Provable Security and Privacy Analysis of WPA3's SAE and SAE-PK Protocols*

Shan Chen[1], Kaige Pan[1], and Olga Sanina[2]

[1] Southern University of Science and Technology, Shenzhen, China**
dragoncs16@gmail.com, 12331240@mail.sustech.edu.cn
[2] Technische Universität Darmstadt, Darmstadt, Germany
olga.sanina@tu-darmstadt.de

Abstract. SAE and SAE-PK are the core security protocols introduced in the latest Wi-Fi security standard, WPA3, to protect personal networks. SAE-PK extends SAE to prevent the so-called evil twin attacks, where an attacker with the knowledge of the password attempts to impersonate a legitimate access point. In this work, we present the *first* provable security and privacy analysis of SAE and SAE-PK. We introduce formal models that capture their intended properties and use these models to analyze the guarantees these protocols provide.

First, we identify an attack that prevents SAE from fulfilling its intended authentication guarantees. As a result, SAE can only be proven secure within a weaker security model, which we also formalize and show the proof in. To achieve the desired level of security, we propose two simple fixes, resulting in two efficient SAE protocols that we call SAEv2 and SAEv3. We prove that both protocols meet the intended security guarantees, with SAEv3 providing greater robustness.

Next, we prove that SAE-PK is indeed secure against evil twin attacks, but its current design introduces a theoretical vulnerability to offline dictionary attacks, which contradicts the expected security guarantees of SAE-PK as an enhanced password-authenticated key exchange protocol. To remedy this, we show that SAE-PK can be modified with minimal changes to fully realize its desired security goals.

Finally, we analyze the privacy guarantees of SAE, SAE-PK, and our proposed enhanced variants. We prove that their cryptographic core preserves the unlinkability of client devices across distinct Wi-Fi networks, if MAC address randomization is properly applied.

Keywords: Wi-Fi · WPA3 · SAE · SAE-PK · Security · Privacy

* © IACR 2026. This is the full version of the EUROCRYPT 2026 article published by Springer-Verlag.

** Shan Chen is affiliated with both the Research Institute of Trustworthy Autonomous Systems and the Department of Computer Science and Engineering of SUSTech.

Table of Contents

1 Introduction . . . 3
2 Preliminaries . . . 4
2.1 Hardness Assumptions . . . 5
3 sPAKE Protocol Syntax and Security Model . . . 5
3.1 Protocol Syntax . . . 6
3.2 Security Model . . . 6
4 Simultaneous Authentication of Equals (SAE) . . . 10
5 Security Analysis of SAE . . . 13
5.1 Role Confusion Attack on SAE . . . 13
5.2 SAE Security . . . 14
5.3 SAEv2 and SAEv3 for Stronger Security . . . 16
6 The SAE-PK Protocol and its Security . . . 17
6.1 Description of SAE-PK . . . 17
6.2 The Randomized Fingerprinting Primitive . . . 18
6.3 Security of SAE-PK . . . 20
6.4 SAE-PK+ for Stronger Security . . . 21
7 Privacy of SAE-Family Protocols . . . 22
7.1 sPAKE Privacy Model . . . 22
7.2 Privacy Mechanisms and Analysis . . . 23
8 Conclusion and Recommendations . . . 24
A Cryptographic Building Blocks . . . 27
A.1 Hash Functions . . . 27
A.2 Pseudorandom Functions (PRFs) . . . 27
A.3 Signature Schemes . . . 28
B SAE-PK Password Encoding and Decoding . . . 28
C Theorems and Security Proofs for SAE-Family Protocols . . . 29
C.1 Theorem Statements . . . 29
C.2 Proof of Theorem 6 (Security of SAE) . . . 31
C.3 Proof of Theorem 7 (Security of SAEv2) . . . 48
C.4 Proof of Theorem 8 (Security of SAEv3) . . . 48
C.5 Proof of Theorem 9 (Security of SAE-PK) . . . 49
C.6 Proof of Theorem 10 (Security of SAE-PK+) . . . 55
D Other Security and Privacy Proofs . . . 55
D.1 Proof of Theorem 2 (Second-Preimage Resistance of PPG) . . . 55
D.2 Proof of Theorem 3 (Fingerprint Indistinguishability of PPG) . . . 56
D.3 Proof of Theorem 5 (Privacy of SAE-Family Protocols) . . . 57

1 Introduction

As the most widely used wireless local area networking technology, Wi-Fi plays a crucial role in enabling billions of devices to access the Internet and exchange data within local networks. The security of Wi-Fi networks is certified by the Wi-Fi Alliance through its Wi-Fi Protected Access (WPA) family of standards. The latest standard, WPA3 [All25], was introduced in 2018 as the successor to WPA2 to address several known design flaws, most notably its vulnerability to *offline dictionary attacks*, where an attacker attempts to guess the password locally using captured WPA2 authentication handshake traffic, as well as flaws that enabled serious practical attacks such as KRACK [VP17].

One of the main updates introduced in WPA3 is the *Simultaneous Authentication of Equals (SAE)* protocol, which is mandated for personal wireless networks. SAE was first specified in the IEEE 802.11s amendment [IEE11], incorporated into the main IEEE 802.11 standard in 2012 [IEE12], and has remained part of it through the latest version [IEE25]. SAE is a *password-authenticated key exchange (PAKE)* protocol that enables two Wi-Fi entities, typically a client station (STA) and a wireless access point (AP), to mutually authenticate and establish a shared secret via a pre-shared password. In personal Wi-Fi networks, this password is often distributed among a group of users, such as customers in a coffee shop. While SAE is designed to thwart offline dictionary attacks by unauthorized outsiders who do not know the password, it remains vulnerable to insider threats: an attacker with the password can impersonate a legitimate AP, a scenario commonly referred to as an *evil twin attack* [GBL+10]. To address this threat, WPA3 introduced an enhanced protocol called *SAE Public Key (SAE-PK)* in its 3.0 version (released in 2020 and subsequently updated to version 3.5 [All25]), enabling strong server authentication by provisioning a signature key pair to the AP.

Despite the critical role that SAE and SAE-PK play in securing personal Wi-Fi networks under WPA3, their security has not been formally analyzed. The initial SAE [Har08a] (that differs from WPA3's SAE protocol) is a variant of the Dragonfly key exchange protocol proposed by Harkins [Har15]. However, SAE used in WPA3 is more complex and diverges in subtle yet important ways from the simplified different SAE variant for which a formal security proof exists [LS15]. For example, WPA3's SAE does not assume that the STA and AP know each other's identifier, namely their media access control (MAC) addresses, prior to initiating the key exchange: The STA typically learns the AP's MAC address during scanning, while the AP obtains the STA's address from handshake message headers. They also differ in other important aspects, such as how their password-based group elements (PWE) and session keys are derived. Then, for SAE-PK, while it was studied in a recent work [VR24], the focus was on implementation and pre-computation attacks rather than formal security guarantees. Besides [VR24], there are some other practical attacks against WPA3 that could affect SAE and SAE-PK, e.g., [VR20,GV24]; however, the security of their cryptographic core remains intact once an appropriate configuration is enforced, such as disabling transition mode [VR20], selecting the hash-to-element method for PWE derivation [VR20,GV24], using strong security parameters and a single public-key size [VR24], etc.

So far, it remains open what security guarantees SAE and SAE-PK can formally achieve when they are properly configured. Our work fills this gap.

Our Contributions. Our main contributions are as follows:

- We define a security model that we call *server-password-authenticated key exchange (sPAKE)*, to formalize the security targeted by SAE and SAE-PK. To this end, our model handles password sharing among multiple clients and insider security when passwords are corrupted, which are not typically considered in PAKE models.
- We identify a *role confusion attack* that allows two client STAs to mistakenly complete an SAE run, undermining the expected security guarantees of SAE. Then, we prove its security in a weaker model that does not regard this attack as a break.

Table 1. Security summary of SAE-family protocols. Green ✓ denotes security, red ✗ denotes insecurity, and orange (✗) denotes theoretical insecurity.

Attack Resistance	Protocol				
	SAE [IEE25]	SAEv2	SAEv3	SAE-PK [All25]	SAE-PK+
Offline Dictionary	✓	✓	✓	(✗)	✓
Role Confusion	✗	✓	✓	✓	✓
Evil Twin	✗	✗	✗	✓	✓

Table 2. Privacy summary of SAE-family protocols with respect to the MAC-address randomization mechanisms employed. Green ✓ denotes unlinkability and red ✗ denotes linkability.

Unlinkability	MAC-Address Randomization Mechanism		
	Fixed [IEE25]	Persistent [And]	Randomized [IEE25]
Across Wi-Fi Networks	✗	✓	✓
Within Wi-Fi Networks	✗	✗	✓

- We propose two simple fixes to SAE, resulting in two efficient variants that we call SAEv2 and SAEv3. We prove that both amended protocols resist the role confusion attack and achieve the desired security guarantees.
- We prove that SAE-PK is secure against evil twin attacks, but offline dictionary attacks become theoretically possible. As part of this effort, to analyze the security guarantees of the password generation and public-key verification algorithms introduced in SAE-PK, we formalize a cryptographic primitive called *randomized fingerprinting*, which might be of independent interest.
- We demonstrate that SAE-PK can be strengthened with minimal modifications to eliminate the above vulnerability, yielding a provably secure and even more efficient protocol that we call SAE-PK+.
- We define a privacy model to analyze the guarantees provided by the above SAE-family protocols under two real-world types of MAC-address randomization mechanisms on client devices: a *persistent* scheme with address randomization per Wi-Fi network, and a *random* scheme that randomizes the address for each connection. We prove that the cryptographic core of all SAE-family protocols considered in this work ensures that client devices are unlinkable across different Wi-Fi networks under the persistent scheme (e.g., as implemented in Android [And]), while the random scheme further ensures unlinkability within each network.

Table 1 summarizes the security results of the above SAE-family protocols; Table 2 summarizes their privacy results, with respect to the MAC-address randomization mechanisms employed.

Responsible Disclosure. We reported our attacks and mitigation proposals to the Wi-Fi Alliance on August 23, 2025 and later we reworked SAEv3 and reported it on January 30, 2026. The Alliance agreed with our results but did not promise to make changes soon.

Paper Organization. The rest of the paper is organized as follows. Section 2 introduces the necessary preliminaries, with additional details provided in Appendix A. Section 3 defines our sPAKE security model. Section 4 describes the SAE protocol, and Section 5 provides its security analysis along with our proposed fixes. Section 6 discusses the additional functionalities introduced by SAE-PK and analyzes its security, with additional details of SAE-PK specified in Appendix B. Section 7 defines the privacy model and presents our privacy results for the SAE-family protocols. Finally, Section 8 concludes the paper with our summarized recommendations for improving the protocols. The proofs are detailed in Appendix C and Appendix D.

2 Preliminaries

Notation. Let ε denote the empty string, $\{0,1\}^n$ denote the set of all bit strings of length $n > 0$, and $\{0,1\}^*$ denote the set of all finite bit strings. Let 0^L and 1^L respectively denote the all-0 and

all-1 strings of length L. Let the special symbol $\perp$ (which is outside $\{0,1\}^*$) denote uninitialized state or failure. Let $|$ denote string concatenation and $\setminus$ denote set subtraction. Let $\min(a,b)$ and $\max(a,b)$ respectively denote the smaller and larger string of a and b when interpreted as non-negative integers. Let $x \leftarrow y$ denote assigning y to x. For a randomized function or algorithm A, let $x \xleftarrow{\$} \mathsf{A}$ denote running A and assigning the output to x; if A is deterministic, we write $x \leftarrow \mathsf{A}$ instead. For a finite set $\mathcal{S}$, let $|\mathcal{S}|$ denote its size and $S \xleftarrow{\$} \mathcal{S}$ denote sampling S from $\mathcal{S}$ uniformly at random. Let $\mathcal{G}$ denote an additive cyclic group of prime order q with generator G, identity element O, and inverse $-A$ for any $A \in \mathcal{G}$. Let $\mathbb{Z}_q$ denote the group of integers modulo a prime q, and a^{-1} denote the multiplicative inverse of $a \in \mathbb{Z}_q$. For a point P on an elliptic curve, let $[P]_x$ denote its x-coordinate.

Advantage Measures for Cryptographic Building Blocks. In Appendix A, we recall the syntax and security definitions of a hash function H, pseudorandom function F, and signature scheme Sig, as well as their corresponding advantage measures for an adversary $\mathcal{A}$: $\mathsf{Adv}^{\mathrm{coll}}_{\mathsf{H}}(\mathcal{A})$, $\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A})$, $\mathsf{Adv}^{\text{uf-cma}}_{\mathsf{Sig}}(\mathcal{A})$.

2.1 Hardness Assumptions

Our security analysis relies on four Diffie–Hellman-related hardness assumptions. In addition to the classical Computational Diffie–Hellman (CDH) assumption and its gap variant GDH [OP01], we employ the Gap Simultaneous Diffie–Hellman (GSDH) assumption introduced by Pointcheval and Wang [PW17], who proved its intractability in the generic group model [Sho97]. To handle certain less complex cases in our security proofs, we also define an intermediate assumption that is harder than GSDH but easier than GDH, which we refer to as the weak GSDH (wGSDH) assumption. The formal definitions of the above assumptions are recalled as follows:

CDH. This assumption states that, given (G, aG, bG) for random $a, b \xleftarrow{\$} \mathbb{Z}_q$, it is computationally infeasible to find abG. The advantage of an adversary $\mathcal{A}$ in breaking this assumption is denoted by $\mathsf{Adv}^{\mathrm{cdh}}_{\mathcal{G},G}(\mathcal{A})$.

GDH. This assumption states that CDH remains computationally infeasible even when the adversary has access to a Decisional Diffie–Hellman (DDH) oracle, which inputs (G, aG, bG, cG) and outputs whether $abG = cG$. The advantage of an adversary $\mathcal{A}$ in breaking this assumption is denoted by $\mathsf{Adv}^{\mathrm{gdh}}_{\mathcal{G},G}(\mathcal{A})$.

GSDH. This assumption states that, given (G, aG, bG) for random $a, b \xleftarrow{\$} \mathbb{Z}_q$, it is computationally infeasible to find $(H, a^{-1}H, b^{-1}H)$ such that $H \in \mathcal{G} \setminus \{O\}$ (i.e., H is a non-zero group element), even with help of a DDH oracle. The advantage of an adversary $\mathcal{A}$ in breaking this assumption is denoted by $\mathsf{Adv}^{\mathrm{gsdh}}_{\mathcal{G},G}(\mathcal{A})$.

wGSDH. This assumption is similar to GSDH but further requires the adversary to find another triple. It states that, given (G, aG, bG) for random $a, b \xleftarrow{\$} \mathbb{Z}_q$, it is computationally infeasible to find two triples, $(H, a^{-1}H, b^{-1}H)$ and $(K, a^{-1}kK, b^{-1}kK)$, such that $H \in \mathcal{G} \setminus \{O\}$ and $K = kG \in \mathcal{G} \setminus \{O\}$, even with help of a DDH oracle. The advantage of an adversary $\mathcal{A}$ in breaking this assumption is denoted by $\mathsf{Adv}^{\mathrm{wgsdh}}_{\mathcal{G},G}(\mathcal{A})$.

3 sPAKE Protocol Syntax and Security Model

In this section, we define the protocol syntax and security model for *server-password-authenticated key exchange (sPAKE)* protocols.

Existing PAKE models are typically defined for a client-server setting where each client holds a private password, so they do not apply to the Wi-Fi setting in which multiple clients may share the same password. Recent works [CNPR22,KR24] address this limitation by binding the shared password to each party's identity. However, their models do not align with the achievable security of SAE and SAE-PK, yet aiming for significantly stronger security guarantees. Our

model is designed to capture the intended security properties of SAE and SAE-PK. It is also universal, in that it can be used to analyze both protocols and to directly compare their security properties within a unified framework.

UC vs. Game-Based. PAKE protocols are typically analyzed via either the universal composability (UC) framework (e.g., [CHK+05,AHH21,BGHJ24]) or game-based notions (e.g., [BPR00,AP05,LS15]). While the UC framework ensures *security under composition*, a valuable property for protocols serving as primitives in greater systems, we adopt the game-based methodology for sPAKE to analyze SAE and SAE-PK. This is motivated by both protocols being primarily used in the Wi-Fi ecosystem, where composability is less critical and game-based notions can more naturally capture the multiple security levels expected from them. Furthermore, this choice facilitates a direct comparison with the game-based Dragonfly analysis [LS15].

3.1 Protocol Syntax

Our sPAKE protocol syntax is similar to that of the classical PAKE protocol defined in the model formalized by Bellare, Pointcheval, and Rogaway [BPR00] (simply referred to as the BPR model in this paper), but with several important differences that we described below.

Recall that a PAKE protocol allows a client and a server to establish a high-entropy session key based on a low-entropy password chosen by the client. This client memorizes the password while the other clients do not know it. The server typically stores only some information about the password (e.g., its hash) such that using it to recover the password is hard but to verify the correct one is easy.

Unlike PAKE, a sPAKE protocol associates a password with each server rather than each client; we refer to this server as the *associated server* of the password. Accordingly, passwords are stored and managed by servers and are hence not required to be human-memorizable, but they are still human-readable and human-writable to enable manual verification and entry of the passwords.[3] Furthermore, sPAKE allows multiple clients to *share* the same server password, which models scenarios typical of personal Wi-Fi networks. Finally, each shared password may be accompanied by some *auxiliary information* to authenticate and verify the identity of the associated server.

Formally, a sPAKE protocol is an interactive protocol between a client and a server, each associated with an identifier. In the beginning of the protocol, they each input their own identifier and a shared password; each party may also input auxiliary information associated with the shared password, which could differ between the client and server sides (e.g., the server holding a secret signing key while the client holding the corresponding public verification key). When the protocol terminates, the client and the server each establish a session key. Correctness imposes that their established session keys are the same.

3.2 Security Model

One important security goal of classical PAKE protocols is to prevent *offline dictionary attacks*, i.e., finding the correct password in use by brute-forcing all possible ones while *passively* eavesdropping protocol executions. However, due to the low entropy of passwords used in PAKE protocols, active *man-in-the-middle (MITM)* attacks can still guess and verify the shared password with *non-negligible* probability. Such attacks are referred to as *online dictionary attacks*, as they have to interact with an "online" honest party to verify its password guess. A secure PAKE protocol guarantees that only a very limited number (e.g., one or two) of passwords can be guessed per manipulated protocol execution.

[3] Here we describe "human-readable", "human-writable" and "human-memorizable" informally; one can refer to [BCDP17] for their formal definitions.

For our sPAKE protocols, the server passwords are human-writable and hence typically not as high-entropy as cryptographic keys. Therefore, as with PAKE, a sPAKE protocol should also be designed to resist offline dictionary attacks.

In the following, we formally define the security model for sPAKE protocols. In addition to the classical PAKE security goals in the BPR model, as we will show, our security model introduces new security notions to capture the weaker and stronger guarantees expected from different sPAKE protocols.

Session Oracles and States. Recall that a sPAKE protocol is executed between clients $\mathcal{C}$ and servers $\mathcal{S}$. As with the Bellare–Rogaway model [BR94], to capture multiple sequential and parallel sPAKE executions (or instances), we associate each party $P \in \mathcal{C} \cup \mathcal{S}$ with a sequence of session oracles $\pi_P^1, \pi_P^2, \ldots$, where π_P^i denotes the i-th instance of P and is also called the i-th *session* of P.

Each session π_P^i keeps the following states (and perhaps other internal states):

- $\mathsf{id} \in \mathcal{ID} \subseteq \{0,1\}^l$: the identifier associated with party P;
- $\mathsf{pw} \in \mathcal{PW} \subseteq \{0,1\}^*$: the password used by the session;
- $\mathsf{S} \in \mathcal{S}$: the server with which the input password pw is associated, which is set along with pw and simply referred to as the session's *associated server*;
- $\mathsf{SN} \in \mathcal{SN} \subseteq \{0,1\}^*$: the name of the associated server, e.g., the network name set by an AP, which may not be unique;
- $\mathsf{aux} \in \mathcal{X} \subseteq \{0,1\}^*$: the auxiliary information associated with pw, which could be the empty string to indicate the absence of input auxiliary information;
- $\mathsf{key} \in \{0,1\}^\lambda \cup \{\perp\}$: the established session key, initialized to $\perp$ (unset);
- $\mathsf{sid} \in \{0,1\}^* \cup \{\perp\}$: the session identifier used as unique session reference, typically defined as a function of the transcript (i.e., exchanged messages) of the session, initialized to $\perp$ (unset);
- $\mathsf{pid} \in \mathcal{ID} \cup \{\perp\}$: the identifier of the party with which the session believes it has just established the session key, initialized to $\perp$ (unset);
- $\mathsf{st}_{\mathsf{exe}} \in \{\mathsf{initiated}, \mathsf{accepted}\}$: the execution state of the session, initialized to $\mathsf{initiated}$, and changed to $\mathsf{accepted}$ when the session *accepts* (at which point the session has set its key, sid and pid);

Security Experiment. The experiment is executed between an adversary $\mathcal{A}$ and a challenger. At the beginning of the experiment, the challenger fixes an arbitrary distribution $\mathscr{D}$ over a (probably high-entropy) password dictionary $\mathcal{PW}$, then it samples an independent random password for each server according to $\mathscr{D}$, together with the associated auxiliary information (if any) respectively for the server itself and clients that use the server password; this sampling process is denoted by $\langle(\mathsf{pw}_S, \mathsf{aux}_S, \overline{\mathsf{aux}}_S) \xleftarrow{\$} \mathcal{PW} \times \mathcal{X} \times \mathcal{X}\rangle_{S \in \mathcal{S}}$. Note that here without loss of generality each server $S \in \mathcal{S}$ is assumed to hold only one password. As a result, all sessions of S share the same password pw_S and server-side auxiliary information aux_S, and all client sessions that use pw_S as the input password share the same client-side auxiliary information $\overline{\mathsf{aux}}_S$; all sessions using pw_S share the same associated server S. Next, the adversary $\mathcal{A}$ assigns a server name to each server $S \in \mathcal{S}$ and then interacts with the challenger by querying the following oracles and learning the execution state of its queried sessions.

$\mathsf{Send}(P, i, m)$ sends a message m to session π_P^i, which consists of two cases:

- The session π_P^i is specified to initiate the protocol run and was *not used* before (i.e., not queried to any allowed oracles). If P is a client, $\mathcal{A}$ is required to specify a server $S \in \mathcal{S}$ by setting $m = S$, indicating that π_P^i uses the password pw_S of the server S and the auxiliary information $\overline{\mathsf{aux}}_S$ and sets $\pi_P^i.\mathsf{S} \leftarrow S$; otherwise, P is a server and the session π_P^i just uses pw_P, aux_P and sets $\pi_P^i.\mathsf{S} \leftarrow P$, so the input message m is useless and required to be set to a special symbol $\top$ that means unspecified. Then, the challenger returns to $\mathcal{A}$ the first message that π_P^i sends to start a protocol execution.

– Otherwise, the challenger asks the session π_P^i (which has not terminated) to process m according to the protocol, updates its session states, and returns the response to $\mathcal{A}$. If the session *accepts* (i.e., $\pi_P^i.\mathsf{st}_{\mathsf{exe}} = \mathsf{accepted}$), its sid and pid are set according to the protocol and also returned to $\mathcal{A}$.

After this query, we say π_P^i was *manipulated*, indicating that $\mathcal{A}$ has interacted with the session and may manipulate its sent messages.

$\mathsf{Execute}(C, i, S, j)$ triggers an honest protocol execution between a client session π_C^i and a server session π_S^j, where both sessions were not used before. The challenger returns the entire communication *transcript* (i.e., all exchanged messages of the protocol execution) and sid, pid to $\mathcal{A}$. As with the BPR model, this oracle is useful in modeling offline dictionary attacks where the adversary has only *passive* access to honest protocol executions. Note that the sessions used in this query are not manipulated, indicating that $\mathcal{A}$ did not manipulate their sent messages.

$\mathsf{Reveal}(P, i)$ returns the session key $\pi_P^i.\mathsf{key}$ to $\mathcal{A}$ if the session has accepted (and hence $\pi_P^i.\mathsf{key} \neq \perp$); otherwise, it returns $\perp$. In the first case, after this query, we say the session key of π_P^i was *revealed*.

$\mathsf{sCorrupt}(S, (\mathsf{pw}, \mathsf{aux}))$ returns to $\mathcal{A}$ the password pw_S of server S and its server-side auxiliary information aux_S. This oracle models the corruption of servers. If $(\mathsf{pw}, \mathsf{aux}) \neq \top$ (i.e., the second input is specified), the challenger further replaces pw_S and aux_S with the input pw and aux respectively, which, for instance, captures attacks that alter the password of S to disable its service to honest clients. After this query, we say the server S was *strongly corrupted*.

$\mathsf{iCorrupt}(S)$ returns to $\mathcal{A}$ the password pw_S of server S and its client-side auxiliary information $\overline{\mathsf{aux}}_S$. This oracle models a server's password corruption on the client side: The client-stored password, along with the associated client-side auxiliary information, is corrupted, but the server-side auxiliary information remains safe. After this query, we say S was *insider-corrupted*.

$\mathsf{Test}(P, i)$ is used to measure adversarial success in breaking session key secrecy and does not reflect real-world adversarial abilities. This query is allowed only if the specified session π_P^i has accepted; if so, the following happens. First, a uniform random bit $b \in \{0, 1\}$ is flipped. Then, if $b = 0$, a random key is sampled uniformly at random from the key space $\{0, 1\}^\lambda$ and returned to $\mathcal{A}$; if $b = 1$, the actual session key $\pi_P^i.\mathsf{key}$ is returned. This oracle, if granted to the adversary, is queried *exactly once*.

Partners and Weak Partners. We follow the BPR model [BPR00] to define *partnered* sessions using session identifiers. Two distinct sessions π_P^i and $\pi_{P'}^{i'}$ are *partnered* if and only if they both have accepted and the following conditions hold:

(1) $P \in \mathcal{C} \wedge P' \in \mathcal{S}$ **or** $P \in \mathcal{S} \wedge P' \in \mathcal{C}$;
(2) $\pi_P^i.\mathsf{S} = \pi_{P'}^{i'}.\mathsf{S}$, $\pi_P^i.\mathsf{key} = \pi_{P'}^{i'}.\mathsf{key}$, $\pi_P^i.\mathsf{sid} = \pi_{P'}^{i'}.\mathsf{sid}$, $\pi_P^i.\mathsf{pid} = \mathsf{id}_{P'}$ **and** $\pi_{P'}^{i'}.\mathsf{pid} = \mathsf{id}_P$;
(3) no other session holds the same sid as π_P^i and $\pi_{P'}^{i'}$.

That is, two sessions are each other's *partner* if and only if: (1) they belong to different types of parties, (2) they are associated with the same server S, establish the same key, agree on the same sid, and have each other's party identity as pid, and (3) they do not share the agreed sid with any other session.

In order to capture the weak security guarantees of SAE, specifically to rule out the role confusion attack (see Section 5.1 for details) from the model, we define a notion of *weak partnership* by dropping the first condition from the above partnership definition. Two sessions are called *weak partners* (or are *weakly partnered*) if and only if the last two conditions (2) (3) hold. As a result, sessions of the same type, e.g., two client sessions, can be weakly partnered. Clearly, any pair of partnered sessions are also weakly partnered.

Advantage Measures. As with the BPR model [BPR00], we define two types of security notions and their corresponding advantage measures for a sPAKE protocol sPAKE to capture its *authenticated-key-exchange (AKE)* security and *explicit mutual authentication (EA)*. For each type, we define three notions: a regular one, similar to that in the BPR model; a weaker one,

reflecting the limited security guarantees of SAE; and a stronger one, which further captures the insider security guarantees expected from a sPAKE protocol. Among them, as shown below, all AKE security notions capture *forward secrecy*, i.e., session keys remain secure even if the password (and auxiliary information) are later corrupted.

AKE security. At the end of the security experiment, an adversary $\mathcal{A}$ against the AKE security of sPAKE outputs a bit $b' \in \{0,1\}$ as its guess for the secret random bit b sampled in the Test query. We say that $\mathcal{A}$ *wins* if and only if $b' = b$ and the tested session π_P^i is *fresh*:

(1) The tested session key was not revealed;
(2) For any session $\pi_{P'}^{i'}$ that is partnered with the tested session π_P^i, if any, its session key was not revealed;
(3) The tested session π_P^i was not manipulated
or the associated server $\pi_P^i.\mathsf{S}$ was *not corrupted* (i.e., neither strongly corrupted nor insider-corrupted) before π_P^i accepted.

The AKE advantage $\mathsf{Adv}^{\text{ake}}_{\mathsf{sPAKE}}(\mathcal{A})$ is defined as $2\Pr[\mathcal{A} \text{ wins}] - 1$.

Weak AKE (wAKE) security. This notion is defined the same as AKE security, except that session freshness is strengthened (making the adversary less powerful) such that even *weakly* partnered session keys are not allowed to be revealed. Formally, weak AKE security is defined by replacing the above AKE freshness condition (2) with the following (with conditions (1) (3) unchanged):

(2)' For any session $\pi_{P'}^{i'}$ that is weakly partnered with the tested session π_P^i, if any, its session key was not revealed.

The wAKE advantage $\mathsf{Adv}^{\text{w-ake}}_{\mathsf{sPAKE}}(\mathcal{A})$ is defined as $2\Pr[\mathcal{A} \text{ wins}] - 1$.

Insider AKE (iAKE) security. This notion is defined similarly to AKE security, except that it further accounts for malicious insiders (e.g., a malicious client in practice) who may have access to the shared password. For such an active insider adversary, there is nothing preventing it from establishing a key with a server once the adversary knows the password, as the auxiliary information is used to verify the server's identity rather than the client's. Therefore, the adversary is allowed to obtain the password (via a $\mathsf{iCorrupt}$ query) only before testing a *client* session that uses it. Formally, insider AKE security is defined by replacing the freshness condition (3) in the AKE security notion with the following (with conditions (1) (2) unchanged):

(3)' The tested session π_P^i was not manipulated
or the associated server $\pi_P^i.\mathsf{S}$ was not corrupted before π_P^i accepted
or $P \in \mathcal{C}$ and $\pi_P^i.\mathsf{S}$ was not strongly corrupted before π_P^i accepted.

Note that the above newly added condition (in blue) allows the adversary to insider-corrupt (via a $\mathsf{iCorrupt}$ query) the password used by the tested client session π_P^i. The iAKE advantage $\mathsf{Adv}^{\text{i-ake}}_{\mathsf{sPAKE}}(\mathcal{A})$ is defined as $2\Pr[\mathcal{A} \text{ wins}] - 1$.

Explicit authentication (EA). An adversary $\mathcal{A}$ attempting to break EA of sPAKE interacts with all experiment oracles except Test. We say that $\mathcal{A}$ *wins* if and only if it causes a session π_P^i to *successfully terminate* (i.e., without aborting) with the following hold:

(a) The session π_P^i has no partner;
(b) The associated server $\pi_P^i.\mathsf{S}$ was not corrupted before π_P^i terminated.

The EA advantage $\mathsf{Adv}^{\text{ea}}_{\mathsf{sPAKE}}(\mathcal{A})$ is defined as $\Pr[\mathcal{A} \text{ wins}]$.

Weak explicit authentication (wEA). This notion is defined the same as EA, except that the terminated session must also have no *weak* partner. That is, the above condition (a) is replaced by the following (with condition (b) unchanged):

(a)' The session π_P^i has no weak partner.

The wEA advantage $\mathsf{Adv}^{\text{w-ea}}_{\mathsf{sPAKE}}(\mathcal{A})$ is defined as $\Pr[\mathcal{A} \text{ wins}]$.

Insider explicit authentication (iEA). Similar to how iAKE captures insider attacks, insider explicit authentication is defined by replacing condition (b) in the EA security notion with the following (with condition (a) unchanged):

(b)' The associated server $\pi^i_P.\mathsf{S}$ was not corrupted before π^i_P terminated
or $P \in \mathcal{C}$ and $\pi^i_P.\mathsf{S}$ was not strongly corrupted before π^i_P terminated.

Similarly, the above newly added condition (in blue) allows the adversary to insider-corrupt (via a $\mathsf{iCorrupt}$ query) the password used by the target client session π^i_P. The iEA advantage $\mathsf{Adv}^{\text{i-ea}}_{\mathsf{sPAKE}}(\mathcal{A})$ is defined as $\Pr[\mathcal{A} \text{ wins}]$.

Notion Relations. Now, we discuss the relations between the six notions defined above. First, recall that by definition a partner is also a weak partner. So, the adversarial abilities modeled in wAKE and wEA are more restricted (or weaker) than those in AKE and EA, respectively, so AKE implies wAKE, and EA implies wEA. Then, since conditions (3)' and (b)' are clearly relaxed compared to (3) and (b), respectively, it follows that iAKE implies AKE, and iEA implies EA. However, the other directions do not hold. As we will show in later sections, SAE is secure in the wAKE and wEA sense, but achieves neither AKE nor EA; our fixed protocols SAEv2 and SAEv3 satisfy AKE and EA, but fail to achieve iAKE and iEA; finally, SAE-PK and our enhanced protocol SAE-PK+ satisfy iAKE and iEA. The above notion relations are summarized in Fig. 1.

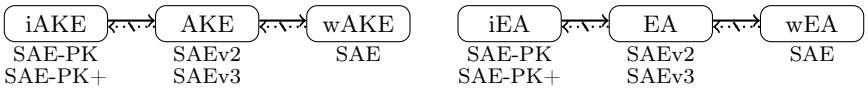


Fig. 1. Notion relations (with example sPAKE protocols below). Solid arrows represent implications, while dotted (crossed-out) arrows indicate separations.

Remark on Passive Insider Security of AKE Notions. We remark that if the insider adversary is *passive* during a sPAKE protocol execution between a client and a server, i.e., not manipulating the exchanged messages, the adversary is not supposed to learn their established session key even if it already knows the shared password beforehand. This is captured by the first condition in (3)' and the original (3): The tested session was not manipulated. Therefore, besides iAKE security, both the original AKE security and weak AKE security also protect against passive insiders who know the shared password (e.g., an honest-but-curious client in a public Wi-Fi setting).

4 Simultaneous Authentication of Equals (SAE)

In this section, we describe the *Simultaneous Authentication of Equals (SAE)* protocol specified in IEEE 802.11-2024 [IEE25]. This protocol, as depicted in Fig. 2, runs between a client station (STA) and a wireless access point (AP) to achieve mutual authentication and shared session key establishment from a pre-shared password. The blue parts in the figure depict the *SAE Public Key (SAE-PK)* extensions, which will be described in Section 6.1.

Public Parameters and Setup. Before describing the protocol, we specify the cryptographic setting first. SAE (and SAE-PK) can be instantiated with either elliptic curve cryptography (ECC) groups or finite field cryptography (FFC) groups; in both cases, the underlying group is a cyclic group of *prime* order, ruling out *small subgroup attacks* [CJ19]. In this work, we focus on ECC groups, as they are more common in practice, and hence consider additive group operations. Then, we assume a fixed protocol configuration and do not consider downgrade attacks. As a

result, related negotiation procedures are omitted; instead, both entities are assumed to agree on public group parameters $(\mathcal{G}, G, q)$, where $\mathcal{G}$ is a cyclic group of prime order q (where $q = |\mathcal{G}|$ is of at least 256-bit length [All25]) with generator G. These group parameters further determines another public parameter $L \in \{256, 384, 512\}$, which indicates the output length of the underlying hash algorithm SHA-L used in the protocol.

Both a STA and an AP possess physical media access control (MAC) addresses as their identifiers, denoted by MAC_C and MAC_S respectively, which are 48-bit values hard-coded into the entity devices. To enhance privacy, the standard allows the use of locally generated random addresses instead of the real MAC addresses. Such privacy mechanisms are described and analyzed later in Section 7.2. Prior to protocol execution, an AP is configured with a password pw and a service set identifier (SSID), which serves as the network name (e.g., "Home") and may not be unique. This password (and optionally SSID) are provided to the STA out of band (e.g., through user input), either before or during the initial protocol phase, known as *scanning*, as described below.

Scanning Phase. In this phase (see lines 1 to 5 of Fig. 2), the entities discover each other. This scanning process may be initiated by either party, depending on which of the following modes is employed.

In the *active* scanning mode, a STA initiates the scanning process to actively attempting a connection to the intended network. The STA sends a probe request containing its MAC_C (and optionally a target SSID), and the AP that received this request replies with a probe response that includes its MAC_S and SSID.

In the *passive* scanning mode, line 2 of Fig. 2 is *not* executed. Instead, an AP periodically broadcasts beacons that advertise its presence and configuration, including its MAC_S and SSID. Such beacons are monitored by STAs to discover nearby APs.

PWE Generation. After scanning, the STA proceed to generate the password-based group element PWE (see lines 6 to 8 of Fig. 2), while on the AP side PWE is generated after receiving the first SAE authentication message from the STA (see lines 6, 14 and 15 of Fig. 2). We now describe how PWE is generated.

First, the password pw is mapped to an elliptic curve point using one of two methods defined in the IEEE standard [IEE25]: *hunt-and-peck* or *hash-to-element*. Both methods rely on the same hash function SHA-L, but differ in their instantiations. We focus on the hash-to-element method, abstract its involved procedures as a single function $PT \leftarrow \mathsf{H2C}(\text{SSID}, \mathsf{pw})$, and refer to [IEE25] for details. Although the AP may pre-compute this PT during setup, it is only used after scanning; we therefore simply present its computation on both sides symmetrically.

Next, the STA and AP each compute a digest value val. It is computed using HMAC [KBC97] instantiated with SHA-L, under key 0^L, on the concatenation of MAC_C and MAC_S, ordered lexicographically with the larger address first. Formally, this is $val \leftarrow \mathsf{HMAC}(0^L, \max(\text{MAC}_C, \text{MAC}_S) | \min(\text{MAC}_C, \text{MAC}_S))$. All subsequent uses of HMAC follow the same hash instantiation. The resulting val is mapped to a nonzero scalar in $\mathbb{Z}_q$, by computing $val \bmod (q-1) + 1$, and then used for scalar multiplication with PT, yielding the group element PWE.

Authentication Flow. After generating PWE, the STA constructs a so-called Commit message and sends it to the AP, as shown in lines 9 to 12 of Fig. 2. This message contains the STA's MAC address MAC_C, a scalar s_C, and a group element CE_C. The scalar s_C is computed as $s_C \leftarrow (r_C + m_C) \bmod q$, where r_C, m_C are secret random values sampled from $\mathbb{Z}_q \setminus \{0, 1\}$ such that $(r_C + m_C) \bmod q$ is not equal to 0 or 1. The element CE_C is derived as $CE_C \leftarrow -m_C \cdot PWE$.

Upon receiving the Commit message, the AP first performs validity checks (see line 13 of Fig. 2), which prevent invalid curve attacks (e.g., [CH14,Sun22]). If they pass, the AP learns the STA's MAC_C, generates PWE, performs operations that mirror the above STA procedures, and sends back a Commit message containing MAC_S, s_S, CE_S, as shown in lines 16 to 19 of

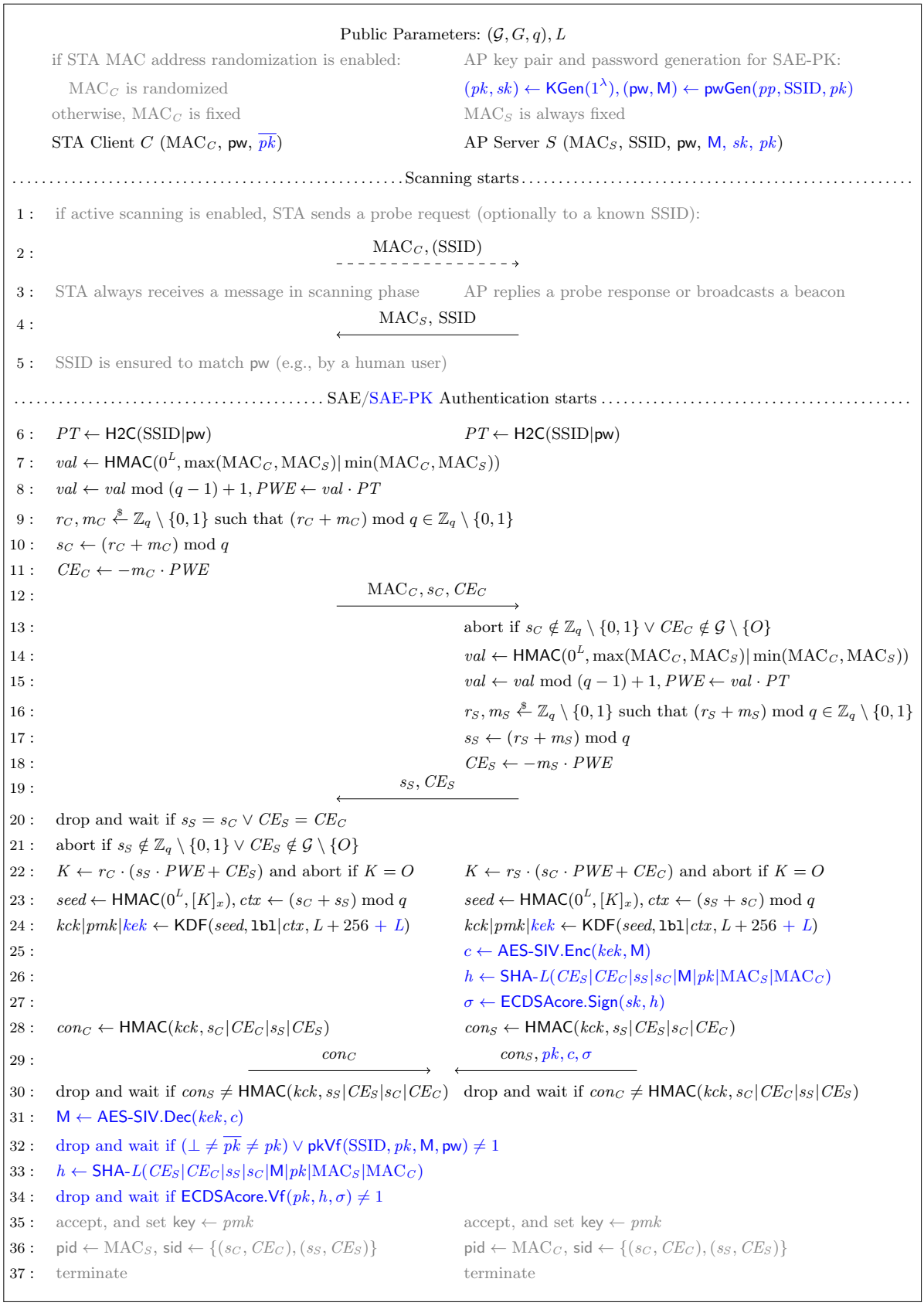


Fig. 2. The SAE and SAE-PK protocols with the scanning phase used in WPA3. Procedures exclusive to SAE-PK are colored in blue. The password generation algorithm pwGen and public-key verification algorithm pkVf of SAE-PK are described later in Fig. 5 and Section 6.1. The key label `lbl` used in KDF is the ASCII string "`SAE KCK and PMK`" for SAE and "`SAE-PK keys`" for SAE-PK.

Fig. 2. Here MAC_S is omitted from the sent message as it is already known by the STA. Then, the STA also checks the validity of the received Commit message (see lines 20 to 21 of Fig. 2).

After exchanging valid Commit messages, both entities proceed to derive the key confirmation key *kck* and pairwise master key *pmk*, as shown in lines 22 to 24 of Fig. 2). First, a non-zero shared key K is computed from the exchanged values, the generated PWE, and the secret random values. Then, its x-coordinate $[K]_x$ is processed using HMAC with key 0^L to obtain the key seed *seed*. Finally, *kck* and *pmk* are derived from *seed* via a key derivation function KDF specified in the IEEE standard [IEE25, Section 12.7.1.6.2], using the fixed label "`SAE KCK and PMK`" concatenated with the context $ctx \leftarrow (s_C + s_S) \bmod q$. Formally, this final step is written as $kck|pmk \leftarrow \mathsf{KDF}(seed, \texttt{lbl}|ctx, L + 256)$. Here *kck* has length L bits, while *pmk* has length either 256 or L bits depending on the configuration; as $L \geq 256$, we for simplicity fix the length of *pmk* to 256 bits in this paper.

After deriving *kck* and *pmk*, the STA and AP proceed to construct and exchange the so-called Confirm messages (see lines 28 to 30 of Fig. 2), which each contain a confirmation value denoted by con_C and con_S respectively. These values are computed using HMAC with key *kck* over the exchanged values as $con_C \leftarrow \mathsf{HMAC}(kck, s_C|CE_C|s_S|CE_S)$ and $con_S \leftarrow \mathsf{HMAC}(kck, s_S|CE_S|s_C|CE_C)$. These values are exchanged, potentially *concurrently*, and verified by the entities. If verification succeeds, they accept *pmk* as the session key and terminate.

Recall that the blue parts specific to SAE-PK is described later in Section 6.1.

5 Security Analysis of SAE

Before diving into the security analysis, we first clarify the necessary assumptions and specify our analysis targets.

By design, SAE treats both entities as equal participants. In practice, however, the protocol runs between a STA, representing a personal device, and an AP, typically a wireless router. This asymmetry is also reflected in the ordering of Commit messages. To align with practical deployments, we assume that the STA initiates SAE authentication after scanning (see Fig. 3).

For protocol configuration, we assume the more common elliptic curve groups are used, as mentioned in Section 4, but our results also apply to finite field groups. For the scanning phase, we assume the STA obtains the password (and optionally the SSID) during a setup phase prior to protocol execution, and can later verify (e.g., by a human user) that the SSID matches the password in use. Finally, for PWE generation, we mentioned that two methods are specified in the IEEE standard [IEE25]: hunt-and-peck and hash-to-element. However, hunt-and-peck has been shown to be vulnerable to timing side-channel attacks [VR20] and hard to fix [DABFS20]; moreover, SAE with hunt-and-peck is susceptible to recently discovered SSID confusion attacks [GV24]. Therefore, we consider hunt-and-peck insecure and analyze the hash-to-element method in this work.

5.1 Role Confusion Attack on SAE

First, we identify a *role confusion attack* that prevents SAE from achieving its intended authentication security.

Our attack resembles the role confusion attack on Bluetooth [TH21], where two devices both think they are the initiator but end up with different keys, causing communication to fail. In contrast, our attack is more subtle and powerful: Two STAs can successfully complete SAE authentication while each mistakenly believes it is the initiator that connects to an AP, and crucially, they derive the *same* session key. Although the full Wi-Fi connection may fail due to asymmetries in the subsequent procedures, our attack still produces a legitimate-looking authentication run that violates the intended guarantees of SAE.

As shown in Fig. 4, the attacker performs two steps: first tampering with the beacon messages such that each STA thinks the other STA's MAC address is the AP's and views itself as the

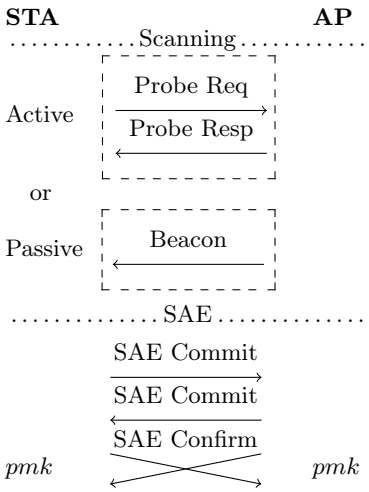


Fig. 3. Scanning and SAE phases.

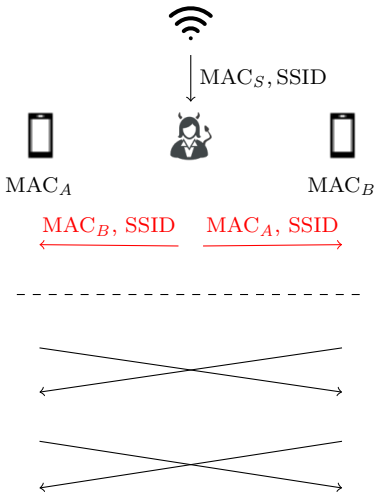


Fig. 4. Role confusion attack on SAE.

initiator, then relaying their Commit and Confirm messages to each other. We note that our attack strategy differs from that of the Selfie attack on TLS-PSK [DG21], which reflects a single entity's messages to cause role confusion. Unlike Selfie, our attack involves two distinct entities and is mounted by modifying and relaying messages between them.

5.2 SAE Security

Due to role confusion attacks, SAE as a sPAKE protocol cannot achieve the desired AKE and EA security. Formally, this is because an adversary $\mathcal{A}$ can simulate the role confusion attack in the security model to ask two honest client sessions to complete an SAE run with each other and establish the same session key *pmk*. By definition, these two client sessions are only weakly partnered but *not partnered*, i.e., having no partnered server session; this already breaks the EA security. Then, $\mathcal{A}$ can reveal either client session's *pmk* and query the Test oracle on the other session, which trivially breaks the AKE security.

The following theorem (with full security bounds in Theorem 6) states the wAKE and wEA security of SAE, which essentially confirms that SAE is secure when role confusion attacks are not a concern. We remark that this security actually ensures that the adversary cannot learn the session key of two connected client sessions resulting from a role confusion attack, unless one of them is compromised.

Theorem 1. *Let* $\mathrm{H}_\infty(\mathscr{D}_\mathcal{D})$ *denote the min-entropy of a password distribution* $\mathscr{D}_\mathcal{D}$ *over the dictionary* $\mathcal{D}$. *Let* $\mathsf{H2C}$ *(see line 6 of Fig. 2),* HMAC *(with fixed key* 0^L *for deriving seed, see line 23 of Fig. 2),* KDF *(see line 24 of Fig. 2), and* HMAC *(for deriving* $con_C, cons_S$*, see lines 28 and 30 of Fig. 2) be modeled as random oracles* H_1*,* H_2*,* H_3*, and* H_4*, respectively. Let* $\mathcal{A}$ *be any efficient adversary that makes at most* q_{se} Send*-queries,* q_{exe} $\mathsf{Execute}$*-queries, and* $q_{\mathsf{H}_1}, q_{\mathsf{H}_2}, q_{\mathsf{H}_3}, q_{\mathsf{H}_4}$ *random oracle queries to* H_1*,* H_2*,* H_3*,* H_4*, respectively. Let* $q_{\mathsf{all}} = q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1} + q_{\mathsf{H}_2} + q_{\mathsf{H}_3} + q_{\mathsf{H}_4}$*. Then, there exist efficient adversaries* $\mathcal{C}$*,* $\mathcal{D}$*,* $\mathcal{E}$*,* $\mathcal{F}$*,* $\mathcal{M}$*, such that* $\mathsf{Adv}^{\text{w-ake}}_{\mathsf{SAE}}(\mathcal{A}) \leq 2T_{\mathsf{SAE}}$ *and* $\mathsf{Adv}^{\text{w-ea}}_{\mathsf{SAE}}(\mathcal{A}) \leq T_{\mathsf{SAE}} + q_{\mathsf{se}}/2^L$*, where*

$$\begin{aligned} T_{\mathsf{SAE}} &= \frac{q_{\mathsf{se}}}{2^{\mathrm{H}_\infty(\mathscr{D}_\mathcal{D})}} + 2q_{\mathsf{exe}} \cdot (q_{\mathsf{se}} + q_{\mathsf{H}_2}) \cdot \mathsf{Adv}^{\mathrm{cdh}}_{\mathcal{G},G}(\mathcal{C}) + 2q_{\mathsf{se}}^2 \cdot \mathsf{Adv}^{\mathrm{gdh}}_{\mathcal{G},G}(\mathcal{D}) \\ &+ 4q_{\mathsf{se}} \cdot (q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1})^2 \cdot (\sqrt{\mathsf{Adv}^{\mathrm{wgsdh}}_{\mathcal{G},G}(\mathcal{E})} + \mathsf{Adv}^{\mathrm{gsdh}}_{\mathcal{G},G}(\mathcal{F})) \\ &+ \mathsf{Adv}^{\mathrm{coll}}_{\mathsf{HMAC}(0^L,\cdot)}(\mathcal{M}) + \frac{O(q_{\mathsf{all}}^2)}{2^{256}}. \end{aligned}$$

Proof Sketch. Our proof follows a similar idea sketched in the BPR paper [BPR00], but with additional subtleties, which we summarize later as proof challenges.

The adversary $\mathcal{A}$ against our sPAKE security can be viewed as trying to guess a target server password. We define a "bad event" such that if it does not occur then one can ensure the following :

(1) Each $\mathcal{A}$'s guess (that must involve an active interaction with the target client or server via Send-queries) can eliminate *at most one* password candidate from $\mathcal{D}$;
(2) The only information that $\mathcal{A}$ gets so far about the password is that its previous guesses are wrong.

This "ideal environment" for adversarial password guessing is often referred to as the *information-theoretic game*. In this game, one can bound $\mathcal{A}$'s winning probability by $q_{\mathsf{se}}/2^{\mathrm{H}_\infty(\mathscr{D}_\mathcal{D})}$, where q_{se} bounds the number of sessions the adversary actively interacted with to guess the password.

Then, we need to bound the probability of the "bad event" by mainly relying on the hardness of several Diffie–Hellman problems. We establish this by dividing sessions into five disjoint cases:

- *Execute Case* (created by Execute-queries), bounded under CDH;
- *Paired Case* (created by two sessions using a password of the same server), bounded under GDH;
- *Cross-Server Paired Case* (created by two sessions using passwords of different servers), bounded under GDH;
- *Malicious-Client Case* (created by the adversary interacting with a single server session), bounded under wGSDH;
- *Malicious-Server Case* (created by the adversary interacting with a single client session), bounded under GSDH.

The full proof is provided in Appendix C.2. □

Proof Challenges. We now outline the proof challenges arising from SAE's special design and our approaches to addressing them.

CDH failing to handle the Paired Case. While the *Paired Case* was usually handled in a similar way to the Execute Case under CDH (e.g., [BCP04,LS15]), the concurrent (unordered) SAE Confirm messages in SAE render the simulation infeasible when embedding a CDH challenge (X, Y) in the *Paired Case*. For instance, a server session embedded with Y may output its Confirm message immediately upon receiving the SAE Commit message from the client session embedded with X, thereby deemed as in the Paired Case. However, the client session is not yet paired and may later accept an adversarial Commit message and fall into the *Malicious-Server Case*, where a corruption query could force the computation of K in simulation, which is infeasible under the CDH embedding. An analogous situation may also arise between two client sessions due to the role confusion attack. To overcome this, the simulator relies on the GDH assumption rather than CDH, introducing a DDH oracle to ensure a faithful simulation.

Tight bound for handling online dictionary attacks. Unlike the analysis of the simplified Dragonfly variant [LS15], we aim for a *tight* non-negligible term that bounds the online dictionary attacks, i.e., $q_{\mathsf{se}}/2^{\mathrm{H}_\infty(\mathscr{D}_\mathcal{D})}$ without any constant factor before q_{se}. To this end, our proof is very careful in arranging the game hops to avoid unnecessary constant factors. As part of this effort, we classify the *Cross-Server Paired Case*, which is intuitively easier to handle than the last two malicious cases as the involved SAE Commit messages still both come from honest sessions. With this case treated separately in our proof, we can show that the non-negligible term is even tighter than $q_{\mathsf{se}}/2^{\mathrm{H}_\infty(\mathscr{D}_\mathcal{D})}$ by dividing the at most q_{se} manipulated sessions into four categories according to our last four session cases. This is because each pair of sessions considered in the *Cross-Server Paired Case* can be shown to contribute only one password guess. To the best of our knowledge, this type of tight bound was never shown in the literature.

Session key collisions across independent sessions. The SAE session key is derived as $pmk = \mathsf{KDF}(\mathsf{HMAC}(0^L, [K]_x), ctx)$, where K is the Diffie–Hellman (DH) shared secret and ctx is the sum of exchanged scalars as described in Section 4. Unlike typical PAKE protocols (e.g., EKE2 [BPR00], Dragonfly [LS15], CPace [HL19], etc.), SAE does *not* incorporate the session transcript into the session key derivation. As a result, whenever two independent sessions obtain the same pair $([K]_x, ctx)$, they inevitably derive identical session keys. Actually, our role confusion attack on SAE is largely due to this special key derivation design. Our proof resolves this by explicitly accounting for potential collisions on $([K]_x, ctx)$.

Additional steps after keyseed derivation complicating forward secrecy proof. In SAE, the password is used to derive a group generator, which typically leads to security proofs under assumptions like DIDH [Mac01] and GSDH [PW17] (e.g., DIDH for Dragonfly [LS15] and GSDH for VTBPEKE [PW17]). These assumptions prevent the adversary from actively testing many passwords in one attempt, rather than hiding the DH shared secret K after corruption. Therefore, when forward secrecy is considered, a corruption query may allow the adversary to recover a previously established K, which requires the challenger to maintain simulation consistency before and after corruption. For instance, if the adversary first obtains a confirmation value con_C from an uncorrupted client and later corrupts it, then under GSDH it may recover the correct K, compute con_C by itself, and verify it against the simulated one. In the simplified Dragonfly variant analysis [LS15], such simulation consistency can be handled straightforwardly, since the confirmation values are direct outputs of a hash function, modeled as a random oracle, on the DH shared secret and other inputs. In contrast, SAE employs additional functions, such as KDF and HMAC, after deriving the keyseed from K via a random oracle. To ensure consistent simulation, we model them as random oracles; similarly, in our SAE-PK security proof, AES-SIV is modeled as an ideal cipher.

5.3 SAEv2 and SAEv3 for Stronger Security

In order to prevent role confusion attacks and achieve the desired security guarantees, we introduce two enhanced SAE variants, named SAEv2 and SAEv3, which are both simpler than the original SAE.[4]

SAEv2. Our first proposal is quite straightforward: just fix the order of input MAC addresses used to derive val (see lines 7 and 14 of Fig. 2) by computing

$$\text{either } val \leftarrow \mathsf{HMAC}(0^L, \mathrm{MAC}_C|\mathrm{MAC}_S) \text{ or } val \leftarrow \mathsf{HMAC}(0^L, \mathrm{MAC}_S|\mathrm{MAC}_C)$$

instead of $val \leftarrow \mathsf{HMAC}(0^L, \max(\mathrm{MAC}_C, \mathrm{MAC}_S)|\min(\mathrm{MAC}_C, \mathrm{MAC}_S))$. However, this fix does not prevent two client STAs with the same MAC address from establishing a connection, so SAEv2 further requires a client STA to drop beacon messages that carry its own MAC address. This behavior is already implemented in practice, e.g., in the Linux kernel (which we tested on version 6.16.8).

SAEv3. Our second proposal is also very simple: just skip the ctx computation (see line 23 of Fig. 2) and use s_C instead, i.e., directly compute

$$kck|pmk \leftarrow \mathsf{KDF}(seed, \texttt{lbl}|s_C, L + 256)$$

instead of $kck|pmk \leftarrow \mathsf{KDF}(seed, \texttt{lbl}|ctx, L + 256)$. Unlike SAEv2, our SAEv3 does not have to handle duplicate MAC addresses and is hence more robust. We note that replacing the above s_C with s_S is not recommended, which leads to a much looser security bound; see Appendix C.4 for details.

[4] We remark that our SAEv2 and SAEv3 are designed to address role confusion attacks on WPA3, but the *asymmetric* features they introduce are not fully consistent with the original SAE design for other use cases, e.g., mesh networks [IEE25,Har08a].

Password Generation Algorithm $\mathsf{pwGen}(pp, \mathrm{SSID}, pk)$:

1 : $(\mathsf{sec}, \mathsf{pwl}) \leftarrow pp$; $\mathsf{M} \xleftarrow{\$} \{0,1\}^{128}$
2 : **while** $\mathsf{LEFT}(\mathsf{SHA}\text{-}L(\mathrm{SSID}|\mathsf{M}|pk), 0, 8{\cdot}\mathsf{sec}) \neq 0$:
3 : $\quad \mathsf{M} \leftarrow (\mathsf{M}+1) \bmod 2^{128}$
4 : $\mathsf{fp} \leftarrow \mathsf{LEFT}(\mathsf{SHA}\text{-}L(\mathrm{SSID}|\mathsf{M}|pk), 8{\cdot}\mathsf{sec}, 19{\cdot}\mathsf{pwl}/4{-}5)$
5 : $\mathsf{pw} \leftarrow \mathsf{Encode}(\mathsf{fp}, \mathsf{sec}, \mathsf{pwl})$
6 : **return** $(\mathsf{pw}, \mathsf{M})$

Public-Key Verification Algorithm $\mathsf{pkVf}(\mathrm{SSID}, pk, \mathsf{M}, \mathsf{pw})$:

1 : $(\mathsf{fp}, \mathsf{sec}, \mathsf{pwl}) \leftarrow \mathsf{Decode}(\mathsf{pw})$; $\mathsf{fp}_{\mathsf{exp}} \leftarrow 0^{8\cdot\mathsf{sec}} | \mathsf{fp}$
2 : $\mathsf{fp}' \leftarrow \mathsf{LEFT}(\mathsf{SHA}\text{-}L(\mathrm{SSID}|\mathsf{M}|pk), 0, 8{\cdot}\mathsf{sec}{+}19{\cdot}\mathsf{pwl}/4{-}5)$
3 : **if** $\mathsf{fp}' = \mathsf{fp}_{\mathsf{exp}}$ **then return** 1
4 : **else return** 0

Fig. 5. The SAE-PK password generation algorithm (left) and public-key verification algorithm (right). The public parameters pp consist of a security parameter $\mathsf{sec} \in \{3, 5\}$ and a password length pwl that is a multiple of 4, such that $8{\cdot}\mathsf{sec}+19{\cdot}\mathsf{pwl}/4-5 \leq L$ holds, where L is the same public parameter as in SAE (Fig. 2). The function $\mathsf{SHA}\text{-}L$ denotes the same underlying hash function used in SAE with the output size L. The function $\mathsf{LEFT}(s, j, k)$ returns the k consecutive bits of a string s starting from the position j from left to right. The function Encode (resp. Decode) maps a bitstring (resp. a character string) to a character string (resp. a bitstring). The computed fingerprint fp is of the length $8{\cdot}\mathsf{sec} + 19{\cdot}\mathsf{pwl}/4 - 5$ bits, and the encoded password pw (when excluding hyphen separators) is of the length $5{\cdot}\mathsf{pwl}$ bits, i.e., pwl characters (5-bit each) from the alphabet $\{a, b, \ldots, z, 2, 3, 4, 5, 6, 7\}$.

Security of SAEv2 and SAEv3. We prove that SAEv2 and SAEv3 both achieve the desired AKE and EA security. Their theorem statements are very similar to that of Theorem 1 and establish the same security bounds, but for the stronger notions of AKE and EA security. We omit them here and refer to Theorems 7 and 8 for the full statements and Appendices C.3 and C.4 for the full proofs.

6 The SAE-PK Protocol and its Security

In some practical scenarios, an attacker can easily obtain an AP's password and SSID. For example, public networks (e.g., in coffee shops) often make these AP credentials available to many customers, sometimes by posting them on a wall. With them, an attacker can set up a rogue AP and trick clients into connecting to it (i.e., mounting an evil twin attack [GBL+10,KH18,GV24]). Indeed, apart from the shared password, an AP typically has no other means to authenticate itself to clients.

To mitigate the above attack, WPA3 introduced the SAE-PK protocol, which enables explicit AP authentication by provisioning the AP with a signature key pair and using the public verification key to derive a fingerprint as the shared password. This authentication mechanism provides password-compromise resistance: An attacker cannot impersonate the AP even if it learns the password. This is captured by our sPAKE model as *insider* security, where the adversary is allowed to corrupt the password via the $\mathsf{iCorrupt}$ oracle.

6.1 Description of SAE-PK

Now, we describe the SAE-PK protocol as specified in the WPA3 standard [All25]. SAE-PK extends SAE with several additional components highlighted in blue in Fig. 2. Among these, SAE-PK introduces two new algorithms: the password (or fingerprint) generator pwGen (see the first blue line of Fig. 2) and the public-key verifier pkVf (see line 32 of Fig. 2); the detailed procedures are shown in Fig. 5.

Algorithms for Password Generation and Public-Key Verification. We now describe these algorithms, which employ an encoding algorithm Encode and a decoding algorithm Decode. The Encode algorithm takes as input a fingerprint fp and two public parameters $(\mathsf{sec}, \mathsf{pwl})$, where the security parameter $\mathsf{sec} \in \{3, 5\}$ and the password length pwl is a multiple of 4, and outputs a human-readable and human-writable password pw of pwl characters (when excluding hyphen separators), with characters in the alphabet $\{a, b, \ldots, z, 2, 3, 4, 5, 6, 7\}$. Conversely, Decode takes as input a password pw and outputs a tuple $(\mathsf{fp}, \mathsf{sec}, \mathsf{pwl})$. The details of Encode and Decode are provided in Appendix B.

The password generation algorithm pwGen is executed by the AP prior to connection establishment, as illustrated on the left side of Fig. 5. It takes as input the public parameters pp, a network name SSID, and a public key pk, and proceeds as follows. First, a 128-bit modifier M is randomly sampled and repeatedly incremented modulo 2^{128} until the first $8 \cdot \mathsf{sec}$ bits of the hash output $\mathsf{SHA}\text{-}L(\mathrm{SSID}|\mathsf{M}|pk)$ are all zeros. Next, the leftmost $19 \cdot \mathsf{pwl}/4 - 5$ bits immediately to the right of the $8 \cdot \mathsf{sec}$-bit 0s are truncated from the hash output to form the fingerprint fp, which is then passed to the Encode algorithm to derive an encoded password pw. Finally, the password-modifier pair $(\mathsf{pw}, \mathsf{M})$ is returned by the algorithm.

The public-key verification algorithm pkVf is executed by the STA to verify a public key pk received from the AP, as illustrated on the right side of Fig. 5. It takes as input the network name SSID, the modifier M, the password pw, and the public key pk, and proceeds as follows. First, the algorithm calls Decode on the input password to derive a decoded fingerprint fp with the public parameters $(\mathsf{sec}, \mathsf{pwl})$; with these values, it prepends $8 \cdot \mathsf{sec}$ zeros to fp to construct the expected fingerprint $\mathsf{fp}_{\mathsf{exp}}$. Then, the algorithm obtains another fingerprint fp' by directly computing $\mathsf{SHA}\text{-}L(\mathrm{SSID}, \mathsf{M}, pk)$ and truncating the derived hash value to the leftmost $8{\cdot}\mathsf{sec} + 19{\cdot}\mathsf{pwl}/4 - 5$ bits. Finally, it compares fp' with $\mathsf{fp}_{\mathsf{exp}}$ and outputs 1 if they match and 0 otherwise.

Authentication Process Extensions. With the pwGen and pkVf algorithms, SAE-PK extends the SAE authentication process as follows.

During setup, the AP uses the key generation algorithm KGen to generate a signature key pair (pk, sk) of key length at least 256 bits, and then generates the password pw and the modifier M using the pwGen algorithm. The STA may be additionally provided with an AP's public key, denoted by $\overline{pk}$, together with the password pw. The protocol then follows SAE up to the key derivation step (see line 24 of Fig. 2). At this step, both entities now use KDF to derive three keys instead of two, and the key label `lbl` in the KDF input is changed from "`SAE KCK and PMK`" to "`SAE-PK keys`" for SAE-PK. The additional key kek, referred to as the key encryption key, is L-bit long and used for later encryption.

After key derivation, as shown in lines 25 to 27 of Fig. 2, the AP computes two additional values that are not present in SAE: a ciphertext c and a signature σ. The former is obtained by encrypting the modifier M with the $\mathsf{AES\text{-}SIV}$ scheme [Har08b], using the derived key encryption key kek. The signature σ is derived by signing the message $CE_S|CE_C|s_S|s_C|\mathsf{M}|pk|\mathrm{MAC}_S|\mathrm{MAC}_C$ with the ECDSA scheme [ISO18]. Concretely, this message is first hashed with $\mathsf{SHA}\text{-}L$ to obtain a digest h, which is then signed with sk to derive a signature σ; the latter step is denoted by $\sigma \leftarrow \mathsf{ECDSAcore.Sign}(sk, h)$. Here, we split the ECDSA signing procedure into two parts rather than treating it as a single algorithm, as our security analysis requires explicitly modeling the hash function $\mathsf{SHA}\text{-}L$ as a random oracle. After deriving the signature, the AP sends the STA a Confirm message carrying pk, c, σ, together with the SAE confirmation value con_S.

Upon receiving the above message, the STA first verifies the confirmation value con_S as in SAE, and then verifies the public key pk and the signature σ, as shown in lines 31 to 34 of Fig. 2. For public key verification, the STA first checks whether a given public key $\overline{pk}$ matches the received pk, and discards the received Confirm message if it does not. Otherwise, it decrypts the ciphertext c with the $\mathsf{AES\text{-}SIV}$ scheme using kek to recover the modifier M, and then calls pkVf on input $(\mathrm{SSID}, pk, \mathsf{M}, \mathsf{pw})$ to verify pk. Then, for signature verification, the STA first computes the hash digest h in the same way as the AP, and then verifies the signature σ with pk; the latter is denoted by $\mathsf{ECDSAcore.Vf}(pk, h, \sigma)$. If either of the above two verifications fails, the Confirm message is also discarded.

The remaining procedures for both entities are the same as in SAE.

6.2 The Randomized Fingerprinting Primitive

In this section, we formalize a cryptographic primitive that we call *randomized fingerprinting (RF)*, and define its syntax and desired security properties. This abstraction enables a formal

Game $\mathsf{2\text{-}PRE}^{\mathcal{A}}_{\mathsf{RF}}$	Oracle $\mathcal{O}_{pp}(id, m)$
$\mathrm{win} \leftarrow 0,\ \mathcal{Q} \leftarrow \varnothing,\ (m', s') \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{pp}}$	$(t, s) \xleftarrow{\$} \mathsf{Gen}(pp, id, m)$
for $(id, m, t, s) \in \mathcal{Q}$ **do**	$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(id, m, t, s)\}$
if $\mathsf{Vf}(id, m', s', t) = 1 \wedge m \neq m'$ **do** $\mathrm{win} \leftarrow 1$	**return** t
return win	

Fig. 6. Second preimage resistance game for a randomized fingerprinting scheme RF.

Game $\mathsf{REAL}^{\mathcal{A}}_{\mathsf{RF}}$	Oracle $\widetilde{\mathcal{O}}_{pp}(id, m)$
$\mathcal{Q} \leftarrow \varnothing$	**if** $(id, m) \in \mathcal{Q}$ **return** $\perp$
$b' \xleftarrow{\$} \mathcal{A}^{\widetilde{\mathcal{O}}_{pp}}$	$(t, s) \xleftarrow{\$} \mathsf{Gen}(pp, id, m)$
return b'	$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(id, m)\}$
	return t

Game $\mathsf{IDEAL}^{\mathcal{A}}_{\mathsf{RF}}$	Oracle $\widetilde{\mathcal{O}}_{pp}(id, m)$
$\mathcal{Q} \leftarrow \varnothing$	**if** $(id, m) \in \mathcal{Q}$ **return** $\perp$
$b' \xleftarrow{\$} \mathcal{A}^{\widetilde{\mathcal{O}}_{pp}}$	$t \xleftarrow{\$} \mathcal{T}_{pp}$
return b'	$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(id, m)\}$
	return t

Fig. 7. Fingerprint indistinguishability games for a randomized fingerprinting scheme RF: the "real" game (**left**) and the "ideal" game (**right**).

analysis of the security guarantees provided by the password generation and public-key verification algorithms of SAE-PK, and might also be of independent interest.

Syntax. A randomized fingerprinting scheme RF is defined as a pair of algorithms $(\mathsf{Gen}, \mathsf{Vf})$ and is associated with an identifier space $\mathcal{I} \subseteq \{0,1\}^*$, message space $\mathcal{M} \subseteq \{0,1\}^*$, fingerprint space $\mathcal{T} \subseteq \{0,1\}^*$, salt space $\{0,1\}^r$, and several sets of public parameters. Its randomized fingerprint generation algorithm Gen takes as input a set of public parameters pp, an identifier $id \in \mathcal{I}$, a message $m \in \mathcal{M}$, and outputs a fingerprint $t \in \mathcal{T}$ with a salt $s \in \{0,1\}^r$. Its deterministic verification algorithm Vf takes as input an identifier $id \in \mathcal{I}$, a message $m \in \mathcal{M}$, a salt $s \in \{0,1\}^r$, a fingerprint $t \in \mathcal{T}$, and outputs a bit $b \in \{0,1\}$ indicating whether the verification is successful or not. Here, Vf is not taking the public parameters pp as input, since they can typically be "decoded" from the fingerprint. Correctness imposes that, for any given set of public parameters pp, any $id \in \mathcal{I}$ and $m \in \mathcal{M}$, it holds that: $\Pr[\mathsf{Vf}(id, m, s, t) = 1 | (t, s) \xleftarrow{\$} \mathsf{Gen}(pp, id, m)] = 1$.

Security. We define two security notions for an RF scheme RF as follows.

First, we define its *second preimage resistance* security, which is similar to that of hash functions; the corresponding security game $\mathsf{2\text{-}PRE}^{\mathcal{A}}_{\mathsf{RF}}$ is shown in Fig. 6. Given a set of public parameters pp, the adversary $\mathcal{A}$ is given access to an oracle $\mathcal{O}_{pp}$ that takes id, m as input, runs $(t, s) \xleftarrow{\$} \mathsf{Gen}(pp, id, m)$ and outputs t. In the end, $\mathcal{A}$ outputs a message-salt pair (m', s'). $\mathcal{A}$ wins if and only if there exists some previous $\mathcal{O}_{pp}(id, m)$ query that output t such that $\mathsf{Vf}(id, m', s', t) = 1$ and $m \neq m'$. The advantage measure $\mathsf{Adv}^{\text{2-pre}}_{\mathsf{RF},pp}(\mathcal{A})$ is defined as $\mathcal{A}$'s winning probability in the game: $\mathsf{Adv}^{\text{2-pre}}_{\mathsf{RF},pp}(\mathcal{A}) = \Pr[\mathsf{2\text{-}PRE}^{\mathcal{A}}_{\mathsf{RF}} \Rightarrow 1]$. We say that RF (with public parameters pp) is secure against second preimage attacks if $\mathsf{Adv}^{\text{2-pre}}_{\mathsf{RF},pp}(\mathcal{A})$ is sufficiently small for any efficient $\mathcal{A}$.

Then, an RF scheme is also expected to generate fingerprints with sufficient entropy, e.g., to thwart online dictionary attacks, for which we define a so-called *fingerprint indistinguishability* notion by considering two games shown in Fig. 7. Given a set of public parameters pp, in both games the adversary $\mathcal{A}$ is given access to a fingerprint generation oracle $\widetilde{\mathcal{O}}_{pp}$ that takes id, m as input and outputs a fingerprint t, but $\mathcal{A}$ cannot repeat its oracle query inputs. In the "real" game $\mathsf{REAL}^{\mathcal{A}}_{\mathsf{RF}}$, the oracle runs $(t, s) \xleftarrow{\$} \mathsf{Gen}(pp, id, m)$ and outputs t; in the "ideal" game $\mathsf{IDEAL}^{\mathcal{A}}_{\mathsf{RF}}$, the oracle outputs a fingerprint sampled independently and uniformly at random from a fingerprint space $\mathcal{T}_{pp} \subseteq \mathcal{T}$, which may vary according to the public parameters pp. In the end, $\mathcal{A}$ outputs a bit b' guessing which game it was in. The advantage measure $\mathsf{Adv}^{\text{ind}}_{\mathsf{RF},\mathcal{T}_{pp}}(\mathcal{A})$ is defined as $\mathcal{A}$'s ability to distinguish the two games: $\mathsf{Adv}^{\text{ind}}_{\mathsf{RF},\mathcal{T}_{pp}}(\mathcal{A}) = \Pr[\mathsf{REAL}^{\mathcal{A}}_{\mathsf{RF}} \Rightarrow 1] - \Pr[\mathsf{IDEAL}^{\mathcal{A}}_{\mathsf{RF}} \Rightarrow 1]$. We say that RF (with public parameters pp) generates *indistinguishable* fingerprints if the advantage

$\mathsf{Adv}^{\mathrm{ind}}_{\mathsf{RF}}(\mathcal{A})$ is sufficiently small for any efficient $\mathcal{A}$. In this case, the generated fingerprints have a min-entropy of approximately $\log_2 |\mathcal{T}_{pp}|$, i.e., the min-entropy of a uniform random distribution over the fingerprint space $\mathcal{T}_{pp}$.

6.3 Security of SAE-PK

Security of SAE-PK's Public-Key-Based Password Generation. First, by viewing SSID as the identifier, *pk* as the message, and M as the salt, we have the password generation and public-key verification algorithms pwGen and pkVf of SAE-PK together constitute an RF scheme that we call *public-key-based password generation (PPG)* and denote it by $\mathsf{PPG} = (\mathsf{pwGen}, \mathsf{pkVf})$.

Given a set of public parameters $pp = (\mathsf{sec}, \mathsf{pwl})$, the following two theorems (with proofs in Appendix D) show the security guarantees of PPG (with any given set of public parameters pp) with respect to second-preimage resistance and fingerprint indistinguishability.

Theorem 2. *Let the underlying* SHA-L *in* PPG *(see Fig. 5) be modeled as a random oracle* H*. For any efficient adversary* $\mathcal{A}$ *making* q_{id} *(resp.* $q_{\mathsf{H}id}$*) queries to its* $\mathcal{O}_{pp}$ *oracle (resp. random oracle) with id as the queried identifier, we have*

$$\mathsf{Adv}^{\text{2-pre}}_{\mathsf{PPG},pp}(\mathcal{A}) \leq \frac{\sum_{id} q_{id}(q_{\mathsf{H}id}+1)}{2^{8\cdot\mathsf{sec}+19\cdot\mathsf{pwl}/4-5}},$$

where $8\cdot\mathsf{sec} + 19\cdot\mathsf{pwl}/4 - 5 \geq 76$.

Theorem 3. *Let the underlying* SHA-L *in* PPG *(see Fig. 5) be modeled as a random oracle* H*. For any efficient adversary* $\mathcal{A}$ *making* q_{H} *random oracle queries, we have*

$$\mathsf{Adv}^{\mathrm{ind}}_{\mathsf{PPG},\mathcal{T}_{pp}}(\mathcal{A}) \leq \frac{q_{\mathsf{H}}}{2^{120-8\cdot\mathsf{sec}}} + \frac{q_{\mathsf{H}}}{2^{63+8\cdot\mathsf{sec}}},$$

where $\log_2 |\mathcal{T}_{pp}| = 19\cdot\mathsf{pwl}/4 - 5 \geq 52$.

Remark on parameter choices. We note that, as shown in Theorem 2, the PPG scheme of SAE-PK is actually not secure against second preimage attacks when the pp parameters are small (e.g., $\mathsf{sec} = 3$ and $\mathsf{pwl} = 12$) and only a few identifiers (i.e., SSIDs) are targeted (which in practice means many servers set up the same SSID), as the security bound then degrades to a birthday bound. This result matches the second-preimage attacks shown in [VR24], which shows that a precomputation attack can reduce the time to find a second preimage of an SAE-PK password from 48 CPU years (as claimed in the WPA3 specification [All25]) to an amortized time of fewer than 12 days. They recommend using large parameters, such as $\mathsf{sec} = 5$ and $\mathsf{pwl} \geq 16$, to mitigate this issue. In order to achieve provable 80-bit security against second preimage attacks in the worst case, we recommend choosing $pp = (\mathsf{sec}, \mathsf{pwl})$ parameters such that $8\cdot\mathsf{sec} + 19\cdot\mathsf{pwl}/4 - 5 > 160$, e.g., $(\mathsf{sec} = 3, \mathsf{pwl} \geq 32)$ or $(\mathsf{sec} = 5, \mathsf{pwl} \geq 28)$. Then, Theorem 3 shows that passwords generated from PPG has min-entropy approximately equal to 52 for the smallest pp parameters $(\mathsf{sec} = 3, \mathsf{pwl} = 12)$. For a larger sec value, the first term in the security bounds becomes worse, so we recommend increasing pwl while keeping $\mathsf{sec} = 3$ when higher password entropy is desired. To sum up, we recommend choosing $pp = (\mathsf{sec} = 3, \mathsf{pwl} \geq 32)$.

Security of Overall SAE-PK. Based on the above results, we state the following theorem (with full security bounds in Theorem 9), which establishes the iAKE and iEA security guarantees of SAE-PK. Since in general such security requires only reasonable password min-entropy rather than fingerprint indistinguishability (i.e., password distribution indistinguishable from uniform), the following theorem does not rely on Theorem 3. Instead, it directly specifies the min-entropy $\mathrm{H}_\infty(\mathscr{D}_{pp})$ of the password distribution derived in SAE-PK, with respect to the password dictionary $\mathcal{T}_{pp}$ for a given set of pp parameters.

Theorem 4. *Let* $\mathrm{H}_\infty(\mathscr{D}_{pp})$ *denote the min-entropy of the password distribution* $\mathscr{D}_{pp}$ *over* $\mathcal{T}_{pp}$. *Let* H_1, H_2, H_3, H_4 *be the same random oracles as defined in Theorem 1. Let* SHA-L *(see lines 26 and 33 of Fig. 2) be modeled as a random oracle* H_5 *and* AES-SIV *(see lines 25 and 31 of Fig. 2) as an ideal cipher. Let* $\mathcal{A}$ *be any efficient adversary that makes at most* q_{se} Send*-queries,* q_{exe} Execute*-queries,* q_{ic} *ideal cipher queries, and* $q_{\mathsf{H}_1}, q_{\mathsf{H}_2}, q_{\mathsf{H}_3}, q_{\mathsf{H}_4}, q_{\mathsf{H}_5}$ *random oracle queries to* H_1, H_2, H_3, H_4, H_5, *respectively. Let* $q_{\mathsf{all}} = q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{ic}} + q_{\mathsf{H}_1} + q_{\mathsf{H}_2} + q_{\mathsf{H}_3} + q_{\mathsf{H}_4}$. *Then, there exist efficient adversaries* $\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$, $\mathcal{F}$, $\mathcal{I}$, $\mathcal{J}$, $\mathcal{M}$, *such that* $\mathsf{Adv}^{\text{i-ake}}_{\mathsf{SAE\text{-}PK}}(\mathcal{A}) \leq 2T_{\mathsf{SAE\text{-}PK}}$ *and* $\mathsf{Adv}^{\text{i-ea}}_{\mathsf{SAE\text{-}PK}}(\mathcal{A}) \leq T_{\mathsf{SAE\text{-}PK}} + q_{\mathsf{se}}/2^L$, *where*

$$\begin{aligned} T_{\mathsf{SAE\text{-}PK}} &= \frac{q_{\mathsf{se}}}{2^{\mathrm{H}_\infty(\mathscr{D}_{pp})}} + \frac{4q_{\mathsf{H}_5}}{2^{120-8\cdot\mathsf{sec}}} + \frac{4q_{\mathsf{H}_5}}{2^{63+8\cdot\mathsf{sec}}} + 2q_{\mathsf{exe}} \cdot (q_{\mathsf{se}} + q_{\mathsf{H}_2}) \cdot \mathsf{Adv}^{\text{cdh}}_{\mathcal{G},G}(\mathcal{C}) \\ &+ 4q_{\mathsf{se}}^2 \cdot \mathsf{Adv}^{\text{gdh}}_{\mathcal{G},G}(\mathcal{D}) + 4q_{\mathsf{se}} \cdot (q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1})^2 \cdot \sqrt{\mathsf{Adv}^{\text{wgsdh}}_{\mathcal{G},G}(\mathcal{E})} \\ &+ 2q_{\mathsf{se}} \cdot (q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1})^2 \cdot \mathsf{Adv}^{\text{gsdh}}_{\mathcal{G},G}(\mathcal{F}) + 8 \cdot \mathsf{Adv}^{\text{2-pre}}_{\mathsf{PPG},pp}(\mathcal{I}) \\ &+ q_{\mathsf{se}}^2 \cdot \mathsf{Adv}^{\text{uf-cma}}_{\mathsf{ECDSA}}(\mathcal{J}) + \mathsf{Adv}^{\text{coll}}_{\mathsf{HMAC}(0^L,\cdot)}(\mathcal{M}) + \frac{12q_{\mathsf{H}_5}}{2^{128}} + \frac{O(q_{\mathsf{all}}^2)}{2^{256}}. \end{aligned}$$

Proof Sketch. The proof proceeds similarly to that of SAE, except that we need to further rely on the second preimage resistance of the RF scheme PPG (introduced by SAE-PK to generate public-key-based passwords) and unforgeability of the ECDSA signature scheme. However, signatures are unforgeable but may not be private, so they may carry information about the password. This yields the two red-colored terms that allow for theoretical offline dictionary attacks.

The full proof is provided in Appendix C.5. □

Remark on offline dictionary attacks. The red-colored terms in the above security bound closely resemble those in Theorem 3 and arise from an offline-dictionary attack that attempts to recover the password by brute-forcing the possible modifiers used in honest pwGen executions. Although these terms are not very large, they are much worse than the other negligible terms in the security bounds and should be avoided for a well-designed SAE-PK protocol as an enhanced PAKE. We therefore propose a minimal change in SAE-PK to fix this issue in the next section. We also remark that such attacks can be mitigated by increasing the modifier size (e.g., from 128 to 256), which can effectively reduce the red-colored terms when appropriate adjusting parameters are chosen, as detailed in Appendix D.2.

6.4 SAE-PK+ for Stronger Security

Now, we introduce a simple enhanced SAE-PK variant that we call SAE-PK+, which is simpler than SAE-PK and fully eliminates offline dictionary attacks.

SAE-PK+. Our proposed scheme SAE-PK+ differs from SAE-PK in only one aspect: It omits the modifier M when computing the SHA-L hash digest for signing (see lines 26 and 33 of Fig. 2), instead just computing

$$h \leftarrow \mathsf{SHA}\text{-}L(CE_S|CE_C|s_S|s_C|pk|\mathrm{MAC}_S|\mathrm{MAC}_C).$$

Security of SAE-PK+. We prove that SAE-PK+ achieves iAKE and iEA security. Due to high similarity between this theorem statement and that of Theorem 4, we omit it here and refer to Theorem 10 and Appendix C.6 for the complete statement and proof. In particular, SAE-PK+ has the same security bounds as SAE-PK, except that the red-colored terms in Theorem 4 disappear, the coefficient of $\mathsf{Adv}^{\text{2-pre}}_{\mathsf{PPG},pp}(\mathcal{I})$ is reduced from 8 to 4, and the coefficient of $q_{\mathsf{H}_5}/2^{128}$ is reduced from 12 to 8.

7 Privacy of SAE-Family Protocols

In contrast to wired communication, Wi-Fi transmits all messages over the air, making them accessible to any nearby device, even if the messages are intended for a specific recipient. Consequently, all devices within range can passively observe the communication and potentially extract information. This behavior can be exploited for device tracking, such as building user profiles, monitoring physical locations, and inferring behavioral patterns [HM10,ZL22].

Although messages transmitted over Wi-Fi are encrypted, they often still contain unencrypted metadata that might reveal the identity of the sender or receiver. This metadata includes physical-layer characteristics such as signal strength [CCLK05,NMSH18] and transmission frequency [Fre15], as well as parts of the cryptographic transcript, such as the SSID and MAC addresses [VMC+16].

In our analysis, we exclude device tracking based on physical-layer characteristics, assuming appropriate countermeasures are already in place. Instead, we focus on the linkability of the cryptographic transcripts and evaluate the effectiveness of the privacy mechanisms like MAC-address randomization.

In the following, we first define the privacy model for sPAKE protocols and then employ it to analyze the privacy guarantees of all SAE-family protocols considered in this work: SAE, SAEv2, SAEv3, SAE-PK, SAE-PK+.

7.1 sPAKE Privacy Model

Our privacy model is similar to the Bluetooth privacy model proposed in [FS21], but unlike theirs, which considers only passive adversaries, our privacy model allows powerful active adversaries to capture MITM attacks in the Wi-Fi setting. Such powerful adversaries can essentially be viewed as *malicious servers* that know all passwords and auxiliary information and try to track clients.

While the sPAKE security guarantees defined in Section 3 cannot hold against the above strong adversaries, privacy remains achievable. Intuitively, this is because the client identifiers may be randomized and such identifiers could be the only way for the malicious servers to link different sessions of the same client. In our privacy model, three types of client identifiers are considered: (1) *fixed* identifiers that never change, (2) *random* identifiers that refresh per session, and (3) *persistent* identifiers that remain the same per server name but refresh across different server names.

Formally, we employ the same notion of session oracles and states from our sPAKE security model, with the following additional session states:

- $\mathsf{idtype} \in \{\mathsf{fix}, \mathsf{rand}, \mathsf{pers}\}$: the type of a client's identifier, initialized to fix;
- $\mathsf{ikey} \in \{0,1\}^\kappa \cup \{\bot\}$: the *identifier key* (if any) used to randomize client identifiers, initialized to $\bot$ (unset).

Our privacy experiment is defined as follows.

Privacy Experiment. The experiment runs between an adversary $\mathcal{A}$ and a challenger. In the beginning of the experiment, the challenger first samples a random bit $b \xleftarrow{\$} \{0,1\}$ and then takes the same experiment setup procedures as in our sPAKE security model. In addition, the adversary $\mathcal{A}$ assigns all server passwords and auxiliary information $\langle(\mathsf{pw}_S, \mathsf{aux}_S, \overline{\mathsf{aux}}_S)\rangle_{S\in\mathcal{S}}$. Next, $\mathcal{A}$ specifies the identifier types of all clients, and the challenger then samples an independent random identifier key $\mathsf{ikey} \xleftarrow{\$} \{0,1\}^\kappa$ for each client with a persistent identifier (i.e., $\mathsf{idtype} = \mathsf{pers}$).[5] After that, the adversary $\mathcal{A}$ can interact with the challenger by querying the following oracles:

[5] Theoretically, these identifier-related procedures can also be incorporated in the sPAKE security model. We chose not to complicate our security model with such privacy-related aspects, since the way identifiers are generated does not affect the model in any essential way even if the adversary knows all identifier keys. sPAKE parties are authenticated using passwords (and perhaps auxiliary information) rather than identifiers, and these identifiers need not be unique or fixed across sessions.

$\mathsf{SendFix}(C, i, m)$ sends message m to a client session π^i_C, where the input client must have $\mathsf{idtype} = \mathsf{fix}$. The challenger runs $\mathsf{Send}(C, i, m)$ as defined in the sPAKE security model and returns the derived output to $\mathcal{A}$.

$\mathsf{TestSend}(C_0, i_0, C_1, i_1, m)$ sends message m to a client session $\pi^{i_b}_{C_b}$, where b is the secret bit sampled in the beginning of the privacy experiment and both input clients must *not* have $\mathsf{idtype} = \mathsf{fix}$; here both tested sessions were not used before and must always be queried together to $\mathsf{TestSend}$ thereafter to complete the protocol run. The challenger runs $\mathsf{Send}(C_b, i_b, m)$ and returns the derived output to $\mathcal{A}$. Note that this oracle can be queried to test multiple different pairs of sessions from the same client or from distinct clients.

$\mathsf{Corrupt}(C)$ returns to $\mathcal{A}$ the current states of all sessions of client C, including its identifier key ikey. After this query, we say the client C was *corrupted*.

Advantage Measure. An adversary $\mathcal{A}$ against the privacy of a sPAKE protocol sPAKE queries the above oracles, and at the end of the privacy experiment it outputs a bit $b' \in \{0, 1\}$ as its guess for the secret random bit b. We say that $\mathcal{A}$ *wins* if and only if $b' = b$ and the following conditions hold:

(1) No client with $\mathsf{idtype} = \mathsf{pers}$ queried to $\mathsf{TestSend}$ was corrupted;
(2) No client with $\mathsf{idtype} = \mathsf{pers}$ was queried to $\mathsf{TestSend}$ to complete two protocol runs under the same server name (chosen by $\mathcal{A}$ via $\mathsf{TestSend}$ input messages).

The privacy advantage $\mathsf{Adv}^{\mathrm{priv}}_{\mathsf{sPAKE}}(\mathcal{A})$ is defined as $2\Pr[\mathcal{A} \text{ wins}] - 1$.

We remark that the above conditions are used to rule out trivial wins, where the adversary could easily break privacy by identifying (1) a persistent client identifier derived using a corrupted identifier key, or (2) a persistent client identifier reused across sessions under the same server name.

7.2 Privacy Mechanisms and Analysis

Privacy Mechanisms in WPA3. MAC addresses are 48-bit identifiers assigned to a device's network interface during manufacturing. To mitigate device tracking based on these static identifiers, IEEE introduced a privacy-enhancing mechanism [IEE25] in which devices can use randomized MAC addresses instead of their physical ones. These randomized addresses are marked by setting the two least-significant bits of the first byte to 10, indicating a locally administered unicast address. As part of this mechanism, a device replaces its MAC address with a random 46-bit string (excluding two fixed bits). IEEE does not mandate how frequently such addresses must be updated, leaving this choice to the implementation. In practice, this mechanism has been widely adopted: For example, devices running Android version 10 and later enable MAC address randomization by default [And], and a similar feature was introduced in iOS version 16.1 [App]. In this work, we assume all 46 unfixed bits of the address are re-randomized per session.

Modern implementations also support *persistent* MAC randomization, which allows a device to consistently use the same randomized MAC address for each network (e.g., per SSID). Such persistent MAC addresses are required by networks to support features such as automatic login and parental controls.

In this work, we take Android's implementation [And] as an example to perform the privacy analysis of persistent MAC addresses. An Android device samples a secret key $ik \xleftarrow{\$} \{0, 1\}^{256}$ (which corresponds to the identifier key ikey defined in the privacy model) and uses it to compute an HMAC (instantiated with the SHA-256 hash function) value over SSID and potentially other parameters like the security type or fast transition identifier. Since these other parameters are either constant or irrelevant for personal networks (the focus of this work), we assume that SSID is the only input to the MAC-address derivation function. The resulting $\mathsf{HMAC}(ik, \mathrm{SSID})$ output is then truncated to 46 bits to produce the persistent randomized MAC address.

Privacy Results. Equipped with our privacy model, we prove that all SAE-family protocols considered in this work ensure privacy under the above Android implementation [And] for generating persistent MAC addresses, as stated in the following theorem (with the proof provided in Appendix D.3).

Theorem 5. *For any efficient adversary $\mathcal{A}$ making at most q_{ts} $\mathsf{TestSend}$-queries (among which n_{pers} distinct clients with $\mathsf{idtype} = \mathsf{pers}$ were queried to $\mathsf{TestSend}$), there exists an efficient adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathrm{priv}}_{\mathsf{sPAKE}}(\mathcal{A}) \leq 2n_{\mathsf{pers}} \cdot \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{HMAC}}(\mathcal{B}) \leq 4q_{\mathsf{ts}} \cdot \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{HMAC}}(\mathcal{B}),$$

where $\mathsf{sPAKE} \in \{\mathsf{SAE}, \mathsf{SAEv2}, \mathsf{SAEv3}, \mathsf{SAE\text{-}PK}, \mathsf{SAE\text{-}PK+}\}$.

Remark on the privacy guarantees. The WPA3 specification [All25] mentions only passive tracking as the design goal of MAC address randomization:

> MAC privacy enhancements are specified in IEEE Std. 802.11 to mitigate passive tracking based on MAC address.

Our results, however, show that this guarantee can be extended in the case of SAE-family protocols: even if all APs are malicious (knowing all passwords with auxiliary information and assigning all server names) and collaborate, an STA using the random privacy mechanism can still preserve privacy, in the sense that APs cannot link sessions originating from the same STA. Under a well-designed persistent privacy mechanism such as that used in Android, STAs may be linkable within the same Wi-Fi network, but remain unlinkable across distinct networks with different SSIDs. However, sessions from STAs with a fixed MAC address can be trivially linked.

Finally, we remark that the privacy guarantees of the persistent MAC randomization mechanism can be further enhanced by including the AP's unique MAC address in the HMAC computation. This way, STAs cannot be linked across distinct networks, even when they share the same SSID. However, this may slightly affect usability, e.g., it may hinder seamless reconnection across different APs sharing the same network name (SSID).

8 Conclusion and Recommendations

We present the first formal analysis of the SAE and SAE-PK protocols used in WPA3. To this end, we define universal models to capture the SAE-family protocols, prove their security and privacy guarantees, identify attacks, and propose efficient and provably secure fixes. Our recommendations in improving the security and privacy of SAE and SAE-PK are summarized as follows:

- Use SAEv2 or SAEv3 to avoid role confusion attacks;
- Use good public parameters ($\mathsf{sec} = 3, \mathsf{pwl} \geq 32$) for SAE-PK password generation to resist second-preimage attacks and ensure high password entropy;
- Use SAE-PK+ to eliminate the threat of offline dictionary attacks, or mitigate it by increasing the modifier size;
- Use persistent MAC addresses for good privacy (client unlinkability across networks) and usability (e.g., automatic login), or use random ones for stronger privacy (client unlinkability across sessions) but worse usability.

Acknowledgments. We thank the EUROCRYPT 2026 anonymous reviewers for the improvement suggestion on SAEv3 and all anonymous reviewers for the valuable comments. Shan Chen and Kaige Pan were supported by the research start-up grant by the Southern University of Science and Technology awarded to Shan Chen. Olga Sanina was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SFB 1119 – 236615297.

Disclosure of Interests. The authors declare that they have no competing interests relevant to the content of this article.

References

[AHH21] Michel Abdalla, Björn Haase, and Julia Hesse. Security analysis of CPace. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part IV*, volume 13093 of *Lecture Notes in Computer Science*, pages 711–741, Singapore, December 6–10, 2021. Springer, Cham, Switzerland. (Cited on page 6.)

[All25] Wi-Fi Alliance. WPA3™ specification, 2025. Version 3.5. (Cited on pages 3, 4, 11, 17, 20, and 24.)

[And] MAC randomization behavior | Android Open Source Project — source.android.com. https://source.android.com/docs/core/connect/wifi-mac-randomization-behavior. (Cited on pages 4, 23, and 24.)

[AP05] Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 191–208, San Francisco, CA, USA, February 14–18, 2005. Springer Berlin Heidelberg, Germany. (Cited on page 6.)

[App] Privacy features when connecting to wireless networks - Apple Support. https://support.apple.com/guide/security/privacy-features-connecting-wireless-networks-secb9cb3140c/web. (Cited on page 23.)

[BCDP17] Alexandra Boldyreva, Shan Chen, Pierre-Alain Dupont, and David Pointcheval. Human computing for handling strong corruptions in authenticated key exchange. In Boris Köpf and Steve Chong, editors, *CSF 2017: IEEE 30th Computer Security Foundations Symposium*, pages 159–175, Santa Barbara, CA, USA, August 21–25, 2017. IEEE Computer Society Press. (Cited on page 6.)

[BCP04] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. New security results on encrypted key exchange. In Feng Bao, Robert Deng, and Jianying Zhou, editors, *PKC 2004: 7th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2947 of *Lecture Notes in Computer Science*, pages 145–158, Singapore, March 1–4, 2004. Springer Berlin Heidelberg, Germany. (Cited on page 15.)

[BGHJ24] Manuel Barbosa, Kai Gellert, Julia Hesse, and Stanislaw Jarecki. Bare PAKE: Universally composable key exchange from just passwords. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part II*, volume 14921 of *Lecture Notes in Computer Science*, pages 183–217, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland. (Cited on page 6.)

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155, Bruges, Belgium, May 14–18, 2000. Springer Berlin Heidelberg, Germany. (Cited on pages 6, 8, 15, and 16.)

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, Santa Barbara, CA, USA, August 22–26, 1994. Springer Berlin Heidelberg, Germany. (Cited on page 7.)

[CCLK05] Yu-Chung Cheng, Yatin Chawathe, Anthony LaMarca, and John Krumm. Accuracy characterization for metropolitan-scale Wi-Fi localization. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, MobiSys '05, page 233–245, New York, NY, USA, 2005. Association for Computing Machinery. (Cited on page 22.)

[CH14] Dylan Clarke and Feng Hao. Cryptanalysis of the Dragonfly key exchange protocol. *IET Information Security*, 8(6):283–289, November 2014. (Cited on page 11.)

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 404–421, Aarhus, Denmark, May 22–26, 2005. Springer Berlin Heidelberg, Germany. (Cited on page 6.)

[CJ19] Cas Cremers and Dennis Jackson. Prime, order please! Revisiting small subgroup and invalid curve attacks on protocols using Diffie-Hellman. In Stephanie Delaune and Limin Jia, editors, *CSF 2019: IEEE 32nd Computer Security Foundations Symposium*, pages 78–93, Hoboken, NJ, USA, June 25–28, 2019. IEEE Computer Society Press. (Cited on page 10.)

[CNPR22] Cas Cremers, Moni Naor, Shahar Paz, and Eyal Ronen. CHIP and CRISP: Protecting all parties against compromise through identity-binding PAKEs. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 668–698, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland. (Cited on page 5.)

[DABFS20] Daniel De Almeida Braga, Pierre-Alain Fouque, and Mohamed Sabt. Dragonblood is still leaking: Practical cache-based side-channel in the wild. In *Proceedings of the 36th Annual Computer Security Applications Conference*, ACSAC '20, page 291–303, New York, NY, USA, 2020. Association for Computing Machinery. (Cited on page 13.)

[DG21] Nir Drucker and Shay Gueron. Selfie: reflections on TLS 1.3 with PSK. *Journal of Cryptology*, 34(3):27, July 2021. (Cited on page 14.)

[FM21] Marc Fischlin and Arno Mittelbach. An overview of the hybrid argument. Cryptology ePrint Archive, Report 2021/088, 2021. (Cited on page 57.)

[Fre15] Julien Freudiger. How talkative is your mobile device? An experimental study of Wi-Fi probe requests. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '15, pages 1–6, New York, NY, USA, 2015. Association for Computing Machinery. (Cited on page 22.)

[FS21] Marc Fischlin and Olga Sanina. Cryptographic analysis of the bluetooth secure connection protocol suite. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 696–725, Singapore, December 6–10, 2021. Springer, Cham, Switzerland. (Cited on page 22.)

[GBL+10] Harold Gonzales, Kevin Bauer, Janne Lindqvist, Damon McCoy, and Douglas Sicker. Practical defenses for evil twin attacks in 802.11. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pages 1–6, 2010. (Cited on pages 3 and 17.)

[GV24] Héloïse Gollier and Mathy Vanhoef. SSID confusion: Making Wi-Fi clients connect to the wrong network. In *Proceedings of the 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '24, page 156–161, New York, NY, USA, 2024. Association for Computing Machinery. (Cited on pages 3, 13, and 17.)

[Har08a] Dan Harkins. Simultaneous authentication of equals: A secure, password-based key exchange for mesh networks. pages 839–844, 2008. (Cited on pages 3 and 16.)

[Har08b] Dan Harkins. Synthetic Initialization Vector (SIV) Authenticated Encryption Using the Advanced Encryption Standard (AES). RFC 5297, October 2008. (Cited on page 18.)

[Har15] Dan Harkins. Dragonfly Key Exchange. RFC 7664, November 2015. (Cited on page 3.)

[HL19] Björn Haase and Benoît Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):1–48, 2019. (Cited on page 16.)

[HM10] Nathaniel Husted and Steven Myers. Mobile location tracking in metro areas: malnets and others. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010: 17th Conference on Computer and Communications Security*, pages 85–96, Chicago, Illinois, USA, October 4–8, 2010. ACM Press. (Cited on page 22.)

[IEE11] 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 amendment 10: Mesh networking, 2011. (Cited on page 3.)

[IEE12] 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, 2012. (Cited on page 3.)

[IEE25] 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, 2025. (Cited on pages 3, 4, 10, 11, 13, 16, and 23.)

[ISO18] ISO/IEC 14888-3:2018 IT Security techniques – Digital signatures with appendix – Part 3, 2018. Edition 4. (Cited on page 18.)

[Jos06] Simon Josefsson. The Base16, Base32, and Base64 Data Encodings. RFC 4648, October 2006. (Cited on page 28.)

[KBC97] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, February 1997. (Cited on page 11.)

[KH18] Christopher P. Kohlios and Thaier Hayajneh. A comprehensive attack flow model and security analysis for Wi-Fi and WPA3. *Electronics*, 7(11), 2018. (Cited on page 17.)

[KR24] Jonathan Katz and Michael Rosenberg. LATKE: A framework for constructing identity-binding PAKEs. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part II*, volume 14921 of *Lecture Notes in Computer Science*, pages 218–250, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland. (Cited on page 5.)

[LS15] Jean Lancrenon and Marjan Skrobot. On the provable security of the Dragonfly protocol. In Javier Lopez and Chris J. Mitchell, editors, *ISC 2015: 18th International Conference on Information Security*, volume 9290 of *Lecture Notes in Computer Science*, pages 244–261, Trondheim, Norway, September 9–11, 2015. Springer, Cham, Switzerland. (Cited on pages 3, 6, 15, and 16.)

[Mac01] Philip MacKenzie. On the security of the SPEKE password-authenticated key exchange protocol. Cryptology ePrint Archive, Report 2001/057, 2001. (Cited on page 16.)

[NMSH18] Tomoya Nakatani, Takuya Maekawa, Masumi Shirakawa, and Takahiro Hara. Estimating the physical distance between two locations with Wi-Fi received signal strength information using obstacle-aware approach. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2(3), September 2018. (Cited on page 22.)

[OP01] Tatsuaki Okamoto and David Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In Kwangjo Kim, editor, *PKC 2001: 4th International Workshop on*

Theory and Practice in Public Key Cryptography, volume 1992 of *Lecture Notes in Computer Science*, pages 104–118, Cheju Island, South Korea, February 13–15, 2001. Springer Berlin Heidelberg, Germany. (Cited on page 5.)

[PW17] David Pointcheval and Guilin Wang. VTBPEKE: Verifier-based two-basis password exponential key exchange. In Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi, editors, *ASIACCS 17: 12th ACM Symposium on Information, Computer and Communications Security*, pages 301–312, Abu Dhabi, United Arab Emirates, April 2–6, 2017. ACM Press. (Cited on pages 5 and 16.)

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, Konstanz, Germany, May 11–15, 1997. Springer Berlin Heidelberg, Germany. (Cited on page 5.)

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. (Cited on pages 29, 56, and 57.)

[Sun22] Sheng Sun. A chosen random value attack on WPA3 SAE authentication protocol. *Digital Threats: Research and Practice (DTRAP)*, 3(2), February 2022. (Cited on page 11.)

[TH21] Michael Troncoso and Britta Hale. The bluetooth CYBORG: Analysis of the full human-machine passkey entry AKE protocol. In *ISOC Network and Distributed System Security Symposium – NDSS 2021*, Virtual, February 21–25, 2021. The Internet Society. (Cited on page 13.)

[Ver69] Jacobus Verhoeff. Error detecting decimal codes. *Mathematical Centre Tracts*, 29, 1969. (Cited on page 29.)

[VMC+16] Mathy Vanhoef, Célestin Matte, Mathieu Cunche, Leonardo S. Cardoso, and Frank Piessens. Why MAC address randomization is not enough: An analysis of wi-fi network discovery mechanisms. In Xiaofeng Chen, XiaoFeng Wang, and Xinyi Huang, editors, *ASIACCS 16: 11th ACM Symposium on Information, Computer and Communications Security*, pages 413–424, Xi'an, China, May 30 – June 3, 2016. ACM Press. (Cited on page 22.)

[VP17] Mathy Vanhoef and Frank Piessens. Key reinstallation attacks: Forcing nonce reuse in WPA2. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1313–1328, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press. (Cited on page 3.)

[VR20] Mathy Vanhoef and Eyal Ronen. Dragonblood: Analyzing the dragonfly handshake of WPA3 and EAP-pwd. In *2020 IEEE Symposium on Security and Privacy*, pages 517–533, San Francisco, CA, USA, May 18–21, 2020. IEEE Computer Society Press. (Cited on pages 3 and 13.)

[VR24] Mathy Vanhoef and Jeroen Robben. A security analysis of WPA3-PK: Implementation and precomputation attacks. In Christina Pöpper and Lejla Batina, editors, *ACNS 2024: 22nd International Conference on Applied Cryptography and Network Security, Part II*, volume 14584 of *Lecture Notes in Computer Science*, pages 217–240, Abu Dhabi, UAE, March 5–8, 2024. Springer, Cham, Switzerland. (Cited on pages 3 and 20.)

[ZL22] Yue Zhang and Zhiqiang Lin. When good becomes evil: Tracking Bluetooth low energy devices via allowlist-based side channel and its countermeasure. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 3181–3194, Los Angeles, CA, USA, November 7–11, 2022. ACM Press. (Cited on page 22.)

A Cryptographic Building Blocks

In this section, we recall the syntax and security definitions of the necessary cryptographic building blocks used in this work.

A.1 Hash Functions

An (unkeyed) hash function $\mathsf{H} : \{0,1\}^* \to \mathcal{D}$ maps a binary string x of arbitrary length to an element d in the message digest space $\mathcal{D} \subseteq \{0,1\}^*$. We say H is *collision-resistant* if it is computational infeasible for any efficient adversary $\mathcal{A}$ to find two distinct messages x, x' hashing to the same digest d, i.e., $\mathsf{H}(x) = \mathsf{H}(x')$. The collision-resistance advantage of $\mathcal{A}$ is denoted by $\mathsf{Adv}^{\text{coll}}_{\mathsf{H}}(\mathcal{A})$.

A.2 Pseudorandom Functions (PRFs)

For a function $\mathsf{F} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^n$, consider an adversary $\mathcal{A}$ interacting in one of the following two worlds. In the "real" world, $\mathcal{A}$ has oracle access to the function $\mathsf{F}_k(\cdot) = \mathsf{F}(k, \cdot)$,

Encoding Algorithm $\mathsf{Encode}(\mathsf{fp}, \mathsf{sec}, \mathsf{pwl})$:

1 : $\mathsf{pw} \leftarrow \varepsilon$
2 : **if** $pp.\mathsf{sec} = 3$ **then** $b_{\mathsf{sec}} = 1$ **else** $b_{\mathsf{sec}} = 0$
3 : **for** $(i = 0, i < pp.\mathsf{pwl}/4{-}1, i{+}{+})$:
4 : $\quad \mathsf{pw} \leftarrow \mathsf{pw}|\mathsf{Base32}(b_{\mathsf{sec}}|\mathsf{LEFT}(\mathsf{fp}, 8{\cdot}\mathsf{sec}{+}19{\cdot}i, 19))$
5 : $\mathsf{pw} \leftarrow \mathsf{pw}|\mathsf{Base32}(b_{\mathsf{sec}}|\mathsf{LEFT}(\mathsf{fp}, 8{\cdot}\mathsf{sec}{+}19{\cdot}i, 14))$
6 : $\mathsf{pw} \leftarrow \mathsf{AddChecksum}(\mathsf{pw})$
7 : **return** pw

Decoding Algorithm $\mathsf{Decode}(\mathsf{pw})$:

1 : $\mathsf{pwl} \leftarrow |\mathsf{pw}| - \lfloor |\mathsf{pw}|/5 \rfloor, \mathsf{pw} \leftarrow \mathsf{RemoveChecksum}(\mathsf{pw})$
2 : $\mathsf{pw} \leftarrow \mathsf{Base32d}(\mathsf{pw}), \mathsf{fp} \leftarrow \varepsilon$
3 : **if** $\mathsf{LEFT}(\mathsf{pw}, 0, 1) = 1$ **then** $\mathsf{sec} \leftarrow 3$ **else** $\mathsf{sec} \leftarrow 5$
4 : **for** $(i = 0, i < \mathsf{pwl}/4{-}1, i{+}{+})$:
5 : $\quad \mathsf{fp} \leftarrow \mathsf{fp}|\mathsf{LEFT}(\mathsf{pw}, 20{\cdot}i{+}1, 19)$
6 : $\mathsf{fp} \leftarrow \mathsf{fp}|\mathsf{LEFT}(\mathsf{pw}, 20{\cdot}i{+}1, 14)$
7 : **return** $(\mathsf{fp}, \mathsf{sec}, \mathsf{pwl})$

Fig. 8. SAE-PK password encoding (left) and password decoding (right) algorithms. Function $\mathsf{LEFT}(s, j, k)$ returns the k consecutive bits of string s starting from position j from left to right. Function $\mathsf{Base32}$ is the Base32 encoding algorithm defined in RFC 4648 [Jos06], which maps a bitstring to a lowercase US-ASCII string (5 bits per character). Function $\mathsf{AddChecksum}$ (resp. $\mathsf{RemoveChecksum}$) adds (resp. removes) a checksum character (5 bits) and inserts (resp. deletes) a hyphen separator between every four Base32 characters (20 bits). The returned fp is of length $19{\cdot}\mathsf{pwl}/4 - 5$ bits.

where k is randomly sampled from the key space $\{0,1\}^\lambda$; in the "ideal" world, it has oracle access to a random function $\mathsf{f}(\cdot)$, where f is uniformly sampled from the set $\mathcal{F}$ of all functions with domain $\{0,1\}^*$ and range $\{0,1\}^n$. In the end, $\mathcal{A}$ outputs a bit b' guessing which world it was in. Its PRF advantage measure $\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A})$ is defined as its ability to distinguish the two worlds: $\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A}) = \Pr[k \xleftarrow{\$} \{0,1\}^\lambda : \mathcal{A}^{\mathsf{F}_k} \Rightarrow 1] - \Pr[\mathsf{f} \xleftarrow{\$} \mathcal{F} : \mathcal{A}^{\mathsf{f}} \Rightarrow 1]$. F is called a *pseudorandom function* if the above advantage is sufficiently small for any efficient adversary $\mathcal{A}$.

A.3 Signature Schemes

A signature scheme Sig consists of three efficient algorithms $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$:

- $\mathsf{KGen}(1^\lambda)$: takes as input the security parameter 1^λ and outputs a pair of keys: a public verification key pk and a private signing key sk.
- Sign: takes as input a signing key sk and a message m, then outputs a signature σ.
- Vf: takes as input a verification key pk, a message m, and a signature σ, then outputs a bit b indicating if the signature is valid.

The *correctness* requires that for any $(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$ and any m it holds that $\mathsf{Vf}(pk, m, \mathsf{Sign}(sk, m)) = 1$.

For security, consider the following *UF-CMA (unforgeability under chosen message attack)* security experiment associated with an adversary $\mathcal{A}$. In the beginning, run $(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$. Then, $\mathcal{A}$ is given pk and access to the oracle $\mathsf{Sign}_{sk}(\cdot) = \mathsf{Sign}(sk, \cdot)$. In the end, $\mathcal{A}$ outputs a message-signature pair (m, σ). $\mathcal{A}$ wins if and only if $\mathsf{Vf}(pk, m, \sigma) = 1$ and m was not queried to the $\mathsf{Sign}_{sk}(\cdot)$ oracle. The UF-CMA advantage measure $\mathsf{Adv}^{\text{uf-cma}}_{\mathsf{Sig}}(\mathcal{A})$ is defined as $\mathcal{A}$'s winning probability.

B SAE-PK Password Encoding and Decoding

This section explains how a fingerprint is encoded as a human-readable password (e.g., "a2bc-de3f-ghi4") and how a password is decoded to recover its fingerprint, as defined by the Encode and Decode algorithms (see Fig. 8).

Encoding Algorithm. The encoding algorithm takes as input the fingerprint fp and the public parameters $pp = (\mathsf{sec}, \mathsf{pwl})$, and outputs a human-readable password pw. It initializes pw as an empty string and derives its final value based on fp and pp. First, it sets $b_{\mathsf{sec}} = 1$ if $pp.\mathsf{sec} = 3$, and $b_{\mathsf{sec}} = 0$ otherwise. Then, it iterates from $i = 0$ to $pp.\mathsf{pwl}/4 - 1$, and in each iteration appends to pw the $\mathsf{Base32}$-encoded bits consisting of b_{sec} concatenated with 19 bits of fp starting

from position $8 \cdot \mathsf{sec} + 19i$. After the loop, it additionally appends to pw the Base32-encoded bits consisting of b_{sec} concatenated with the last 14 bits of fp starting from $8 \cdot \mathsf{sec} + 19 \cdot (\mathsf{pwl}/4)$. Finally, a hyphen separator between every four Base32 characters and a checksum character are added using the Verhoeff algorithm [Ver69], implemented in the function AddChecksum shown in Fig. 8.

Decoding Algorithm. The decoding algorithm performs the inverse operation of the encoding algorithm. It takes as input a password pw and outputs a fingerprint fp and the public parameters $pp = (\mathsf{sec}, \mathsf{pwl})$. First, it computes $\mathsf{pwl} = |\mathsf{pw}| - \lfloor |\mathsf{pw}|/5 \rfloor$. Second, it removes the checksum character and hyphen separators from pw, and then applies the Base32d function to recover its bitstring representation. From this bitstring, the value of sec is derived. Next, it iterates for $i = 0$ to $\mathsf{pwl}/4 - 1$ to reconstruct the fingerprint fp. In each iteration, it appends to fp the 19 bits corresponding to positions $20i+1$ to $20i+19$ of the bitstring. After the loop, it additionally appends the last 14 bits from positions $20 \cdot (\mathsf{pwl}/4) + 1$ to $20 \cdot (\mathsf{pwl}/4) + 14$. Finally, the algorithm outputs the recovered fingerprint fp and the public parameters $pp = (\mathsf{sec}, \mathsf{pwl})$.

C Theorems and Security Proofs for SAE-Family Protocols

In this section, we present the formal statements and proofs of the security theorems for SAE-family protocols. The theorems are presented in the order of SAE, SAEv2, SAEv3, SAE-PK, and SAE-PK+ in Appendix C.1, with their proofs given in Appendices C.2 to C.6, respectively. Our proofs follow the *game-based proof technique* (see [Sho04] for a tutorial).

C.1 Theorem Statements

Theorem 6 (Security of SAE). *Let $\mathrm{H}_\infty(\mathscr{D}_\mathcal{D})$ denote the min-entropy of a password distribution $\mathscr{D}_\mathcal{D}$ over the dictionary $\mathcal{D}$. Let* H2C *(see line 6 of Fig. 2),* HMAC *(with fixed key 0^L for deriving seed, see line 23 of Fig. 2),* KDF *(see line 24 of Fig. 2), and* HMAC *(for deriving con_C, con_S, see lines 28 and 30 of Fig. 2) be modeled as random oracles H_1, H_2, H_3, and H_4, respectively. Let $\mathcal{A}$ be an efficient adversary that issues at most q_{se}* Send*-queries, q_{exe}* Execute*-queries, and $q_{\mathsf{H}_1}, q_{\mathsf{H}_2}, q_{\mathsf{H}_3}, q_{\mathsf{H}_4}$ random oracle queries to H_1, H_2, H_3, and H_4, respectively. Let $q_{\mathsf{se}}^{cs\text{-}pair}$, q_{se}^S, and q_{se}^C denote the numbers of* Send*-queries corresponding to the Cross-Server Paired, Malicious-Client, and Malicious-Server cases, respectively.*

Then, there exist efficient adversaries $\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$, $\mathcal{F}$, and $\mathcal{M}$ such that

$$\mathsf{Adv}^{\text{w-ake}}_{\mathsf{SAE}}(\mathcal{A}) \le 2T_1 \quad \textit{and} \quad \mathsf{Adv}^{\text{w-ea}}_{\mathsf{SAE}}(\mathcal{A}) \le T_1 + \frac{q_{\mathsf{se}}}{2^L}, \ \textit{where}$$

$$\begin{aligned} T_1 = {} & \frac{\frac{1}{2}q_{\mathsf{se}}^{cs\text{-}pair} + q_{\mathsf{se}}^S + q_{\mathsf{se}}^C}{2^{\mathrm{H}_\infty(\mathscr{D}_\mathcal{D})}} + 2q_{\mathsf{exe}} \cdot (q_{\mathsf{H}_2} + q_{\mathsf{se}}) \cdot \mathsf{Adv}^{\mathrm{cdh}}_{\mathcal{G},G}(\mathcal{C}) + 2q_{\mathsf{se}}^2 \cdot \mathsf{Adv}^{\mathrm{gdh}}_{\mathcal{G},G}(\mathcal{D}) \\ & + 4q_{\mathsf{se}} \cdot (q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1})^2 \cdot (\sqrt{\mathsf{Adv}^{\mathrm{wgsdh}}_{\mathcal{G},G}(\mathcal{E})} + \mathsf{Adv}^{\mathrm{gsdh}}_{\mathcal{G},G}(\mathcal{F})) + \mathsf{Adv}^{\mathrm{coll}}_{\mathsf{HMAC}(0^L,\cdot)}(\mathcal{M}) \\ & + \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3})^2}{2^{257}} + \frac{5q_{\mathsf{se}}^2 + 2q_{\mathsf{exe}}^2}{|\mathcal{G}| - 1} \\ & + \frac{6q_{\mathsf{se}} + 12q_{\mathsf{exe}} + 2q_{\mathsf{se}}^2 + 6q_{\mathsf{se}} \cdot q_{\mathsf{exe}} + 4q_{\mathsf{exe}}^2 + \frac{1}{2}(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1})^2}{|\mathcal{G}|} \\ & + \frac{5q_{\mathsf{se}} + 2q_{\mathsf{H}_3} + 3q_{\mathsf{H}_4} + 2q_{\mathsf{se}}^2 + 3q_{\mathsf{se}} \cdot q_{\mathsf{exe}} + 2q_{\mathsf{se}} \cdot q_{\mathsf{H}_3} + q_{\mathsf{exe}} \cdot q_{\mathsf{H}_3}}{2^L} \\ & + \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_2})^2 + (q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3})^2 + (q_{\mathsf{se}} + 2q_{\mathsf{exe}} + q_{\mathsf{H}_4})^2}{2^{L+1}}. \end{aligned} \tag{1}$$

Theorem 7 (Security of SAEv2). *We consider the SAEv2 protocol. All notation and assumptions are inherited from Theorem 6 without exception, which we omit to avoid repetition.*

Then, for any efficient adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathrm{ake}}_{\mathsf{SAEv2}}(\mathcal{A}) \;\le\; 2T_2 \quad \textit{and} \quad \mathsf{Adv}^{\mathrm{ea}}_{\mathsf{SAEv2}}(\mathcal{A}) \;\le\; T_2 + \frac{q_{\mathsf{se}}}{2^L},$$

where T_2 is identical to T_1 in equation (1).

Theorem 8 (Security of SAEv3). *We consider the SAEv3 protocol. All notation and assumptions are inherited from Theorem 6 without exception, which we omit to avoid repetition. Then, for any efficient adversary $\mathcal{A}$, we have*

$$\mathsf{Adv}^{\mathrm{ake}}_{\mathsf{SAEv3}}(\mathcal{A}) \le 2T_3 \quad \textit{and} \quad \mathsf{Adv}^{\mathrm{ea}}_{\mathsf{SAEv3}}(\mathcal{A}) \le T_3 + \frac{q_{\mathsf{se}}}{2^L},$$

where T_3 is identical to T_1 in equation (1).

Theorem 9 (Security of SAE-PK). *We consider the SAE-PK protocol. Let $\mathrm{H}_\infty(\mathscr{D}_{pp})$ denote the min-entropy of the password distribution $\mathscr{D}_{pp}$ over $\mathcal{T}_{pp}$. Let* H2C *(see line 6 of Fig. 2),* HMAC *(with a fixed key 0^L for deriving seed, see line 23 of Fig. 2),* KDF *(see line 24 of Fig. 2), and* HMAC *(for deriving con_C, con_S, see lines 28 and 30 of Fig. 2), and* SHA-L *(see lines 26 and 33 of Fig. 2) of SAE-PK be modeled as random oracles* H_1, H_2, H_3, H_4, H_5, *respectively. Let* AES-SIV *(see lines 25 and 31 of Fig. 2) be modeled as an ideal cipher. Let $\mathcal{A}$ be any efficient adversary that makes at most q_{se}* Send*-queries, q_{exe}* Execute*-queries, q_{ic} ideal cipher queries, and q_{H_1}, q_{H_2}, q_{H_3}, q_{H_4}, q_{H_5} random oracle queries to* H_1, H_2, H_3, H_4, H_5, *respectively. Let $q_{\mathsf{se}}^{cs\text{-}pair}$, q_{se}^{S}, and q_{se}^{C} denote the numbers of* Send*-queries corresponding to the Cross-Server Paired, Malicious-Client, and Malicious-Server cases, respectively.*

Then, there exist efficient adversaries $\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$, $\mathcal{F}$, $\mathcal{I}$, $\mathcal{J}$, and $\mathcal{M}$ such that

$$\mathsf{Adv}^{\text{i-ake}}_{\mathsf{SAE\text{-}PK}}(\mathcal{A}) \;\le\; 2T_4 \quad \textit{and} \quad \mathsf{Adv}^{\text{i-ea}}_{SAE\text{-}PK}(\mathcal{A}) \;\le\; T_4 + \frac{q_{\mathsf{se}}}{2^L},$$

where T_4 is defined in equation (2).

$$\begin{aligned}
T_4 = \;& \frac{\frac{1}{2}q_{\mathsf{se}}^{cs\text{-}pair} + q_{\mathsf{se}}^{S} + q_{\mathsf{se}}^{C}}{2^{\mathrm{H}_\infty(\mathscr{D}_{\mathcal{D}})}} + \frac{4q_{\mathsf{H}_5}}{2^{120-8\cdot\mathsf{sec}}} + \frac{4q_{\mathsf{H}_5}}{2^{63+8\cdot\mathsf{sec}}} \\
& + 2q_{\mathsf{exe}} \cdot (q_{\mathsf{H}_2} + q_{\mathsf{se}}) \cdot \mathsf{Adv}^{\mathrm{cdh}}_{\mathcal{G},G}(\mathcal{C}) + 2q_{\mathsf{se}}^2 \cdot \mathsf{Adv}^{\mathrm{gdh}}_{\mathcal{G},G}(\mathcal{D}) \\
& + 4q_{\mathsf{se}} \cdot (q_{\mathsf{H}_1} + q_{\mathsf{se}} + q_{\mathsf{exe}})^2 \cdot (\sqrt{\mathsf{Adv}^{\mathrm{wgsdh}}_{\mathcal{G},G}(\mathcal{E})} + \mathsf{Adv}^{\mathrm{gsdh}}_{\mathcal{G},G}(\mathcal{F})) \\
& + 8 \cdot \mathsf{Adv}^{\text{2-pre}}_{\mathsf{PPG},pp}(\mathcal{I}) + q_{\mathsf{se}}^2 \cdot \mathsf{Adv}^{\text{uf-cma}}_{\mathsf{ECDSA}}(\mathcal{J}) + \mathsf{Adv}^{\mathrm{coll}}_{\mathsf{HMAC}(0^L,\cdot)}(\mathcal{M}) \\
& + \frac{12q_{\mathsf{H}_5}}{2^{128}} + \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3})^2}{2^{257}} + \frac{5q_{\mathsf{se}}^2 + 2q_{\mathsf{exe}}^2}{|\mathcal{G}| - 1} \\
& + \frac{6q_{\mathsf{se}} + 12q_{\mathsf{exe}} + 3q_{\mathsf{se}}^2 + 7q_{\mathsf{se}} \cdot q_{\mathsf{exe}} + \frac{9}{2}q_{\mathsf{exe}}^2 + \frac{1}{2}(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1})^2}{|\mathcal{G}|} \\
& + \frac{5q_{\mathsf{se}} + 2q_{\mathsf{H}_3} + 3q_{\mathsf{H}_4} + 2q_{\mathsf{se}}^2 + 3q_{\mathsf{se}} \cdot q_{\mathsf{exe}} + (2q_{\mathsf{se}} + q_{\mathsf{exe}}) \cdot (q_{\mathsf{H}_3} + q_{\mathsf{ic}})}{2^L} \\
& + \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_2})^2 + 2(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3})^2 + (q_{\mathsf{se}} + 2q_{\mathsf{exe}} + q_{\mathsf{H}_4})^2}{2^{L+1}}.
\end{aligned} \tag{2}$$

Theorem 10 (Security of SAE-PK+). *We consider the SAE-PK protocol. Let $\mathrm{H}_\infty(\mathscr{D}_{pp})$ denote the min-entropy of the password distribution $\mathscr{D}_{pp}$ over $\mathcal{T}_{pp}$. Let* H_1, H_2, H_3, *and* H_4 *be the same random oracles as defined in Theorem 9. Let* AES-SIV *(see lines 25 and 31 of Fig. 2) be modeled as an ideal cipher. Let $\mathcal{A}$ be any efficient adversary that makes at most q_{se}* Send*-queries, q_{exe}* Execute*-queries, q_{ic} ideal cipher queries, and q_{H_1}, q_{H_2}, q_{H_3}, q_{H_4} random-oracle queries to* H_1, H_2, H_3, H_4, *respectively. Let $q_{\mathsf{se}}^{cs\text{-}pair}$, q_{se}^{S}, and q_{se}^{C} denote the numbers of* Send*-queries corresponding to the Cross-Server Paired, Malicious-Client, and Malicious-Server cases, respectively.*

Then, there exist efficient adversaries $\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$, $\mathcal{F}$, $\mathcal{I}$, $\mathcal{J}$, $\mathcal{M}$, and $\mathcal{K}$ such that

$$\mathsf{Adv}^{\text{i-ake}}_{\mathsf{SAE\text{-}PK+}}(\mathcal{A}) \le 2T_5 \quad \textit{and} \quad \mathsf{Adv}^{\text{i-ea}}_{\mathsf{SAE\text{-}PK+}}(\mathcal{A}) \le T_5 + \frac{q_{\mathsf{se}}}{2^L},$$

where T_5 is defined in equation (3), with terms in red indicating the differences between SAE-PK and SAE-PK+.

$$\begin{aligned}
T_5 = {} & \frac{\frac{1}{2}q_{\mathsf{se}}^{cs\text{-}pair} + q_{\mathsf{se}}^S + q_{\mathsf{se}}^C}{2^{\mathrm{H}_\infty(\mathscr{D}_\mathcal{D})}} + \frac{4q_{\mathsf{H}_5}}{2^{120-8\cdot\mathsf{sec}}} + \frac{4q_{\mathsf{H}_5}}{2^{63+8\cdot\mathsf{sec}}} \\
& + 2q_{\mathsf{exe}} \cdot (q_{\mathsf{H}_2} + q_{\mathsf{se}}) \cdot \mathsf{Adv}^{\text{cdh}}_{\mathcal{G},G}(\mathcal{C}) + 2q_{\mathsf{se}}^2 \cdot \mathsf{Adv}^{\text{gdh}}_{\mathcal{G},G}(\mathcal{D}) \\
& + 4q_{\mathsf{se}} \cdot (q_{\mathsf{H}_1} + q_{\mathsf{se}} + q_{\mathsf{exe}})^2 \cdot (\sqrt{\mathsf{Adv}^{\text{wgsdh}}_{\mathcal{G},G}(\mathcal{E})} + \mathsf{Adv}^{\text{gsdh}}_{\mathcal{G},G}(\mathcal{F})) \\
& + 4 \cdot \mathsf{Adv}^{\text{2-pre}}_{\mathsf{PPG},pp}(\mathcal{I}) + q_{\mathsf{se}}^2 \cdot \mathsf{Adv}^{\text{uf-cma}}_{\mathsf{ECDSA}}(\mathcal{J}) + \mathsf{Adv}^{\text{coll}}_{\mathsf{HMAC}(0^L,\cdot)}(\mathcal{M}) \\
& + \frac{8q_{\mathsf{H}_5}}{2^{128}} + \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3})^2}{2^{257}} + \frac{5q_{\mathsf{se}}^2 + 2q_{\mathsf{exe}}^2}{|\mathcal{G}| - 1} \\
& + \frac{6q_{\mathsf{se}} + 12q_{\mathsf{exe}} + 3q_{\mathsf{se}}^2 + 7q_{\mathsf{se}} \cdot q_{\mathsf{exe}} + \frac{9}{2}q_{\mathsf{exe}}^2 + \frac{1}{2}(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1})^2}{|\mathcal{G}|} \\
& + \frac{5q_{\mathsf{se}} + 2q_{\mathsf{H}_3} + 3q_{\mathsf{H}_4} + 2q_{\mathsf{se}}^2 + 3q_{\mathsf{se}} \cdot q_{\mathsf{exe}} + (2q_{\mathsf{se}} + q_{\mathsf{exe}}) \cdot (q_{\mathsf{H}_3} + q_{\mathsf{ic}})}{2^L} \\
& + \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_2})^2 + 2(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3})^2 + (q_{\mathsf{se}} + 2q_{\mathsf{exe}} + q_{\mathsf{H}_4})^2}{2^{L+1}}.
\end{aligned} \tag{3}$$

C.2 Proof of Theorem 6 (Security of SAE)

The proof of Theorem 6 proceeds through a sequence of game hops, ending in a final game in which all *fresh* sessions can be perfectly simulated without knowledge of passwords.

Game $\mathbf{G}_0$ corresponds to the real protocol execution in the random oracle model. From $\mathbf{G}_1$ to $\mathbf{G}_3$, we simulate the random oracles and all sessions exactly as honest participants would, answer adversarial queries, rule out certain rare group-element choices, and prevent undesirable collisions. We then analyze the five cases described in the proof sketch, corresponding to the game hops from $\mathbf{G}_4$ to $\mathbf{G}_{11}$. Specifically, we first analyze the *Execute Case* ($\mathbf{G}_4$–$\mathbf{G}_6$), followed by the *Paired Case* ($\mathbf{G}_7$–$\mathbf{G}_{10}$). Subsequently, to obtain a tighter overall bound, we handle the remaining three cases simultaneously in $\mathbf{G}_{11}$, namely the *Cross-Server Paired Case*, *Malicious-Client Case*, and *Malicious-Server Case*, rather than analyzing them separately. These cases were introduced in Section 5.2 and will be recalled below as needed. Finally, the proof concludes with the information-theoretic game $\mathbf{G}_{12}$.

In each game, we define the following notation and events. For each session with $(\mathrm{MAC}_C, s_C, CE_C)$ and $(\mathrm{MAC}_S, s_S, CE_S)$, letting $(\mathrm{MAC}_1, s_1, CE_1)$ denote the lexicographically smaller tuple, we define the session identifier as

$$\mathsf{sid} = (s_1, CE_1, s_2, CE_2),$$

which is equivalent to the order-insensitive set-based definition $\{(s_C, CE_C), (s_S, CE_S)\}$ shown in line 36 of Fig. 2.

We then define the following events for game $\mathbf{G}_m$:

wS_m**:** The adversary $\mathcal{A}$ correctly guesses the hidden bit b under weak AKE security;
wAuth_m**:** Some *fresh* session π_P^i accepts without a weak partner.

Game $\mathbf{G}_0$: This is the original security experiment in the random oracle model. Specifically, the functions used to derive PT, *seed*, $kck|pmk$, as well as the confirmation values con_C and con_S, are modeled as random oracles. By definition,

$$\mathsf{Adv}^{\text{w-ake}}_{\mathsf{SAE}}(\mathcal{A}) = 2\Pr[\mathsf{wS}_0] - 1 \quad \text{and} \quad \mathsf{Adv}^{\text{w-ea}}_{\mathsf{SAE}}(\mathcal{A}) = \Pr[\mathsf{wAuth}_0]. \tag{4}$$

H_1 oracle simulation:
For each $\mathsf{H}_1(w)$ query, the simulator first checks if the same query was previously asked. If so, it retrieves the record $(w, \cdot, r)$ from the list $\mathcal{L}_{\mathsf{H}_1}$ and responds with r. Otherwise, the response r is determined as follows:

► **Rule** $\mathsf{H}_1^{(1)}$– Choose $\alpha \in \mathbb{Z}_q$ and compute $r = \alpha G$.

Record the tuple (w, r, α) in $\mathcal{L}_{\mathsf{H}_1}$.

H_2 and H_2' oracles simulation:
For each query $\mathsf{H}_2(w)$ (resp. $\mathsf{H}_2'(w)$), if the same query was previously asked, the simulator retrieves the record (w, r) from the list $\mathcal{L}_{\mathsf{H}_2}$ (resp. $\mathcal{L}'_{\mathsf{H}_2}$) and responds with r. Otherwise, the response r is generated as follows:

► **Rule** $\mathsf{H}_2^{(1)}$– Choose $r \in \{0,1\}^L$.

Record the tuple (w, r) in $\mathcal{L}_{\mathsf{H}_2}$ (resp. $\mathcal{L}'_{\mathsf{H}_2}$).

H_3 and H_3' oracles simulation:
For each query $\mathsf{H}_3(w)$ (resp. $\mathsf{H}_3'(w)$), if the same query was previously asked, the simulator retrieves the record (w, r) from the list $\mathcal{L}_{\mathsf{H}_3}$ (resp. $\mathcal{L}'_{\mathsf{H}_3}$) and responds with r. Otherwise, the response r is generated as follows:

► **Rule** $\mathsf{H}_3^{(1)}$– Choose $r \in \{0,1\}^{L+256}$.

Record the tuple (w, r) in $\mathcal{L}_{\mathsf{H}_3}$ (resp. $\mathcal{L}'_{\mathsf{H}_3}$).

H_4 and H_4' oracles simulation:
For each query $\mathsf{H}_4(w)$ (resp. $\mathsf{H}_4'(w)$), if the same query was previously asked, the simulator retrieves the record (w, r) from the list $\mathcal{L}_{\mathsf{H}_4}$ (resp. $\mathcal{L}'_{\mathsf{H}_4}$) and responds with r. Otherwise, the response r is obtained as follows:

► **Rule** $\mathsf{H}_4^{(1)}$– Choose $r \in \{0,1\}^L$.

Record the tuple (w, r) in $\mathcal{L}_{\mathsf{H}_4}$ (resp. $\mathcal{L}'_{\mathsf{H}_4}$).

Fig. 9. Simulations of random oracles used in the proof of Theorem 6 (Security of SAE).

In the following, for notational convenience, we use $\mathbf{G}_m$ to denote any event defined in Game $\mathbf{G}_m$. If the game aborts, or if the adversary $\mathcal{A}$ exceeds the allowed number of queries or the prescribed time bound, then a random bit b' is returned. This guarantees that every game terminates with a well-defined output.

Game $\mathbf{G}_1$: In this game, we simulate the random oracles and all adversarial interactions. For random oracles:

- We model the H2C used to derive PT (see line 6 of Fig. 2) as a random oracle H_1.
- We model the HMAC with fixed key 0^L used to derive the key seed *seed* (see line 23 of Fig. 2) as a random oracle H_2.
- We model the KDF used to derive $kck|pmk$ (see line 24 of Fig. 2) as a random oracle H_3.
- We model the HMAC used to compute con_C and con_S (see lines 28 and 30 of Fig. 2) as a random oracle H_4.

For each random oracle H_i (with $i = 1, 2, 3, 4$), we maintain a list $\mathcal{L}_{\mathsf{H}_i}$ to record all queries and responses during the simulation. Additionally, we maintain separate lists $\mathcal{L}^{\mathcal{A}}_{\mathsf{H}_i}$ to track queries issued directly by the adversary. To facilitate subsequent game hops, we introduce three additional private random oracles: $\mathsf{H}_2' : \{0,1\}^* \to \{0,1\}^L$, $\mathsf{H}_3' : \{0,1\}^* \to \{0,1\}^{L+256}$, and $\mathsf{H}_4' : \{0,1\}^* \to \{0,1\}^L$, together with the corresponding lists $\mathcal{L}_{\mathsf{H}_2'}, \mathcal{L}_{\mathsf{H}_3'}, \mathcal{L}_{\mathsf{H}_4'}$ used to record all oracle queries. The complete simulation details of these random oracles are provided in Fig. 9.

A $\mathsf{Send}(S, j, \mathrm{MAC}_C)$-query, $\mathsf{Send}(S, j, (\mathrm{MAC}_C, \mathrm{SSID}))$-query or $\mathsf{Send}(S, j, \top)$-query is answered with $(\mathrm{MAC}_S, \mathrm{SSID})$.

A $\mathsf{Send}(S, j, (\mathrm{MAC}_C, s_C, CE_C))$-query is processed as:

- **Rule S1**$^{(1)}$ Abort if $s_C \notin \mathbb{Z}_q \setminus \{0, 1\}$ or $CE_C \notin \mathbb{Z}_q \setminus \{0\}$.
- **Rule S2**$^{(1)}$ $PT = \mathsf{H}_1(\mathrm{SSID} \mid \mathsf{pw})$ and $val = \mathsf{HMAC}(0^L, \max(\mathrm{MAC}_C, \mathrm{MAC}_S) \mid \min(\mathrm{MAC}_C, \mathrm{MAC}_S)) \bmod (q-1)+1$. Set $PWE = val \cdot PT$.
- **Rule S3**$^{(1)}$ Uniformly select $r_S, m_S \in \mathbb{Z}_q \setminus \{0, 1\}$ such that $(r_S + m_S) \bmod q \in \mathbb{Z}_q \setminus \{0, 1\}$. Compute $s_S = (r_S + m_S) \bmod q$ and $CE_S = -m_S \cdot PWE$.
- **Rule S4**$^{(1)}$ Compute $K = r_S \cdot (s_C \cdot PWE + CE_C)$ and abort if $K = 0$.
- **Rule S5**$^{(1)}$ Compute $seed = \mathsf{H}_2([K]_x)$.
- **Rule S6**$^{(1)}$ Compute $ctx = (s_S + s_C) \bmod q$.
- **Rule S7**$^{(1)}$ Compute $kck \mid pmk = \mathsf{H}_3(seed, ctx)$.
- **Rule S8**$^{(1)}$ Compute $con_S = \mathsf{H}_4(kck, s_S \mid CE_S \mid s_C \mid CE_C)$.

Return $((\mathrm{MAC}_S, s_S, CE_S), con_S)$ and transition to the expected state. The simulation also records $((\mathrm{MAC}_S, s_S, CE_S), (\mathrm{MAC}_C, s_C, CE_C), con_S)$ in $\mathcal{L}_{\mathcal{T}}$.

A $\mathsf{Send}(S, j, con_C)$-query is processed as:

- **Rule S9**$^{(1)}$ Drop and wait if $con_C \neq \mathsf{H}_4(kck, s_S \mid CE_S \mid s_C \mid CE_C)$; otherwise, the server instance accepts, stores pmk, and terminates.

Fig. 10. Simulation of Send queries to the server in the proof of Theorem 6 (Security of SAE).

A $\mathsf{Send}(C, i, \top)$-query is answered with $(\mathrm{MAC}_C, \mathrm{SSID})$.

A $\mathsf{Send}(C, i, (\mathrm{MAC}_S, \mathrm{SSID}))$-query is processed as:

- **Rule C1**$^{(1)}$ Abort if SSID does not match.
- **Rule C2**$^{(1)}$ $PT = \mathsf{H}_1(\mathrm{SSID} \mid \mathsf{pw})$ and $val = \mathsf{HMAC}(0^L, \max(\mathrm{MAC}_C, \mathrm{MAC}_S) \mid \min(\mathrm{MAC}_C, \mathrm{MAC}_S)) \bmod (q-1)+1$. Set $PWE = val \cdot PT$.
- **Rule C3**$^{(1)}$ Choose $r_C, m_C \in \mathbb{Z}_q \setminus \{0, 1\}$ uniformly at random, ensuring $(r_C + m_C) \bmod q \in \mathbb{Z}_q \setminus \{0, 1\}$. Set $s_C = (r_C + m_C) \bmod q$ and $CE_C = -m_C \cdot PWE$.

Return $(\mathrm{MAC}_C, s_C, CE_C)$ and transition to the expected state.

A $\mathsf{Send}(C, i, (\mathrm{MAC}_S, s_S, CE_S))$-query is processed as:

- **Rule C4**$^{(1)}$ Drop and wait if $s_C = s_S \vee CE_S = CE_C$.
- **Rule C5**$^{(1)}$ Abort if $s_S \notin \mathbb{Z}_q \setminus \{0, 1\}$ or $CE_S \notin \mathbb{Z}_q \setminus \{0\}$.
- **Rule C6**$^{(1)}$ Compute $K = r_C \cdot (s_S \cdot PWE + CE_S)$ and abort if $K = 0$.
- **Rule C7**$^{(1)}$ Compute $seed = \mathsf{H}_2([K]_x)$.
- **Rule C8**$^{(1)}$ Set $ctx = (s_C + s_S) \bmod q$.
- **Rule C9**$^{(1)}$ Compute $kck \mid pmk = \mathsf{H}_3(seed, ctx)$.
- **Rule C10**$^{(1)}$ Compute $con_C = \mathsf{H}_4(kck, s_C \mid CE_C \mid s_S \mid CE_S)$.

Return con_C, and transition to the expected state.

A $\mathsf{Send}(C, i, con_S)$-query is processed as:

- **Rule C11**$^{(1)}$ Drop and wait if $con_S \neq \mathsf{H}_4(kck, s_S \mid CE_S \mid s_C \mid CE_C)$; otherwise, the client instance accepts, stores pmk, and terminates.

A $\mathsf{Send}(C, i, (\mathrm{MAC}_S, s_S, CE_S, con_S))$-query is processed by executing Rule C4–Rule C10, returning con_C, and then performing Rule C11.

Fig. 11. Simulation of Send queries to the client in the proof of Theorem 6 (Security of SAE).

An $\mathsf{Execute}(C, i, S, j)$-query is simulated by sequentially executing the corresponding Send-queries. Once completed, the resulting transcript is given to the adversary.

A $\mathsf{Reveal}(P, i)$-query returns the session key pmk of instance (P, i), provided that the instance has already accepted and terminated.

A $\mathsf{sCorrupt}(S, (pw, aux))$-query returns to the adversary the password pw_S of server S and its server-side auxiliary information aux_S. If $(pw, aux) \neq \perp$, the challenger replaces pw_S and aux_S with the provided values.

A $\mathsf{iCorrupt}(S)$-query returns to the adversary the password pw_S of server S and its client-side auxiliary information $\overline{aux}_S$.

A $\mathsf{Test}(P, i)$-query operates as follows: flip a fair coin $b \in \{0, 1\}$. If $b = 1$ and the instance has already accepted with a session key, return pmk; otherwise, return a random value from $\{0, 1\}^{256}$.

Fig. 12. Simulation of $\mathsf{Execute}$, $\mathsf{sCorrupt}$, $\mathsf{iCorrupt}$, Reveal, and Test queries in the proof of Theorem 6 (Security of SAE).

All interactions with the adversary are handled exactly as in the real protocol, including the simulation of all protocol participants and the handling of Send, $\mathsf{Execute}$, Reveal, $\mathsf{sCorrupt}$, $\mathsf{iCorrupt}$, and Test queries, as specified in Figs. 10 to 12.

For Send queries, the special message $\top$ denotes session initiation. Although a $\mathsf{Send}(C, i, \top)$-query may receive no response, since this message is optional, we may assume without loss of generality that a response is always returned, as it is not cryptographically protected. This assumption only strengthens the adversary. Send queries involving $\top$ are handled identically.

It follows that

$$\left|\Pr[\mathbf{G}_1] - \Pr[\mathbf{G}_0]\right| = 0. \tag{5}$$

Game $\mathbf{G}_2$: In this game, we sample $r_C, m_C, r_S, m_S \leftarrow\!\!\$\ \mathbb{Z}_q$ uniformly at random and do not abort the game even if any of r_C, m_C, r_S, m_S, $(r_C + m_C) \bmod q$, $(r_S + m_S) \bmod q$ lies in $\{0, 1\}$. The details are as follows:

- **Rule $\mathbf{C3}^{(2)}$** Choose $r_C, m_C \in \mathbb{Z}_q$ uniformly at random. Compute $s_C = (r_C + m_C) \bmod q$ and $CE_C = -m_C \cdot PWE$.
- **Rule $\mathbf{S3}^{(2)}$** Choose $r_S, m_S \in \mathbb{Z}_q$ uniformly at random. Compute $s_S = (r_S + m_S) \bmod q$ and $CE_S = -m_S \cdot PWE$.

The distinguishing advantage arises from detecting whether any of these variables, or their modular sums, equal 0 or 1, which are special values that are excluded in the real protocol to ensure correctness and uniformity in subsequent operations. We have

$$\left|\Pr[\mathbf{G}_2] - \Pr[\mathbf{G}_1]\right| \ \leq\ \frac{6q_{\mathsf{se}} + 12q_{\mathsf{exe}}}{q}. \tag{6}$$

In the following games, r_C, m_C, r_S, m_S are sampled independently and uniformly at random from $\mathbb{Z}_q$.

Game $\mathbf{G}_3$: In this game, we enforce collision resistance among several outputs. The modifications are as follows:

1. **Avoiding collisions in val.** This makes a distinction unless a val-collision occurs, which happens with probability at most

$$\mathsf{Adv}^{\mathrm{coll}}_{\mathsf{HMAC}(0^L,\cdot)}(\mathcal{M}).$$

2. **Avoiding collisions in random-oracle outputs.** We rule out output collisions in the random oracles H_1, H_2, H'_2, H_3, H'_3, H_4, and H'_4. In particular, we avoid collisions among the derived values PT, *seed*, *kck*, *pmk*, con_C and con_S. Since there are at most $q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1}$, $q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_2}$, $q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3}$ and $q_{\mathsf{se}} + 2q_{\mathsf{exe}} + q_{\mathsf{H}_4}$ queries to H_1, H_2 and H'_2, H_3 and H'_3, H_4 and H'_4, respectively, the birthday bound implies that the distinguishing advantage due to such collisions is bounded by:

$$\frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1})^2}{2q} + \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_2})^2}{2^{L+1}} + \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3})^2}{2^{L+1}} + \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3})^2}{2^{257}} + \frac{(q_{\mathsf{se}} + 2q_{\mathsf{exe}} + q_{\mathsf{H}_4})^2}{2^{L+1}}$$

3. **Avoiding collisions in transcripts.** We ensure that any pair (s_C, CE_C) or (s_S, CE_S) newly generated by an honest party does not collide with any previously generated value, whether produced by the simulator or by the adversary. There are at most $2q_{\mathsf{se}} + 2q_{\mathsf{exe}}$ possible pairs (s_C, CE_C) or (s_S, CE_S), and at most $q_{\mathsf{se}} + 2q_{\mathsf{exe}}$ new generations. By the uniform randomness of each new generation, the probability that a newly generated pair collides with a previous one is upper bounded by

$$\frac{(q_{\mathsf{se}} + 2q_{\mathsf{exe}})(2q_{\mathsf{se}} + 2q_{\mathsf{exe}})}{q} = \frac{2q_{\mathsf{se}}^2 + 6q_{\mathsf{se}} \cdot q_{\mathsf{exe}} + 4q_{\mathsf{exe}}^2}{q}.$$

Combining all terms, we obtain the following bound on the total distinguishing advantage due to collision-avoidance changes:

$$\begin{aligned} |\Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_2]| \leq \mathsf{Adv}^{\mathrm{coll}}_{\mathsf{HMAC}(0^L,\cdot)}(\mathcal{M}) &+ \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1})^2}{2q} + \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_2})^2}{2^{L+1}} \\ &+ \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3})^2}{2^{L+1}} + \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3})^2}{2^{257}} \\ &+ \frac{(q_{\mathsf{se}} + 2q_{\mathsf{exe}} + q_{\mathsf{H}_4})^2}{2^{L+1}} + \frac{2q_{\mathsf{se}}^2 + 6q_{\mathsf{se}} \cdot q_{\mathsf{exe}} + 4q_{\mathsf{exe}}^2}{q}. \end{aligned} \tag{7}$$

Game $\mathbf{G}_4$: We now proceed to handle each case stated in the proof sketch of Theorem 1 in turn, transitioning to an information-theoretic game in which passwords of *fresh* sessions are no longer required. From game $\mathbf{G}_4$ to $\mathbf{G}_6$, we consider *Execute Case*, in which sessions are generated via Execute queries.

In this game $\mathbf{G}_4$, we replace the hash function H_2 with a secret random oracle H'_2 for all sessions generated via Execute queries. For readability, we refer to such sessions as Execute sessions and use a superscript Execute to denote rules specific to *Execute Case*. The modification is carried out as follows:

- **Rule C7$^{(4)\text{-Execute}}$** Compute $seed = \mathsf{H}'_2([K]_x)$.
- **Rule S5$^{(4)\text{-Execute}}$** Compute $seed = \mathsf{H}'_2([K]_x)$.

This modification introduces no distinguishing advantage unless the event $\mathsf{AskH}_2\text{-}\mathsf{Execute}^{(4)}$ occurs, namely, either the adversary or the simulator queries H_2 on a value $[K]_x$ corresponding to an Execute session.

Note that the case where the simulator itself queries H_2 also constitutes a bad event. Specifically, when an Execute session computes $seed' = \mathsf{H}'_2([K]_x)$ and a non-Execute session later recomputes $seed = \mathsf{H}_2([K]_x)$ on the same input $[K]_x$, the fact that $seed' \neq seed$ enables the adversary to distinguish the two games.

Consequently, the games $\mathbf{G}_4$ and $\mathbf{G}_3$ are perfectly indistinguishable unless the event $\mathsf{AskH}_2\text{-}\mathsf{Execute}^{(4)}$ occurs:

$$|\Pr[\mathbf{G}_4] - \Pr[\mathbf{G}_3]| \leq \Pr[\mathsf{AskH}_2\text{-}\mathsf{Execute}^{(4)}]. \tag{8}$$

The probability of AskH_2-$\mathsf{Execute}$ is given in the following game $\mathbf{G}_5$.

Game $\mathbf{G}_5$: In this game, we replace the internal value $[K]_x$ with the session identifier $\mathsf{sid} = (s_1, CE_1, s_2, CE_2)$ for all $\mathsf{Execute}$ sessions. Specifically, we update the computation of *seed* as follows:

- **Rule C7$^{(5)}$-Execute** Compute $seed = \mathsf{H}'_2(\mathsf{sid})$.
- **Rule S5$^{(5)}$-Execute** Compute $seed = \mathsf{H}'_2(\mathsf{sid})$.

This game is perfectly indistinguishable from $\mathbf{G}_4$ except in the event that two distinct sids of $\mathsf{Execute}$ sessions lead to the same $[K]_x$ value.

From the protocol specification, the shared key is computed as $K = r_C \cdot (s_S \cdot PWE + CE_S)$. By the randomness of $r_C \in \mathbb{Z}_q$ and its independence from all other values, each nonzero K is uniformly distributed over a set of size $q-1$. Since each value $[K]_x$ corresponds to at most two distinct K values, the probability of an $[K]_x$ collision is bounded by $2/(q-1)$.

Consequently, among the at most q_{exe} $\mathsf{Execute}$ sessions under consideration, the probability that two of them yield the same $[K]_x$ can be bounded via the birthday paradox:

$$\left|\Pr[\mathbf{G}_5] - \Pr[\mathbf{G}_4]\right| \;\leq\; \frac{q_{\mathsf{exe}}^2}{q-1}. \tag{9}$$

Moreover, in this game, we can bound $\Pr[\mathsf{AskH}_2\text{-}\mathsf{Execute}^{(5)}]$ as stated in the following Lemma 1:

Lemma 1. *Let $\mathcal{A}$ be any efficient adversary that triggers the event AskH_2-$\mathsf{Execute}^{(5)}$, i.e., there exists an entry $([K]_x, \cdot) \in \mathcal{L}_{\mathsf{H}_2}$ corresponding to an $\mathsf{Execute}$ session. Then there exists an algorithm $\mathcal{C}$ that solves the CDH problem in the group $\mathcal{G}$ such that*

$$\Pr[\mathsf{AskH}_2\text{-}\mathsf{Execute}^{(5)}] \;\leq\; 2q_{\mathsf{exe}} \cdot (q_{\mathsf{se}} + q_{\mathsf{H}_2}) \cdot \mathsf{Adv}^{\mathrm{cdh}}_{\mathcal{G},G}(\mathcal{C}). \tag{10}$$

Proof. We prove this lemma by constructing a reduction algorithm $\mathcal{C}$ that solves the CDH problem in the group $\mathcal{G}$ by interacting with an adversary $\mathcal{A}$ that triggers the event AskH_2-$\mathsf{Execute}^{(5)}$.

The challenger provides $\mathcal{C}$ with a CDH instance $(G, X = xG, Y = yG)$, and the goal of $\mathcal{C}$ is to compute $Z = DH_G(X, Y) = xyG$.

To achieve this, $\mathcal{C}$ simulates the protocol environment for $\mathcal{A}$ as in $\mathbf{G}_5$, with the following modification: upon receiving an $\mathsf{Execute}$ query, it embeds the challenge values X and Y into the protocol messages CE_C and CE_S, respectively. The details are as follows:

- **Rule C3$^{\text{-}\mathsf{Execute}}$** Choose $r_C, m_C \in \mathbb{Z}_q$ uniformly at random, and compute $s_C = (r_C + m_C) \bmod q$ and $CE_C = -m_C \cdot PWE + X$.
- **Rule S3$^{\text{-}\mathsf{Execute}}$** Choose $r_S, m_S \in \mathbb{Z}_q$ uniformly at random, and compute $s_S = (r_S + m_S) \bmod q$ and $CE_S = -m_S \cdot PWE + Y$.

In this simulation, once an $\mathsf{Execute}$ session and its associated value K are identified, the CDH value xyG can be recovered from the following computation:

$$\begin{aligned}
K =& \mathrm{DH}_{PWE}(s_C \cdot PWE + CE_C,\ s_S \cdot PWE + CE_S) \\
&\mathbin{/\!/} \text{(definition of } K) \\
=& \mathrm{DH}_{PWE}((s_C - m_C) \cdot PWE + X,\ (s_S - m_S) \cdot PWE + Y) \\
&\mathbin{/\!/} \text{(using } CE_C = -m_C PWE + X,\ CE_S = -m_S PWE + Y) \\
=& \mathrm{DH}_{PWE}(r_C \cdot PWE + X,\ r_S \cdot PWE + Y) \\
&\mathbin{/\!/} \text{(where } r_C = s_C - m_C,\ r_S = s_S - m_S) \\
=& r_C r_S \cdot PWE + r_C Y + r_S X + \tfrac{XY}{PWE} \\
&\mathbin{/\!/} \text{(expansion of } \mathrm{DH}_{PWE}) \\
=& r_C r_S \cdot PWE + r_C Y + r_S X + \tfrac{xyG}{val \cdot \alpha} \\
&\mathbin{/\!/} \text{(since } X = xG,\ Y = yG,\ PWE = val \cdot \alpha G).
\end{aligned}$$

The reduction $\mathcal{C}$ identifies the Execute session described above and the corresponding value K by proceeding as follows. It uniformly guesses an Execute-originated transcript among at most q_{exe} candidates and an element $[K]_x \in \mathcal{L}_{\mathsf{H}_2}$ among at most $q_{\mathsf{H}_2} + q_{\mathsf{se}}$ possibilities. When the event AskH_2-Execute occurs—namely, when either the adversary $\mathcal{A}$ or the simulator queries H_2 on a value $[K]_x$ corresponding to an Execute session—the probability that $\mathcal{C}$ correctly guesses the corresponding Execute session and its associated $[K]_x$ is $1/q_{\mathsf{exe}} \cdot (q_{\mathsf{H}_2} + q_{\mathsf{se}})$. Conditioned on a correct guess, the reduction identifies the Execute session and the corresponding value K with probability $\frac{1}{2}$, since $[K]_x$ corresponds to two possible K. Finally, since this yields a perfect simulation of $\mathbf{G}_5$, the reduction $\mathcal{C}$ computes xyG with probability $\Pr[\mathsf{AskH}_2\text{-}\mathsf{Execute}^{(5)}]/2q_{\mathsf{exe}} \cdot (q_{\mathsf{se}} + q_{\mathsf{H}_2})$. □

Game $\mathbf{G}_6$: In this game, we replace H_3 with a secret random function H'_3 for all Execute sessions. The computation is modified as follows:

- **Rule** $\mathbf{C9}^{(6)}$-Execute $kck|pmk = \mathsf{H}'_3(\mathsf{sid})$.
- **Rule** $\mathbf{S7}^{(6)}$-Execute $kck|pmk = \mathsf{H}'_3(\mathsf{sid})$.

Since we avoid collisions on *seed*, this modification only makes a distinction if the adversary or the simulator query H_3 at *seed*. However, as *seed* is derived from the secret random oracle H'_2, this occurs randomly. Therefore, at most $q_{\mathsf{H}_3} + q_{\mathsf{se}}$ queries imply

$$|\Pr[\mathbf{G}_6] - \Pr[\mathbf{G}_5]| \leq \frac{q_{\mathsf{exe}}(q_{\mathsf{se}} + q_{\mathsf{H}_3})}{2^L} = \frac{q_{\mathsf{se}} \cdot q_{\mathsf{exe}}}{2^L} + \frac{q_{\mathsf{exe}} \cdot q_{\mathsf{H}_3}}{2^L}. \tag{11}$$

Game $\mathbf{G}_7$: We now complete the *Execute Case* and turn to the *Paired Case*, which ends in game $\mathbf{G}_{10}$. In this case, we consider $\mathsf{Send}(S, j, (\mathrm{MAC}_C, s_C, CE_C))$-queries with (s_C, CE_C) originating from a non-Execute session associated with the same server as π^j_S. We also consider $\mathsf{Send}(C, i, (s_S, CE_S))$-queries with (s_S, CE_S) originating from a non-Execute session associated with the same server as π^i_C. For simplicity, we refer to such queries as $\mathsf{Send}_{\mathsf{pair}}$ queries and the corresponding sessions as $\mathsf{Send}_{\mathsf{pair}}$ sessions.

In this game $\mathbf{G}_7$, we simulate $\mathsf{Send}_{\mathsf{pair}}$ queries using a secret random oracle H'_2 instead of the original H_2. We denote the corresponding query rules by the superscript $\mathsf{Send}_{\mathsf{pair}}$. The simulation proceeds as follows:

- **Rule** $\mathbf{C7}^{(7)}$-$\mathsf{Send}_{\mathsf{pair}}$ Compute $seed = \mathsf{H}'_2([K]_x)$.
- **Rule** $\mathbf{S5}^{(7)}$-$\mathsf{Send}_{\mathsf{pair}}$ Compute $seed = \mathsf{H}'_2([K]_x)$.

As in the Execute case analysis of $\mathbf{G}_4$, games $\mathbf{G}_6$ and $\mathbf{G}_7$ differ only if the event $\mathsf{AskH}_2\text{-}\mathsf{Send}_{\mathsf{pair}}{}^{(7)}$ occurs, namely when the adversary or simulator queries $\mathsf{H}_2([K]_x)$ of a $\mathsf{Send}_{\mathsf{pair}}$ session. Hence,

$$|\Pr[\mathbf{G}_7] - \Pr[\mathbf{G}_6]| \leq \Pr\left[\mathsf{AskH}_2\text{-}\mathsf{Send}_{\mathsf{pair}}{}^{(7)}\right]. \tag{12}$$

The bound on $\Pr[\mathsf{AskH}_2\text{-}\mathsf{Send}_{\mathsf{pair}}{}^{(7)}]$ is given in $\mathbf{G}_9$ (see Lemma 2).

Game $\mathbf{G}_8$: In this game, we replace the random oracle H_3 with a secret random oracle H'_3 for all $\mathsf{Send}_{\mathsf{pair}}$ sessions. The computation rules are updated as follows:

- **Rule** $\mathbf{C9}^{(8)}$-$\mathsf{Send}_{\mathsf{pair}}$ Compute $kck|pmk = \mathsf{H}'_3(seed, ctx)$.
- **Rule** $\mathbf{S7}^{(8)}$-$\mathsf{Send}_{\mathsf{pair}}$ Compute $kck|pmk = \mathsf{H}'_3(seed, ctx)$.

This change is indistinguishable unless the adversary or the simulator queries the H_3 oracle on the correct *seed*. Since *seed* is sampled uniformly at random from the secret random oracle H'_2, we have

$$|\Pr[\mathbf{G}_8] - \Pr[\mathbf{G}_7]| \leq \frac{q_{\mathsf{se}} + q_{\mathsf{H}_3}}{2^L}. \tag{13}$$

Game $\mathbf{G}_9$: In this game, $[K]_x$ is replaced by the session identifier $\mathsf{sid} = (s_1, CE_1, s_2, CE_2)$ and the MAC addresses for all $\mathsf{Send}_{\mathsf{pair}}$ sessions. We update the computation of *seed* as follows:

- **Rule C9$^{(9)}$-Send$_{\mathsf{pair}}$** Compute $kck|pmk = \mathsf{H}'_3(\mathrm{MAC}_1, \mathrm{MAC}_2, \mathsf{sid})$.
- **Rule S7$^{(9)}$-Send$_{\mathsf{pair}}$** Compute $kck|pmk = \mathsf{H}'_3(\mathrm{MAC}_1, \mathrm{MAC}_2, \mathsf{sid})$.

Moreover, there is no need to compute *seed* any further, and thus the rules are modified as follows:

- **Rule C7$^{(9)}$-Send$_{\mathsf{pair}}$** Do nothing.
- **Rule S5$^{(9)}$-Send$_{\mathsf{pair}}$** Do nothing.

This change is indistinguishable unless two $\mathsf{Send}_{\mathsf{pair}}$ sessions have distinct $(\mathrm{MAC}_1, \mathrm{MAC}_2, \mathsf{sid})$ but share the same $(seed, ctx)$. Since collisions in H'_2 are ruled out, sharing the same $(seed, ctx)$ implies sharing the same $([K]_x, ctx)$. So it suffices to consider the event that, among $\mathsf{Send}_{\mathsf{pair}}$ sessions, there exist two sessions with distinct $(\mathrm{MAC}_1, \mathrm{MAC}_2, \mathsf{sid})$ but sharing the same $([K]_x, ctx)$. By considering the following two cases, we obtain

$$|\Pr[\mathbf{G}_9] - \Pr[\mathbf{G}_8]| \leq \frac{2q_{\mathsf{se}}^2}{q-1}. \tag{14}$$

Case 1: The sids are identical, but the MAC addresses differ. Fix two sessions with $(s_1, CE_1), PWE_1, K_1$ and $(s_2, CE_2), PWE_2, K_2$, respectively. We then have

$$K_1 = s_1 s_2 PWE_1 + s_1 CE_2 + s_2 CE_1 + \tfrac{CE_1 CE_2}{PWE_1},$$
$$K_2 = s_1 s_2 PWE_2 + s_1 CE_2 + s_2 CE_1 + \tfrac{CE_1 CE_2}{PWE_2}.$$

Since both sessions are associated with the same server, they necessarily share the value $PT = \mathsf{H}_1(\mathrm{SSID}|\mathsf{pw})$. Nevertheless, the ordered pairs $(\mathrm{MAC}_1, \mathrm{MAC}_2)$ are distinct, which implies different values of $val = \mathsf{HMAC}(0^L, \max(\mathrm{MAC}_C, \mathrm{MAC}_S)|\min(\mathrm{MAC}_C, \mathrm{MAC}_S)) \bmod (q-1)+1$, as we exclude val-collisions. Consequently, $PWE_1 \neq PWE_2$.

By the order of message flow, the pair (s_2, CE_2) is generated after PWE_2 and thus independent of PWE_2. It then follows from the above computation that K_2 is random and independent of (s_1, CE_1). Without loss of generality, assume (s_1, CE_1) is chosen after PWE_1, PWE_2. We then have

$$K_1 = K_2 - s_1 s_2 PWE_2 - \tfrac{CE_1 CE_2}{PWE_2} + s_1 s_2 PWE_1 + \tfrac{CE_1 CE_2}{PWE_1}.$$

By the randomness of (s_1, CE_1) with respect to all other values, K_1 is random and independent of K_2. Since each x-coordination corresponds to at most two possible values of K, we conclude $\Pr\left[[K_1]_x = [K_2]_x\right] \leq 2/(q-1)$.

There are at most q_{se}^2 pairs of sessions with the same sid considered, by the union bound, the probability of existing $([K]_x, ctx)$ collision for this kind is bounded by $q_{\mathsf{se}}^2/(q-1)$.

Case 2: sids are distinct. This case can be proved similarly as above case. Let's consider two sessions with derived shared keys be

$$K_1 = s_1 s_2 PWE_1 + s_1 CE_2 + s_2 CE_1 + \tfrac{CE_1 CE_2}{PWE_1},$$
$$K_2 = s_3 s_4 PWE_2 + s_3 CE_4 + s_4 CE_3 + \tfrac{CE_3 CE_4}{PWE_2}.$$

Since sids are distinct, we can assume that (s_4, CE_4) differs from both (s_1, CE_1) and (s_2, CE_2).

By the randomness and independence of CE_4 from all other values, the probability that $[K_1]_x = [K_2]_x$ is at most $2/(q-1)$. Similarly, by the randomness of s_4, the probability that the same $ctx = s_C + s_S$ occurs is at most $1/q$. Hence, the probability of a collision in $([K]_x, ctx)$ is at most $2/(q-1)q$. Since there are at most q_{se} Send-queries, there are at most q_{se}^2 session pairs under consideration. By the birthday bound, the overall probability of a $([K]_x, ctx)$ collision is bounded by $q_{\mathsf{se}}^4/(q-1)q \leq (q_{\mathsf{se}}^2/q-1)^2 \leq q_{\mathsf{se}}^2/q-1$, where the last inequality follows from the fact that $q_{\mathsf{se}}^2/(q-1) \leq 1$.

In addition, we bound $\Pr[\mathsf{AskH}_2\text{-}\mathsf{Send}_{\mathsf{pair}}{}^{(9)}]$ by the following Lemma 2:

Lemma 2. *Let $\mathcal{A}$ be any adversary that triggers the event $\mathsf{AskH}_2\text{-}\mathsf{Send}_{\mathsf{pair}}^{(9)}$, that is, there exists an entry $([K]_x, \cdot) \in \mathcal{L}_{\mathsf{H}_2}$ corresponding to a $\mathsf{Send}_{\mathsf{pair}}$ session. Then there exists an algorithm $\mathcal{D}$ that solves the GDH problem in the group $\mathcal{G}$ such that*

$$\Pr[\mathsf{AskH}_2\text{-}\mathsf{Send}_{\mathsf{pair}}^{(9)}] \leq q_{\mathsf{se}}^2 \cdot \mathsf{Adv}_{\mathcal{G},G}^{\mathrm{gdh}}(\mathcal{D}). \tag{15}$$

Proof. We prove this by constructing a reduction algorithm $\mathcal{D}$. It solves the GDH problem in the group $\mathcal{G}$ by interacting with an adversary $\mathcal{A}$ that triggers the event $\mathsf{AskH}_2\text{-}\mathsf{Send}_{\mathsf{pair}}^{(9)}$.

The GDH challenger provides $\mathcal{D}$ with an instance $(G, X = xG, Y = yG)$. The goal of $\mathcal{D}$ is to compute $Z = DH_G(X, Y) = xyG$ with the DDH oracle.

To this end, $\mathcal{D}$ simulates the protocol environment for $\mathcal{A}$ as in $\mathbf{G}_9$, with the following modification. Upon receiving a $\mathsf{Send}_{\mathsf{pair}}$ query, it embeds the challenge values X and Y into the protocol messages CE_C and CE_S, respectively. The details are as follows. The reduction algorithm $\mathcal{D}$ randomly chooses $d_1, d_2 \in \{1, 2, \ldots, q_{\mathsf{se}}\}$ and proceeds as follows.

- **Rule C3-$\mathsf{Send}_{\mathsf{pair}}$** Choose $r_C, m_C \in \mathbb{Z}_q$ uniformly at random, and compute $s_C = (r_C + m_C) \bmod q$ and

$$CE_C = \begin{cases} -m_C \cdot PWE + X, & \text{for the } d_1\text{-th query,} \\ -m_C \cdot PWE + Y, & \text{for the } d_2\text{-th query,} \\ -m_C \cdot PWE, & \text{otherwise.} \end{cases}$$

- **Rule S3-$\mathsf{Send}_{\mathsf{pair}}$** Choose $r_S, m_S \in \mathbb{Z}_q$ uniformly at random, and compute $s_S = (r_S + m_S) \bmod q$ and

$$CE_S = \begin{cases} -m_S \cdot PWE + X, & \text{for the } d_1\text{-th query,} \\ -m_S \cdot PWE + Y, & \text{for the } d_2\text{-th query,} \\ -m_S \cdot PWE, & \text{otherwise.} \end{cases}$$

Note that the two challenge values X and Y are embedded into two different sessions. It may happen that only one of these sessions becomes a $\mathsf{Send}_{\mathsf{pair}}$ session, while the other does not, which prevents a perfect simulation.

Consider the case where a server session embedded with Y receives a commit message from a client session embedded with X. In this case, the server session forms a $\mathsf{Send}_{\mathsf{pair}}$ session, whereas the client session does not necessarily do so. In particular, the client session does not receive the corresponding server commit, becomes corrupted, and receives an adversarially forged commit message. As a result, the client session needs to compute K, but it fails to do so since it is embedded with X, and hence the simulation cannot be answered perfectly.

To resolve this issue, we rely on a DDH oracle to achieve a perfect simulation. Specifically, consider the session for which K cannot be directly derived. If the queried value $[K]_x$ already appears in the list $\mathcal{L}_{\mathsf{H}_2}$, then we recover the corresponding K using the DDH oracle. Otherwise, we sample a random value $a \leftarrow\!\$\ \{0,1\}^L$, assign it to $\mathsf{H}_2([K]_x)$, and record this assignment. For any subsequent $\mathsf{H}_2([K]_x)$ query, we use the DDH oracle to test whether the queried key corresponds to the session associated with a, thereby ensuring perfect consistency throughout the simulation.

By the randomness of d_1 and d_2, the event $\mathsf{AskH}_2\text{-}\mathsf{Send}_{\mathsf{pair}}^{(9)}$ corresponds to a $\mathsf{Send}_{\mathsf{pair}}$ session embedded with (X, Y) with probability at least $1/q_{\mathsf{se}}^2$. If the adversary $\mathcal{A}$ triggers $\mathsf{AskH}_2\text{-}\mathsf{Send}_{\mathsf{pair}}^{(9)}$, then, with access to the DDH oracle, the value $[K]_x$ of this session can be found in the list $\mathcal{L}_{\mathsf{H}_2}$ efficiently.

As in the discussion of Lemma 1, once we obtain the corresponding session and $[K]_x$, xyG can be extracted. This completes the proof. □

Game $\mathbf{G}_{10}$: In this game, we replace the random oracle H_4 with a secret random oracle H_4' for all $\mathsf{Send}_{\mathsf{pair}}$ sessions. The computation rules are updated as follows:

- **Rule C10$^{(10)}$-Send$_{\text{pair}}$** Let $con_C = \mathsf{H}'_4(\mathrm{MAC}_1, \mathrm{MAC}_2, \mathsf{sid}, s_C|CE_C|s_S|CE_S)$.
- **Rule C11$^{(10)}$-Send$_{\text{pair}}$** Drop and wait if $con_S \neq \mathsf{H}'_4(\mathrm{MAC}_1, \mathrm{MAC}_2, \mathsf{sid}, s_S|CE_S|s_C|CE_C)$. Otherwise, the client session accepts, stores pmk and terminates.
- **Rule S8$^{(10)}$-Send$_{\text{pair}}$** Let $con_S = \mathsf{H}'_4(\mathrm{MAC}_1, \mathrm{MAC}_2, \mathsf{sid}, s_S|CE_S|s_C|CE_C)$.
- **Rule S9$^{(10)}$-Send$_{\text{pair}}$** Drop and wait if $con_C \neq \mathsf{H}'_4(\mathrm{MAC}_1, \mathrm{MAC}_2, \mathsf{sid}, s_S|CE_S|s_C|CE_C)$. Otherwise, the server session accepts, stores pmk and terminates.

Since we avoid collisions of kck, this transition only differs if the adversary makes a H_4 query on kck. However, as kck is derived from the secret random function H'_3, such an event can only occur by chance. Hence, we obtain

$$|\Pr[\mathbf{G}_{10}] - \Pr[\mathbf{G}_9]| \leq \frac{q_{\mathsf{H}_4}}{2^L}. \tag{16}$$

Game $\mathbf{G}_{11}$: We have now completed the analysis of *Execute Case* (from game $\mathbf{G}_4$ to $\mathbf{G}_6$) and *Paired Case* (from game $\mathbf{G}_7$ to $\mathbf{G}_{10}$). Three cases remain to be handled: *Cross-Server Paired Case*, *Malicious-Client Case*, and *Malicious-Server Case*. Their definitions are recalled in cases 1–3 below. These cases could be analyzed separately in a sequence of games; however, doing so would lead to a looser final bound. Instead, we handle all three cases simultaneously in $\mathbf{G}_{11}$, which may introduce several bad events. We therefore analyze these events carefully in order to obtain a tight bound.

First, we define a predicate uncorrupted, indicating that a session's parties have not been corrupted; namely, neither a iCorrupt nor a sCorrupt query has been issued about it. We refer to such a session as an uncorrupted session. In contrast, we define a predicate corrupted, indicating that a session's parties have been corrupted, and refer to it as an corrupted session.

All uncorrupted sessions not covered by the above analysis reject any authenticator; this includes sessions of *Cross-Server Paired Case*, *Malicious-Client Case*, and *Malicious-Server Case*. The behavior of corrupted sessions remains unchanged. We present the details in the following order.

1. **Cross-Server Paired Case.** Transcripts created via Send queries in which the two commit elements originate from sessions associated with *different* servers. Concretely, consider a $\mathsf{Send}(C, i, (s_S, CE_S))$ query where (s_S, CE_S) originates from a non-Execute session associated with a server different from that of π^i_C; the case of $\mathsf{Send}(S, j, (\mathrm{MAC}_C, s_C, CE_C))$ is symmetric. For simplicity, we refer to such sessions as $\mathsf{Send}_{\text{cs-pair}}$ sessions and denote the corresponding queries by the superscript $\mathsf{Send}_{\text{cs-pair}}$. The rules are modified as follows. We use the secret random oracle H'_4 for uncorrupted sessions, and all such sessions reject any authenticator. The behavior of corrupted sessions remains unchanged.
   - **Rule C10$^{(11)}$-Send$_{\text{cs-pair}}$**
     - If uncorrupted: $con_C = \mathsf{H}'_4(s_C|CE_C|s_S|CE_S)$.
     - If corrupted: $con_C = \mathsf{H}_4(kck, s_C|CE_C|s_S|CE_S)$.
   - **Rule C11$^{(11)}$-Send$_{\text{cs-pair}}$**
     - If uncorrupted: Drop and wait.
     - If corrupted: Drop and wait if $con_S \neq \mathsf{H}_4(kck, s_S|CE_S|s_C|CE_C)$; otherwise, accept, store pmk and terminate.
   - **Rule S8$^{(11)}$-Send$_{\text{cs-pair}}$**
     - If uncorrupted: $con_S = \mathsf{H}'_4(s_S|CE_S|s_C|CE_C)$.
     - If corrupted: $con_S = \mathsf{H}_4(kck, s_S|CE_S|s_C|CE_C)$.
   - **Rule S9$^{(11)}$-Send$_{\text{cs-pair}}$**
     - If uncorrupted: Drop and wait.
     - If corrupted: Drop and wait if $con_C \neq \mathsf{H}_4(kck, s_C|CE_C|s_S|CE_S)$; otherwise, accept, store pmk and terminate.

2. **Malicious-Client Case.** Generated by $\mathsf{Send}(S, j, (\mathrm{MAC}_C, s_C, CE_C))$ queries in which the commit element (s_C, CE_C) is adversarially forged. We refer to sessions containing such adversarially forged commit messages as $\mathsf{Send}_{\mathcal{A}}$ sessions; this also covers the following *Malicious-Server Case*. We use the secret random oracle H'_4 for uncorrupted sessions, and all such sessions reject any authenticator. The behavior of corrupted sessions remains unchanged. The details are as follows:
   - ▶ **Rule** $\mathbf{S8}^{(11)\text{-}\mathsf{Send}_{\mathcal{A}}}$
     - If uncorrupted: $con_S = \mathsf{H}'_4(s_S|CE_S|s_C|CE_C)$.
     - If corrupted: $con_S = \mathsf{H}_4(kck,\ s_S|CE_S|s_C|CE_C)$.
   - ▶ **Rule** $\mathbf{S9}^{(11)\text{-}\mathsf{Send}_{\mathcal{A}}}$
     - If uncorrupted: Drop and wait.
     - If corrupted: Drop and wait if $con_C \neq \mathsf{H}_4(kck,\ s_C|CE_C|s_S|CE_S)$; otherwise, accept, store pmk and terminate.
3. **Malicious-Server Case.** Generated by $\mathsf{Send}(C, i, (s_S, CE_S))$ where (s_S, CE_S) is adversarially forged. The rules are modified as follows. We use H'_4 for uncorrupted sessions, and all such sessions reject any authenticator. The behavior of corrupted sessions remains unchanged.
   - ▶ **Rule** $\mathbf{C10}^{(11)\text{-}\mathsf{Send}_{\mathcal{A}}}$
     - If uncorrupted: $con_C = \mathsf{H}'_4(s_C|CE_C|s_S|CE_S)$.
     - If corrupted: $con_C = \mathsf{H}_4(kck,\ s_C|CE_C|s_S|CE_S)$.
   - ▶ **Rule** $\mathbf{C11}^{(11)\text{-}\mathsf{Send}_{\mathcal{A}}}$
     - If uncorrupted: Drop and wait.
     - If corrupted: Drop and wait if $con_S \neq \mathsf{H}_4(kck,\ s_S|CE_S|s_C|CE_C)$; otherwise, accept, store pmk and terminate.

We modify the random-oracle responses to $\mathsf{H}_4(kck, \cdot)$ queries such that, in the post-corruption setting, the view of any $\mathsf{Send}_{\mathcal{A}}$ session (i.e., *Malicious-Client Case* and *Malicious-Server Case*) is identical to that in the previous game, except with negligible probability. The modification proceeds as follows. For H_4 queries corresponding to a corrupted session, the simulator returns the value generated by the secret random oracle H'_4 used in the simulation. Since the simulator learns the password after corruption, it can compute the correct shared key K and all related values, which enables the reprogramming of H_4. Specifically, once the password pw is known, the password element PWE can be computed as $PWE = val \cdot \mathsf{H2C}(\mathrm{SSID} \mid \mathsf{pw})$. Consequently, each client (resp., server) session computes the shared key $K = \left(s_C + \frac{-m_C}{val \cdot \alpha} G\right)(s_S \cdot PWE + CE_S)$ (resp., $K = \left(s_S + \frac{-m_S}{val \cdot \alpha} G\right)(s_C \cdot PWE + CE_C)$).

- ▶ **Rule** $\mathsf{H}_4^{(11)\text{-}\mathsf{Send}_{\mathcal{A}}}$ **(using secret oracle** H'_4**)** Upon a query $\mathsf{H}_4(kck, w)$:
  - If (kck, w) corresponds to a corrupted $\mathsf{Send}_{\mathcal{A}}$ session, return $r = \mathsf{H}'_4(w)$.
  - Otherwise, sample $r \leftarrow\!\!\$\ \{0,1\}^L$ uniformly.

We begin by analyzing the distinguishing advantage between the last game $\mathbf{G}_{10}$ and the current game $\mathbf{G}_{11}$. We first handle *Cross-Server Paired Case* and define the following events.

- $\mathsf{AskH}_4\text{-}\mathsf{Send}_{\mathsf{cs\text{-}pair}}{}^{(11)}$: the adversary makes a $\mathsf{H}_4(kck, w)$ query matching a $\mathsf{Send}_{\mathsf{cs\text{-}pair}}$ session;
- $\mathsf{wAuth}\text{-}\mathsf{Send}_{\mathsf{cs\text{-}pair}}{}^{(11)}$: the adversary sends a valid authenticator to a $\mathsf{Send}_{\mathsf{cs\text{-}pair}}$ session, while the event $\mathsf{AskH}_4\text{-}\mathsf{Send}_{\mathsf{cs\text{-}pair}}{}^{(11)}$ does not occur;
- $\mathsf{KColl}\text{-}\mathsf{Send}_{\mathsf{cs\text{-}pair}}{}^{(11)}$: there exist two uncorrupted $\mathsf{Send}_{\mathsf{cs\text{-}pair}}$ sessions with the same (sid, kck).

Next, we handle *Malicious-Client Case* and *Malicious-Server Case* by defining the following events. Here, we emphasize that the first event below is considered in the *before corruption* setting; that is, the adversary does not issue any iCorrupt or sCorrupt about the server of related sessions.

- $\mathsf{AskH}_4\mathsf{bC}\text{-}\mathsf{Send}_{\mathcal{A}}{}^{(11)}$: the adversary makes an $\mathsf{H}_4(kck, w)$ query matching a $\mathsf{Send}_{\mathcal{A}}$ session *before corruption*;

– $\mathsf{wAuth\text{-}Send}_{\mathcal{A}}{}^{(11)}$: the adversary sends a valid authenticator to an uncorrupted $\mathsf{Send}_{\mathcal{A}}$ session, while the event $\mathsf{AskH_4bC\text{-}Send}_{\mathcal{A}}{}^{(11)}$ does not occur.

If neither $\mathsf{AskH_4\text{-}Send_{cs\text{-}pair}}{}^{(11)}$ nor $\mathsf{AskH_4bC\text{-}Send}_{\mathcal{A}}{}^{(11)}$ occurs, then replacing H_4 with H'_4 yields a distribution identical to that of the previous game.

If neither $\mathsf{wAuth\text{-}Send_{cs\text{-}pair}}{}^{(11)}$ nor $\mathsf{wAuth\text{-}Send}_{\mathcal{A}}{}^{(11)}$ occurs, then discarding only adversarial authenticators suffices to obtain an identical distribution. Moreover, if the event $\mathsf{KColl\text{-}Send_{cs\text{-}pair}}{}^{(11)}$ does not occur, then any authenticator generated by the simulator will not be authenticated by *Cross-Server Paired Case*; hence, discarding all authenticators does not affect the distribution.

We first give bounds for two events: $\mathsf{wAuth\text{-}Send_{cs\text{-}pair}}{}^{(11)}$ and $\mathsf{wAuth\text{-}Send}_{\mathcal{A}}{}^{(11)}$.

For the event $\mathsf{wAuth\text{-}Send_{cs\text{-}pair}}{}^{(11)}$, since H_4 is modeled as a random oracle, if the event $\mathsf{AskH_4\text{-}Send_{cs\text{-}pair}}{}^{(11)}$ does not occur, then the adversary can only guess the corresponding authenticator. Therefore, we have

$$\Pr\left[\mathsf{wAuth\text{-}Send_{cs\text{-}pair}}{}^{(11)}\right] \;\le\; \frac{2q_{\mathsf{se}}}{2^L},$$

where the factor 2 accounts for the fact that, for a fixed session identifier sid, two distinct $\mathsf{Send_{cs\text{-}pair}}$ sessions may derive two different key confirmation keys kck.

Similarly, since each $\mathsf{Send}_{\mathcal{A}}$ session always has a distinct sid, we have

$$\Pr[\mathsf{wAuth\text{-}Send}_{\mathcal{A}}{}^{(11)}] \;\le\; \frac{q_{\mathsf{se}}}{2^L}.$$

Therefore, it remains to bound three additional events, and we obtain

$$\begin{aligned}\left|\Pr[\mathbf{G}_{11}] - \Pr[\mathbf{G}_{10}]\right| \;\le\; &\frac{3q_{\mathsf{se}}}{2^L} + \Pr\left[\mathsf{KColl\text{-}Send_{cs\text{-}pair}}{}^{(11)}\right] \\ &+ \Pr\left[\mathsf{AskH_4\text{-}Send_{cs\text{-}pair}}{}^{(11)}\right] + \Pr\left[\mathsf{AskH_4bC\text{-}Send}_{\mathcal{A}}{}^{(11)}\right].\end{aligned} \tag{17}$$

Since we drop all authenticators except after corruption, the simulator never needs to compute K, *seed*, *kck*, or *pmk* beforehand. Hence, without loss of generality, we may postpone these computations until the moment *after* corruption, which results in an identical distribution from the adversary's perspective. Under this modification, we proceed to bound three additional events of equation (17); $\Pr[\mathsf{KColl\text{-}Send_{cs\text{-}pair}}{}^{(11)}]$, $\Pr[\mathsf{AskH_4\text{-}Send_{cs\text{-}pair}}{}^{(11)}]$, and $\Pr[\mathsf{AskH_4bC\text{-}Send}_{\mathcal{A}}{}^{(11)}]$ are bounded in the following Lemma 3, Lemma 4 and Lemma 5, respectively.

Lemma 3. *Let* $\mathsf{Coll}_{PWE}{}^{(11)}$ *denote the event that there exist two* uncorrupted $\mathsf{Send_{cs\text{-}pair}}$ *sessions with the same* sid *and the same PWE. Then*

$$\Pr[\mathsf{KColl\text{-}Send_{cs\text{-}pair}}{}^{(11)}] \;\le\; \frac{q_{\mathsf{se}}^2}{q-1} \;+\; \Pr[\mathsf{Coll}_{PWE}{}^{(11)}], \tag{18}$$

where $\Pr[\mathsf{Coll}_{PWE}{}^{(11)}]$ *is shown in Lemma 7.*

Proof. Since

$$\begin{aligned}\Pr\left[\mathsf{KColl\text{-}Send_{cs\text{-}pair}}{}^{(11)}\right] &= \Pr\left[\mathsf{KColl\text{-}Send_{cs\text{-}pair}}{}^{(11)} \wedge \neg\mathsf{Coll}_{PWE}{}^{(11)}\right] \\ &\quad + \Pr\left[\mathsf{KColl\text{-}Send_{cs\text{-}pair}}{}^{(11)} \wedge \mathsf{Coll}_{PWE}{}^{(11)}\right] \\ &\le \Pr\left[\mathsf{KColl\text{-}Send_{cs\text{-}pair}}{}^{(11)} \wedge \neg\mathsf{Coll}_{PWE}{}^{(11)}\right] \\ &\quad + \Pr\left[\mathsf{Coll}_{PWE}{}^{(11)}\right],\end{aligned}$$

it suffices to show that $\Pr[\mathsf{KColl\text{-}Send_{cs\text{-}pair}}{}^{(11)} \wedge \neg\mathsf{Coll}_{PWE}{}^{(11)}] \;\le\; q_{\mathsf{se}}^2/(q-1)$. The lemma then follows.

The event $\mathsf{KColl\text{-}Send_{cs\text{-}pair}}^{(11)} \wedge \neg \mathsf{Coll}_{PWE}{}^{(11)}$ means that there exist two uncorrupted $\mathsf{Send_{cs\text{-}pair}}$ sessions sharing the same (sid, kck) pair but having distinct password elements PWE.

As stated in $\mathbf{G}_9$, Case 1, we have $\Pr\left[[K_1]_x = [K_2]_x \mid PWE_1 \neq PWE_2\right] \leq 2/(q-1)$. Moreover, since collisions of the key encryption key kck derived from distinct $[K]_x$ values are avoided in $\mathbf{G}_3$, this bound applies to each fixed pair of $\mathsf{Send_{cs\text{-}pair}}$ sessions with the same session identifier sid. There are at most $\binom{q_{\mathsf{se}}}{2} \leq q_{\mathsf{se}}^2/2$ such $\mathsf{Send_{cs\text{-}pair}}$ session pairs. Applying a union bound over all these pairs yields the claimed bound. □

Next, we bound the second term $\Pr\left[\mathsf{AskH}_4\text{-}\mathsf{Send_{cs\text{-}pair}}^{(11)}\right]$ in equation (17).

Lemma 4. *Recall that* $\mathsf{AskH}_4\text{-}\mathsf{Send_{cs\text{-}pair}}^{(11)}$ *is the event that the adversary makes a* $\mathsf{H}_4(kck, w)$ *query matching an* uncorrupted $\mathsf{Send_{cs\text{-}pair}}$ *session. Then*

$$\Pr[\mathsf{AskH}_4\text{-}\mathsf{Send_{cs\text{-}pair}}^{(11)}] \leq \frac{q_{\mathsf{H}_4}}{2^L} + \frac{q_{\mathsf{se}}(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3})}{2^L} + q_{\mathsf{se}}^2 \cdot \mathsf{Adv}^{\mathrm{gdh}}_{\mathcal{G},G}(\mathcal{D}). \tag{19}$$

Proof. For any uncorrupted $\mathsf{Send_{cs\text{-}pair}}$ session, the key kck is derived as $kck = \mathsf{H}_3(seed, \cdot)$. Thus, the adversary can trigger this bad event only if it obtains the correct kck. This can happen in the following ways.

1. *Guessing kck directly in a* H_4 *query.* Since H_3 is modeled as a random oracle, the value kck is uniformly distributed over $\{0,1\}^L$ conditioned on the adversary's view. Therefore, over at most $|\mathcal{L}_{\mathsf{H}_4}| \leq q_{\mathsf{H}_4}$ queries, the probability that the adversary queries the correct pair (kck, w) is at most $q_{\mathsf{H}_4}/2^L$.
2. *Computing kck by querying* H_3 *on the correct input* $(seed, ctx)$. The adversary must obtain the correct *seed* for some uncorrupted $\mathsf{Send_{cs\text{-}pair}}$ session. This can occur in two subcases:
   - *Guessing seed and querying* $\mathsf{H}_3(seed, \cdot)$. For each target session there is one correct value of *seed*, and there are at most q_{se} such target sessions. Since *seed* is from random oracle, over at most $|\mathcal{L}_{\mathsf{H}_3}| \leq q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3}$ queries to H_3, the probability of ever hitting a correct *seed* is at most $q_{\mathsf{se}}(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3})/2^L$.
   - *Deriving seed by querying* H_2 *on the correct* $[K]_x$ *of an* uncorrupted $\mathsf{Send_{cs\text{-}pair}}$ *session.* This case is identical to that analyzed in Lemma 2, and its probability is bounded by $q_{\mathsf{se}}^2 \cdot \mathsf{Adv}^{\mathrm{gdh}}_{\mathcal{G},G}(\mathcal{D})$.

Applying a union bound over the above cases yields the claimed inequality.

□

In the following Lemma 5, we bound the third term $\Pr[\mathsf{AskH}_4\mathsf{bC\text{-}Send}_{\mathcal{A}}{}^{(11)}]$ in equation (17), and defer the analysis of all remaining events to the final game $\mathbf{G}_{12}$.

Lemma 5. *Let* $\mathsf{AskH}_2\mathsf{bC\text{-}Send}_{\mathcal{A}}\mathsf{S}^{(11)}$ *denote the event that, before corruption, there exists a correct* $\mathsf{H}_2([K]_x)$ *query corresponding to some* $\mathsf{Send}_{\mathcal{A}}$ *server session. Let* $\mathsf{Coll_S}^{(11)}$ *denote the event that, before corruption, there exist one or more* $\mathsf{H}_2([K]_x)$ *queries on* $[K]_x$ *values induced by one or more passwords for a fixed* $\mathsf{Send}_{\mathcal{A}}$ *server session. Similarly, let* $\mathsf{AskH}_2\mathsf{bC\text{-}Send}_{\mathcal{A}}\mathsf{C}^{(11)}$ *and* $\mathsf{Coll_C}^{(11)}$ *denote the analogous events for an* uncorrupted $\mathsf{Send}_{\mathcal{A}}$ *client session. Then*

$$\begin{aligned}
\Pr[\mathsf{AskH}_4\mathsf{bC{-}Send}_{\mathcal{A}}{}^{(11)}] \leq\ & \frac{q_{\mathsf{H}_4}}{2^L} + \frac{q_{\mathsf{se}}(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3})}{2^L} \\
&+ 4q_{\mathsf{se}} \cdot (q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1})^2 \cdot \sqrt{\mathsf{Adv}^{\mathrm{wgsdh}}_{\mathcal{G},G}(\mathcal{E})} \\
&+ 4q_{\mathsf{se}} \cdot (q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1})^2 \cdot \mathsf{Adv}^{\mathrm{gsdh}}_{\mathcal{G},G}(\mathcal{F}) \\
&+ \Pr[\mathsf{AskH}_2\mathsf{bC\text{-}Send}_{\mathcal{A}}\mathsf{S}^{(11)} | \neg \mathsf{Coll_S}^{(11)}] \\
&+ \Pr[\mathsf{AskH}_2\mathsf{bC\text{-}Send}_{\mathcal{A}}\mathsf{C}^{(11)} | \neg \mathsf{Coll_C}^{(11)}],
\end{aligned} \tag{20}$$

where the remaining two terms are shown in Lemma 7.

Proof. Recall that $\mathsf{AskH_4bC{-}Send}_{\mathcal{A}}^{(11)}$ is the event that the adversary makes a $\mathsf{H}_4(kck, w)$ query matching an $\mathsf{Send}_{\mathcal{A}}$ session. For any $\mathsf{uncorrupted\ Send_{cs\text{-}pair}}$ session, the key kck is derived as $kck = \mathsf{H}_3(seed, \cdot)$. Thus, the adversary can trigger this bad event only if it obtains the correct kck. This can happen in the following ways.

1. *Direct guessing of kck at* H_4. With at most $|\mathcal{L}_{\mathsf{H}_4}| \leq q_{\mathsf{H}_4}$ queries, the success probability is bounded by $q_{\mathsf{H}_4}/2^L$.
2. *Querying* H_3 *with the correct input* $(seed, \cdot)$. The only unknown is *seed*. The adversary can either
   - obtain *seed* by querying H_3, since at most q_{se} correct kck values exist and $|\mathcal{L}_{\mathsf{H}_3}| \leq q_{\mathsf{H}_3} + q_{\mathsf{se}} + q_{\mathsf{exe}}$. The success probability in this case is bounded by $q_{\mathsf{se}}(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3})/2^L$, or
   - obtain *seed* by querying H_2 on the correct $[K]_x$ of some $\mathsf{uncorrupted\ Send}_{\mathcal{A}}$ session. This event is captured by $\mathsf{AskH_2bC\text{-}Send}_{\mathcal{A}}\mathsf{S}^{(11)}$ or $\mathsf{AskH_2bC\text{-}Send}_{\mathcal{A}}\mathsf{C}^{(11)}$, depending on whether the target session is server-side or client-side.

Moreover, we can decompose:

$$\Pr[\mathsf{AskH_2bC\text{-}Send}_{\mathcal{A}}\mathsf{S}^{(11)}] \leq \Pr[\mathsf{Coll_S}^{(11)}] + \Pr[\mathsf{AskH_2bC\text{-}Send}_{\mathcal{A}}\mathsf{S}^{(11)} | \neg \mathsf{Coll_S}^{(11)}],$$
$$\Pr[\mathsf{AskH_2bC\text{-}Send}_{\mathcal{A}}\mathsf{C}^{(11)}] \leq \Pr[\mathsf{Coll_C}^{(11)}] + \Pr[\mathsf{AskH_2bC\text{-}Send}_{\mathcal{A}}\mathsf{C}^{(11)} | \neg \mathsf{Coll_C}^{(11)}].$$

By the following Lemma 6, we have a bound on $\Pr[\mathsf{Coll_S}^{(11)}]$. By the same reasoning as in Lemma 6, but noting that it suffices to find the first triple to reduce to the GSDH assumption, we obtain

$$\Pr[\mathsf{Coll_C}^{(11)}] \ \leq \ 4q_{\mathsf{se}} \cdot (q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1})^2 \cdot \mathsf{Adv}^{\mathrm{gsdh}}_{\mathcal{G},G}(\mathcal{F}).$$

The claim follows by a union bound over the above events. □

The following lemma bounds the term $\Pr[\mathsf{Coll_S}^{(11)}]$ left from Lemma 5.

Lemma 6. *Let* $\mathcal{A}$ *be any adversary that triggers the event* $\mathsf{Coll_S}^{(11)}$*, i.e., there exists an* $\mathsf{uncorrupted\ Send}_{\mathcal{A}}$ *server session's transcript* $\mathcal{T}$ *such that two distinct values* pw_1 *and* pw_2 *yield shared keys* $[K_1]_x, [K_2]_x \in \mathcal{L}_{\mathsf{H}_2}$*. Then, there exists an algorithm* $\mathcal{E}$ *solving the wSDH problem in group* $\mathcal{G}$ *such that:*

$$\Pr[\mathsf{Coll_S}^{(11)}] \leq 4q_{\mathsf{se}} \cdot (q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1})^2 \cdot \sqrt{\mathsf{Adv}^{\mathrm{wgsdh}}_{\mathcal{G},G}(\mathcal{E})}.$$

Proof. We construct an algorithm $\mathcal{B}$ that, given random Diffie–Hellman values $\langle A, B\rangle$ (where $A = aG$ and $B = bG$), attempts to find two non - zero triple $(H, a^{-1}H, b^{-1}H), (K = kG, a^{-1}kK, b^{-1}kK) \in \mathcal{G}^3$.

1. Find the first non-zero triple $(H, a^{-1}H, b^{-1}H)$**.**

It selects two indices $d_1, d_2 \leftarrow\!\!\$\ \{1, 2, \ldots, |\mathcal{L}_{\mathsf{H}_1}|\}$ uniformly at random, intending to guess the queries corresponding to pw_1 and pw_2 in the H_1 random oracle. Then it programs the oracle according to the following rule.

▶ **Rule** H_1

Upon receiving a query w to H_1, proceed as follows:

1. If w is the d_1-th query, set $r := A$ and $\alpha := \perp$.
2. If w is the d_2-th query, set $r := B$ and $\alpha := \perp$.
3. Otherwise, sample $\alpha \leftarrow\!\!\$\ \mathbb{Z}_q$ uniformly at random and compute $r := \alpha G$.

And the randomly choose $d_3 \in \{1, 2, \ldots, q_{\mathsf{se}}\}$ — to guess the transcript $\mathcal{T}$. We further modify the simulation as follows:

▶ **Rule S3**$^{\mathsf{Send}_{\mathcal{A}}}$ For generating (s_S, CE_S), proceed as follows:

1. For the d_3-th query (where y is the difference): sample $s_S, y \leftarrow\!\!\$\ \mathbb{Z}_q$ and set $CE_S = yG$.

2. For all other queries (without modification): sample $s_S, m_S \leftarrow\!\$\, \mathbb{Z}_q \setminus \{0,1\}$ and set $CE_S = -m_S \cdot PWE$.

Due to the randomness of the guess, the probability that the d_1-th and d_2-th queries to H_1 match pw_1 and pw_2, respectively, is at least $1/|\mathcal{L}_{\mathsf{H}_1}|^2 \geq 1/(q_{\mathsf{H}_1} + q_{\mathsf{se}} + q_{\mathsf{exe}})^2$. If either password has not been queried to H_1, we program the oracle by assigning $A = \mathsf{H}_1(\mathrm{SSID}|\mathsf{pw}_1)$, $B = \mathsf{H}_1(\mathrm{SSID}|\mathsf{pw}_2)$, since H_1 is modeled as a random oracle. Thus, with probability at least $1/(q_{\mathsf{H}_1} + q_{\mathsf{se}} + q_{\mathsf{exe}})^2$, we successfully embed the challenge values A and B into the corresponding H_1 queries, i.e., $A = \mathsf{H}_1(\mathrm{SSID}|\mathsf{pw}_1), B = \mathsf{H}_1(\mathrm{SSID}|\mathsf{pw}_2)$.

Next, we guess the target transcript $\mathcal{T}$. Since there are at most q_{se} sessions, the probability that $\mathcal{T}$ coincides with the d_3-th session is at least $1/q_{\mathsf{se}}$ due to the randomness of d_3.

Finally, we invoke the DDH oracle to obtain $[K_1]_x$ and $[K_2]_x$ in $\mathcal{L}_{\mathsf{H}_2}$. Since in an elliptic curve group each x-coordinate corresponds to at most two group elements, we can recover K_1 and K_2 with probability at least $1/4$.

When all guesses are correct, we have

$$\begin{aligned}
K_1 =& \mathrm{DH}_{PWE}(s_C \cdot PWE + CE_C,\ s_S \cdot PWE + CE_S) \\
& /\!/ \text{ (definition of } K) \\
=& \mathrm{DH}_{PWE}(s_C \cdot PWE + xG,\ s_S \cdot PWE + yG) \\
& /\!/ \text{ (assuming } CE_C = xG \text{ and known } CE_S = yG,) \\
=& \mathrm{DH}_{val \cdot aG}(s_C \cdot val \cdot aG + xG,\ s_S \cdot val \cdot aG + yG) \\
& (\text{since } PWE = val \cdot A = val \cdot aG) \\
=& (s_C + \tfrac{x}{val \cdot a})(s_S \cdot val \cdot aG + yG) \\
& /\!/ \text{ (expansion of DH}_{PWE}) \\
=& s_C s_S \cdot val \cdot aG + s_C \cdot yG + s_S \cdot xG + \tfrac{y \cdot xG}{val \cdot a} \\
& /\!/ \text{ (straightforward computation)} \\
=& s_C s_S \cdot val \cdot A + s_C\, CE_S + s_S\, CE_C + \tfrac{y \cdot CE_C}{val \cdot a} \\
& /\!/ \text{ (since } A = aG,\ CE_C = xG,\ CE_S = yG).
\end{aligned}$$

Then, we can extract

$$a^{-1} CE_C = (K_1 - s_C s_S \cdot val \cdot A - s_C\, CE_S - s_S\, CE_C) \cdot val \cdot y^{-1}.$$

Similarly, we get

$$b^{-1} CE_C = (K_2 - s_C s_S \cdot val \cdot B - s_C\, CE_S - s_S\, CE_C) \cdot val \cdot y^{-1}.$$

Therefore, upon the event $\mathsf{Coll}_{\mathsf{S}}{}^{(11)}$, with probability at least $1/4q_{\mathsf{se}}(q_{\mathsf{H}_1} + q_{\mathsf{se}} + q_{\mathsf{exe}})^2$, the reduction algorithm $\mathcal{E}$ succeeds in finding the first non-zero triple $(H, a^{-1}H, b^{-1}H)$:

$$\left(CE_C,\ a^{-1} CE_C,\ b^{-1} CE_C\right).$$

2. Find the second non-zero triple $(K = kG, a^{-1}kK, b^{-1}kK)$**.**

It is the same discussion as above, except that we modify the simulation of generating (s_S, CE_S) as follows: Upon the d_3-th query, the reply is determined based on the received CE_C.

▶ **Rule S3**$^{\mathsf{Send}_{\mathcal{A}}}$ For generating (s_S, CE_S), proceed as follows:

1. For the d_3-th query (based on the received CE_C): sample $s_S, y \leftarrow\!\$\, \mathbb{Z}_q \setminus \{0,1\}$ and set $CE_S = y \cdot CE_C$.
2. For all other queries (without modification): sample $s_S, m_S \leftarrow\!\$\, \mathbb{Z}_q \setminus \{0,1\}$ and set $CE_S = -m_S \cdot PWE$.

By the same reasoning as above, the probability of correctly guessing $\mathcal{T}, \mathsf{pw}_1, \mathsf{pw}_2, K_1, K_2$ is at least $1/4q_{\mathsf{se}}(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1})^2$.

When all guesses are correct, we have

$$\begin{aligned}
K_1 =& \mathrm{DH}_{PWE}(s_C \cdot PWE + CE_C,\ s_S \cdot PWE + CE_S) \\
& /\!/ \text{ (definition of } K) \\
=& \mathrm{DH}_{PWE}(s_C \cdot PWE + xG,\ s_S \cdot PWE + xyG) \\
& /\!/ \text{ (assuming } CE_C = xG, \text{ knowing } CE_S = yCE_C) \\
=& \mathrm{DH}_{val \cdot aG}(s_C \cdot val \cdot aG + xG,\ s_S \cdot val \cdot aG + xyG) \\
& /\!/ \text{ (since } PWE = val \cdot A = val \cdot aG) \\
=& (s_C + \tfrac{x}{val \cdot a})(s_S \cdot val \cdot aG + xyG) \\
& /\!/ \text{ (expansion of } \mathrm{DH}_{PWE}) \\
=& s_C s_S \cdot val \cdot aG + s_C \cdot xyG + s_S \cdot xG + \tfrac{y \cdot x^2 G}{val \cdot a} \\
& /\!/ \text{ (straightforward computation)} \\
=& s_C s_S \cdot val \cdot A + s_C\, CE_S + s_S\, CE_C + \tfrac{y \cdot x\, CE_S}{val \cdot a} \\
& /\!/ \text{ (since } A = aG,\ CE_C = xG,\ CE_S = yCE_C = xyG).
\end{aligned}$$

Then, we can extract:

$$a^{-1} x\, CE_C = (K_1 - s_C s_S \cdot val \cdot A - s_C\, CE_S - s_S\, CE_C) \cdot val \cdot y^{-1}.$$

Similarly, we get

$$b^{-1} x\, CE_C = (K_2 - s_C s_S \cdot val \cdot B - s_C\, CE_S - s_S\, CE_C) \cdot val \cdot y^{-1}.$$

Namely, when the event $\mathsf{Coll}_{\mathsf{S}}{}^{(11)}$ occurs, we have a probability of at least $1/q_{\mathsf{se}}(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1})^2 4(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_2})^2$ finding the second non-zero triple ($K = kG, a^{-1}kG, b^{-1}G$):

$$(CE_C = xG, a^{-1} x\, CE_C, b^{-1} x\, CE_C).$$

Above all, the probability of solving wSDH is bounded as

$$\frac{\Pr[\mathsf{Coll}_{\mathsf{S}}{}^{(11)}]^2}{16 q_{\mathsf{se}}^2 (q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1})^4} \leq \mathsf{Adv}^{\mathrm{wgsdh}}_{\mathcal{G},G}(\mathcal{E}).$$

□

Game $\mathbf{G}_{12}$: In this game, we provide the probability that was not derived above. For all $\mathsf{uncorrupted}$ sessions, the game can be perfectly simulated without passwords according to the games proved above. In particular, we may set CE_C, CE_S as

$$CE_C = -m_C G, \qquad CE_S = -m_S G,$$

which avoids the use of pw. Hence, the value of pw is not fixed until corruption occurs or until the very end of the game. This game is indistinguishable from the previous one:

$$|\Pr[\mathbf{G}_{12}] - \Pr[\mathbf{G}_{11}]| = 0. \tag{21}$$

Finally, we give the following Lemmas 7 and 8 to conclude the proof.

Lemma 7. *We have the following bounds on the events* $\mathsf{Coll}_{PWE}{}^{(12)}$, $\mathsf{AskH_2bC\text{-}Send}_{\mathcal{A}}\mathsf{S}^{(12)}$, *and* $\mathsf{AskH_2bC\text{-}Send}_{\mathcal{A}}\mathsf{C}^{(12)}$*:*

1. *Let* $q_{\mathsf{se}}^{cs\text{-}pair}$ *be the number of* $\mathsf{Send}_{\mathsf{cs\text{-}pair}}$ *sessions. Then the probability that two* uncorrupted $\mathsf{Send}_{\mathsf{cs\text{-}pair}}$ *sessions share the same* sid *and PWE satisfies*

$$\Pr[\mathsf{Coll}_{PWE}{}^{(12)}] \;\leq\; \frac{\frac{1}{2}q_{\mathsf{se}}^{cs\text{-}pair}}{2^{\mathrm{H}_\infty(\mathscr{D}_\mathcal{D})}}. \tag{22}$$

2. *Let* q_{se}^{S} *denote the number of* $\mathsf{Send}_{\mathcal{A}}$ *server sessions. Conditioned on* $\neg\mathsf{Coll}_{\mathsf{S}}{}^{(12)}$*, the probability that some* uncorrupted $\mathsf{Send}_{\mathcal{A}}$ *server session makes a correct* H_2 *query is bounded by*

$$\Pr[\mathsf{AskH_2bC\text{-}Send}_{\mathcal{A}}\mathsf{S}^{(12)}|\neg\mathsf{Coll}_{\mathsf{S}}{}^{(12)}] \;\leq\; \frac{q_{\mathsf{se}}^{S}}{2^{\mathrm{H}_\infty(\mathscr{D}_\mathcal{D})}}. \tag{23}$$

3. *Let* q_{se}^{C} *denote the number of* $\mathsf{Send}_{\mathcal{A}}$ *client sessions. Conditioned on* $\neg\mathsf{Coll}_{\mathsf{C}}{}^{(12)}$*, the probability that some* uncorrupted $\mathsf{Send}_{\mathcal{A}}$ *client session makes a correct* H_2 *query is bounded by*

$$\Pr[\mathsf{AskH_2bC\text{-}Send}_{\mathcal{A}}\mathsf{C}^{(12)}|\neg\mathsf{Coll}_{\mathsf{C}}{}^{(12)}] \;\leq\; \frac{q_{\mathsf{se}}^{C}}{2^{\mathrm{H}_\infty(\mathscr{D}_\mathcal{D})}}. \tag{24}$$

Proof. We consider the three bounds separately.

1. **Bounding** $\Pr[\mathsf{Coll}_{\boldsymbol{PWE}}{}^{(12)}]$**.**
   This is the probability that two uncorrupted $\mathsf{Send}_{\mathsf{cs\text{-}pair}}$ sessions share the same sid and PWE. Clearly, this probability is upper bounded by $\Pr[PWE_1 = PWE_2]$ for two sessions holding the same sid.
   Recall that

$$PWE_1 = val_1 \cdot \mathsf{H}_1(\mathrm{SSID}_1|\mathsf{pw}_1), \qquad PWE_2 = val_2 \cdot \mathsf{H}_1(\mathrm{SSID}_2|\mathsf{pw}_2),$$

   so equality implies

$$\mathsf{H}_1(\mathrm{SSID}_1|\mathsf{pw}_1) = \frac{val_2}{val_1} \cdot \mathsf{H}_1(\mathrm{SSID}_2|\mathsf{pw}_2).$$

   Since neither password is corrupted when val_1 and val_2 are fixed, we may assume without loss of generality that pw_1 is chosen last. By the independence of pw_1 from all other values, and since H_1 is modeled as a random oracle, we obtain that $\Pr[PWE_1 = PWE_2]$ is bounded by the maximum point probability of the password distribution, i.e., $1/2^{\mathrm{H}_\infty(\mathscr{D}_\mathcal{D})}$. Since there are at most $q_{\mathsf{se}}^{cs\text{-}pair}/2$ pairs of sessions sharing the same sid, the overall collision probability follows.
2. **Bounding** $\Pr[\mathsf{AskH_2bC\text{-}Send}_{\mathcal{A}}\mathsf{S}^{(12)}|\neg\mathsf{Coll}_{\mathsf{S}}{}^{(12)}]$**.**
   Since passwords are only fixed at the corruption query or at the end of the game, before corruption, the probability that the correct password results in a $[K]_x$ appearing in $\mathcal{L}_{\mathsf{H}_2}$ is bounded by the maximum point probability of the dictionary distribution, i.e., $1/2^{\mathrm{H}_\infty(\mathscr{D}_\mathcal{D})}$. Conditioned on $\neg\mathsf{Coll}_{\mathsf{S}}{}^{(12)}$, for each uncorrupted $\mathsf{Send}_{\mathcal{A}}$ server session, at most one password maps to a $[K]_x$ value contained in $\mathcal{L}_{\mathsf{H}_2}$. Hence, the total number of such $[K]_x$ values is at most q_{se}^{S}, implying the final bound $q_{\mathsf{se}}^{S}/2^{\mathrm{H}_\infty(\mathscr{D}_\mathcal{D})}$.
3. **Bounding** $\Pr[\mathsf{AskH_2bC\text{-}Send}_{\mathcal{A}}\mathsf{C}^{(12)}|\neg\mathsf{Coll}_{\mathsf{C}}{}^{(12)}]$**.**
   The analysis for client sessions is entirely analogous.

□

Finally, we consider the two events wS_{12} and wAuth_{12} to complete the proof.

Lemma 8. *The probabilities of the events* wS_{12} *and* wAuth_{12} *are bounded as follows:*

$$\Pr[\mathsf{wS}_{12}] = \frac{1}{2}, \tag{25}$$

$$\Pr[\mathsf{wAuth}_{12}] \leq \frac{q_{\mathsf{se}}}{2^L}. \tag{26}$$

Proof. The event wS_{12} corresponds to the adversary correctly guessing the hidden bit used in the Test query. Since the derived uncorrupted session key pmk is uniformly random and independent of the adversary's view, the best possible strategy for the adversary is to guess the bit at random. Hence, $\Pr[\mathsf{wS}_{12}] = 1/2$.

For the event wAuth_{12}, recall that, except for the dropped cases, any uncorrupted authenticator is derived from the secret random oracle H'_4, with a unique input for each session. Thus, the adversary can only guess its value. With at most q_{se} relevant Send queries, we obtain $\Pr[\mathsf{wAuth}_{12}] \leq q_{\mathsf{se}}/2^L$. □

From equations (4) to (26), the Theorem 6 follows. This completes the proof.

C.3 Proof of Theorem 7 (Security of SAEv2)

Note that this fix works provided that each client session rejects any peer whose MAC address equals its own.

The protocol SAEv2 is a slight modification of SAE, where the value val (see lines 7 and 14 of Fig. 2) is computed as

$$val = \mathsf{HMAC}(0^L, \mathrm{MAC}_C|\mathrm{MAC}_S).$$

The proof structure follows that of SAE. In particular, it suffices to show that no *fresh* session can accept without a partner. To this end, we introduce the following events, which serve as the counterparts of wS and wAuth for capturing AKE and EA security:

S_m**:** The adversary $\mathcal{A}$ correctly guesses the hidden bit b under AKE security;
Auth_m**:** Some *fresh* session accepts without a partner.

Observe that the only relevant situations are the *Paired Cases*, since all other cases do not involve weak partners and therefore remain unchanged from the proof of SAE. In particular, for the *Paired Case*:

– When answering random-oracle queries, i.e., H'_3 and H'_4, we replace the input $(\mathrm{MAC}_1, \mathrm{MAC}_2, \mathsf{sid})$ with $(\mathrm{MAC}_C, \mathrm{MAC}_S, \mathsf{sid})$, where $(\mathrm{MAC}_C, \mathrm{MAC}_S)$ is arranged in a fixed order.

Under the assumption that each client session rejects any peer whose MAC address equals its own, two client sessions can never authenticate with each other, which follows directly from the confirmation values. Furthermore, two server sessions can never share the same sid, and hence can never authenticate with each other. As a result, no session can accept without a weak partner in SAEv2.

All game-hop bounds remain unchanged, so the final distinguishing advantage is identical to that of SAE.

Remark 1. We observe that it suffices to fix a consistent ordering of MAC addresses; there is no need to enforce that MAC_C or MAC_S comes first. Both orderings satisfy the same security bound.

C.4 Proof of Theorem 8 (Security of SAEv3)

Note that in the SAEv2 case, security relies on each client session rejecting any peer holding its own MAC address. For SAEv3, however, such rejection is unnecessary.

The protocol SAEv3 is a minor modification of SAE, where the key seed $kck|pmk$ (see line 24 of Fig. 2) is redefined as

$$kck|pmk \leftarrow \mathsf{KDF}(seed, \mathtt{lbl}|s_C, L + 256).$$

With S_m and Auth_m defined exactly as in SAEv2, the overall proof strategy is identical to that of SAE. The only modification is that ctx is replaced by s_C throughout the proof.

More precisely, ctx appears only in Game $\mathbf{G}_9$, Case 2, where the sids are distinct, when analyzing collisions in $([K]_x, ctx)$. In the SAEv3 case, this analysis instead considers collisions in $([K]_x, s_C)$. The modified proof is as follows.

By the randomness and independence of CE_4 from all other values, the probability that $[K_1]_x = [K_2]_x$ is at most $2/(q-1)$. For two client sessions, the probability that they share the same s_C is $1/q$. Hence, the probability of a collision in $([K]_x, s_C)$ is at most $2/((q-1)q)$. For two server sessions, or for one client session and one server session, the probability that they share the same s_C is at most 1, and therefore the corresponding collision probability is bounded by $2/(q-1)$.

Assume that there are a Send queries to client sessions, which generate at most a^2 pairs $([K]_x, s_C)$, and at most $q_{\mathsf{se}} - a$ Send queries to server sessions, which generate at most $q_{\mathsf{se}} - a$ such pairs. By the birthday bound, the probability of a collision in $([K]_x, s_C)$ among client sessions is bounded by $a^4/((q-1)q) \leq a^2/(q-1)$, where the inequality follows from $a^2/(q-1) \leq 1$. Similarly, the collision probability among server sessions is bounded by $(q_{\mathsf{se}} - a)^2/(q-1)$. Finally, by a union bound, the collision probability between a client session and a server session is bounded by $2a(q_{\mathsf{se}} - a)/(q-1)$. Overall, the probability of a collision in $([K]_x, s_C)$ is at most $a^2/(q-1) + (q_{\mathsf{se}} - a)^2/(q-1) + 2a(q_{\mathsf{se}} - a)/(q-1) \leq q_{\mathsf{se}}^2/(q-1)$.

Remark 2. We note that the alternative fix

$$kck|pmk \leftarrow \mathsf{KDF}(seed, \mathtt{lbl}|s_S, L + 256),$$

i.e., replacing the context ctx with s_S instead of s_C, does not achieve the same bound under our proof strategy. In this case, the probability of a collision in $([K]_x, s_S)$ among client sessions is bounded by $2/(q-1)$ rather than $2/(q-1)q$. Consequently, the overall bound weakened to at most $q_{\mathsf{se}}^4/(q-1)$.

C.5 Proof of Theorem 9 (Security of SAE-PK)

The security proof of SAE-PK is structurally very similar to that of SAE. Nevertheless, certain protocol specific subtleties require additional treatment, and we therefore present the proof for SAE-PK in full detail below.

We define the following events for each game $\mathbf{G}_m$:

iS_m**:** The adversary $\mathcal{A}$ correctly guesses the hidden bit b under insider AKE security;
iAuth_m**:** Some *fresh* session π_P^i accepts without a partner.

Game $\mathbf{G}_0$: This is the original security experiment in the random oracle and ideal cipher model. Specifically, the functions used to derive PT, $seed$, $kck|pmk$, the confirmation values con_C and con_S, as well as SHA-L, are modeled as independent random oracles, while $\mathsf{AES\text{-}SIV}$ is modeled as an ideal-cipher oracle. By definition,

$$\mathsf{Adv}^{\text{i-ake}}_{\mathsf{SAE\text{-}PK}}(\mathcal{A}) = 2\Pr[\mathsf{iS}_0] - 1 \quad \text{and} \quad \mathsf{Adv}^{\text{i-ea}}_{\mathsf{SAE\text{-}PK}}(\mathcal{A}) = \Pr[\mathsf{iAuth}_0]. \tag{27}$$

Game $\mathbf{G}_1$: In this game, compared with game $\mathbf{G}_1$ in the proof of SAE, we additionally model:

- SHA-L (see lines 26 and 33 in Fig. 2, lines 2 and 4 in Fig. 5 left, and line 2 in Fig. 5 right) as an independent random oracle H_5;
- $\mathsf{AES\text{-}SIV}$ (see lines 25 and 31 in Fig. 2) as an ideal cipher oracle.

H_5 and H'_5 oracles simulation:
For each query $\mathsf{H}_5(w)$ (resp., $\mathsf{H}'_5(w)$), if the same query was previously asked, the simulator retrieves the record (w, r) from the list $\mathcal{L}_{\mathsf{H}_5}$ (resp., $\mathcal{L}'_{\mathsf{H}_5}$) and responds with r. Otherwise, it generates the response r as follows:

- **Rule** $\mathsf{H}_5^{(1)}$– Choose $r \in \{0,1\}^L$.

Record the tuple (w, r) in the list $\mathcal{L}_{\mathsf{H}_5}$ (resp., $\mathcal{L}'_{\mathsf{H}_5}$).

Ideal Cipher Encryption-Query $\mathcal{E}$ oracle simulation:
For an encryption-query $\mathcal{E}(k, Z)$, such that a record $(k, Z, Z^\star)$ appears in $\mathcal{L}_{\mathsf{ic}}$, the answer is $Z^\star$. Otherwise the answer $Z^\star$ is defined according to the following rule:

- **Rule** $\mathcal{E}^{(1)}$– Choose a random element $Z^\star \in \{0,1\}^{128}$.

Record the tuple (k, Z, Z^*) in $\mathcal{L}_{\mathsf{ic}}$.

Ideal Cipher Decryption-Query $\mathcal{D}$ oracle simulation:
For a decryption-query $\mathcal{D}(k, Z^\star)$, such that a record $(k, Z, Z^\star)$ appears in $\mathcal{L}_{\mathsf{ic}}$, the answer is Z. Otherwise, one applies the following rule to obtain the answer Z:

- **Rule** $\mathcal{D}^{(1)}$– Choose $r \in \{0,1\}^{128}$.

Record the tuple (k, Z, Z^*) in $\mathcal{L}_{\mathsf{ic}}$.

Fig. 13. Simulation of random oracles $\mathsf{H}_5, \mathsf{H}'_5$ and ideal ciphers in the proof of Theorem 9 (Security of SAE-PK).

A $\mathsf{Send}(S, j, \mathrm{MAC}_C)$-query, $\mathsf{Send}(S, j, (\mathrm{MAC}_C, \mathrm{SSID}))$-query or $\mathsf{Send}(S, j, \top)$-query returns $(\mathrm{MAC}_S, \mathrm{SSID})$.

A $\mathsf{Send}(S, j, (\mathrm{MAC}_C, s_C, CE_C))$-query is processed as follows:

- **Rule S1**$^{(1)}$ Abort if $s_C \notin \mathbb{Z}_q \setminus \{0,1\} \vee CE_C \notin \mathbb{Z}_q \setminus \{0\}$.
- **Rule S2**$^{(1)}$ $PT = \mathsf{H}_1(\mathrm{SSID} \mid \mathsf{pw})$ and $val = \mathsf{HMAC}(0^L, \max(\mathrm{MAC}_C, \mathrm{MAC}_S) \mid \min(\mathrm{MAC}_C, \mathrm{MAC}_S)) \bmod (q-1) + 1$. Set $PWE = val \cdot PT$.
- **Rule S3**$^{(1)}$ Uniformly random select $r_S, m_S \in \mathbb{Z}_q \setminus \{0,1\}$ such that $(r_S + m_S) \bmod q \in \mathbb{Z}_q \setminus \{0,1\}$. Set $s_S = (r_S + m_S) \bmod q$ and $CE_S = -m_S \cdot PWE$.
- **Rule S4**$^{(1)}$ Compute $K = r_S \cdot (s_C \cdot PWE + CE_C)$ and abort if $K = 0$.
- **Rule S5**$^{(1)}$ Compute $seed = \mathsf{H}_2([K]_x)$.
- **Rule S6**$^{(1)}$ Let $ctx = (s_S + s_C) \bmod q$.
- **Rule S7**$^{(1)}$ Compute $kck \mid pmk \mid kek = \mathsf{H}_3(seed, \mathtt{lbl} \mid ctx, L + 256)$.
- **Rule S8**$^{(1)}$ Compute $c = \mathcal{E}(kek, M)$.
- **Rule S9**$^{(1)}$ Compute $h = \mathsf{H}_5(CE_S \mid CE_C \mid s_S \mid s_C \mid \mathsf{M} \mid pk \mid \mathrm{MAC}_S \mid \mathrm{MAC}_C)$.
- **Rule S10**$^{(1)}$ Compute $\sigma = \mathsf{ECDSAcore.Sign}(sk, h)$.
- **Rule S11**$^{(1)}$ Compute $con_S = \mathsf{H}_4(kck, s_S \mid CE_S \mid s_C \mid CE_C)$.

Return $((\mathrm{MAC}_S, s_S, CE_S), (con_S, pk, c, \sigma))$ and transition to the expected state.

A $\mathsf{Send}(S, j, con_C)$-query is processed as follows:

- **Rule S12**$^{(1)}$ Abort if $con_C \neq \mathsf{H}_4(kck, s_S \mid CE_S \mid s_C \mid CE_C)$.

The server instance stores pmk and terminates, unless the session is aborted.

Fig. 14. Simulation of Send queries to the server in the proof of Theorem 9 (Security of SAE-PK). Differences between SAE-PK and SAE are highlighted in blue.

The complete simulation procedures for the random oracles $\mathsf{H}_5, \mathsf{H}'_5$ and the ideal-cipher oracle are given in Fig. 13. For simplicity, we assume that both the input and the output of the ideal cipher are 128-bit strings. Furthermore, we simulate all protocol sessions exactly as real players would: the handling of Send-queries is shown in Figs. 14 and 15, whereas the simulation

A $\mathsf{Send}(C, i, \top)$-query is answered with $(\mathrm{MAC}_C, \mathrm{SSID})$.

A $\mathsf{Send}(C, i, (\mathrm{MAC}_S, \mathrm{SSID}))$-query is processed as:

- **Rule C1**$^{(1)}$ Abort if SSID does not match.
- **Rule C2**$^{(1)}$ $PT = \mathsf{H}_1(\mathrm{SSID} \mid \mathsf{pw})$ and $val = \mathsf{HMAC}(0^L, \max(\mathrm{MAC}_C, \mathrm{MAC}_S) \mid \min(\mathrm{MAC}_C, \mathrm{MAC}_S)) \bmod (q-1) + 1$. Set $PWE = val \cdot PT$.
- **Rule C3**$^{(1)}$ Choose $r_C, m_C \in \mathbb{Z}_q \setminus \{0, 1\}$ uniformly at random, ensuring $(r_C + m_C) \bmod q \in \mathbb{Z}_q \setminus \{0, 1\}$. Set $s_C = (r_C + m_C) \bmod q$ and $CE_C = -m_C \cdot PWE$.

Return $(\mathrm{MAC}_C, s_C, CE_C)$ and transition to the expected state.

A $\mathsf{Send}(C, i, (\mathrm{MAC}_S, s_S, CE_S))$-query is processed as:

- **Rule C4**$^{(1)}$ Drop and wait if: $s_C = s_S \vee CE_S = CE_C$.
- **Rule C5**$^{(1)}$ Abort if: $s_S \notin \mathbb{Z}_q \setminus \{0, 1\}$, $CE_S \notin \mathbb{Z}_q \setminus \{0\}$.
- **Rule C6**$^{(1)}$ Compute $K = r_C \cdot (s_S \cdot PWE + CE_S)$ and abort if $K = 0$.
- **Rule C7**$^{(1)}$ Compute $seed = \mathsf{H}_2([K]_x)$.
- **Rule C8**$^{(1)}$ Set $ctx = (s_C + s_S) \bmod q$.
- **Rule C9**$^{(1)}$ Compute $kck \mid pmk \mid kek = \mathsf{H}_3(seed, \mathtt{lbl} \mid ctx, L + 256)$.
- **Rule C10**$^{(1)}$ Compute $con_C = \mathsf{H}_4(kck, s_C \mid CE_C \mid s_S \mid CE_S)$.

Return con_C, and transition to the expected state.

A $\mathsf{Send}(C, i, (con_S, pk, c, \sigma))$-query is processed as:

- **Rule C11**$^{(1)}$ Drop and wait if: $con_S \neq \mathsf{H}_4(kck, s_S \mid CE_S \mid s_C \mid CE_C)$.
- **Rule C12**$^{(1)}$ Compute $M = \mathcal{D}(kek, c)$.
- **Rule C13**$^{(1)}$ Drop and wait if $(\overline{pk} \neq \perp \wedge \overline{pk} \neq pk) \vee \mathsf{pkVf}(\mathrm{SSID}, pk, \mathsf{pw}, \mathsf{M}) \neq 1$.
- **Rule C14**$^{(1)}$ Set $h = \mathsf{H}_5(CE_S \mid CE_C \mid s_S \mid s_C \mid M \mid pk \mid \mathrm{MAC}_S \mid \mathrm{MAC}_C)$.
- **Rule C15**$^{(1)}$ Drop and wait if: $\mathsf{ECDSAcore.Vf}(pk, h, \sigma) \neq 1$.

The client instance accepts, stores pmk and terminates.

A $\mathsf{Send}(C, i, (\mathrm{MAC}_S, s_S, CE_S, con_S, pk, c, \sigma))$-query is processed by executing Rule C4–Rule C10, returning con_C, and then performing Rule C11–Rule C15.

Fig. 15. Simulation of Send queries to the client in the proof of Theorem 9 (Security of SAE-PK). Differences between SAE-PK and SAE are highlighted in blue.

of Execute, Reveal, sCorrupt, iCorrupt, and Test-queries follows the same rules as in SAE (see Fig. 12). The components specific to SAE-PK, as opposed to SAE, are highlighted in blue. Clearly,

$$|\Pr[\mathbf{G}_1] - \Pr[\mathbf{G}_0]| = 0 \tag{28}$$

Game $\mathbf{G}_2$: This game is identical to $\mathbf{G}_2$ of SAE (see Appendix C.2, $\mathbf{G}_2$):

$$|\Pr[\mathbf{G}_2] - \Pr[\mathbf{G}_1]| \leq \frac{6q_{\mathsf{se}} + 12q_{\mathsf{exe}}}{q}. \tag{29}$$

Game $\mathbf{G}_3$: In this game, we add some collision-avoidance rules, some of which are the same as in SAE (see Appendix C.2, $\mathbf{G}_3$):

1. **Avoiding *val* collisions.** Identical to the analysis in SAE.
2. **Avoiding transcript collisions.** Identical to the analysis in SAE.
3. **Avoiding *pk* collisions.** Since there are at most $q_{\mathsf{se}} + q_{\mathsf{exe}}$ pk involved and pk is randomly chosen in group $\mathcal{G}$ with order q, this yields a birthday bound:

$$\frac{(q_{\mathsf{se}} + q_{\mathsf{exe}})^2}{2q}.$$

4. **Avoiding output collisions in random oracles.** As in the case of SAE, but with the additional possibility of kek colliding. This yields the bound:

$$\frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3})^2}{2^{L+1}}.$$

Combining all terms, we bound the total distinguishing advantage due to collision-avoidance changes by:

$$\begin{aligned} |\Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_2]| \leq \mathsf{Adv}^{\mathrm{coll}}_{\mathsf{HMAC}(0^L,\cdot)}(\mathcal{M}) + \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}})^2}{2q} + \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_1})^2}{2q} \\ + \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_2})^2}{2^{L+1}} + \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3})^2}{2^L} + \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}} + q_{\mathsf{H}_3})^2}{2^{257}} \\ + \frac{(q_{\mathsf{se}} + 2q_{\mathsf{exe}} + q_{\mathsf{H}_4})^2}{2^{L+1}} + \frac{2(q_{\mathsf{se}} + 2q_{\mathsf{exe}})(q_{\mathsf{se}} + q_{\mathsf{exe}})}{q^2}. \end{aligned} \tag{30}$$

Games $\mathbf{G}_4$–$\mathbf{G}_{10}$ are identical to those in SAE (see Appendix C.2, $\mathbf{G}_4$–$\mathbf{G}_{10}$), except for the additional computation of kek. Since this additional computation does not affect the game-hop bounds, the details of those games are omitted here.

Game $\mathbf{G}_{11}$: In this game, we choose c uniformly at random for the *Execute* and *Paired Cases*.

- Execute Case
  - **Rule $\mathbf{S8}^{(11)}$-Execute** Choose $c \in \{0,1\}^{128}$ uniformly at random.
  - **Rule $\mathbf{C12}^{(11)}$-Execute** Do nothing.
- Paired Case
  - **Rule $\mathbf{S8}^{(11)}$-$\mathsf{Send}_{\mathsf{pair}}$** Choose $c \in \{0,1\}^{128}$ uniformly at random.
  - **Rule $\mathbf{C12}^{(11)}$-$\mathsf{Send}_{\mathsf{pair}}$** If c is from partner, then let M be the one used. Otherwise, choose random $\mathsf{M} \in \{0,1\}^{128}$.

This game makes a distinction only if the adversary queries the ideal cipher on kek derived from an $\mathsf{Execute}$ or $\mathsf{Send}_{\mathsf{pair}}$ session. Since kek is derived by a secret random oracle H'_3 and hidden from the adversary's view, at most q_{ic} ideal cipher queries and at most $q_{\mathsf{se}} + q_{\mathsf{exe}}$ right kek imply

$$|\Pr[\mathbf{G}_{11}] - \Pr[\mathbf{G}_{10}]| \leq \frac{(q_{\mathsf{se}} + q_{\mathsf{exe}})q_{\mathsf{ic}}}{2^L}. \tag{31}$$

Game $\mathbf{G}_{12}$: This game is identical to $\mathbf{G}_{11}$ of SAE (see Appendix C.2), except that in the $\mathsf{uncorrupted}$ cases, that is, when neither $\mathsf{iCorrupt}$ nor $\mathsf{sCorrupt}$ occurs, the ciphertext $c = \mathcal{E}(kek, \mathsf{M})$ is sampled uniformly at random. The modification is as follows:

- **Rule $\mathbf{S8}^{(12)}$-$\mathsf{Send}_{\mathsf{cs\text{-}pair}}$,$\mathsf{Send}_{\mathcal{A}}$**
  - If $\mathsf{uncorrupted}$, choose $c \in \{0,1\}^{128}$ uniformly at random.
  - Otherwise, compute $c = \mathcal{E}(kek, \mathsf{M})$.
- **Rule $\mathbf{C12}^{(12)}$-$\mathsf{Send}_{\mathsf{cs\text{-}pair}}$,$\mathsf{Send}_{\mathcal{A}}$**
  - If $\mathsf{uncorrupted}$, drop and wait.
  - Otherwise, compute $\mathsf{M} = \mathcal{D}(kek, c)$.

Once corruption occurs, the correct ciphertext c is returned for cipher queries of a $\mathsf{Send}_{\mathcal{A}}$ session to ensure consistency before and after corruption.We further modify the game as follows:

- **Rule $\mathcal{E}^{(12)}$-$\mathsf{Send}_{\mathcal{A}}$** For a query $\mathcal{E}(kek, \mathsf{M})$, define the response r as follows:
  - If (kek, M) corresponds to a $\mathsf{corrupted}$ $\mathsf{Send}_{\mathcal{A}}$ session, set $r = c$ where c is the one used in this session.
  - Otherwise, choose $r \in \{0,1\}^{128}$ randomly.
- **Rule $\mathcal{D}^{(12)}$-$\mathsf{Send}_{\mathcal{A}}$** For a query $\mathcal{D}(kek, c)$, define the response r as follows:

- If (kek, c) corresponds to a corrupted $\mathsf{Send}_{\mathcal{A}}$ session, set $r = \mathsf{M}$.
- Otherwise, choose $r \in \{0,1\}^{128}$ randomly.

The distinction introduced by this modification is identical to that analyzed in $\mathbf{G}_{11}$ of SAE, except for the following two cases:

– When $\neg\mathsf{AskH}_3\text{-}\mathsf{Send}_{\mathsf{cs\text{-}pair}}$ and $\neg\mathsf{AskH}_3\mathsf{bC}\text{-}\mathsf{Send}_{\mathcal{A}}$, the adversary may query the correct kek to the ideal cipher.
Since kek is derived from the random oracle H_3 and does not appear elsewhere, and since there are at most q_{se} valid values of kek, this event occurs with probability at most $q_{\mathsf{se}} \cdot q_{\mathsf{ic}}/2^L$.
– Two uncorrupted server sessions with distinct sid derive the same kek.
By the randomness in independence the (s_S, CE_S), this probability is clearly bounded by $q_{\mathsf{se}}^2/2q$ (only consider ctx collision).

Thus, as the same discussion of $\mathbf{G}_{11}$ in SAE, we have

$$\begin{aligned}|\Pr[\mathbf{G}_{12}] - \Pr[\mathbf{G}_{11}]| \;\leq\; & \frac{q_{\mathsf{se}}^2}{q-1} + \frac{q_{\mathsf{se}}^2}{2q} + \frac{2q_{\mathsf{H}_4} + 2q_{\mathsf{se}}^2 + 2q_{\mathsf{se}} \cdot q_{\mathsf{exe}} + 2q_{\mathsf{exe}}^2 + q_{\mathsf{se}} \cdot q_{\mathsf{ic}}}{2^L} \\ & + q_{\mathsf{se}}^2 \cdot \mathsf{Adv}^{\mathrm{gdh}}_{\mathcal{G},G}(\mathcal{D}) + 4q_{\mathsf{se}} \cdot (q_{\mathsf{H}_1} + q_{\mathsf{se}} + q_{\mathsf{exe}})^2 \cdot \sqrt{\mathsf{Adv}^{\mathrm{wgsdh}}_{\mathcal{G},G}(\mathcal{E})} \\ & + 4q_{\mathsf{se}} \cdot (q_{\mathsf{H}_1} + q_{\mathsf{se}} + q_{\mathsf{exe}})^2 \cdot \mathsf{Adv}^{\mathrm{gsdh}}_{\mathcal{G},G}(\mathcal{F}) \\ & + \Pr[\mathsf{Coll}_{PWE}{}^{(12)}] + \Pr[\mathsf{AskH}_2\mathsf{bC}\text{-}\mathsf{Send}_{\mathcal{A}}\mathsf{S}^{(12)} | \neg\mathsf{Coll}_{\mathsf{S}}{}^{(12)}] \\ & + \Pr[\mathsf{AskH}_2\mathsf{bC}\text{-}\mathsf{Send}_{\mathcal{A}}\mathsf{C}^{(12)} | \neg\mathsf{Coll}_{\mathsf{C}}{}^{(12)}],\end{aligned} \tag{32}$$

where the remaining three probabilities are bounded in Game $\mathbf{G}_{15}$.

Game $\mathbf{G}_{13}$: In this game, the random oracle H_5 of uncorrupted sessions is replaced with an independently sampled secret random oracle H'_5. The modification is defined as follows:

▶ **Rule C14$^{(13)}$**
- If uncorrupted: $h = \mathsf{H}'_5(CE_S|CE_C|s_S|s_C|pk|\mathrm{MAC}_S|\mathrm{MAC}_C)$.
- Otherwise: $h = \mathsf{H}_5(CE_S|CE_C|s_S|s_C|\mathsf{M}|pk|\mathrm{MAC}_S|\mathrm{MAC}_C)$.

▶ **Rule S9$^{(13)}$**
- If uncorrupted: $h = \mathsf{H}'_5(CE_S|CE_C|s_S|s_C|pk|\mathrm{MAC}_S|\mathrm{MAC}_C)$.
- Otherwise: $h = \mathsf{H}_5(CE_S|CE_C|s_S|s_C|\mathsf{M}|pk|\mathrm{MAC}_S|\mathrm{MAC}_C)$.

Similarly to H_4, in order to maintain consistency, after corruption, we return the h value used before. We model the random oracle H_5 as follows:

▶ **Rule H_5** Upon a query $\mathsf{H}_5(w)$:
- If the input w corresponds to a corrupted $\mathsf{Send}_{\mathcal{A}}$ session, return $r = \mathsf{H}'_5(w')$, where w' denotes the input used prior to corruption.
- Otherwise, sample $r \in \{0,1\}^L$ uniformly.

The only way the adversary can distinguish this game from the previous one is by directly querying the input $CE_S|CE_C|s_S|s_C|\mathsf{M}|pk|\mathrm{MAC}_S|\mathrm{MAC}_C$ to H_5 for some uncorrupted session. Let this event be $\mathsf{AskH}_5^{(13)}$. We then obtain

$$|\Pr[\mathbf{G}_{13}] - \Pr[\mathbf{G}_{12}]| \leq \Pr[\mathsf{AskH}_5^{(13)}]. \tag{33}$$

The probability $\Pr[\mathsf{AskH}_5^{(13)}]$ will be deduced in the following game $\mathbf{G}_{14}$.

Game $\mathbf{G}_{14}$: In this game, we enforce that the challenger only accepts honest public keys. Concretely, we modify the challenger as follows:

▶ **Rule C13$^{(14)}$** Drop and wait if pk is not honest.

Since pk-collisions are ruled out in game $\mathbf{G}_3$, the above modification introduces a distinguishing event only if the event $\mathsf{badpk}^{(14)}$ occurs; namely, the adversary submits a public key pk such that $\mathsf{pkVf}(\mathrm{SSID}, pk, \mathsf{pw}, \mathsf{M}) = 1$, while pk is distinct from the honest public key. Thus,

$$|\Pr[\mathbf{G}_{14}] - \Pr[\mathbf{G}_{13}]| \leq \Pr[\mathsf{badpk}^{(14)}]. \tag{34}$$

Observe that the pair $(\mathsf{pw}, \mathsf{M})$ is no longer used for $\mathsf{uncorrupted}$ parties. Hence, we do not generate $(\mathsf{pw}, \mathsf{M})$ for these sessions in the simulation. This is analogous to $\mathbf{G}_{12}$ of SAE (see Appendix C.2).

Since no information about the $(\mathsf{pw}, \mathsf{M})$ values of $\mathsf{uncorrupted}$ sessions is leaked, the adversary can only guess them. Therefore, the probability that a correct M is queried to the random oracle H_5 is bounded by the maximum point probability of the modifier M distribution, as shown in the proof of Theorem 3, bounded by $1/2^{120-8\cdot\mathsf{sec}} + 1/2^{63+8\cdot\mathsf{sec}}$. Thus, at most q_{H_5} H_5 queries imply

$$\Pr[\mathsf{AskH}_5^{(14)}] \leq \frac{q_{\mathsf{H}_5}}{2^{120-8\cdot\mathsf{sec}}} + \frac{q_{\mathsf{H}_5}}{2^{63+8\cdot\mathsf{sec}}}.$$

Game $\mathbf{G}_{15}$: In this game, we sample M uniformly at random from $\{0,1\}^{128}$ instead of generating it via pwGen. When a $\mathsf{sCorrupt}$ query occurs, i.e., when M is required, we set

$$\mathsf{fp}' = \mathsf{LEFT}(\mathsf{SHA}\text{-}L(\mathrm{SSID}|\mathsf{M}|pk), 0,\ 8\cdot\mathsf{sec} + 19\cdot\mathsf{pwl}/4 - 5),$$

where $(\mathsf{fp}, \mathsf{sec}, \mathsf{pwl}) = \mathsf{Decode}(\mathsf{pw})$ and $\mathsf{fp}_{\mathsf{exp}} = 0^{8\cdot\mathsf{sec}}|\mathsf{fp}$. This is accessible since $\mathsf{SHA}\text{-}L$ is modeled as a random oracle H_5.

This modification introduces a distinguishing event only if the adversary queries the correct value $\mathrm{SSID}|\mathsf{M}|pk$ to H_5. Since pk-collisions are ruled out in game $\mathbf{G}_3$, the adversary can make at most q_{H_5} oracle queries, and hence by the randomness of M:

$$|\Pr[\mathbf{G}_{15}] - \Pr[\mathbf{G}_{14}]| \leq \frac{q_{\mathsf{H}_5}}{2^{128}}. \tag{35}$$

Besides, we bound $\Pr[\mathsf{badpk}^{(15)}]$ as follows. In this game. we no longer use pwGen to generate M. Instead, we generate pw via the oracle $\mathcal{O}_{pp}(id, m)$ as defined in Fig. 6. By Theorem 2, it holds that

$$\Pr[\mathsf{badpk}^{(15)}] \leq \mathsf{Adv}^{\text{2-pre}}_{\mathsf{PPG},pp}(\mathcal{A}). \tag{36}$$

Moreover, the password pw has min-entropy $\mathrm{H}_\infty(\mathscr{D}_{pp})$, so all remaining event probabilities are the same discussion as those in game $\mathbf{G}_{12}$ of SAE. Finally, we deal with the signature.

Game $\mathbf{G}_{16}$: In this game, the challenger rejects any signature that is sent to a $\mathsf{Send}_{\mathcal{A}}$ or $\mathsf{Send}_{\mathsf{cs\text{-}pair}}$ session whose party is not strongly corrupted.

- **Rule C15$^{(16)}$-$\mathsf{Send}_{\mathsf{cs\text{-}pair}}$,$\mathsf{Send}_{\mathcal{A}}$**
  - If not strongly corrupted, drop all non-honest signatures σ.
  - Otherwise, drop and wait if $\mathsf{ECDSAcore.Vf}(pk, h, \sigma) \neq 1$.

This makes a distinction unless $\mathsf{Sign}^{(16)}$ occurs, i.e., when the adversary outputs a valid signature σ to some $\mathsf{Send}_{\mathcal{A}}$ or $\mathsf{Send}_{\mathsf{cs\text{-}pair}}$ session before corrupting sk. Thus, by the following Lemma 9, we have

$$|\Pr[\mathbf{G}_{16}] - \Pr[\mathbf{G}_{15}]| \leq q_{\mathsf{se}}^2 \cdot \mathsf{Adv}^{\text{uf-cma}}_{\mathsf{ECDSA}}(\mathcal{J}). \tag{37}$$

Lemma 9. *Let $\mathcal{A}$ be an adversary that triggers the event $\mathsf{Sign}^{(16)}$. Then there exists a signature forger $\mathcal{J}$ that breaks the signature security of* ECDSA *such that*

$$\Pr[\mathsf{Sign}^{(16)}] \leq q_{\mathsf{se}}^2 \cdot \mathsf{Adv}^{\text{uf-cma}}_{\mathsf{ECDSA}}(\mathcal{J}).$$

Proof. We construct a forger $\mathcal{J}$ against ECDSA from the adversary $\mathcal{A}$. The forger $\mathcal{J}$ is given a public key pk by the ECDSA challenger, which it embeds as the public key of a uniformly random server session among the q_{se} possible server sessions. Whenever this server session requires a signature σ, $\mathcal{J}$ forwards the corresponding message $(CE_S|CE_C|s_S|s_C|pk|\mathrm{MAC}_S|\mathrm{MAC}_C)$, which is clearly unique since the tuple $(CE_S|CE_C|s_S|s_C)$ is unique, to its ECDSA signing oracle and obtains the signature directly, without explicitly recomputing the hash value h or invoking the ECDSAcore.Sign algorithm.

Observe that the combination of the hash step (i.e., Rule S9) and the core signing step (i.e., Rule S10) exactly implements the standard ECDSA.Sign algorithm. Therefore, the above procedure constitutes a perfect simulation of game $\mathbf{G}_{16}$. Conditioned on this, $\mathcal{J}$ outputs the forgery σ produced by $\mathcal{A}$. If the event $\mathsf{Sign}^{(16)}$ occurs, then with probability $1/q_{\mathsf{se}}$, the forgery is generated against the embedded server. Since there are at most q_{se} Send queries, the probability that the forged signature corresponds to the event $\mathsf{Sign}^{(16)}$ is at least $1/q_{\mathsf{se}}$.

Putting things together, we obtain that $\mathcal{J}$ succeeds in breaking the signature security of ECDSA with probability at least $1/q_{\mathsf{se}}^2 \cdot \Pr[\mathsf{Sign}^{(16)}]$. □

As in the discussion of Lemma 8, we obtain

$$\Pr[\mathsf{iS}_{16}] = \frac{1}{2} \quad \text{and} \quad \Pr[\mathsf{iAuth}_{16}] \leq \frac{q_{\mathsf{se}}}{2^L}. \tag{38}$$

This concludes the proof from equations (27) to (38).

C.6 Proof of Theorem 10 (Security of SAE-PK+)

To eliminate offline dictionary attacks, we define a variant of SAE-PK, denoted SAE-PK+, by redefining the hash value h (see lines 26 and 33 of Fig. 2) as

$$h = \mathsf{SHA}\text{-}L(CE_S|CE_C|s_S|s_C|pk|\mathrm{MAC}_S|\mathrm{MAC}_C),$$

where the modifier M is removed from the hash input.

The proof proceeds exactly as for SAE-PK (see Appendix C.5), with the sole exception that the event AskH_5 is eliminated. The resulting modifications are as follows:

$\mathbf{G}_1$: It is no longer necessary to model the above SHA-L as a random oracle.
$\mathbf{G}_{13}$: This game hop becomes unnecessary, and hence

$$\left|\Pr[\mathbf{G}_{13}] - \Pr[\mathbf{G}_{12}]\right| = 0.$$

Consequently, compared with SAE-PK, the security bound of SAE-PK+ no longer contains the offline dictionary attack term $4q_{\mathsf{H}_5}/2^{120-8\cdot\mathsf{sec}} + 4q_{\mathsf{H}_5}/2^{63+8\cdot\mathsf{sec}}$, while also reducing the coefficient of $\mathsf{Adv}^{\text{2-pre}}_{\mathsf{PPG},pp}(\mathcal{I})$ from 8 to 4 and the coefficient of $q_{\mathsf{H}_5}/2^{128}$ from 12 to 8, resulting in a tighter final bound.

D Other Security and Privacy Proofs

D.1 Proof of Theorem 2 (Second-Preimage Resistance of PPG)

Proof. To break the second-preimage resistance of PPG with public parameters pp, the adversary $\mathcal{A}$ needs to output (pk', M') such that $\mathcal{A}$ has made a $\mathsf{pw} \xleftarrow{\$} \mathcal{O}_{pp}(\mathrm{SSID}, pk)$ query and it holds that $pk \neq pk'$ and $\mathsf{pkVf}(\mathrm{SSID}, pk', \mathsf{M}', \mathsf{pw}) = 1$. By construction of pkVf (see Fig. 5), this implies that the left $8 \cdot \mathsf{sec} + 19 \cdot \mathsf{pwl}/4 - 5$ bits of $\mathsf{SHA}\text{-}L(\mathrm{SSID}|\mathsf{M}|pk)$ and $\mathsf{SHA}\text{-}L(\mathrm{SSID}|\mathsf{M}'|pk')$ are identical. Since SHA-L is modeled as a random oracle H, for any 5-tuple $(\mathrm{SSID}, pk, \mathsf{M}, pk', \mathsf{M}')$, the probability that the above equality holds is $2^{-(8\cdot\mathsf{sec}+19\cdot\mathsf{pwl}/4-5)}$.

For any SSID value id, the adversary can test at most $q_{id}(q_{\mathsf{H}id}+1)$ of the above 5-tuples to find one that passes the pkVf check, where the extra 1 in $q_{\mathsf{H}id}+1$ accounts for the output (pk', M') of the adversary that may not have been queried to the random oracle. Then, by applying a union bound on the queried SSID values, we have:

$$\mathsf{Adv}^{\text{2-pre}}_{\mathsf{PPG},pp}(\mathcal{A}) \leq \frac{\sum_{id} q_{id}(q_{\mathsf{H}id}+1)}{2^{8\cdot\mathsf{sec}+19\cdot\mathsf{pwl}/4-5}}.$$

□

D.2 Proof of Theorem 3 (Fingerprint Indistinguishability of PPG)

Proof. By construction of the password generation algorithm pwGen, the fingerprint space $\mathcal{T}_{pp}$ consists of all possible passwords derived by encoding the output of the truncated hash $\mathsf{LEFT}(\mathsf{SHA}\text{-}L(\cdot), 0, 8\cdot\mathsf{sec}+19\cdot\mathsf{pwl}/4-5)$. Recall that the first $8\cdot\mathsf{sec}$ bits are all 0s, so the size of $\mathcal{T}_{pp}$ is $2^{19\cdot\mathsf{pwl}/4-5}$. For a $\mathsf{pwGen}(pp,\cdot,\cdot)$ execution that outputs a password pw, as shown in Fig. 5, after a random 128-bit modifier M is sampled, it is incremented one by one until the first $8\cdot\mathsf{sec}$ bits of the hash output becomes all 0s; we call such a modifier *good* and denote it by $\overline{\mathsf{M}}_{\mathsf{pw}}$.

Since $\mathsf{SHA}\text{-}L$ is modeled as a random oracle H, the adversary $\mathcal{A}$'s views in the two worlds are the same unless it queries H on the good modifier $\overline{\mathsf{M}}_{\mathsf{pw}}$ used in some $\mathsf{pwGen}(pp,\cdot,\cdot)$ execution for generating a password pw in the real world, which we call the bad event. By the Difference Lemma of the game-based proof technique [Sho04], we have

$$\mathsf{Adv}^{\text{ind}}_{\mathsf{PPG},\mathcal{T}_{pp}}(\mathcal{A}) \leq \Pr[\mathsf{bad}]. \tag{39}$$

Among the q_{H} random oracle queries made by the adversary, let q denote the number of good modifiers. Since by definition all $\mathsf{pwGen}(pp,\cdot,\cdot)$ executions in the real world are on different inputs, each of the above q good modifiers can only be used to match at most one password generated from the same (SSID, pk) associated with the good modifier queried to the random oracle. Let F_i denote the event that the i-th query matches a password. Therefore, by applying a union bound, we have

$$\Pr[\mathsf{bad}] \leq q\cdot\max_{1\leq i\leq q}\{\Pr[F_i]\} \leq q_{\mathsf{H}}\cdot\max_{1\leq i\leq q}\{\Pr[F_i]\}. \tag{40}$$

Then, it suffices to compute $\max_{1\leq i\leq q}\{\Pr[F_i]\}$, which can be bounded by the probability of the most likely good modifier used in a pwGen execution, i.e., the one with the most non-good modifiers before it. Note that this probability is independent of the SSID, pk values, since H is modeled as a random oracle.

We split the modifier space $\{0,1\}^{128}$ into 2^{128-m} *consecutively disjoint* windows of size 2^m, with each window consisting of 2^m modifiers, where $m \geq 8\cdot\mathsf{sec} \geq 24$. Let E denote the event that there exists a window that has no good modifier for any given pwGen execution. If E does not occur (i.e., every window has a good identifier), then there are at most $2\cdot 2^m$ consecutive modifiers that are not good, so for any good modifier the probability of it being used in a pwGen execution is bounded by the size of two windows divided by the modifier space size, i.e., $2^{m+1}/2^{128}$. Therefore,

$$\max_{1\leq i\leq q}\{\Pr[F_i \mid \neg E]\} \leq \frac{1}{2^{127-m}}. \tag{41}$$

Now, we bound the probability that event E occurs, i.e., there exists a window without any good modifier. Recall that a modifier is good if its hash has $8\cdot\mathsf{sec}$ leading 0s, so any modifier is good with probability $2^{-8\cdot\mathsf{sec}}$ in the random oracle model. The probability that a window has no good modifier is

$$(1-\frac{1}{2^{8\cdot\mathsf{sec}}})^{2^m} = (1-\frac{1}{2^{8\cdot\mathsf{sec}}})^{2^{8\cdot\mathsf{sec}}\cdot 2^{m-8\cdot\mathsf{sec}}} \leq e^{-2^{m-8\cdot\mathsf{sec}}}.$$

Applying a union bound on the 2^{128-m} windows yields

$$\Pr[E] \leq \frac{2^{128-m}}{e^{2^{m-8\cdot\mathsf{sec}}}} = \frac{(2/e)^{128}\cdot 2^{-m}}{e^{2^{m-8\cdot\mathsf{sec}}-128}} \leq \frac{2^{-56-m}}{e^{2^{m-8\cdot\mathsf{sec}}-2^7}}. \tag{42}$$

Note that when $m = 8\cdot\mathsf{sec}+7$, the bound becomes $2^{-8\cdot\mathsf{sec}-63} \leq 2^{-87}$, which is sufficiently small for practical use; note that the bound decreases further as m increases.

Recall that for any events F_i and E, $\Pr[F_i] = \Pr[F_i \wedge \neg E] + \Pr[F_i \wedge E] \leq \Pr[F_i \mid \neg E] + \Pr[E]$. The proof is concluded by combining equations (39) $\sim$ (42) and choosing $m = 8\cdot\mathsf{sec}+7$. □

D.3 Proof of Theorem 5 (Privacy of SAE-Family Protocols)

Proof. The proof consists of two games. For simplicity, let $\Pr[\mathbf{G}_i]$ denote $\Pr[\mathcal{A}$ wins in $\mathbf{G}_i]$.

Game $\mathbf{G}_0$: This is the original privacy experiment. By definition, we have $\mathsf{Adv}^{\mathrm{priv}}_{\mathsf{sPAKE}}(\mathcal{A}) = 2\Pr[\mathbf{G}_0] - 1$.

Game $\mathbf{G}_1$: This game is the same as $\mathbf{G}_0$, except that, for all clients with $\mathsf{idtype} = \mathsf{pers}$ queried to $\mathsf{TestSend}$, each of their MAC address randomization functions $\mathsf{HMAC}(ik,\cdot)$ is replaced with an independent random function $f(\cdot)$.

To bound the gap between $\mathbf{G}_0$ and $\mathbf{G}_1$, we apply the hybrid argument (see [FM21] for a tutorial) to construct a series of hybrid games $\mathbf{H}_0, \mathbf{H}_1, \ldots, \mathbf{H}_{n_{\mathsf{pers}}}$ to replace the at most n_{pers} keyed HMAC functions used by the queried clients with random functions one by one. Each pair of consecutive hybrid games $\mathbf{H}_i$, $\mathbf{H}_{i+1}$ is bounded by the PRF advantage of some adversary $\mathcal{B}_i$, which uses its PRF oracle to simulate $\mathcal{A}$'s view in $\mathbf{H}_i$ in the real world and its view in $\mathbf{H}_{i+1}$ in the ideal world. Note that this simulation can be done perfectly because by definition of privacy, for the adversary $\mathcal{A}$ to win, no client with a persistent MAC address queried to $\mathsf{TestSend}$ was corrupted, then $\mathcal{B}_i$ can abort the simulation when such a corruption is detected. Then, by the Difference Lemma of the game-based proof technique [Sho04] and the hybrid argument, we can construct an efficient adversary $\mathcal{B}$ against the PRF security of HMAC, such that $\Pr[\mathbf{G}_0] - \Pr[\mathbf{G}_1] \leq n_{\mathsf{pers}} \cdot \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{HMAC}}(\mathcal{B})$.

Since $\mathsf{Adv}^{\mathrm{priv}}_{\mathsf{sPAKE}}(\mathcal{A}) = 2\Pr[\mathbf{G}_0] - 1 = 2(\Pr[\mathbf{G}_0] - \Pr[\mathbf{G}_1]) + 2(\Pr[\mathbf{G}_1] - 1/2)$ and $n_{\mathsf{pers}} \leq 2q_{\mathsf{ts}}$ holds (as each $\mathsf{TestSend}$ query can input at most 2 distinct clients), it is left to show that $\Pr[\mathbf{G}_1] = 1/2$.

In $\mathbf{G}_1$, all persistent MAC addresses of the clients queried to $\mathsf{TestSend}$ are generated from random functions and re-randomized per different SSID, and all random MAC addresses are re-randomized per session. By construction of the SAE-family protocols, MAC addresses are the only information in the cryptographic core (that we consider in this work) of these protocols that can link sessions from the same client. More precisely, the adversary $\mathcal{A}$ learns nothing about the identity of the queried client sessions in $\mathbf{G}_1$ if no client with persistent MAC addresses was queried to $\mathsf{TestSend}$ to complete two protocol runs under the same SSID, which is required for $\mathcal{A}$ to win the game. Also, for clients with random MAC addresses, $\mathcal{A}$ obviously learns nothing about their identity.

Therefore, to win in $\mathbf{G}_1$, the adversary $\mathcal{A}$ can only make a blind guess, and then we have $\Pr[\mathbf{G}_1] = 1/2$ and conclude the proof. □