password-based wireless authentication, WPA3-Personal [4] adopts the Simultaneous Authentication of Equals (SAE) protocol [5], also known as Dragonfly [6], to replace the pre-shared key mechanism used in WPA2-Personal. As shown in Fig. 1, SAE enables a station (STA) and an AP sharing a human-memorable password to perform the commit and confirm exchanges and establish a protected wireless connection. As specified in IEEE Std 802.11-2024 [7], SAE derives a high-entropy session key from a low-entropy password and provides important security properties such as resistance to transcript-based offline dictionary attacks and forward secrecy.

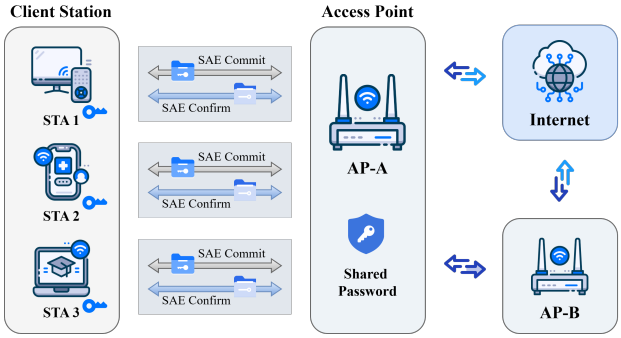


Fig. 1. SAE-based secure wireless access process in a WPA3-Personal network. Each STA independently performs SAE Commit and SAE Confirm exchanges with AP-A using the pre-shared password without transmitting the password itself. After successful SAE authentication, AP-A establishes protected wireless communication with the STA and forwards authenticated traffic to the Internet or other network infrastructure.

Despite these advantages, conventional SAE is essentially a symmetric password-authenticated key exchange (PAKE) protocol. Both the STA and the AP rely on the same password or password-equivalent credential during authentication. This design does not fully meet the security requirements of modern verifier-based systems, where the AP should avoid storing plaintext passwords or reusable password-equivalent secrets. This concern is particularly important in personal Wi-Fi networks, where the AP is continuously online and exposed at the wireless network boundary. Prior studies [8]–[10] have shown that firmware vulnerabilities, insecure default configurations, leakage through local management interfaces, and physical-

This work was supported in part by the National Natural Science Foundation of China under Grants 62472438 and 62572483, and in part by the Young Scientists Fund of the National Natural Science Foundation of China under Grant 62502241. (Corresponding author: Xuexian Hu)

Zhiqiang Zhao, Xuexian Hu, and Jiahui Gao are with the State Key Laboratory of Mathematical Engineering and Advanced Computing, Information Engineering University, Zhengzhou 450000, China. (Email: zhiqiang0909@126.com, xuexian_hu@hotmail.com, gaojiahui0509@163.com).

Jingwei Jiang is with the College of Cryptology and Cyber Science, Nankai University, Tianjin 300350, China. (Email: jiangjingwei@nankai.edu.cn).

Wei Guo is with the Department of Electrical and Electronic Engineering, University of Cagliari, Via Marengo 3, Cagliari 09100, Italy. (Email: wei.guo.cn@outlook.com).

Yining Liu is with the School of Data Science and Artificial Intelligence, Wenzhou University of Technology, Wenzhou 325027, China. (Email: lyn7311@sina.com).

access attacks may all result in the disclosure of credentials stored on the AP. If the AP-side credential information is leaked, the shared Wi-Fi password may be exposed or directly reused, enabling unauthorized access and affecting all STAs that rely on the same network password. Therefore, it is desirable to design an asymmetric SAE protocol in which the STA authenticates using the password, while the AP stores the password-derived verifier. Such a verifier-based (augmented PAKE) design allows the AP to authenticate the STA without storing the plaintext password, so that compromise of the AP exposes only a password-derived verifier, rather than the plaintext password. Consequently, AP-side credential leakage has a reduced impact, and an attacker cannot directly recover the password or impersonate the STA without first mounting an offline dictionary attack against the verifier [11].

A. Challenges

Recent asymmetric PAKE protocols [12]–[15] show that server-side passwords can be replaced with password-derived verifiers. However, directly applying such techniques to SAE is non-trivial: the protocol *must* preserve the SAE commit-and-confirm workflow, withstand AP-side compromise and active attacks, and remain deployable on existing Wi-Fi devices. Our analysis of SAE+ [16] further shows that insufficient binding among the password-derived verifier, ephemeral randomness, and protocol transcripts can lead to KCI attacks and offline dictionary attacks (see Section IV). Therefore, designing a secure and practical asymmetric SAE protocol poses the following three challenges.

Challenge 1: Verifier-Based Asymmetry for SAE. Existing asymmetric PAKE protocols include SRP [12], SPAKE2+ [13], AuCPace [14], and OPAQUE [15]. Among them, AuCPace is a verifier-based PAKE submitted to the IETF Crypto Forum Research Group (CFRG) as a candidate proposal, while OPAQUE has been recommended by the IETF CFRG and standardized as RFC 9807 [17]. However, they are mainly designed for classical client–server settings and cannot be directly applied to SAE. SAE relies on both parties to derive the same password element, exchange scalar-and-element commit messages, and confirm the same session key bound to the password, ephemeral randomness, and protocol transcripts. If the AP stores only a verifier, this symmetric use of the password no longer holds. Therefore, the commit and confirm computations *must* be redesigned so that the STA can authenticate with the password while the AP verifies the authentication without storing the password. Moreover, this setting also requires a suitable security model. Existing PAKE security models [18]–[21], including the recent sPAKE model [22], do not directly capture this verifier-based SAE setting, where the STA holds the password while the AP stores the password-derived verifier.

Challenge 2: Security against credential leakage and active attacks. Among prior SAE variants, SAE+ [16] builds on the SAE design to construct an asymmetric PAKE protocol in a general client–server setting, where each client holds an independent password. This setting differs from personal Wi-Fi networks, where multiple STAs may share one AP password and remain compatible with the IEEE 802.11 SAE workflow. More critically, our analysis shows that SAE+ remains vulnerable to two attacks: key-compromise impersonation (KCI) attacks and offline dictionary attacks. Specifically, server-side credential leakage may enable client impersonation; missing validity checks may enable offline password testing. These observations show that directly adapting asymmetric PAKE techniques to SAE is insufficient. A secure design must instead tightly bind the verifier, password-derived values, ephemeral randomness, and protocol transcripts.

Challenge 3: Preserving deployability and minimizing protocol overhead. WPA3-Personal is deployed on a massive number of Wi-Fi devices, ranging from APs to resource-constrained IoT devices [23]. Therefore, any protocol change should be minimal and easy to integrate into existing Wi-Fi stacks. An asymmetric SAE design should preserve the original SAE workflow, avoid additional communication rounds and extra communication overhead, and minimize both computational overhead and authentication latency.

B. Contributions

To address these challenges, we propose AsymSAE[1], a verifier-based asymmetric SAE protocol tailored for WPA3-Personal networks. AsymSAE follows the STA–AP authentication workflow of IEEE Std 802.11-2024, while replacing the AP-side plaintext password with a password-derived verifier. The protocol consists of four phases: initialization, active scanning, SAE commit exchange, and SAE key confirmation. To resist AP-side KCI attacks, AsymSAE binds the verifier to the AP's fresh randomness and to a transcript-dependent challenge, so that leakage of the AP-side verifier cannot be directly transformed into a valid STA impersonation credential. In addition, AsymSAE incorporates validity checks and direction-bound confirmation tags to prevent invalid-message attacks and reflection of authentication tokens. In summary, our contributions are threefold:

1) We reveal the fundamental security principle for transforming SAE [22] into an asymmetric SAE protocol, namely, strong binding among the password-derived secret, ephemeral randomness, and protocol transcripts. Based on this principle, we demonstrate that SAE+ [16] is vulnerable to KCI attacks and offline dictionary attacks, and identify the underlying causes, including insufficient binding of the password-derived secret and inadequate validation of protocol messages.
2) We propose AsymSAE, the first secure asymmetric variant of SAE for personal Wi-Fi networks. Our AsymSAE retains the SAE commit-and-confirm paradigm defined in IEEE Std 802.11-2024 while replacing AP-side password storage with verifier-based credentials. AsymSAE preserves SAE's inherited properties, and further strengthens AP-side security by providing KCI resistance and limiting the consequences of AP-side credential leakage. Specifically, exposure of the AP-side verifier neither directly reveals the password nor enables passwordless

[1]The implementation is publicly available at https://github.com/zhiqiang0909/AsymSAE-Protocol-for-Personal-Wi-Fi-Networks

STA impersonation without an offline dictionary attack against the verifier.

3) We provide a comprehensive security analysis and performance evaluation of AsymSAE. Specifically, we extend the standard sPAKE security model [22] to the verifier-based asymmetric setting and prove the AKE security of AsymSAE under the extended model. Formal verification with ProVerif [24] further confirms session-key confidentiality, mutual authentication, resistance to transcript-based offline dictionary attacks, forward secrecy, and KCI resistance. Finally, experimental, theoretical, and ns-3-based simulation results show that, compared with SAE [22], AsymSAE maintains the same communication overhead and requires only moderate extra computation, thereby incurring minimal additional authentication latency while achieving stronger security guarantees.

The remainder of this paper is organized as follows. Section II introduces the security model and complexity assumptions. Section III reviews the SAE+ protocol. Section IV analyzes the security flaws of SAE+. Section V presents the proposed AsymSAE protocol. Section VI gives the security analysis and formal verification. Section VII evaluates the protocol properties and performance. Section VIII concludes the paper.

II. Preliminaries

A. Security Model

In this section, we extend the standard sPAKE security model [22] to a verifier-based setting, termed v-sPAKE. This model captures the asymmetric credential structure in which the STA holds the password and the AP holds only a password verifier. A more detailed description is provided below.

Communication model. The protocol participants communicate over a public, fully asynchronous, point-to-point network. The adversary $\mathcal{A}$ controls the entire network and may intercept, delay, modify, replay, and delete messages at will.

Protocol participants. The protocol participants consist of a set of client stations $\mathcal{C} := \{C_1, \cdots, C_n\}$ and a set of access points $\mathcal{S} = \{S_1, \cdots, S_m\}$. Each participant $P \in \mathcal{C} \cup \mathcal{S}$ is modeled as a probabilistic polynomial-time (PPT) Turing machine and may execute a polynomial number of protocol instances in parallel. We use the notation π_P^i to denote the i-th protocol instance of participant P.

Long-lived keys. For each access point $S \in \mathcal{S}$, the challenger samples a password $pw \leftarrow \mathcal{D}$, where $\mathcal{D}$ is a password dictionary. The AP is also associated with public network information, including its network identifier SSID and a salt value s. The password-derived values are computed as $(\pi_0, \pi_1) = \mathcal{H}_0(pw, \mathsf{SSID}, s)$, and the AP-side verifier is defined as $V = \pi_1 \mathsf{H2C}(\pi_0 | \mathsf{SSID})$. The AP stores the verifier file $(\pi_0, V, \mathsf{SSID}, s)$. A STA that attempts to connect to the AP obtains pw out of band, for example through user input, and obtains (SSID, s) during the scanning phase.

Adversarial capabilities. The adversary $\mathcal{A}$ is modeled as a PPT Turing machine, and its capabilities are formalized by allowing it to query the following oracles in any order.

- $\mathsf{Send}(P, i, m)$: This oracle models the adversary's active attacks, where the adversary sends a message m to instance π_P^i, and the challenger returns the response generated by the protocol.
- $\mathsf{Execute}(C, i, S, j)$: This oracle models passive attacks, where the adversary eavesdrops on the honest execution of the protocol between a STA instance π_C^i and an AP instance π_S^j, and the oracle returns the transcript of all exchanged messages.
- $\mathsf{Reveal}(P, i)$: This oracle models the misuse of the session key held by instance π_P^i. If π_P^i has accepted and $key \neq \perp$, this oracle returns the session key held by π_P^i. Otherwise, it returns $\perp$. A session whose key has been returned by this oracle is said to be revealed.
- $\mathsf{VCorrupt}(S)$: This oracle models AP-side verifier compromise. It returns the verifier file $(\pi_0, V, \mathsf{SSID}, s)$ stored by the AP. After this query, S is said to be verifier-corrupted. It captures the main leakage scenario: the AP is compromised and its stored verifier is exposed.
- $\mathsf{PwCorrupt}(C)$: This oracle models password disclosure on the user/client side. It returns the network password pw, together with the client-side public auxiliary information associated with S. After this query, C is said to be password-corrupted.
- $\mathsf{Test}(P, i)$: This oracle models the AKE security of the session key. It may be queried only once and only on an accepted fresh session. The challenger samples an uniformly random bit $b \in \{0, 1\}$. If $b = 1$, the oracle returns the real session key of π_P^i; if $b = 0$, it returns a uniformly random string of the same length.

Partnering. Two instances π_C^i and π_S^j are said to be partnered if and only if the following conditions are satisfied: (1) Both π_C^i and π_S^j have accepted a session key; (2) π_C^i and π_S^j share the same session identifier; (3) The partner identification of π_C^i is π_S^j, and vice versa; (4) Except for π_C^i and π_S^j, no other instance has accepted with a partner identification equal to either π_C^i or π_S^j.

Freshness. An instance π_C^i (resp., π_S^j) is said to be fresh if it has accepted a session key and the following conditions hold. If π_C^i is partnered with π_S^j, then the adversary has queried neither $\mathsf{Reveal}(C, i)$ nor $\mathsf{Reveal}(S, j)$. In addition, the adversary has not both queried $\mathsf{VCorrupt}(S)$ and issued $\mathsf{Send}(C, i, m)$ to the tested STA instance π_C^i (resp., both queried $\mathsf{PwCorrupt}(C)$ and issued $\mathsf{Send}(S, j, m)$ to the tested AP instance π_S^j), for any message m.

AKE security. This security goal is defined via a security game between a challenger and a PPT adversary $\mathcal{A}$. In this game, the challenger maintains the execution of the protocol and responds to the adversary's queries to the oracles. Notably, the adversary may query the Test oracle only once on a fresh instance, and then output a guess bit b'. If $b' = b$, then the adversary is said to win the security game, where b is the hidden bit used by the Test oracle. Let Succ denote the event that the adversary wins the game and $H_\infty(D)$ denote the min-entropy of the password distribution. Then, the protocol $\mathcal{P}$ is said to achieve AKE security, if for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}()$ such that

$$\mathsf{Adv}_{\mathcal{P},\mathcal{D}}^{\mathsf{AKE}}(\mathcal{A}) = 2\Pr[\mathsf{Succ}_{\mathcal{P},\mathcal{D}}^{\mathsf{AKE}}(\mathcal{A})] - 1 \leqslant q_{se}/2^{H_\infty(D)} + \mathsf{negl}(\lambda),$$

where q_{se} denotes the number of $\mathsf{Send}(P, *)$ queries.

B. Cryptographic Hardness Assumptions

Definition 1 (CDH [25]): Let $\mathbb{G}$ be an additive cyclic group of prime order p with generator G. Given a tuple (G, aG, bG), where a, b are independently and uniformly chosen from $\mathbb{Z}_p^*$, the Computational Diffie–Hellman (CDH) problem is to compute the shared group element abG. We say that the CDH assumption holds in $\mathbb{G}$ if any PPT adversary $\mathcal{A}$ can solve this problem only with negligible probability. Formally,

$$\mathsf{Adv}_{\mathbb{G}}^{\mathsf{CDH}}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[\mathcal{A}(G, aG, bG) = abG\right] \leq \mathsf{negl}(\lambda).$$

Definition 2 (GDH [26]): Let $\mathbb{G}$ be an additive cyclic group of prime order p with generator G. The Gap Diffie–Hellman (GDH) assumption states that the Computational Diffie–Hellman problem remains hard even if the adversary is given access to a Decisional Diffie–Hellman (DDH) oracle. Given a tuple (G, aG, bG), where a, b are independently and uniformly chosen from $\mathbb{Z}_p^*$, the adversary is required to compute abG with the help of a DDH oracle $\mathcal{O}_{\mathsf{DDH}}$. We say that the GDH assumption holds in $\mathbb{G}$ if any PPT adversary $\mathcal{A}$ can solve this problem only with negligible probability. Formally,

$$\mathsf{Adv}_{\mathbb{G}}^{\mathsf{GDH}}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[\mathcal{A}^{\mathcal{O}_{\mathsf{DDH}}}(G, aG, bG) = abG\right] \leq \mathsf{negl}(\lambda).$$

Definition 3 (GSDH [27]): Let $\mathbb{G}$ be an additive cyclic group of prime order p with generator G. Given a tuple (G, aG, bG), where a, b are independently and uniformly chosen from $\mathbb{Z}_p^*$, the Gap Simultaneous Diffie–Hellman (GSDH) problem is to find a non-zero group element $H \in \mathbb{G} \setminus \{O\}$ together with the pair $(a^{-1}H, b^{-1}H)$, even with access to a DDH oracle $\mathcal{O}_{\mathsf{DDH}}$. We say that the GSDH assumption holds in $\mathbb{G}$ if any PPT adversary $\mathcal{A}$ can output such a valid triple only with negligible probability. Formally,

$$\begin{aligned}\mathsf{Adv}_{\mathbb{G}}^{\mathsf{GSDH}}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\big[&(H, H_a, H_b) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{DDH}}}(G, aG, bG) : \\ &H \in \mathbb{G} \setminus \{O\} \wedge H_a = a^{-1}H \wedge H_b = b^{-1}H\big] \\ &\leq \mathsf{negl}(\lambda).\end{aligned}$$

III. Review of The SAE+ Protocol

This section reviews the one-round asymmetric SAE+ protocol proposed by Qi et al. [16] under the client-server model. This protocol primarily consists of a registration phase and the commit exchange phase as defined in the original SAE protocol [28]. In addition, the global public parameters of the protocol include an elliptic curve additive group $\mathbb{G}$ of order p and three one-way hash functions $\mathcal{H}_0 : \{0,1\}^* \to \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, $\mathcal{H}_1 : \{0,1\}^* \to \{0,1\}^\ell$, and $\mathsf{H2C} : \mathbb{Z}_p^* \to \mathbb{G}\backslash\{\mathcal{O}\}$, where $\mathcal{O}$ denotes the point at infinity of the elliptic curve.

Registration phase. Once the global public parameters are published, the user selects a password pw, and then the client C computes $(\pi_0, \pi_1) = \mathcal{H}_0(C, S, pw)$ and $V = \pi_1 \mathsf{H2C}(\pi_0)$. Subsequently, the client sends (C, π_0, V) to the server via a secure channel. Upon receiving these values, the server registers them as the client C's password file and then stores the file in the local database.

Commit exchange phase. In this phase, the client C and the server S establish a high-entropy session key sk through one-round exchange. A high-level description of the commit exchange phase of the SAE+ protocol is shown in Fig. 2. The detailed description is as follows.

1) The user inputs the password pw into the client C. The client C computes $(\pi_0, \pi_1) = \mathcal{H}_0(C, S, pw)$. Then, the client C samples $r_1, m_1 \leftarrow_R \mathbb{Z}_p^*$, and computes $s_1 = r_1 + m_1 \bmod p$, $\mathsf{PE} = \mathsf{H2C}(\pi_0)$, and $E_1 = -m_1\mathsf{PE}$. Finally, the client C sends (C, E_1, s_1) to the server S over a public channel.
2) The server S samples $r_2, m_2 \leftarrow_R \mathbb{Z}_p^*$, and computes $s_2 = r_2 + m_2 \bmod p$, $\mathsf{PE} = \mathsf{H2C}(\pi_0)$, and $E_2 = -m_2\mathsf{PE}$. Finally, the server S sends (S, E_2, s_2) to the client C over a public channel.
3) Upon receiving (C, E_1, s_1), the server S computes $K_B = r_2(E_1 + s_1\mathsf{PE} + V)$, and then derives a session key $sk = \mathcal{H}_1(C, S, E_1, s_1, E_2, s_2, \pi_0, K_B)$.
4) Upon receiving (S, E_2, s_2), the client C computes $K_A = (r_1 + \pi_1)(E_2 + s_2\mathsf{PE})$, and then derives a session key $sk = \mathcal{H}_1(C, S, E_1, s_1, E_2, s_2, \pi_0, K_A)$.

IV. Security Flaws of The SAE+ Protocol

In this section, we demonstrate that the SAE+ protocol [16] is susceptible to both KCI attacks and offline dictionary attacks, and further analyze their root causes.

A. Key-compromise Impersonation Attack

A KCI attack means that once Alice's long-term secret value is compromised, an adversary $\mathcal{A}$ can impersonate Bob to Alice [29]. Therefore, we can assume that the adversary $\mathcal{A}$ has queried the $\mathsf{VCorrupt}(S)$ oracle to obtain the server's long-term secret values (π_0, V). Under this condition, the adversary $\mathcal{A}$ will attempt to impersonate the client C to the server S and establish a session key. We now present the successful KCI attack available to the adversary.

1) Attack strategy: Given (π_0, V), the adversary $\mathcal{A}$ attempts to construct (E_1, s_1) while impersonating the client C, so as to derive the same session key sk as the server S without knowing client C's password pw, and thereby pass the server's authentication in the subsequent key-confirmation phase. A high-level description of the interaction between the adversary $\mathcal{A}$ and the server S is shown in Fig. 3. The detailed description is as follows.

1) The adversary $\mathcal{A}$ samples $r_1, m_1 \leftarrow_R \mathbb{Z}_p^*$, and computes $s_1 = r_1 + m_1 \bmod p$, $\mathsf{PE} = \mathsf{H2C}(\pi_0)$, and $E_1 = -V + s_1\mathsf{PE}$. Finally, the adversary $\mathcal{A}$ sends (C, E_1, s_1) to the server S over a public channel.
2) Upon receiving (C, E_1, s_1), the server S samples $r_2, m_2 \leftarrow \mathbb{Z}_p^*$, computes $s_2 = r_2 + m_2 \bmod p$, $\mathsf{PE} = \mathsf{H2C}(\pi_0)$, and $E_2 = -m_2\mathsf{PE}$, and sends (S, E_2, s_2) to the client C. Then, the server S computes $K_B = r_2(E_1 + s_1\mathsf{PE} + V) = r_2((-V + s_1\mathsf{PE}) + s_1\mathsf{PE} + V) = 2r_2s_1\mathsf{PE}$, and derives a session key $sk_s = \mathcal{H}_1(C, S, E_1, s_1, E_2, s_2, \pi_0, K_B)$.
3) Upon receiving (S, E_2, s_2), the adversary $\mathcal{A}$ computes $K_A = 2s_1(E_2 + s_2\mathsf{PE}) = 2s_1r_2\mathsf{PE}$, and then derives a session key $sk_a = \mathcal{H}_1(C, S, E_1, s_1, E_2, s_2, \pi_0, K_A)$.

It is evident that, without knowing client C's password pw, the adversary can impersonate client C to the server S

$\underline{\text{Client } C}$ (pw)		$\underline{\text{Server } S}$ $(\pi_0, V = \pi_1\mathsf{H2C}(\pi_0))$
Global parameters: Elliptic curve additive group $\mathbb{G}$ with prime order p, $\mathsf{H2C}$, $\mathcal{H}_i$ $(i = 0, 1)$		
$(\pi_0, \pi_1) = \mathcal{H}_0(C, S, pw)$		
$r_1, m_1 \leftarrow_R \mathbb{Z}_p^*$		$r_2, m_2 \leftarrow_R \mathbb{Z}_p^*$
$s_1 = r_1 + m_1 \bmod p$		$s_2 = r_2 + m_2 \bmod p$
$\mathsf{PE} = \mathsf{H2C}(\pi_0)$		$\mathsf{PE} = \mathsf{H2C}(\pi_0)$
$E_1 = -m_1\mathsf{PE}$	$\xrightarrow{C, E_1, s_1}$	$E_2 = -m_2\mathsf{PE}$
$K_A = (r_1 + \pi_1)(E_2 + s_2\mathsf{PE})$	$\xleftarrow{S, E_2, s_2}$	$K_B = r_2(E_1 + s_1\mathsf{PE} + V)$
$sk = \mathcal{H}_1(C, S, E_1, s_1, E_2, s_2, \pi_0, K_A)$		$sk = \mathcal{H}_1(C, S, E_1, s_1, E_2, s_2, \pi_0, K_B)$

Fig. 2. The commit exchange phase of the SAE+ protocol. Client C derives (π_0, π_1) from the password pw, while server S stores (π_0, V) with $V = \pi_1\mathsf{H2C}(\pi_0)$. The two parties exchange commit messages and derive the session key sk from the protocol transcripts.

$\underline{\text{Adversary } \mathcal{A}}$ (π_0, V)		$\underline{\text{Server } S}$ $(\pi_0, V = \pi_1\mathsf{H2C}(\pi_0))$
Global parameters: Elliptic curve additive group $\mathbb{G}$ with prime order p, $\mathsf{H2C}$, $\mathcal{H}_i$ $(i = 0, 1)$		
$r_1, m_1 \leftarrow_R \mathbb{Z}_p^*$		$r_2, m_2 \leftarrow_R \mathbb{Z}_p^*$
$s_1 = r_1 + m_1 \bmod p$		$s_2 = r_2 + m_2 \bmod p$
$\mathsf{PE} = \mathsf{H2C}(\pi_0)$		$\mathsf{PE} = \mathsf{H2C}(\pi_0)$
$E_1 = -V + s_1\mathsf{PE}$	$\xrightarrow{C, E_1, s_1}$	$E_2 = -m_2\mathsf{PE}$
$K_A = 2s_1(E_2 + s_2\mathsf{PE})$	$\xleftarrow{S, E_2, s_2}$	$K_B = r_2(E_1 + s_1\mathsf{PE} + V)$
$\boxed{K_A = 2s_1r_2\mathsf{PE}}$		$\boxed{K_B = 2r_2s_1\mathsf{PE}}$
$sk = \mathcal{H}_1(C, S, E_1, s_1, E_2, s_2, \pi_0, K_A)$		$sk = \mathcal{H}_1(C, S, E_1, s_1, E_2, s_2, \pi_0, K_B)$

Fig. 3. The key-compromise impersonation attack on the SAE+ protocol. Given the compromised credentials (π_0, V), the adversary $\mathcal{A}$ forges the first-round message to impersonate client C to server S. Red entries denote values available to or computed by $\mathcal{A}$, and the boxed equations show that both parties derive the same key material, i.e., $K_A = K_B = 2s_1r_2\mathsf{PE}$, enabling $\mathcal{A}$ to compute the session key and successfully impersonate C.

and establish a session key that is accepted by the server. Therefore, the KCI attack is complete. One may then ask whether the server can prevent such a KCI attack by checking whether E_1 is a specific value. Clearly, both E_1 and s_1 sent by the adversary are random values, and K_B is likewise random. Therefore, the server cannot resist KCI attacks merely by checking whether the client's message is equal to some specific value. From the above KCI attack strategy, we can conclude that the SAE+ protocol cannot be patched in a simple ad hoc manner to achieve the claimed KCI resistance property.

2) Root cause analysis: After the adversary $\mathcal{A}$ obtains the server's long-term secret values (π_0, V), the SAE+ protocol cannot resist a KCI attack. The root cause is that the protocol's authentication process relies excessively on the PE derived from π_0 and does not strongly bind π_1. Specifically, the adversary can deliberately construct (E_1, s_1) so that the verifier term V is canceled in the server-side computation, enabling the adversary to derive the same key material $K_B = 2r_2s_1\mathsf{PE}$ from the server's response (E_2, s_2). Therefore, to resist such attacks, the protocol design should bind the server-side verifier V to the server's fresh randomness and a transcript-dependent challenge, so that V cannot be canceled or reused by the adversary. Meanwhile, the client should incorporate the password-derived secret π_1 into the corresponding key computation, ensuring that both parties derive the same verifier-bound key material only in an honest execution.

B. Offline Dictionary Attack

An offline dictionary attack means that after intercepting the interaction messages between legitimate protocol instances, an adversary $\mathcal{A}$ can offline verify the correctness of guessed passwords. Notably, the assumption of the adversary's ability here does not involve the compromise of long-term private keys and password files [30]. Therefore, we can assume that the adversary has queried the $\mathsf{Execute}(C^i, S^j)$ oracle to obtain E_1 and s_1, and has queried the $\mathsf{Send}(C^i, M)$ oracle to send a forged message $M = (S, E_2', s_2')$ to the client C. Under this condition, the adversary $\mathcal{A}$ can successfully launch an offline dictionary attack during the key-confirmation phase. Notably, since the SAE+ protocol does not explicitly specify the operations in the key-confirmation phase and considers the explicit key-confirmation procedure to be redundant in protocols such as TLS 1.3. However, PAKE protocols still require key confirmation through the TLS 1.3 Finished messages [31]. Therefore, we adopt the key-confirmation procedure of the original SAE protocol.

1) Attack strategy: Given a password dictionary $\mathcal{D}$, the adversary $\mathcal{A}$ sets E_2 to the point at infinity $\mathcal{O}$, namely, $E_2 = \mathcal{O}$, and sets $s_2 = 0$. The adversary then impersonates the server S and sends the message (S, E_2, s_2) to the client C. Since the SAE+ protocol does not specify any validity checks on E_2 and s_2, upon receiving

(S, E_2, s_2), the client C computes $K_A = (r_1 + \pi_1)(E_2 + s_2\mathsf{PE}) = \mathcal{O}$ and $sk = \mathcal{H}_1(C, S, E_1, s_1, E_2, s_2, \pi_0, \mathcal{O})$. Subsequently, during the key confirmation phase, the client sends $tok_A = \mathcal{H}_1(sk, C, S, E_1, s_1, E_2, s_2)$ to the server S. The adversary queries the $\mathsf{Execute}(C^i, S^j)$ oracle to intercept tok_A, and then selects a password pw' from the password dictionary $\mathcal{D}$ and computes $(\pi'_0, \pi'_1) = \mathcal{H}_0(C, S, pw')$, $sk' = \mathcal{H}_1(C, S, E_1, s_1, E_2, s_2, \pi'_0, \mathcal{O})$, and $tok'_A = \mathcal{H}_1(sk', C, S, E_1, s_1, E_2, s_2)$. Clearly, the adversary can verify whether the current password guess is correct by checking whether tok_A equals tok'_A. Repeating this process, the adversary $\mathcal{A}$ can successfully launch an offline dictionary attack with only forgery and eavesdropping capabilities.

2) Root cause analysis: If the adversary $\mathcal{A}$ has access to the password dictionary $\mathcal{D}$ and can query both the Send and $\mathsf{Execute}$ oracles, SAE+ cannot resist offline dictionary attacks. The root cause is the lack of validity checks on the server-sent group element E_2 and scalar s_2. As a result, an adversary can force the client to derive the session key sk from attacker-controlled invalid values, making the key-confirmation token tok_A an offline password-verification oracle. Specifically, for each password guess $pw' \leftarrow \mathcal{D}$, the adversary can compute $(\pi'_0, \pi'_1) = \mathcal{H}_0(C, S, pw')$ and verify the guess by comparing the corresponding confirmation token with tok_A. To prevent such attacks, the protocol must follow the validation requirements of RFC 7664 [6]: the scalar must lie strictly between 1 and p, and the received group element must be a valid curve point with coordinates in the underlying field and must not be the point at infinity.

V. Our Proposed Protocol

This section presents the proposed AsymSAE protocol for WPA3-Personal networks. Instead of repeating each computational step in Fig. 4, this section focuses on the design rationale of AsymSAE and highlights the key protocol modifications that enable verifier-based asymmetric authentication while preserving the deployability of SAE in personal Wi-Fi networks.

A. Design Rationale

AsymSAE is designed to introduce verifier-based asymmetric authentication into SAE while preserving its original message flow. During initialization, the password is mapped to $(\pi_0, \pi_1) = \mathcal{H}_0(pw, \mathsf{SSID}, s)$,where SSID denotes the service set identifier and s is a salt value; the AP stores the verifier $V = \pi_1\mathsf{H2C}(\pi_0|\mathsf{SSID})$ instead of the plaintext password. Thus, the STA authenticates with the password-derived secrets π_0 and π_1, while the AP verifies it using only V, reducing the impact of AP-side credential leakage. AsymSAE also binds authentication to the current STA–AP link by setting $\mathsf{PE} = val \cdot \mathsf{H2C}(\pi_0|\mathsf{SSID})$, where val is derived from the ordered MAC-address pair. The main design for resisting AP-side KCI attacks is to make the verifier non-reusable and transcript-dependent. Specifically, AsymSAE derives $c = \mathcal{H}(E_1, s_1, E_2, s_2, \pi_0)$ from both parties' commit messages and embeds the verifier into the AP-side computation as $K_S = r_2(E_1 + s_1\mathsf{PE} + c \cdot val \cdot V)$, where $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p^*$. Since a forged STA commit must be fixed before the AP's fresh commit is known, an adversary with the leaked verifier cannot predict c and thus cannot cancel or reuse the verifier term to force a known session key. This protection requires no additional communication rounds and mainly introduces one extra point multiplication for the term $c \cdot val \cdot V$ at the AP.

B. Key Modifications

Verifier-based credential separation. During initialization, the password is processed together with network-specific information to derive $(\pi_0, \pi_1) = \mathcal{H}_0(pw, \mathsf{SSID}, s)$. The value π_0 is used to derive the password element, while π_1 is used to construct the AP-side verifier $V = \pi_1\mathsf{H2C}(\pi_0|\mathsf{SSID})$. After initialization, the user only remembers pw, whereas the AP stores $(\pi_0, V, \mathsf{SSID}, s)$. This separates the STA-side password secret from the AP-side stored credential and avoids storing the plaintext password at the AP.

SAE-compatible password-element construction. To preserve the SAE-style commit exchange, AsymSAE still derives a password element PE and uses scalar-and-element commitments. Specifically, the protocol computes $val = \mathsf{HMAC}(0^L, \max(\mathsf{MAC_C}, \mathsf{MAC_S})|\min(\mathsf{MAC_C}, \mathsf{MAC_S}))$, maps it into $\mathbb{Z}_p^*$, and derives $\mathsf{PE} = val \cdot \mathsf{H2C}(\pi_0|\mathsf{SSID})$. The ordered address input ensures that both parties derive the same value independently of message direction, while binding the password element to the specific STA–AP link.

KCI-resistant asymmetric key computation. The core change of AsymSAE lies in the asymmetric computation of the shared group element. After exchanging commitments (E_1, s_1) and (E_2, s_2), both parties compute the transcript-dependent challenge $c = \mathcal{H}(E_1, s_1, E_2, s_2, \pi_0)$. The STA computes $K_C = (r_1 + c \cdot \pi_1)(E_2 + s_2\mathsf{PE})$, whereas the AP computes $K_S = r_2(E_1 + s_1\mathsf{PE} + c \cdot val \cdot V)$. More importantly, the verifier term V is bound to the AP's fresh randomness r_2 and to the transcript-dependent challenge c. Thus, even if the AP-side verifier is leaked, an adversary cannot directly transform it into a valid STA impersonation credential without predicting or controlling the challenge-dependent key computation. This design preserves the original SAE commit structure without introducing additional communication rounds.

C. Correctness

For two honest parties executing the protocol, both parties derive the same password element $\mathsf{PE} = val \cdot \mathsf{H2C}(\pi_0|\mathsf{SSID})$ and the same challenge $c = \mathcal{H}(E_1, s_1, E_2, s_2, \pi_0)$. Since $E_2 + s_2\mathsf{PE} = -m_2\mathsf{PE} + (r_2 + m_2)\mathsf{PE} = r_2\mathsf{PE}$ and $E_1 + s_1\mathsf{PE} = -m_1\mathsf{PE} + (r_1 + m_1)\mathsf{PE} = r_1\mathsf{PE}$, the STA computes $K_C = (r_1 + c\pi_1)r_2\mathsf{PE}$. On the AP side, since $V = \pi_1\mathsf{H2C}(\pi_0|\mathsf{SSID})$ and $\mathsf{PE} = val \cdot \mathsf{H2C}(\pi_0|\mathsf{SSID})$, we obtain $val \cdot V = \pi_1\mathsf{PE}$. Thus, the AP computes $K_S = r_2(E_1 + s_1\mathsf{PE} + c \cdot val \cdot V) = r_2(r_1\mathsf{PE} + c\pi_1\mathsf{PE}) = r_2(r_1 + c\pi_1)\mathsf{PE}$. It follows that $K_C = K_S$. Consequently, both parties derive the same $seed$, the same key-confirmation key kck, and the same pairwise master key pmk. Therefore, if the protocol execution is completed honestly and all validation checks pass, the STA and the AP output identical keying material.

Public parameters: Elliptic curve additive group $\mathbb{G}$ with prime order p, $\mathcal{H}, \mathcal{H}_0, \mathsf{H2C}, \mathsf{HMAC}, \mathsf{KDF}$

	Client Station STA (pw)		*Access Point* AP ($\pi_0, V, \mathsf{SSID}, s$)
	Initialization Phase		
1:			$(\pi_0, \pi_1) = \mathcal{H}_0(pw, \mathsf{SSID}, s)$
2:			$V = \pi_1 \mathsf{H2C}(\pi_0 \vert \mathsf{SSID})$
3:	User memorizes pw.		AP stores $(\pi_0, V, \mathsf{SSID}, s)$.
	Active Scanning Starts		
1:	STA sends a probe request.	$\xrightarrow{\mathsf{MAC}_C}$	
2:		$\xleftarrow{\mathsf{MAC}_S, \mathsf{SSID}, s}$	AP replies a probe response.
	SAE Commit Exchange Phase		
3:	User inputs pw.		
4:	$(\pi_0, \pi_1) = \mathcal{H}_0(pw, \mathsf{SSID}, s)$		
5:	$val = \mathsf{HMAC}(0^L, \max(\mathsf{MAC}_C, \mathsf{MAC}_S) \vert \min(\mathsf{MAC}_C, \mathsf{MAC}_S))$		
6:	$val = val \bmod (p-1) + 1$		
7:	$\mathsf{PE} = val \cdot \mathsf{H2C}(\pi_0 \vert \mathsf{SSID})$		
8:	$r_1, m_1 \leftarrow_R \mathbb{Z}_p^*$		$r_2, m_2 \leftarrow_R \mathbb{Z}_p^*$
9:	$s_1 = r_1 + m_1 \bmod p$		$s_2 = r_2 + m_2 \bmod p$
10:	$E_1 = -m_1 \mathsf{PE}$	$\xrightarrow{\mathsf{MAC}_C, E_1, s_1}$	Abort if$(s_1 \notin \mathbb{Z}_p \backslash \{0,1\} \vee E_1 \notin \mathbb{G} \backslash \{\mathcal{O}\})$
11:			$val = \mathsf{HMAC}(0^L, \max(\mathsf{MAC}_C, \mathsf{MAC}_S) \vert \min(\mathsf{MAC}_C, \mathsf{MAC}_S))$
12:			$val = val \bmod (p-1) + 1$
13:			$\mathsf{PE} = val \cdot \mathsf{H2C}(\pi_0 \vert \mathsf{SSID})$
14:	Drop and wait if$(E_2 = E_1 \vee s_2 = s_1)$	$\xleftarrow{E_2, s_2}$	$E_2 = -m_2 \mathsf{PE}$
15	Abort if$(s_2 \notin \mathbb{Z}_p \backslash \{0,1\} \vee E_2 \notin \mathbb{G} \backslash \{\mathcal{O}\})$		
16	$c = \mathcal{H}(E_1, s_1, E_2, s_2, \pi_0)$		$c = \mathcal{H}(E_1, s_1, E_2, s_2, \pi_0)$
17	$K_C = (r_1 + c \cdot \pi_1)(E_2 + s_2 \mathsf{PE})$ and abort if $K_C = \mathcal{O}$		$K_S = r_2(E_1 + s_1 \mathsf{PE} + c \cdot val \cdot V)$ and abort if $K_S = \mathcal{O}$
	SAE Key-confirmation Phase		
18	$seed_C = \mathsf{HMAC}(0^L, [K_C]_x)$		$seed_S = \mathsf{HMAC}(0^L, [K_S]_x)$
19	$kck_c \vert pmk_c = \mathsf{KDF}(seed_C, s_1, L+256)$		$kck_s \vert pmk_s = \mathsf{KDF}(seed_S, s_1, L+256)$
20	$Con_C = \mathsf{HMAC}(kck_c, s_1 \vert E_1 \vert s_2 \vert E_2)$	$\xrightarrow{Con_C}$	$Con_S = \mathsf{HMAC}(kck_s, s_2 \vert E_2 \vert s_1 \vert E_1)$
21	Drop and wait if$(Con_S \neq \mathsf{HMAC}(kck_c, s_2 \vert E_2 \vert s_1 \vert E_1))$	$\xleftarrow{Con_S}$	Drop and wait if$(Con_C \neq \mathsf{HMAC}(kck_s, s_1 \vert E_1 \vert s_2 \vert E_2))$
22	Accept and set $key \leftarrow pmk_c$		Accept and set $key \leftarrow pmk_s$
23	$pid \leftarrow \mathsf{MAC}_S, sid \leftarrow \{(s_1, E_1), (s_2, E_2)\}$		$pid \leftarrow \mathsf{MAC}_C, sid \leftarrow \{(s_1, E_1), (s_2, E_2)\}$
24	Terminate		Terminate

Fig. 4. Protocol flow of the proposed asymmetric SAE in WPA3-Personal networks. The client station STA retains the password pw, while the access point AP stores the derived credential $(\pi_0, V, \mathsf{SSID}, s)$. The protocol retains the SAE commit-and-confirm paradigm defined in IEEE Std 802.11-2024 while enabling asymmetric password storage.

VI. Security Analysis

This section evaluates the security of AsymSAE through three complementary analyses: a game-based formal proof, heuristic attack-oriented analysis, and ProVerif-based symbolic verification, which respectively provide theoretical guarantees, attack-oriented reasoning, and automated verification evidence for the security of the proposed protocol.

A. Formal Security Proof

Theorem 1 (AKE security): Let Π denote the AsymSAE protocol and $\mathcal{D}$ be a password dictionary. Assume that: 1) $\mathcal{H}$, $\mathcal{H}_0$, H2C, HMAC, and KDF are modeled as idealized random oracles H, H_0, H_2, H_3, H_4, respectively; 2) H2C outputs only valid non-identity points in the prime-order subgroup $\mathbb{G}$; 3) All received scalars and group elements are validated exactly as in Fig. 4; 4) The CDH, GDH, and GSDH assumptions hold in $\mathbb{G}$. Let $p = |\mathbb{G}|$ and $L = |Con_C|$. Then for every PPT adversary $\mathcal{A}$ that issues at most q_{se} Send-queries, q_e Execute-queries, and $q_{\mathcal{H}}$, $q_{\mathcal{H}_0}$, q_{H2C}, q_{HMAC}, q_{KDF} random oracle queries, there exist PPT adversaries $\mathcal{B}_{\mathsf{CDH}}$, $\mathcal{B}_{\mathsf{GDH}}$, $\mathcal{B}_{\mathsf{GSDH}}$, such that

$$\begin{aligned}\mathsf{Adv}^{\mathsf{AKE}}_{\Pi,\mathcal{D}}(\mathcal{A}) \leqslant\ & O\left(\frac{q_{se}+q_e+q_1}{p}\right) + O\left(\frac{q_{se}+q_e+q_2}{2^L}\right) \\ & + 4q_e(q_{se}+q_{\mathsf{HMAC}}) \cdot \mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}}(\mathcal{B}_{\mathsf{CDH}}) + \frac{2q_e(q_{se}+q_{\mathsf{KDF}})}{2^L} \\ & + 2q_{se}^2 \cdot \mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(\mathcal{B}_{\mathsf{GDH}}) + \frac{2(q_{se}+q_{\mathsf{KDF}})}{2^L} + \frac{2(q_{se}+q_{\mathsf{HMAC}})}{2^L} \\ & + \frac{2q_{se}}{2^L} + \frac{2q_{se}}{2^{H_\infty(\mathcal{D})}} + \frac{2q_{\mathcal{H}_0}}{2^{H_\infty(\mathcal{D})}} + \frac{2q_{se}}{p-1} \\ & + 2q_{se}q_{\mathsf{HMAC}} \cdot \mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(\mathcal{B}_{\mathsf{GDH}}) + 2q_{se}q_{\mathcal{H}_0}^2 q_{\mathsf{HMAC}} \cdot \mathsf{Adv}^{\mathsf{GSDH}}_{\mathbb{G}}(\mathcal{B}_{\mathsf{GSDH}}),\end{aligned}$$

where $q_1 = q_{\mathcal{H}} + q_{\mathcal{H}0} + q_{\mathsf{H2C}}$ and $q_2 = q_{\mathsf{HMAC}} + q_{\mathsf{KDF}}$.

Proof: We prove the theorem through a sequence of games, beginning with the real attack and ending with a game where the adversary's advantage is 0. Let $\mathsf{S_i}$ denote the event that the adversary correctly guesses the hidden bit b' of the Test oracle in Game G_i.

Game G_0. This game corresponds to the real attack scenario. All oracles behave exactly as specified by the protocol and the v-sPAKE model. By definition,

$$\mathsf{Adv}^{\mathsf{AKE}}_{\Pi,\mathcal{D}}(\mathcal{A}) = 2 \cdot \Pr[\mathsf{S_0}] - 1.$$

Game G_1. This game is identical to game G_0, except that the simulator explicitly simulates the random oracles by lazy sampling. For every random oracle, the simulator maintains lists $L_{\mathcal{H}}$, $L_{\mathcal{H}_0}$, L_{H2C}, L_{HMAC}, L_{KDF} to record all queries and responses during the simulation. Specifically, for each query to the random oracles H, H_0, H_3, and H_4, the simulator returns the stored response if the query has already been made; otherwise, it samples a random value output from the corresponding range, stores the query-response pair, and returns the output. On a query $H_2(x)$, if x already appears in the list L_{H2C}, the same group element is returned. Otherwise, the simulator samples $\alpha \leftarrow_R \mathbb{Z}_p^*$, returns αG, and stores $(x, \alpha, \alpha G)$. Because the simulation reproduces the oracle computations in a fully equivalent manner, we can obtain $\Pr[\mathsf{S_0}] = \Pr[\mathsf{S_1}]$.

Game G_2. In G_2, the simulator aborts if any of the following bad events occurs: 1) $s_i \in \{0, 1\}$ or $E_i = \mathcal{O}$; 2) two distinct random-oracle queries collide and yield the same $\pi_0, \pi_1, val, c, \mathsf{PE}, seed, kck, pmk, Con_C$, and Con_S; 3) two independent sessions produce the same commit pair (s_1, E_1) or (s_2, E_2). By the birthday bound,

$$|\Pr[\mathsf{S_2}] - \Pr[\mathsf{S_1}]| \leqslant O(\frac{q_{se} + q_e + q_1}{p}) + O(\frac{q_{se} + q_e + q_2}{2^L}),$$

Game G_3. We first handle Execute sessions. This game is identical to game G_2, except that the simulator replaces $H_3(0^L, [K]_x)$ with a secret random oracle $H'_3(0^L, sid)$ for all sessions generated via Execute queries. The adversary gains no distinguishing advantage from this modification unless AskH3-Execute occurs, meaning that H_3 is queried on a value $[K]_x$. Consequently, the games G_3 and G_2 are perfectly indistinguishable unless the event AskH3-Execute occurs:

$$|\Pr[\mathsf{S_3}] - \Pr[\mathsf{S_2}]| \leqslant \Pr[\mathsf{AskH3\text{-}Execute}].$$

In G_3, $\mathcal{B}_{\mathsf{CDH}}$ embeds the challenge values X and Y of a CDH instance $(G, X = xG, Y = yG)$ into the protocol messages E_1 and E_2, respectively. Specifically, it sets $\mathsf{PE} = val \cdot \alpha G, E_1 = -m_1\mathsf{PE} + X, E_2 = -m_2\mathsf{PE} + Y$. In this simulation, once the event AskH3-Execute occurs, the CDH value xyG can be extracted from the following computation: $K = \mathsf{CDH}_{\mathsf{PE}}(E_2 + s_2\mathsf{PE}, E_1 + s_1\mathsf{PE} + c \cdot val \cdot V) = r_2(r_1 + c\pi_1)\mathsf{PE} + r_2X + (r_1 + c\pi_1)Y + \frac{xyG}{val \cdot \alpha}$. Therefore,

$$\Pr[\mathsf{AskH3\text{-}Execute}] \leqslant 2q_e(q_{se} + q_{\mathsf{HMAC}}) \cdot \mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G}}(\mathcal{B}_{\mathsf{CDH}}).$$

Game G_4. This game is identical to G_3, except that $H_4(seed, s_1, L+256)$ is replaced with a secret random oracle $H'_4(sid, s_1, L + 256)$ for all Execute sessions. With this modification, G_3 and G_4 can be distinguished only if the adversary explicitly queries H_4 on $seed$, causing $H_4(seed)$ to differ from $H'_4(seed)$, where $H'_4(seed)$ is returned as the answer to a Reveal query. Since, in G_4, $seed$ is derived from the secret random oracle H'_3 and collisions have already been ruled out, it follows that:

$$|\Pr[\mathsf{S_4}] - \Pr[\mathsf{S_3}]| \leqslant q_e(q_{se} + q_{\mathsf{KDF}})/2^L.$$

Game G_5. We handle the paired honest send sessions corresponding to $\mathsf{Send}(S, j, (\mathsf{MAC}_C, s_1, E_1))$ and $\mathsf{Send}(C, i, (s_2, E_2))$. This game is identical to G_4, except that $H_3(0^L, [K_C]_x)$ is replaced with a secret random oracle $H'_3(0^L, [K_C]_x)$. The adversary gains no distinguishing advantage from this modification unless AskH5-Send occurs, meaning that the adversary or simulator queries $H_3(0^L, [K_C]_x)$ of a Send session. Hence,

$$|\Pr[\mathsf{S_5}] - \Pr[\mathsf{S_4}]| \leqslant \Pr[\mathsf{AskH5\text{-}Send}].$$

We construct a reduction algorithm $\mathcal{B}_{\mathsf{GDH}}$ that solves the GDH problem in the group $\mathbb{G}$ by interacting with an adversary $\mathcal{A}$. $\mathcal{B}_{\mathsf{GDH}}$ embeds the challenge values X and Y into the protocol messages E_1 and E_2, respectively. Specifically, the random indices $t_1, t_2 \in \{1, \ldots, q_{se}\}$ denote the two guessed target Send-query positions. If the query at position t_1 is a STA query, the simulator sets $E_1 = -m_1\mathsf{PE} + X$; if it is a AP query, it sets $E_2 = -m_2\mathsf{PE} + X$, and similarly embeds Y at position t_2. Therefore, the computation of K_C in these paired sessions contains the GDH value $\mathsf{DH}(X, Y)$. Any query to $H_3(0^L, [K_C]_x)$ for such a Send session thus reveals the value needed by $\mathcal{B}_{\mathsf{GDH}}$ to solve the GDH instance. Since the reduction guesses the two relevant Send-query positions among at most q_{se} queries, its guesses hit the two positions involved in AskH5-Send with probability at least $1/q_{se}^2$. It follows that

$$\Pr[\mathsf{AskH5\text{-}Send}] \leq q_{se}^2 \cdot \mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(\mathcal{B}_{\mathsf{GDH}}).$$

Game G_6. This game is identical to G_5, except that $H_4(seed, s_1, L+256)$ is replaced with a secret random oracle $H'_4(seed, s_1, L+256)$ for all Send sessions. This modification can affect the adversary's view only when the correct $seed$ is submitted to the H_4 oracle by either the adversary or the simulator. Since $seed$ is sampled uniformly via the secret random oracle H'_3, we have

$$|\Pr[\mathsf{S_6}] - \Pr[\mathsf{S_5}]| \leqslant (q_{se} + q_{\mathsf{KDF}})/2^L.$$

Game G_7. This game is identical to G_6, except that in every session established through a Send query, the random oracle H_3 used for key confirmation is replaced by a secret random oracle H''_3; for instance, $H_3(kck_c, s_1|E_1|s_2|E_2)$ is replaced with $H''_3(kck_c, s_1|E_1|s_2|E_2)$. This change is detectable only if the adversary directly queries H_3 on the correct kck. Since kck is derived from H'_4, it follows that:

$$|\Pr[\mathsf{S_7}] - \Pr[\mathsf{S_6}]| \leqslant (q_{se} + q_{\mathsf{HMAC}})/2^L.$$

Game G_8. We next handle active sessions in which the adversary injects a forged commit message. We consider two cases: the malicious-AP case and the malicious-STA case. For the adversary to compute the correct confirmation tag, it must

either: 1. guess the password online; 2. guess the challenge c; 3. query H_3 at the correct $[K]_x$; 4. guess the confirmation tag; 5. guess the password offline. Let $\mathsf{OnlineGuess}$ denote the event that the adversary succeeds in an active online password-guessing attempt against an uncorrupted password, let AskK be the event that the adversary queries H_3 on $[K]_x$ for some fresh session, let BadC denote the event that the adversary predicts the challenge c before fixing the corresponding forged commit, let $\mathsf{OffVGuess}$ be the event that, after obtaining an AP verifier file, the adversary queries $\mathcal{H}_0(pw', SSID, s)$ for the correct password $pw' = pw$, or for a value that produces the same verifier file, and let $\mathsf{ForgeConf}$ denote a forgery of an L-bit confirmation value without querying the corresponding oracle. Then G_7 and G_8 are indistinguishable unless one of these events occurs, and therefore

$$|\Pr[S_8] - \Pr[S_7]| \leq \Pr[\mathsf{CollH8}],$$

where $\mathsf{CollH8} = \mathsf{AskK} \vee \mathsf{BadC} \vee \mathsf{ForgeConf} \vee \mathsf{OnlineGuess} \vee \mathsf{OffVGuess}$, $\Pr[\mathsf{ForgeConf}] \leq q_{se}/2^L$, $\Pr[\mathsf{OnlineGuess}] \leq q_{se}/2^{H_\infty(\mathcal{D})}$, $\Pr[\mathsf{OffVGuess}] \leq q_{\mathcal{H}_0}/2^{H_\infty(\mathcal{D})}$, and $\Pr[\mathsf{BadC}] \leq q_{se}/(p-1)$.

We now bound the event AskK by considering the following two cases. First, consider the malicious-STA case, where the adversary tries to make an honest AP accept. This case includes the AP-side KCI setting, where the adversary may know the verifier file $(\pi_0, V, \mathsf{SSID}, s)$, but $\mathsf{OffVGuess}$ has not occurred. Let $R = E_2 + s_2\mathsf{PE} = r_2\mathsf{PE}$, $Q = E_1 + s_1\mathsf{PE} + c \cdot val \cdot V$. The AP-side key is $K = r_2 Q = \mathsf{DH}_{\mathsf{PE}}(R, Q)$. If the adversary knows a scalar μ such that $Q = \mu\mathsf{PE}$, then it can compute $K = \mu R$. However, since $val \cdot V = \pi_1\mathsf{PE}$ and π_1 is unknown under $\neg\mathsf{OffVGuess}$, the adversary can force Q to be a known multiple of PE only by canceling the verifier term, which requires predicting c before the forged E_1, s_1 is fixed. This is exactly the event BadC. Conditioned on $\neg\mathsf{BadC}$, Q is not known as a scalar multiple of PE, and any query to the seed oracle on K yields a solution to a GDH instance with base PE, inputs R and Q. Therefore,

$$\Pr[\mathsf{AskK}_{STA}] \leq q_{se} \cdot q_{\mathsf{HMAC}} \cdot \mathsf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(\mathcal{B}_{\mathsf{GDH}}).$$

Second, consider the malicious-AP case, where the adversary tries to make an honest STA accept. In this case, freshness excludes $\mathsf{VCorrupt}(S)$ together with a forged message to the tested STA, and the adversary does not know the password-derived scalar π_1. For a candidate password pw_ℓ, define $\pi_{0,\ell}, \pi_{1,\ell} = \mathcal{H}_0(pw_\ell, \mathsf{SSID}, s)$, $\mathsf{PE}_\ell = val \cdot \mathsf{H2C}(\pi_{0,\ell}|\mathsf{SSID})$, $c_\ell = \mathcal{H}(E_1, s_1, E_2, s_2, \pi_{0,\ell})$, and $\alpha_\ell = s_1 + c_\ell \pi_{1,\ell}$. The candidate key is $K_\ell = DH_{\mathsf{PE}_\ell}(E_1 + \alpha_\ell \mathsf{PE}_\ell, E_2 + s_2\mathsf{PE}_\ell)$. If the adversary obtains the correct seed without an online password guess, then it must have queried the seed oracle using a key value derived from some entry in the $\mathcal{H}_0$-list. If this entry is not the real password entry, then the two distinct H_0-entries are both consistent with the same active transcript, which yields a solution to the GSDH problem. The reduction $\mathcal{B}_{\mathsf{GSDH}}$ guesses the two $\mathcal{H}_0$-entries corresponding to pw_1, pw_2, programs $\mathsf{H2C}(\pi_{0,1}|\mathsf{SSID}) = A = aG$, $\mathsf{H2C}(\pi_{0,2}|\mathsf{SSID}) = B = bG$, and obtains two corresponding seed-oracle queries on K_1, K_2. Let the adversarially supplied non-identity element be denoted by X, and let the honest-side element be programmed as $Y = yG$ for known $y \in \mathbb{Z}_p^*$. For $\ell = 1$, we have $K_1 = \alpha_1 Y + \alpha_1 s_2 val \cdot A + s_2 X + \frac{y}{val \cdot a} X$, and hence $a^{-1}X = (K_1 - \alpha_1 Y - \alpha_1 s_2 val \cdot A - s_2 X) \cdot val \cdot y^{-1}$. Similarly, from the second password-induced query, $b^{-1}X = (K_2 - \alpha_2 Y - \alpha_2 s_2 val \cdot B - s_2 X) \cdot val \cdot y^{-1}$. Since $X \neq O$ by the validation checks, the tuple $(X, a^{-1}X, b^{-1}X)$ is a valid solution to the GSDH instance, we obtain

$$\Pr[\mathsf{AskK}_{AP}] \leq q_{se} \cdot q^2_{\mathcal{H}_0} \cdot q_{\mathsf{HMAC}} \cdot \mathsf{Adv}^{\mathsf{GSDH}}_{\mathbb{G}}(\mathcal{B}_{\mathsf{GSDH}}).$$

$\mathsf{Game}\ G_9$. This game is identical to G_8, except that it aborts whenever any bad event defined in G_2–G_8 occurs. In this game, the session key pmk of the fresh Test session is uniform and independent of the adversary's view. Therefore, replacing the real test key by an independent uniform string is perfectly indistinguishable, and the hidden bit in the Test query is information theoretically hidden. Consequently, $\Pr[S_9] = 1/2$.

In summary, we can obtain that the advantage $\mathsf{Adv}^{\mathsf{AKE}}_{\Pi,\mathcal{D}}(\mathcal{A})$ is negligible. This completes the proof of Theorem 1. ∎

B. Heuristic Security Analysis

1) Forward Secrecy: In the protocol shown in Fig. 4, the STA and the AP choose fresh ephemeral randomness r_1, m_1 and r_2, m_2, respectively, and use them to generate the corresponding commit messages (E_1, s_1) and (E_2, s_2). For an honest execution, we have $E_2 + s_2\mathsf{PE} = r_2\mathsf{PE}$ and $E_1 + s_1\mathsf{PE} = r_1\mathsf{PE}$. Thus, the core keying material computed by the STA and the AP contains $r_1(E_2 + s_2\mathsf{PE}) = r_1 r_2 \mathsf{PE}$ and $r_2(E_1 + s_1\mathsf{PE}) = r_2 r_1 \mathsf{PE}$. Therefore, the session key depends on the Diffie-Hellman shared value $r_1 r_2 \mathsf{PE}$, which is jointly contributed by both parties' ephemeral randomness, rather than solely on the long-term password pw or the AP-side verifier (π_0, V). Even if long-term secret values are compromised after the session, an adversary can only derive $r_1\mathsf{PE}$ and $r_2\mathsf{PE}$ from the transcript, but cannot compute $r_1 r_2 \mathsf{PE}$ once r_1, r_2, m_1, m_2 have been erased, unless it solves the elliptic curve discrete logarithm problem (ECDLP) or the CDH problem. Hence, the AsymSAE protocol supports forward secrecy.

2) Resistance to Offline Dictionary Attacks: A passive adversary may obtain public values MAC_C, MAC_S, SSID, s, the commit messages (E_1, s_1), (E_2, s_2), and the confirmation messages Con_C,Con_S. For any password candidate pw', the adversary can compute $(\pi'_0, \pi'_1) = \mathcal{H}_0(pw', \mathsf{SSID}, s)$ and $\mathsf{PE}' = val \cdot \mathsf{H2C}(\pi'_0|\mathsf{SSID})$. It may then attempt to evaluate $E_1 + s_1\mathsf{PE}'$ and $E_2 + s_2\mathsf{PE}'$. However, if $pw' \neq pw$, then $\mathsf{PE} \neq \mathsf{PE}'$, and the resulting values $r_1\mathsf{PE}'$ and $r_2\mathsf{PE}'$ still appear as random group elements. They do not provide an effective test for determining whether the password guess is correct. Moreover, the confirmation messages Con_C and Con_S are derived from the session key. Since the adversary does not know r_1 or r_2, it cannot derive the correct session key. As a result, the confirmation messages cannot be used as an offline password-verification oracle. Under the security assumptions of the hash function, HMAC, and KDF, the protocol resists transcript-based offline dictionary attacks.

TABLE I
PROVERIF VERIFICATION QUERIES AND RESULTS

ProVerif query	Description	Verification result
`Weak secret pw`	Checks resistance to offline password guessing. A true result means the adversary cannot validate password guesses from public messages.	`True`
`Query not attacker(testSecret[])`	Checks session-key confidentiality. A true result means the adversary cannot obtain the secret protected by the established session key.	`True`
`inj-event(c_accept())==> inj-event(s_accept())`	Checks AP-to-STA injective authentication. Each STA acceptance must correspond to a unique matching AP acceptance.	`True`
`inj-event(s_accept())==> inj-event(c_running())`	Checks STA-to-AP injective authentication. Each AP acceptance must correspond to a unique honest STA execution.	`True`
`Query not attacker_p1(fsSecret[])`	Checks forward secrecy after long-term secret leakage. A true result means past session secrets remain protected.	`True`
`inj-event(s_accept())==> inj-event(c_running())`	Checks AP-side KCI resistance under AP credential leakage. The adversary cannot make the AP accept without an honest STA run.	`True`

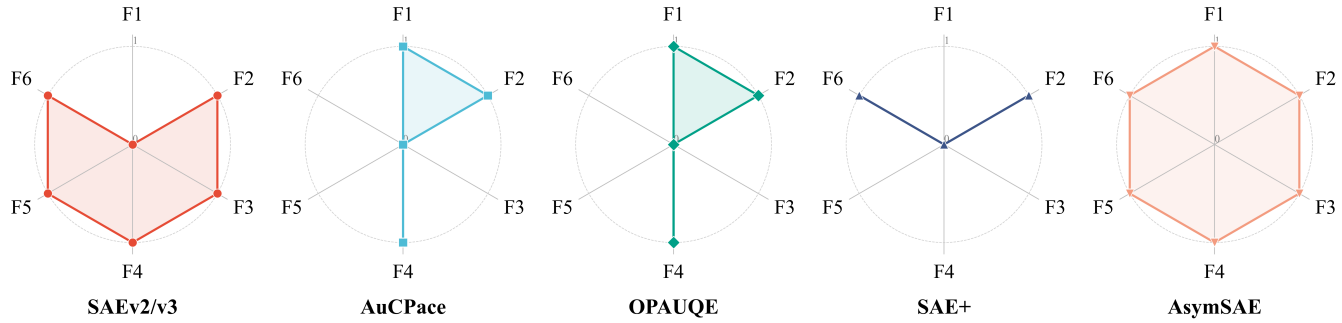


Fig. 5. Comparison of protocol properties and features. F1: resistance to KCI attacks; F2: forward secrecy; F3: communication privacy; F4: resistance to transcript-based offline dictionary attacks; F5: support for personal Wi-Fi networks; and F6: simultaneous two-way messaging. 0 indicates that the property is not satisfied, whereas 1 indicates that it is satisfied.

3) Resistance to KCI Attacks: Consider an adversary who compromises the AP-side verifier state $(\pi_0, V, \mathsf{SSID}, s)$, where $V = \pi_1 \mathsf{H2C}(\pi_0|\mathsf{SSID})$. Given the public MAC addresses, the adversary can compute $\mathsf{PE} = val \cdot \mathsf{H2C}(\pi_0|\mathsf{SSID})$ and $val \cdot V = \pi_1 \mathsf{PE}$, but cannot derive the scalar π_1 from V, assuming the hardness of the discrete-logarithm problem and no successful offline recovery of pw. To impersonate a legitimate STA, the adversary must make the AP derive a session key that it can also compute. Since the AP computes $K_S = r_2(E_1 + s_1\mathsf{PE} + c \cdot val \cdot V)$, where $c = \mathcal{H}(E_1, s_1, E_2, s_2, \pi_0)$, the adversary would try to force $E_1 + s_1\mathsf{PE} + c \cdot val \cdot V = \alpha\mathsf{PE}$ for some known α, which requires setting $E_1 = (\alpha - s_1)\mathsf{PE} - c \cdot val \cdot V$. However, in a normal execution, (E_1, s_1) must be fixed before the AP generates its fresh commit (E_2, s_2); hence the adversary would have to predict both (E_2, s_2) and the corresponding challenge c. In the random oracle model, each such attempt succeeds with probability at most $1/(p-1)^3$, and if the adversary launches at most q_{send} impersonation sessions, the total success probability is bounded by $q_{\mathsf{send}}/(p-1)^3$, which is negligible. Even in the stronger setting where (E_2, s_2) is known before choosing (E_1, s_1), the adversary still has to solve the fixed-point constraint induced by $c = \mathcal{H}(E_1, s_1, E_2, s_2, \pi_0)$; it can only guess $\hat{c}$, set $E_1 = (\alpha - s_1)\mathsf{PE} - \hat{c} \cdot val \cdot V$, and succeed only if $\hat{c} = c$, which occurs with negligible probability at most $q_{\mathsf{send}}/(p-1)$. Thus, the protocol resists KCI attacks in the AP-verifier compromise setting.

C. Formal Verification with ProVerif

We formally verified the proposed protocol using ProVerif under the Dolev–Yao symbolic adversary model, where the adversary fully controls the public channel and can eavesdrop, modify, replay, block, and forge messages. The cryptographic primitives, including hash functions, HMAC, KDF, H2C, and elliptic-curve operations, are modeled as ideal primitives. To distinguish different leakage assumptions, we built three models: a normal attack model, a post-compromise long-term secret leakage model, and an AP-side credential leakage model. Under the normal attack model, the results `Weak secret pw is true` and `Query not attacker(testSecret[]) is true` show resistance to offline password guessing and session-key confidentiality, respectively. The injective correspondences `c_accept()==>s_accept()` and `s_accept()==>c_running()` both hold, confirming that the STA and AP agree on the identities, protocol context, and session key. Under the post-compromise model, ProVerif's phase mechanism is used to leak long-term secrets after session establishment, and the result `Query not attacker_p1(fsSecret[]) is true` confirms forward secrecy. Under the AP-side credential leakage model, the AP's stored long-term credentials are released without revealing the user's password, and the verification result shows that the adversary cannot make the

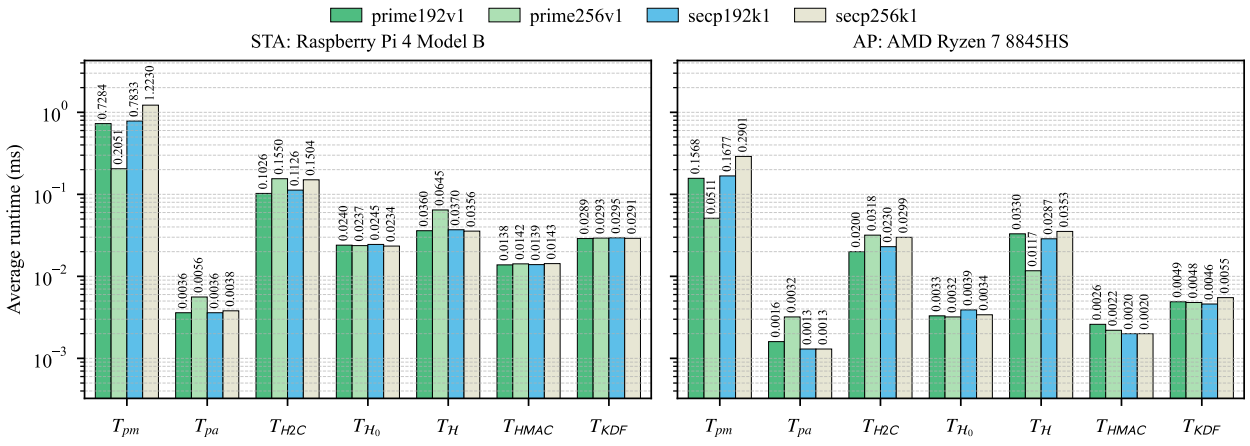


Fig. 6. Average runtime comparison of basic cryptographic operations on the STA and AP platforms under different elliptic curves. T_{pm}: Point multiplication; T_{pa}: Point addition; T_{H2C}: Hash function H2C; $T_{\mathcal{H}_0}$: Hash function $\mathcal{H}_0$; $T_{\mathcal{H}}$: Hash function $\mathcal{H}$; T_{HMAC}: $\mathsf{HMAC}(kck, s_1|E_1|s_2|E_2)$; T_{KDF}: $\mathsf{KDF}(seed, s_1, L+256)$.

AP accept without a genuine STA execution, confirming AP-side KCI resistance. In summary, as shown in Table I, the ProVerif results demonstrate that the proposed protocol achieves resistance to offline password guessing, session-key confidentiality, injective mutual authentication, forward secrecy, and AP-side KCI resistance in the symbolic model. This verification focuses on the logical security of the protocol under ideal cryptographic assumptions and does not cover implementation-level threats such as side-channel leakage, poor randomness, small-subgroup attacks, or weaknesses in concrete cryptographic implementations.

VII. Performance Evaluation

A. Security and Functionality Comparison

We compare AuCPace [14], OPAQUE [15], SAEv2/v3 [22], SAE+ [16], and the proposed AsymSAE in terms of six properties: resistance to KCI attacks, forward secrecy, communication privacy, resistance to transcript-based offline dictionary attacks, support for personal Wi-Fi networks, and simultaneous two-way messaging. As shown in Fig. 5, SAEv2/v3 cannot resist KCI attacks due to its symmetric password-based structure. AuCPace provides forward secrecy and resistance to offline dictionary attacks, but it is not designed for STA–AP wireless access scenarios and therefore does not directly support personal Wi-Fi networks, communication privacy, or SAE-style simultaneous two-way messaging. OPAQUE offers resistance to KCI attacks, forward secrecy, and resistance to offline dictionary attacks through its asymmetric verifier-based PAKE design. However, it is not tailored to WPA3-Personal or the STA–AP architecture, and thus does not directly support personal Wi-Fi access procedures or SAE-style bidirectional message exchange. SAE+ supports forward secrecy and simultaneous two-way messaging, but it does not simultaneously provide KCI resistance, communication privacy, resistance to offline dictionary attacks, and support for personal Wi-Fi networks. In contrast, AsymSAE satisfies all six properties. It achieves forward secrecy using fresh ephemeral randomness, improves resistance to KCI and offline dictionary attacks through an asymmetric verifier-based structure, and preserves the SAE-style interaction suitable for STA–AP authentication. For communication privacy, AsymSAE is compatible with the privacy mechanism of IEEE Std 802.11-2024, where STAs may use locally generated random addresses instead of permanent MAC addresses. Since the link-specific value is derived from the exchanged STA and AP addresses, AsymSAE can operate with randomized STA addresses without exposing the permanent device identity. Therefore, AsymSAE provides a more balanced solution in terms of security, privacy, and practicality for personal Wi-Fi networks.

B. Computation Overhead

To evaluate the protocol's performance more comprehensively in practical environments, we used a Raspberry Pi 4 Model B running Ubuntu 20.04.6 with an ARM Cortex-A72 1.5 GHz CPU and 4 GB of LPDDR4 RAM as the client, and a virtual machine running Ubuntu 18.04.6 with an AMD Ryzen 7 8845HS CPU and 12 GB of RAM as the server. In addition, we implemented the AsymSAE protocol by using Python language and Charm-Crypto library 0.62 version. Meanwhile, to evaluate the protocol performance under different security levels, we ran the protocol 1000 times on four elliptic curves, namely, prime192v1: NIST/X9.62/SECG curve over a 192-bit prime field, prime256v1: X9.62/SECG curve over a 256-bit prime field, secp192k1: SECG curve over a 192-bit prime field, and secp256k1: SECG curve over a 256-bit prime field, and used the time.perf_counter() function in the time library to measure the average runtime of each operation. The experimental results are shown in Fig. 6.

Finally, we compared AsymSAE with existing state-of-the-art protocols. Note that the computation overhead of SAE+ reported in the table follows the original analysis, whereas SAE+ requires four point multiplications when instantiated under IEEE Std 802.11–2024. Against this baseline, AsymSAE upgrades implicit authentication to explicit authentication and additionally resists KCI attacks and offline dictionary attacks compared with SAE+, while incurring only one extra point multiplication on the server side. Similarly, compared

TABLE II
PERFORMANCE COMPARISON

Protocol	Type	Approx. Computation Overhead		Communication Overhead		Flows
		STA	AP	STA	AP	
AuCPace [14]	Asymmetric	$3T_{pm} + 1T_{\mathsf{H2C}}$	$4T_{pm} + 1T_{\mathsf{H2C}}$	$\mathcal{C}_{Ele} + \mathcal{C}_H$	$2\mathcal{C}_{Ele} + \mathcal{C}_l + \mathcal{C}_H$	6
OPAQUE [15]	Asymmetric	$6T_{pm} + 1T_{\mathsf{H2C}}$	$5T_{pm}$	$2\mathcal{C}_{Ele} + \mathcal{C}_f$	$3\mathcal{C}_{Ele} + \mathcal{C}_f$	3
SAEv2/v3 [22]	Symmetric	$4T_{pm} + 1T_{\mathsf{H2C}}$	$4T_{pm} + 1T_{\mathsf{H2C}}$	$\mathcal{C}_{Sca} + \mathcal{C}_{Ele} + \mathcal{C}_H$	$\mathcal{C}_{Sca} + \mathcal{C}_{Ele} + \mathcal{C}_H$	4
SAE+ [16]	Asymmetric	$3T_{pm} + 1T_{\mathsf{H2C}}$	$3T_{pm} + 1T_{\mathsf{H2C}}$	$\mathcal{C}_{Sca} + \mathcal{C}_{Ele}$	$\mathcal{C}_{Sca} + \mathcal{C}_{Ele}$	2
AsymSAE	Asymmetric	$4T_{pm} + 1T_{\mathsf{H2C}}$	$5T_{pm} + 1T_{\mathsf{H2C}}$	$\mathcal{C}_{Sca} + \mathcal{C}_{Ele} + \mathcal{C}_H$	$\mathcal{C}_{Sca} + \mathcal{C}_{Ele} + \mathcal{C}_H$	4

$\mathcal{C}_{Sca}$: The bit length of the scalar; $\mathcal{C}_{Ele}$: The bit length of the group element; $\mathcal{C}_l$: The bit length of the salt value; $\mathcal{C}_f$: The bit length of the pseudorandom function; $\mathcal{C}_H$: The bit length of the HMAC function;

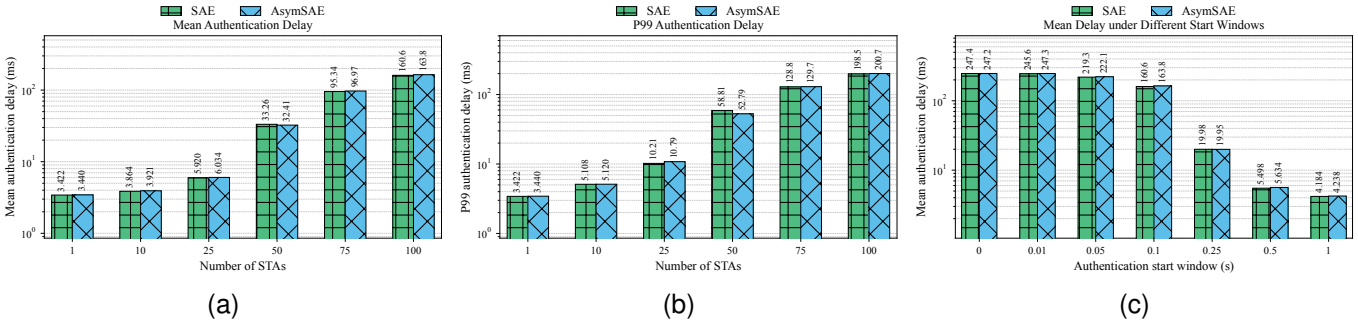


Fig. 7. Authentication delay comparison between SAE and AsymSAE in the ns-3 IEEE 802.11ax simulation. (a) Mean authentication delay under different numbers of STAs. (b) P99 authentication delay under different numbers of STAs. (c) Mean authentication delay under different authentication start windows with 100 STAs.

with SAEv2/v3, AsymSAE provides KCI resistance with only one additional server-side point multiplication and no extra communication cost. Since Moore's law has told us that computational capability grows faster than communication capability, it is therefore a natural and acceptable choice for AsymSAE to trade computational cost for system security without increasing communication overhead.

C. Communication Overhead

Table II compares AsymSAE with AuCPace [14], OPAQUE [15], SAEv2/v3 [22], and SAE+ [16] in terms of communication overhead and protocol flow. For AsymSAE, the communication overhead of the client or the server is $\mathcal{C}_{Sca}+\mathcal{C}_{Ele}+\mathcal{C}_H$. Thus, the total communication overhead is $2(\mathcal{C}_{Sca} + \mathcal{C}_{Ele} + \mathcal{C}_H)$. This is the same as that of SAEv2/v3 [22], which shows that AsymSAE does not introduce additional communication cost despite adopting an asymmetric authentication structure. In comparison, AuCPace requires $\mathcal{C}_{Ele}+\mathcal{C}_H$ on the client side and $2\mathcal{C}_{Ele} + \mathcal{C}_l + \mathcal{C}_H$ on the server side, resulting in a total communication overhead of $3\mathcal{C}_{Ele} + \mathcal{C}_l + 2\mathcal{C}_H$, with 6 flows. OPAQUE requires $2\mathcal{C}_{Ele}+\mathcal{C}_f$ on the client side and $3\mathcal{C}_{Ele}+\mathcal{C}_f$ on the server side, giving a total overhead of $5\mathcal{C}_{Ele} + 2\mathcal{C}_f$. Although OPAQUE uses only 3 flows, it transmits more group elements and pseudorandom-function outputs, leading to a higher overall communication cost. SAE+ requires fewer protocol flows than AsymSAE because it mainly relies on a one-round commit exchange and does not explicitly retain the SAE-style key-confirmation messages. In terms of protocol flow, AsymSAE requires 4 flows, the same as SAEv2/v3, fewer than AuCPace, but more than SAE+ and OPAQUE. This is because AsymSAE preserves the SAE-style commit-and-confirm structure while supporting explicit authentication.

Therefore, AsymSAE maintains the lightweight communication overhead of SAE-like protocols while providing stronger security properties and better suitability for STA–AP wireless access scenarios.

D. Network Simulation

We conducted an ns-3-based performance simulation of SAE [22] and AsymSAE over an IEEE 802.11ax infrastructure network. The authentication exchange is modeled using the Commit/Confirm message workflow, while protocol-specific cryptographic operations are represented by configurable per-stage processing delays measured under the prime256v1 elliptic curve. Specifically, for SAE, the delays of STA Commit generation, AP Commit processing, STA Confirm generation, AP Confirm processing, and STA-side Confirm verification are set to 760 µs, 250 µs, 540 µs, 20 µs, and 50 µs, respectively. For AsymSAE, the corresponding delays are set to 660 µs, 310 µs, 600 µs, 20 µs, and 60 µs. To model AP-side computation contention under concurrent authentication requests, the AP processes authentication jobs through a finite FIFO worker queue. As shown in Fig. 7a and Fig. 7b, SAE and AsymSAE exhibit very similar mean and P99 authentication delays as the number of STAs increases. At 100 STAs, the mean delay is approximately 160 ms for SAE and 164 ms for AsymSAE,

while the P99 delay is approximately 199 ms and 201 ms, respectively. Fig. 7c further evaluates the mean delay under different authentication start windows with 100 STAs.

The results show that increasing the start window significantly reduces the mean delay, indicating that bursty simultaneous authentication requests are the main cause of latency growth. Overall, AsymSAE introduces limited latency overhead compared with SAE under the same IEEE 802.11ax simulation settings. The performance difference between the two protocols is minor, and the dominant factor affecting authentication latency is the concurrency level of STA requests.

VIII. Conclusion

We proposed AsymSAE, a verifier-based asymmetric SAE protocol for personal Wi-Fi networks. AsymSAE strengthens WPA3-Personal authentication while preserving compatibility with the existing IEEE 802.11 SAE workflow. It does not introduce additional communication rounds or increase communication overhead compared with SAEv2/v3, and the additional computation has only limited impact on authentication latency. By allowing the STA to authenticate with the password while the AP stores the password-derived verifier, AsymSAE reduces the security impact of AP-side credential compromise. Our security analysis, formal verification, and performance evaluation show that AsymSAE provides stronger protection against offline dictionary attacks and AP-side KCI attacks while remaining practical for deployment in existing personal Wi-Fi environments. Therefore, AsymSAE is a practical and secure asymmetric SAE protocol for personal Wi-Fi networks.

References

[1] H. A. Omar, K. Abboud, N. Cheng, K. R. Malekshan, A. T. Gamage, and W. Zhuang, "A survey on high efficiency wireless local area networks: Next generation wifi," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2315–2344, 2016.

[2] Wi-Fi Alliance, "Wi-Fi Protected Access: Security Technologies for Wi-Fi Networks," Wi-Fi Alliance Security Highlights, 2021, [Online]. Available: https://www.wifi.org/system/files/Security_Highlights_202101.pdf.

[3] M. Vanhoef and F. Piessens, "Key reinstallation attacks: Forcing nonce reuse in wpa2," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1313–1328.

[4] Wi-Fi Alliance, "WPA3 Specification," Wi-Fi Alliance Specification, Version 3.3, 2024, [Online]. Available: https://www.wi-fi.org/system/files/WPA3%20Specification%20v3.3.pdf.

[5] D. Harkins, "Simultaneous authentication of equals: A secure, password-based key exchange for mesh networks," in *2008 Second International Conference on Sensor Technologies and Applications (sensorcomm 2008)*. IEEE, 2008, pp. 839–844.

[6] "Dragonfly Key Exchange," RFC 7664, Nov. 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7664

[7] "IEEE standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2024 (Revision of IEEE Std 802.11-2020)*, pp. 1–5956, 2025.

[8] J. Ye, X. D. C. De Carnavalet, L. Zhao, M. Zhang, L. Wu, and W. Zhang, "Exposed by default: A security analysis of home router default settings," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 63–79.

[9] X. Zhou, Q. Deng, J. Pu, K. Man, Z. Qian, and S. V. Krishnamurthy, "Untangling the knot: Breaking access control in home wireless mesh networks," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 2072–2086.

[10] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, "A large-scale analysis of the security of embedded firmwares," in *Proceedings of the 23rd USENIX Conference on Security Symposium*, ser. SEC'14. USA: USENIX Association, 2014, p. 95–110.

[11] "Efficient Augmented Password-Only Authentication and Key Exchange for IKEv2," RFC 6628, Jun. 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6628

[12] "Using the Secure Remote Password (SRP) Protocol for TLS Authentication," RFC 5054, Nov. 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc5054

[13] "SPAKE2+, an Augmented Password-Authenticated Key Exchange (PAKE) Protocol," RFC 9383, Sep. 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9383

[14] B. Haase and B. Labrique, "AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 2, p. 1–48, Feb. 2019.

[15] S. Jarecki, H. Krawczyk, and J. Xu, "OPAQUE: An asymmetric pake protocol secure against pre-computation attacks," in *Advances in Cryptology – EUROCRYPT 2018*, J. B. Nielsen and V. Rijmen, Eds. Cham: Springer International Publishing, 2018, pp. 456–486.

[16] M. Qi, W. Hu, and Y. Tai, "SAE+: One-round provably secure asymmetric sae protocol for client-server model," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 3906–3913, 2024.

[17] "The OPAQUE Augmented Password-Authenticated Key Exchange (aPAKE) Protocol," RFC 9807, Jul. 2025. [Online]. Available: https://www.rfc-editor.org/info/rfc9807

[18] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *Advances in Cryptology — EUROCRYPT 2000*, B. Preneel, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 139–155.

[19] M. Abdalla, P.-A. Fouque, and D. Pointcheval, "Password-based authenticated key exchange in the three-party setting," in *Proceedings of the 8th International Conference on Theory and Practice in Public Key Cryptography*, ser. PKC'05. Berlin, Heidelberg: Springer-Verlag, 2005, p. 65–84.

[20] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, 2001, pp. 136–145.

[21] V. Boyko, P. MacKenzie, and S. Patel, "Provably secure password-authenticated key exchange using diffie-hellman," in *Advances in Cryptology — EUROCRYPT 2000*, B. Preneel, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 156–171.

[22] S. Chen, K. Pan, and O. Sanina, "Provable security and privacy analysis of WPA3's SAE and SAE-PK protocols," Cryptology ePrint Archive, Paper 2026/348, 2026. [Online]. Available: https://eprint.iacr.org/2026/348

[23] M. Vanhoef and E. Ronen, "Dragonblood: Analyzing the dragonfly handshake of wpa3 and eap-pwd," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 517–533.

[24] B. Blanchet, "Modeling and verifying security protocols with the applied pi calculus and proverif," *Found. Trends Priv. Secur.*, vol. 1, no. 1–2, p. 1–135, Oct. 2016.

[25] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[26] T. Okamoto and D. Pointcheval, "The gap-problems: A new class of problems for the security of cryptographic schemes," in *Public Key Cryptography (PKC 2001)*, ser. Lecture Notes in Computer Science, vol. 1992. Springer, 2001, pp. 104–118.

[27] D. Pointcheval and G. Wang, "Vtbpeke: Verifier-based two-basis password exponential key exchange," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 301–312.

[28] D. Harkins, "Simultaneous authentication of equals: A secure, password-based key exchange for mesh networks," in *2008 Second International Conference on Sensor Technologies and Applications (sensorcomm 2008)*, 2008, pp. 839–844.

[29] Y. Wu, H. Guo, Y. Han, S. Li, and J. Liu, "A security-enhanced authentication and key agreement protocol in smart grid," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 9, pp. 11 449–11 457, 2024.

[30] D. Wang, G. Li, and J. Jiang, "Password-authenticated key exchange protocols: A survey," *ACM Comput. Surv.*, vol. 58, no. 6, Dec. 2025.

[31] L. Bauman, D. Benjamin, S. Menon, and C. A. Wood, "A Password Authenticated Key Exchange Extension for TLS 1.3," Internet Engineering Task Force, Internet-Draft draft-ietf-tls-pake-01, Mar. 2026, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-tls-pake/01/