Cryptanalysis and Enhancement of VTBSAE: Provably Secure Verifier-Based PAKE Protocols for the Industrial Internet of Things

Jiahui Gao, Zhiqiang Zhao, and Xuexian Hu

***Abstract*—Verifier-based password-authenticated key exchange (PAKE) enables a client and a server to establish high-entropy session keys from low-entropy passwords without requiring the server to store plaintext passwords. This asymmetric design reduces the risk that leakage of server-side credentials directly leads to client impersonation, making verifier-based PAKE well suited to authentication and session-key establishment for Industrial Internet of Things (IIoT) devices. In this paper, we analyze the recently proposed VTBSAE, a verifier-based PAKE protocol, and reveal that it fails to provide the claimed resistance against server-compromise client-impersonation attacks. To fix this vulnerability, we propose E-VTBSAEv1. While preserving the lightweight structure of VTBSAE, E-VTBSAEv1 binds the client-generated point E_1 to fresh server-side randomness. We also briefly discuss E-VTBSAEv2 as a challenge-based variant. We provide a formal security proof for E-VTBSAEv1 in the random-oracle model under the CDH assumption, covering AKE security, forward secrecy, and resistance to server-compromise client-impersonation attacks after server-side verifier-file leakage. We evaluate E-VTBSAEv1 under two practical deployment settings: a Raspberry Pi 4B server with an AutoDL client, and a Raspberry Pi 4B server with a Raspberry Pi 4B client. The experimental results show that E-VTBSAEv1 preserves the communication and storage overheads of VTBSAE and achieves runtime performance comparable to that of VTBSAE.**

***Index Terms*—Verifier-based password-authenticated key exchange, Industrial Internet of Things, protocol cryptanalysis, provable security, random-oracle model**

I. Introduction

THE Industrial Internet of Things (IIoT) has been widely deployed in industrial control [1], remote monitoring [2], and device configuration scenarios [3], where industrial devices communicate with management terminals, mobile clients, or human–machine interfaces over open networks. In such deployments, devices commonly act as servers, while administrators use client terminals to perform configuration, status query, and control operations. Therefore, mutual authentication and session-key establishment are essential before transmitting control commands or operational data. Although certificate-based authenticated key exchange protocols can provide these functions, they require certificate issuance, verification, renewal, and revocation, which introduce considerable management complexity and implementation overhead for large-scale, resource-constrained, and long-lived IIoT devices. These limitations motivate lightweight and easily deployable authentication and key-establishment mechanisms for IIoT environments [4].

Password-authenticated key exchange (PAKE) [5] enables two parties to establish a high-entropy session key [6] from a low-entropy password [7] over an insecure channel, thereby avoiding the certificate-management burden of traditional public-key infrastructures [8]. In client–server IIoT deployments, however, symmetric PAKE protocols, such as EKE [9], SPEKE [10], J-PAKE [11], and SAE [12], are less desirable because the server must store a plaintext password or password-equivalent secret, and server compromise may directly enable client impersonation. Verifier-based PAKE addresses this limitation by allowing the client to hold the password while the server stores only verifier information derived from it [13]. A secure verifier-based PAKE should ensure that leakage of the server-side verifier file does not allow an adversary to impersonate the client without recovering the password. Representative verifier-based protocols include SRP [14], SPAKE2+ [16], AuCPace [17], OPAQUE [18], KHAPE [19], and SAE+ [20].

In elliptic-curve settings, many PAKE protocols require a hash-to-curve (H2C) mapping, which is nontrivial to implement securely in constant time [21] and may introduce side-channel vulnerabilities [22]. To avoid the security pitfalls of H2C mappings, Pointcheval and Wang [23] introduced the two-basis construction, which generates password-derived group elements from two public base points and forms the basis of their VTBPEKE protocol.

Recently, Qi proposed VTBSAE, a verifier-based PAKE protocol for IIoT based on SAE and the two-basis technique [24]. VTBSAE uses two public base points to construct password-derived group elements and lets the server store only the client identity, salt, and verifier information. VTBSAE directly targets IIoT deployment and claims resistance to server-compromise client-impersonation attacks.

Our cryptanalysis, however, reveals a critical flaw in VTBSAE. We demonstrate that an adversary who obtains the server-side verifier file can successfully impersonate a legitimate client without recovering the plaintext password. By exploiting the linear structure of the server-side key computation, the adversary can craft a malicious elliptic-curve point. This crafted point causes the server to derive the same key material as in an honest execution and accept the corresponding authentication tag.

This work was supported in part by the National Natural Science Foundation of China under Grants 62672510 and 62472438. (Corresponding author: Xuexian Hu.)

Jiahui Gao, Zhiqiang Zhao, and Xuexian Hu are affiliated with the Information Engineering University, ZhengZhou 450001, China (e-mail: gaojiahui0509@163.com, lanren0909@gmail.com, xuexian_hu@hotmail.com).

In practical IIoT deployments, this flaw is particularly relevant because industrial devices usually have long deployment periods, large deployment scales, and persistent exposure to complex network environments. Once the device-side verifier database is leaked, an adversary capable of impersonating an administrator client may obtain unauthorized configuration capabilities and affect device operation or control-process security. This shows that the two-basis structure alone does not automatically guarantee resistance to client impersonation after server compromise.

To address this flaw while retaining the lightweight two-basis structure of VTBSAE, we propose E-VTBSAEv1 as the main repair. E-VTBSAEv1 binds the client-generated point to fresh server-side randomness, so canceling the verifier term V would require computing a CDH-hard value involving the server's ephemeral scalar. E-VTBSAEv1 preserves the main communication flow, storage structure, and lightweight features of VTBSAE.

The main contributions of this paper are summarized as follows.

1) **Cryptanalysis and enhancement of VTBSAE.** We identify a server-compromise client-impersonation attack against VTBSAE, demonstrate that this attack arises from the verifier-cancellation strategy enabled by the linear server-side key computation, and propose E-VTBSAEv1 to address this flaw while retaining the lightweight design of the original protocol.
2) **Formal security analysis.** We present a formal security proof for E-VTBSAEv1 in the random-oracle model under the CDH assumption, covering AKE security, forward secrecy, and resistance to server-compromise client-impersonation attacks after server-side verifier-file leakage. We further verify E-VTBSAEv1 using Scyther under server-side verifier-file leakage, with all 12 claims satisfied and no attacks found within the explored bounds.
3) **Performance evaluation.** We evaluate E-VTBSAEv1 against related protocols in terms of security properties, communication overhead, storage overhead, and computational overhead, and conduct implementation experiments under two practical deployment settings. The results indicate that E-VTBSAEv1 retains the communication and storage overheads of VTBSAE while achieving comparable runtime performance.

II. Preliminaries

This section introduces the notation and hardness assumption used in the protocol descriptions and security analysis.

Elliptic-curve additive group. Let $\mathbb{G}$ be an elliptic-curve additive group of prime order p, and let $\mathbb{Z}_p^*$ denote the set of nonzero elements modulo p. Elements of $\mathbb{G}$ are represented as elliptic-curve points, and the group operation is written additively. For a point $P \in \mathbb{G}$ and a scalar $a \in \mathbb{Z}_p^*$, aP denotes scalar multiplication. The notation $x \leftarrow_R S$ means that x is chosen uniformly at random from the set S, and a^{-1} denotes the multiplicative inverse of a modulo p. All random scalars used as inversion inputs in this paper are selected from $\mathbb{Z}_p^*$, so the corresponding inverses always exist. All elliptic-curve points used in the protocol are assumed to be canonically encoded and validated.

Computational Diffie–Hellman Assumption. Let G_0 be a generator of $\mathbb{G}$. Given (G_0, aG_0, bG_0), where $a, b \leftarrow_R \mathbb{Z}_p^*$, the Computational Diffie–Hellman (CDH) problem is to compute abG_0 without knowing a and b. For any probabilistic polynomial-time algorithm $\mathcal{B}$, its advantage in solving the CDH problem is defined as $\mathrm{Adv}_{\mathbb{G}}^{\mathrm{CDH}}(\mathcal{B}) = \Pr[\, abG_0 \leftarrow \mathcal{B}(G_0, aG_0, bG_0)\,]$. The CDH assumption states that this advantage is negligible in the security parameter.

III. Security Model

This section presents the security model used for the formal AKE proof of E-VTBSAEv1, following the BPR oracle framework introduced by Bellare, Pointcheval, and Rogaway and the key-exchange modeling methodology of Canetti and Krawczyk [25], while incorporating server-side verifier-file leakage [26].

Participants: The model uses C and S to denote a client and a server, respectively. The notations C^i and S^j denote different protocol instances of C and S. The client C holds a low-entropy password π_C, chosen from a password dictionary $\mathcal{PW}$; for a non-uniform dictionary, the usual replacement by the maximum password probability $P_{\max}$ applies. For each registered client C, the server S stores a server-side verifier file $vf_{C,S} = (C, s, \alpha, V)$, where $(\alpha, \beta) \leftarrow H_0(C, S, s, \pi_C)$, $D = P + \alpha Q$, and $V = \beta D$. Each protocol instance maintains a session identifier sid, a partner identifier pid, a session key sk, and an acceptance state acc.

Partnering: Two instances C^i and S^j are considered partnered if both are in the accepted state, share the same non-null session identifier sid, identify each other as the intended communication partner, and output the same session key. Except with negligible probability, no other distinct instance accepts with the same session identifier.

Adversary Capabilities: A probabilistic polynomial-time adversary $\mathcal{A}$ is assumed to control the communication network. In particular, $\mathcal{A}$ can eavesdrop on, block, replay, modify, and inject protocol messages. The adversary interacts with the challenger via the following oracle queries.

- $\mathsf{Execute}(C^i, S^j)$: The adversary obtains the public transcript of an honest protocol execution between the client instance C^i and the server instance S^j. This oracle models passive eavesdropping attacks.
- $\mathsf{Send}(U^i, m)$: The adversary sends an arbitrary message m to an instance U^i and receives the response generated by U^i according to the protocol specification. The adversary may invoke $\mathsf{Send}(U^i, \mathsf{Start})$ to activate a new protocol instance. This oracle models active attacks.
- $\mathsf{Reveal}(U^i)$: If the instance U^i has accepted and established a session key, the challenger returns sk_{U^i} to the adversary. This oracle models known session-key attacks.
- $\mathsf{Corrupt}(U)$: The adversary obtains the long-term secret information of participant U. If $U = C$, the challenger returns the client's password π_C. If $U = S$, the challenger returns the server-side verifier files stored by the server for registered clients.

- $\mathsf{Test}(U^i)$: The adversary may issue this query at most once, and only to a fresh instance. The challenger chooses a random bit $b \in \{0,1\}$. If $b = 1$, the challenger returns the real session key of U^i; if $b = 0$, it returns a random string of the same length.

Freshness: A test instance U^i is fresh if neither U^i nor its partnered instance has been the target of a Reveal query, the adversary has not obtained the client password π_C during the AKE experiment, and the corresponding server-side verifier file has not been exposed before U^i accepts. Exposure of the verifier file after acceptance does not violate freshness, provided that neither the test instance nor its partnered instance has been revealed. Pre-acceptance verifier-file exposure is excluded from AKE freshness and is analyzed separately through the server-compromise client-impersonation attacks considered in Theorem 1.

AKE Security: Let $b \in \{0,1\}$ denote the random challenge bit selected by the challenger for the unique Test query issued by the adversary to a fresh instance. After receiving the Test response and making any allowed subsequent oracle queries, the adversary outputs a guess bit b'. The advantage of $\mathcal{A}$ against protocol P is defined as $\mathrm{Adv}_P^{\mathrm{ake}}(\mathcal{A}) = \left|\Pr[b' = b] - \frac{1}{2}\right|$. A protocol P is AKE-secure in this model if the advantage of every probabilistic polynomial-time adversary $\mathcal{A}$ is bounded by the online password-guessing terms and a negligible function of the security parameter.

Forward-Secrecy Security: we consider a completed partnered session in which the ephemeral scalars of both partnered instances have been erased after acceptance. The adversary may subsequently obtain the client password, the corresponding server-side verifier file, or both, while neither the target session nor its partnered instance may be the target of a Reveal query. The adversary may then issue a Test query to the target session, which is answered as in the AKE experiment. Let b denote the challenge bit and b' the adversary's guess. The forward-secrecy advantage of $\mathcal{A}$ against protocol P is defined as $\mathrm{Adv}_P^{\mathrm{fs}}(\mathcal{A}) = \left|\Pr[b' = b] - \frac{1}{2}\right|$.

IV. Cryptanalysis of the VTBSAE Protocol

In this section, we first review the VTBSAE protocol [24] and then present server-compromise client-impersonation attacks against it. The symbols used in the protocol are summarized in Table I.

TABLE I
Symbols and Descriptions in the VTBSAE Protocol

Symbol	Description
C, S	Client and server
π	Client password
$\mathbb{G}$	Elliptic curve additive group
p	Prime order of $\mathbb{G}$
P, Q	Public generator points in $\mathbb{G}$
H_0, H_1, H_2	Cryptographic hash functions
s	Salt value
α, β	Password-related values derived by H_0
V	Server-side password verifier
r_1, m_1	Random values chosen by the client
r_2, m_2	Random values chosen by the server
s_1, s_2	Public scalar values in the protocol
E_1, E_2	Exchanged elliptic-curve points
K_A, K_B	Key materials computed by the client and server
σ, τ	Authentication tags
sk	Final session key

A. Overview of the VTBSAE Protocol

VTBSAE [24] is a verifier-based PAKE protocol designed for client–server IIoT environments. Let $\mathbb{G}$ be an elliptic-curve additive group of prime order p, and let $P, Q \in \mathbb{G}$ be two public generator points. The protocol uses three hash functions $H_0 : \{0,1\}^* \to \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, $H_1 : \{0,1\}^* \to \{0,1\}^{\ell_1}$, and $H_2 : \{0,1\}^* \to \{0,1\}^{\ell_2}$. VTBSAE consists of two phases: the registration phase and the PAKE session phase.

Registration phase: In this phase, the client generates a password verifier and registers it with the server over a secure channel.

The client C holds a password π and selects a fresh salt s. It computes $(\alpha, \beta) \leftarrow H_0(C, S, s, \pi)$. For convenience, we define $D = P + \alpha Q$ to simplify the subsequent description. The client then derives the password verifier $V = \beta D$ and sends the verifier file (C, s, α, V) to the server S over a secure channel. Upon receiving the registration request, the server checks that $\alpha \in \mathbb{Z}_p^*$, recomputes $D = P + \alpha Q$ and verifies that $D \neq \mathcal{O}$, and validates that V is a non-identity point in $\mathbb{G}$. If all checks pass, the server stores (C, s, α, V) in its database.

PAKE session phase: In this phase, the client and the server perform mutual authentication and establish a shared session key.

1) The client C samples two random scalars $r_1, m_1 \leftarrow_R \mathbb{Z}_p^*$ and computes $s_1 = r_1 + m_1 \bmod p$. The client then sends (C, s_1) to the server S.

2) Upon receiving (C, s_1), the server S uses C as the index to retrieve the corresponding verifier file (C, s, α, V). The server then samples two random scalars $r_2, m_2 \leftarrow_R \mathbb{Z}_p^*$ and computes $s_2 = r_2 + m_2 \bmod p$ and $E_2 = -m_2 D$. The server sends (S, s, E_2, s_2) to the client.

3) Upon receiving (S, s, E_2, s_2), the client recomputes $(\alpha, \beta) \leftarrow H_0(C, S, s, \pi)$, computes $E_1 = -m_1 D$ and $K_A = (r_1 + \beta)(E_2 + s_2 D)$, and derives $\sigma \parallel \hat{\tau} = H_1(C, S, E_1, s_1, E_2, s_2, s, \alpha, K_A)$. The client sends (E_1, σ) to the server.

4) Upon receiving (E_1, σ), the server computes $K_B = r_2(E_1 + s_1 D + V)$ and $\hat{\sigma} \parallel \tau = H_1(C, S, E_1, s_1, E_2, s_2, s, \alpha, K_B)$. If $\sigma \neq \hat{\sigma}$, the server aborts. Otherwise, it sends τ to the client and derives $sk = H_2(C, S, E_1, s_1, E_2, s_2, s, \alpha, K_B)$.

5) Upon receiving τ, the client checks whether $\tau = \hat{\tau}$ holds. If verification fails, the client aborts. Otherwise, the client derives $sk = H_2(C, S, E_1, s_1, E_2, s_2, s, \alpha, K_A)$.

VTBSAE claims to resist server-compromise client-impersonation attacks because the server only stores the verifier values α and V, from which the plaintext password π and the secret value β cannot be directly recovered.

B. Server-Compromise Client-Impersonation Attacks

This subsection describes in detail a server-compromise client-impersonation attack against VTBSAE. In this attack, an adversary needs only to obtain the server-side verifier file (C, s, α, V) to impersonate the client and interact with the

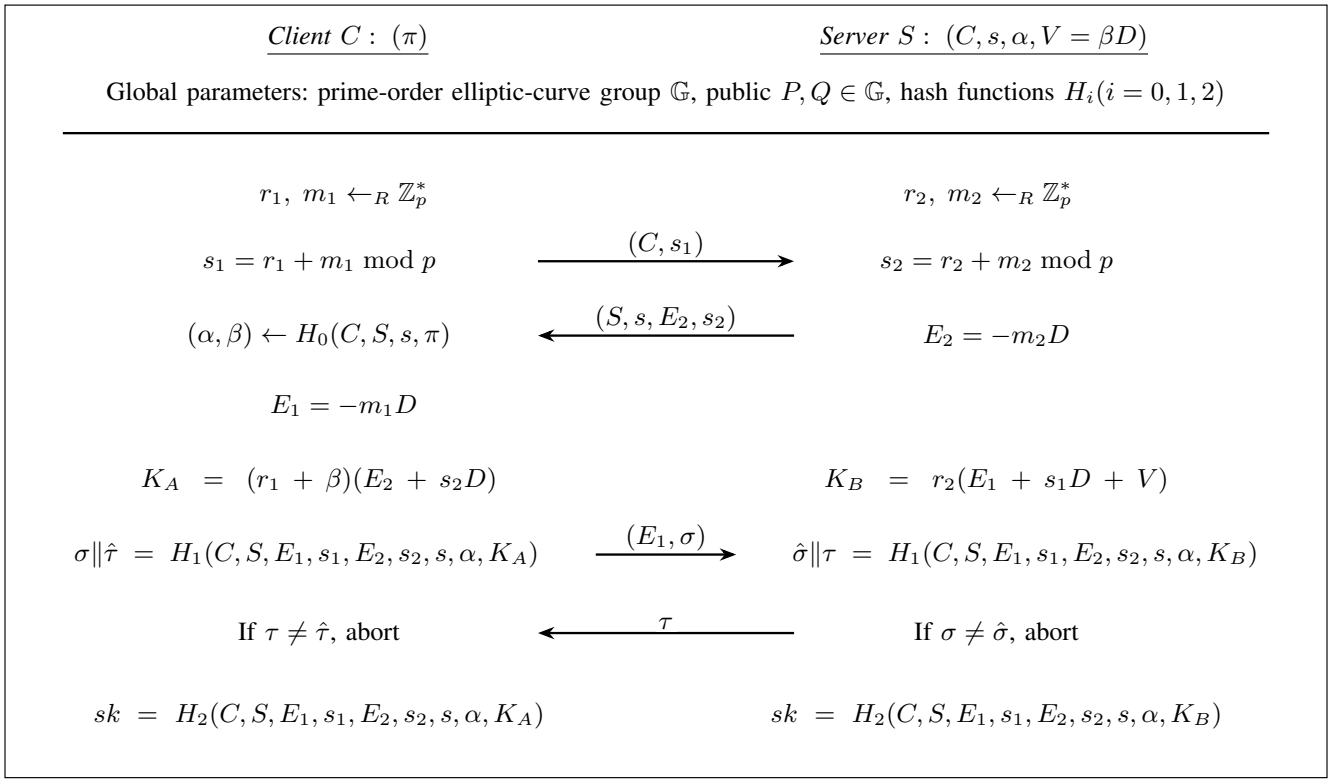


Fig. 1. The VTBSAE protocol.

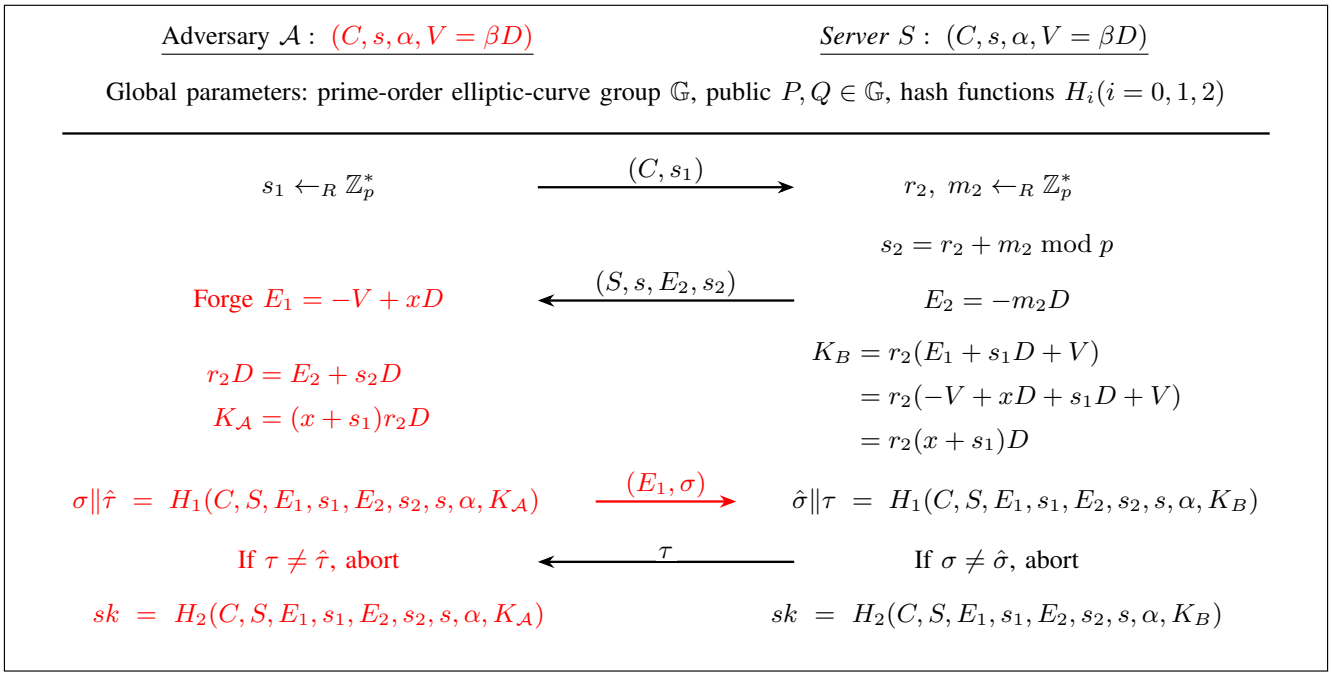


Fig. 2. Server-compromise client-impersonation attack against VTBSAE.

server by constructing $E_1 = -V$ or $E_1 = -V + xD$. The complete attack procedure is illustrated in Fig. 2, where the red elements indicate adversarial actions.

1) Assume that the adversary $\mathcal{A}$ compromises the server-side verifier database and obtains the verifier file (C, s, α, V). To impersonate the client C, $\mathcal{A}$ chooses $s_1 \leftarrow_R \mathbb{Z}_p^*$ and sends (C, s_1) to the server S. The server samples $r_2, m_2 \leftarrow_R \mathbb{Z}_p^*$, computes $s_2 = r_2 + m_2 \bmod p$ and $E_2 = -m_2 D$, and returns (S, s, E_2, s_2). Since $\mathcal{A}$ knows α, it computes $D = P + \alpha Q$ and derives $E_2 + s_2 D = -m_2 D + (r_2 + m_2) D = r_2 D$.

2) The adversary first forges $E_1 = -V$. The server then computes $K_B = r_2(E_1 + s_1 D + V) = r_2(-V + s_1 D + V) =$

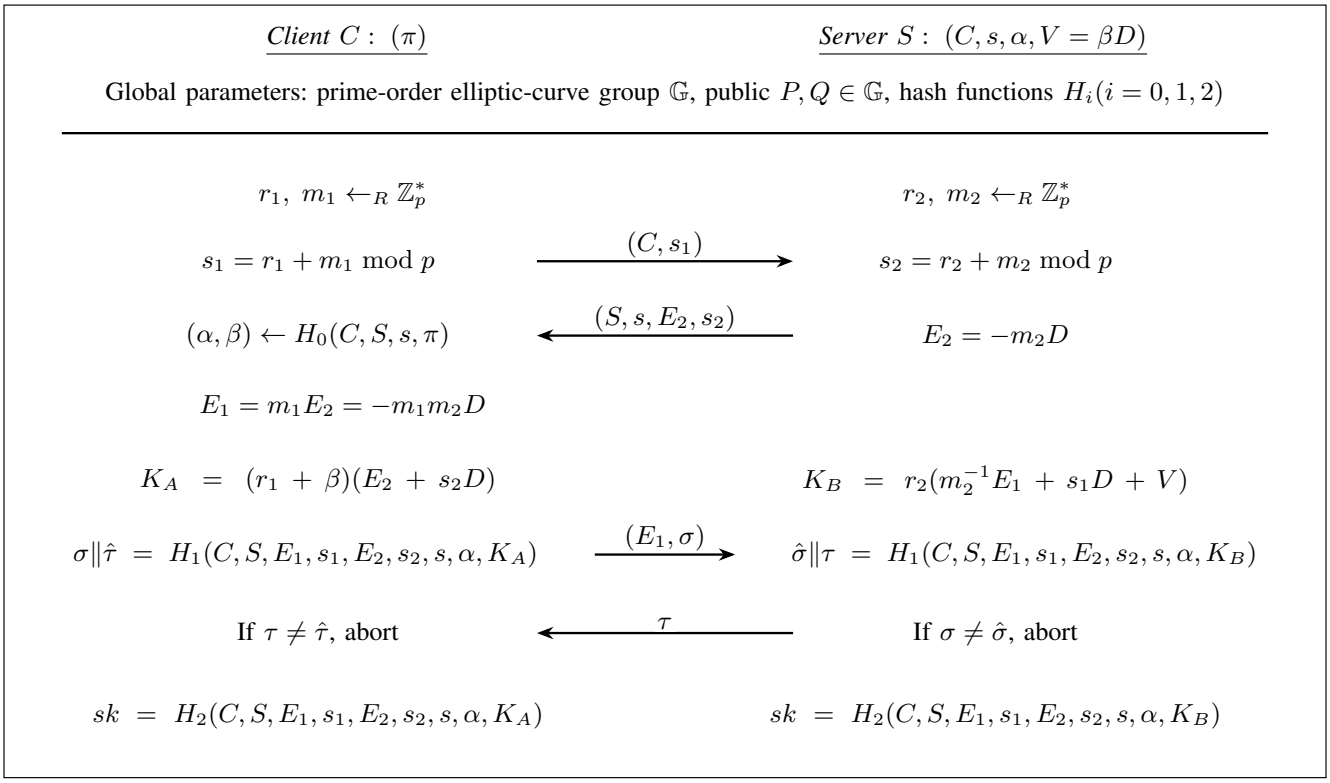


Fig. 3. The E-VTBSAEv1 protocol.

$r_2 s_1 D$. Since $\mathcal{A}$ knows s_1 and $r_2 D = E_2 + s_2 D$, it derives the same key material: $K_{\mathcal{A}} = s_1(E_2 + s_2 D) = s_1 r_2 D = K_B$.

Even if the server adds an extra check to reject the special value $E_1 = -V$, the attack still works. The adversary chooses an arbitrary known scalar $x \leftarrow_R \mathbb{Z}_p^*$ such that $x + s_1 \neq 0 \bmod p$ and forges $E_1 = -V + xD$. The server then computes $K_B = r_2(E_1 + s_1 D + V) = r_2(-V + xD + s_1 D + V) = r_2(x + s_1)D$. Since $\mathcal{A}$ has computed $r_2 D = E_2 + s_2 D$, it obtains $K_{\mathcal{A}} = (x + s_1)(E_2 + s_2 D) = (x + s_1) r_2 D = K_B$. As shown in Fig. 2, the verifier term V is algebraically canceled from the server-side key computation. We refer to this algebraic elimination of V as *verifier cancellation*.

3) After obtaining $K_{\mathcal{A}}$, the adversary computes $\sigma \parallel \hat{\tau} = H_1(C, S, E_1, s_1, E_2, s_2, s, \alpha, K_{\mathcal{A}})$ and sends (E_1, σ) to the server. Because $K_{\mathcal{A}} = K_B$, the server verifies σ and accepts $\mathcal{A}$ as client C. The adversary then derives $sk = H_2(C, S, E_1, s_1, E_2, s_2, s, \alpha, K_{\mathcal{A}})$, which is identical to the session key computed by the honest server.

Thus, for either $E_1 = -V$ or $E_1 = -V + xD$, the adversary derives the same key material as the server and can therefore generate valid inputs to H_1 and H_2. Consequently, VTBSAE does not provide the claimed resistance to server-compromise client-impersonation attacks.

V. Enhanced VTBSAE Protocol

A. Design Rationale of E-VTBSAEv1

The structural vulnerability of VTBSAE stems from the server-side key computation $K_B = r_2(E_1 + s_1 D + V)$, in which the long-term verifier V appears as a fixed linear term and the client-supplied point E_1 is not algebraically bound to the server's fresh ephemeral randomness. After server-side verifier-file leakage, these properties enable an adversary to cancel V, derive key material consistent with the server-side computation, and impersonate the client.

To fix this vulnerability, E-VTBSAEv1 modifies the server-side key-material computation by replacing the E_1 term in the original protocol with $m_2^{-1} E_1$. Under this modification, the server computes $K_B = r_2\big(m_2^{-1} E_1 + s_1 D + V\big)$. The client computes $E_1 = m_1 E_2$ from the server message $E_2 = -m_2 D$, replacing the original computation $E_1 = -m_1 D$. The modified computation preserves the consistency of the key material derived by the client and the server.

B. Specific Steps of the E-VTBSAEv1 Protocol

E-VTBSAEv1 follows the notation system and basic interaction flow of VTBSAE. The registration phase of E-VTBSAEv1 is identical to that of VTBSAE: the client computes $(\alpha, \beta) \leftarrow H_0(C, S, s, \pi)$ from the password π, salt s, and identity information, sets $D = P + \alpha Q$, generates the verifier $V = \beta D$, and submits the verifier file (C, s, α, V) to the server over a secure channel. The PAKE session phase of E-VTBSAEv1 is described as follows.

Throughout the protocol descriptions, every received group-element encoding is required to be canonical and to decode to a non-identity point in the correct prime-order subgroup. Any invalid encoding or invalid group element causes the protocol to abort. These validation checks are omitted from the individual steps for conciseness. In session phase, the client

and the server perform mutual authentication and establish a shared session key.

1) The client C samples two random scalars $r_1, m_1 \leftarrow_R \mathbb{Z}_p^*$ and computes $s_1 = r_1 + m_1 \bmod p$. The client then sends (C, s_1) to the server S.

2) Upon receiving (C, s_1), the server S uses the client identity C as the index to retrieve the corresponding verifier file (C, s, α, V). The server then computes $D = P + \alpha Q$, samples two random scalars $r_2, m_2 \leftarrow_R \mathbb{Z}_p^*$, computes $s_2 = r_2 + m_2 \bmod p$ and $E_2 = -m_2 D$, and sends (S, s, E_2, s_2) to the client.

3) Upon receiving (S, s, E_2, s_2), the client first verifies that the received server identity S is the intended peer identity for the current session. It then validates E_2. If any check fails, the client aborts. The client then recomputes $(\alpha, \beta) \leftarrow H_0(C, S, s, \pi)$ and sets $D = P + \alpha Q$. Next, the client computes $E_1 = m_1 E_2$ and $K_A = (r_1 + \beta)(E_2 + s_2 D)$. Using K_A, the client computes $\sigma \parallel \hat{\tau} = H_1(C, S, E_1, s_1, E_2, s_2, s, \alpha, K_A)$. The client then sends (E_1, σ) to the server.

4) Upon receiving (E_1, σ), the server validates E_1. If the validation fails, the server aborts. The server then computes the key material $K_B = r_2(m_2^{-1} E_1 + s_1 D + V)$. The inverse m_2^{-1} exists because $m_2 \in \mathbb{Z}_p^*$. Next, the server computes $\hat{\sigma} \parallel \tau = H_1(C, S, E_1, s_1, E_2, s_2, s, \alpha, K_B)$. If $\sigma \neq \hat{\sigma}$, the server aborts. Otherwise, the server sends τ to the client and derives the final session key $sk = H_2(C, S, E_1, s_1, E_2, s_2, s, \alpha, K_B)$.

5) Upon receiving τ, the client checks whether $\tau = \hat{\tau}$ holds. If verification fails, the client aborts. Otherwise, the client derives the final session key $sk = H_2(C, S, E_1, s_1, E_2, s_2, s, \alpha, K_A)$. The session identifier is set as $sid = (C, S, E_1, s_1, E_2, s_2, s, \alpha)$.

Correctness: $K_A = (r_1 + \beta)(E_2 + s_2 D) = (r_1 + \beta)\big(- m_2 D + (r_2 + m_2) D\big) = (r_1 + \beta) r_2 D = r_2\big(m_2^{-1} E_1 + s_1 D + V\big) = K_B$.

C. Discussion of E-VTBSAEv2

In addition to E-VTBSAEv1, we consider E-VTBSAEv2 as a challenge-based variant designed to avoid the same verifier cancellation. Its registration phase and basic message flow remain unchanged from VTBSAE, with modifications limited to the session-dependent challenge and key-material computation.

E-VTBSAEv2 introduces a session-dependent challenge $c = H_c(C, S, E_1, s_1, E_2, s_2, s, \alpha)$, where $H_c : \{0,1\}^* \to \mathbb{Z}_p$. The client retains $E_1 = -m_1 D$ and computes $K_A = (r_1 + c\beta)(E_2 + s_2 D)$, whereas the server computes $K_B = r_2(E_1 + s_1 D + cV)$. The correctness follows directly from $K_A = (r_1 + c\beta)(E_2 + s_2 D) = (r_1 + c\beta) r_2 D = r_2(E_1 + s_1 D + cV) = K_B$.

Under server-side verifier-file leakage, the adversary obtains the verifier information and can compute $D = P + \alpha Q$. To apply the same verifier cancellation used in the server-compromise client-impersonation attacks against VTBSAE, the adversary would need to make $E_1 + s_1 D + cV = xD$ for some known scalar x, which requires $E_1 = (x - s_1) D - H_c(C, S, E_1, s_1, E_2, s_2, s, \alpha) V$. Unlike in VTBSAE, the cancellation coefficient now depends on E_1 itself. Hence, the verifier cancellation used against VTBSAE no longer applies directly.

We provide an informal analysis of the remaining security properties of E-VTBSAEv2. A password guess from a public transcript does not reveal the session key material required to verify σ or τ, and thus does not provide a direct offline password-verification predicate. Replay attacks are mitigated because fresh session values, including E_2 and s_2, are incorporated into the challenge c and the authentication tags. Moreover, after session completion, subsequent compromise of the password or the server-side verifier file does not reveal the erased ephemeral scalars, and recovering the previous session key material from the recorded transcript still requires solving the corresponding CDH problem.

VI. Security Analysis

In this section, we first examine the main security properties of E-VTBSAEv1 through an informal analysis, followed by a formal security proof covering AKE security, resistance to server-compromise client-impersonation attacks after server-side verifier-file leakage, and forward secrecy, as well as symbolic verification using Scyther.

A. Informal Security Analysis

We analyze E-VTBSAEv1 with respect to the following security properties:

Resistance to server-compromise client-impersonation attacks: Under server-side verifier-file leakage, the adversary obtains (C, s, α, V) and can compute $D = P + \alpha Q$ and observe $E_2 = -m_2 D$ from the public transcript, but does not learn the ephemeral scalar m_2. Because the server scales the client-supplied point E_1 by m_2^{-1}, canceling V requires constructing E_1 such that $m_2^{-1} E_1$ contains the term $-V$, which requires the group element $m_2 V$. Deriving $m_2 V$ from D, $E_2 = -m_2 D$, and $V = \beta D$ corresponds to solving a CDH instance. Therefore, server-side verifier-file leakage alone does not enable the adversary to reproduce the server's session key material through the verifier cancellation used against VTBSAE.

Resistance to offline dictionary attacks based on public transcripts: An offline dictionary attack requires a password guess to yield candidate session key material that can be verified against the authentication tags σ or τ. In E-VTBSAEv1, a password guess allows the adversary to compute candidate values of α, β, and $D = P + \alpha Q$. The public transcript provides $E_2 = -m_2 D$, $s_2 = r_2 + m_2$, $s_1 = r_1 + m_1$, and $E_1 = m_1 E_2$. These values, however, do not determine the session key material used in the real protocol execution, since computing the corresponding Diffie–Hellman group element is infeasible under the CDH assumption. Consequently, the adversary cannot reconstruct $K_A = (r_1 + \beta)(E_2 + s_2 D)$ or $K_B = r_2(m_2^{-1} E_1 + s_1 D + V)$ for a password guess. The authentication tags therefore do not provide an offline verification predicate for password guesses, and E-VTBSAEv1 resists offline dictionary attacks based on public transcripts.

Forward secrecy: The freshness definition permits post-acceptance corruption of long-term credentials as long as

neither the tested instance nor its partner has been revealed. In an accepted E-VTBSAEv1 session, the session key is derived from $H_2(C, S, E_1, s_1, E_2, s_2, s, \alpha, K)$, where $K = r_2(r_1 + \beta)D$. Assuming that the ephemeral scalars r_1, m_1, r_2, m_2 are erased after session completion, subsequent compromise of the client password allows the adversary to recompute α, β, D, and V, whereas compromise of the server-side verifier file reveals α and V but not the scalar β. In either case, recovering the previous session key material K from the public transcript requires solving the corresponding CDH problem. Therefore, unless the adversary has already obtained the relevant key material through an allowed oracle query or solved CDH, the previously established session key remains indistinguishable from random under the assumptions of Theorem 1.

B. Formal Security Proof

We establish the security of E-VTBSAEv1 in the random-oracle model [27] under the CDH assumption. The proof covers AKE security, resistance to server-compromise client-impersonation attacks after server-side verifier-file leakage, and forward secrecy.

Theorem 1 (Security of E-VTBSAEv1): Let $\mathcal{A}$ be a probabilistic polynomial-time adversary against E-VTBSAEv1 in the random-oracle model. Let q_{H_0}, q_{H_1}, and q_{H_2} denote the numbers of random-oracle queries, let q_s and q_e denote the numbers of active and passive sessions, and let $|\mathcal{PW}|$ denote the size of the password dictionary. Assume that the CDH problem is hard in the prime-order group $\mathbb{G}$. Then the following properties hold.

(i) *AKE security.* For a fresh test session as defined in Section III,

$$\mathrm{Adv}_{v1}^{\mathrm{ake}}(\mathcal{A}) \le \frac{q_{H_0}}{|\mathcal{PW}|} + \frac{2q_s}{|\mathcal{PW}|} + \frac{2q_s}{2^{\ell_\sigma}} + \epsilon_{\mathrm{bad}} + (q_e + q_s)(q_{H_1} + q_{H_2})\mathrm{Adv}_{\mathbb{G}}^{\mathrm{CDH}}(\mathcal{B}_1),$$

where $\mathcal{B}_1$ is a polynomial-time CDH solver, ℓ_σ is the bit-length of each authentication tag, and ϵ_{bad} denotes the negligible probability of random-oracle collisions, direct guesses of unrevealed random-oracle outputs, session-identifier collisions, and degenerate group-element events. For a non-uniform password distribution, each factor $1/|\mathcal{PW}|$ is replaced by the maximum password probability $P_{\max}$.

(ii) *Resistance to server-compromise client-impersonation attacks.* After exposure of the server-side verifier file (C, s, α, V), let Imp denote the event that the adversary constructs E_1 such that $m_2^{-1}E_1 + s_1D + V = tD$ for a scalar t known to the adversary. Then

$$\Pr[\mathsf{Imp}] \le q_s\,\mathrm{Adv}_{\mathbb{G}}^{\mathrm{CDH}}(\mathcal{B}_2) + \mathrm{negl}(\lambda),$$

where $\mathcal{B}_2$ is a polynomial-time CDH solver.

(iii) *Forward secrecy.* For a completed partnered session in which the ephemeral scalars r_1, m_1, r_2, m_2 are erased after acceptance, subsequent compromise of the client password, the corresponding server-side verifier file, or both does not enable a polynomial-time adversary to distinguish the established session key from random except with probability

$$\mathrm{Adv}_{v1}^{\mathrm{fs}}(\mathcal{A}) \le (q_e + q_s)(q_{H_1} + q_{H_2}) \cdot \mathrm{Adv}_{\mathbb{G}}^{\mathrm{CDH}}(\mathcal{B}_{\mathrm{FS}}) + \mathrm{negl}(\lambda),$$

where $\mathcal{B}_{\mathrm{FS}}$ is a polynomial-time CDH solver.

Proof: We prove the three parts in turn.

Proof of part (i). **Game G_0.** This is the real AKE experiment for E-VTBSAEv1. The adversary issues the oracle queries allowed by the security model, subject to freshness. Let Succ_0 denote the event that $\mathcal{A}$ guesses the hidden Test bit correctly.

Game G_1. The challenger implements H_0, H_1, and H_2 by lazy sampling, returning an independent random value for each new query and the stored value for repeated queries. This is a perfect simulation of the random-oracle model; hence

$$\Pr[\mathsf{Succ}_1] = \Pr[\mathsf{Succ}_0].$$

Game G_2. The game aborts on a random-oracle or session-identifier collision, a direct guess of an unrevealed random-oracle output relevant to the target session, or a degenerate group element, including $D = \mathcal{O}$, $E_1 = \mathcal{O}$, $E_2 = \mathcal{O}$, $Y = E_2 + s_2D = \mathcal{O}$, or $X = m_2^{-1}E_1 + s_1D + V = \mathcal{O}$. A birthday/union bound over polynomially many sessions and oracle queries gives

$$|\Pr[\mathsf{Succ}_2] - \Pr[\mathsf{Succ}_1]| \le \epsilon_{\mathrm{bad}}.$$

Game G_3. The game aborts if the adversary queries H_0 on the correct password input (C, S, s, π) for the test client during the AKE experiment. Thus

$$|\Pr[\mathsf{Succ}_3] - \Pr[\mathsf{Succ}_2]| \le \frac{q_{H_0}}{|\mathcal{PW}|}.$$

For a non-uniform password distribution, the right-hand side is $q_{H_0}P_{\max}$.

Game G_4. The game aborts on a successful online password guess through an active Send query. Each active attempt tests at most one password in one authentication direction, and mutual authentication gives

$$|\Pr[\mathsf{Succ}_4] - \Pr[\mathsf{Succ}_3]| \le \frac{2q_s}{|\mathcal{PW}|}.$$

Game G_5. The game aborts if an instance accepts without an H_1 query on the corresponding authentication key material. Otherwise, acceptance requires guessing either σ or τ, each modeled as an independent ℓ_σ-bit random value; therefore

$$|\Pr[\mathsf{Succ}_5] - \Pr[\mathsf{Succ}_4]| \le \frac{2q_s}{2^{\ell_\sigma}}.$$

Game G_6. The game aborts if the adversary queries H_1 or H_2 on the real key material of the fresh test session. Conditioned on G_2–G_5, an accepting fresh target session has a matching honest partner: otherwise G_5 would require an exact H_1 query containing the correct password-derived value α, while freshness and G_2–G_4 exclude obtaining that value before acceptance.

For the partnered target session, let $Y = E_2 + s_2D = r_2D$, $X = m_2^{-1}E_1 + s_1D + V = (r_1 + \beta)D$, and $K = r_2X = (r_1+\beta)Y$. A solver $\mathcal{B}_1$ receives a CDH challenge $(D, A, B) =$

(D, aD, bD), treats D as the target-session generator, embeds $Y = A$ and $V = B$, samples s_2 subject to $s_2 D \neq A$, and sets $E_2 = A - s_2 D$. This implicitly gives $r_2 = a$ and $m_2 = s_2 - a \neq 0$, so the server message has the honest distribution. It then samples $m_1 \leftarrow_R \mathbb{Z}_p^*$ and s_1 subject to $s_1 \neq m_1$, sets $E_1 = m_1 E_2$, and thereby obtains $X = (s_1 - m_1)D + B$ and $K = (s_1 - m_1)A + abD$. The target transcript and random-oracle outputs are simulated by lazy sampling; post-acceptance verifier-file exposure can be answered with $V = B$. If the guessed H_1 or H_2 query contains the correct K^*, then $\mathcal{B}_1$ outputs $K^* - (s_1 - m_1)A = abD$. Hence

$$|\Pr[\mathsf{Succ}_6] - \Pr[\mathsf{Succ}_5]| \leq (q_e + q_s)(q_{H_1} + q_{H_2}) \mathrm{Adv}_{\mathbb{G}}^{\mathrm{CDH}}(\mathcal{B}_1).$$

Game G_7. After G_6, the adversary has not queried H_1 or H_2 on the real key material, so the test-session input to H_2 remains unqueried and its output is uniform and independent of the adversary's view. Therefore

$$\Pr[\mathsf{Succ}_7] = \frac{1}{2}.$$

Summing the differences between consecutive games proves part (i).

Proof of part (ii). Assume that the adversary has obtained (C, s, α, V) and attempts to mount server-compromise client-impersonation attacks using verifier cancellation. If it constructs E_1 such that $X = m_2^{-1} E_1 + s_1 D + V = tD$ for a known scalar t, then $m_2 V = m_2(t - s_1)D - E_1 = -(t - s_1)E_2 - E_1$. A solver $\mathcal{B}_2$ guesses one of the at most q_s active server sessions, receives $(D, A, B) = (D, m_2 D, \beta D)$, sets $E_2 = -A$ and $V = B$, and samples s_2 subject to $s_2 D \neq A$, which yields the honest server-response distribution. From any successful pair (E_1, t), it outputs $(t - s_1)A - E_1 = m_2 \beta D$, solving CDH. Therefore

$$\Pr[\mathsf{Imp}] \leq q_s \, \mathrm{Adv}_{\mathbb{G}}^{\mathrm{CDH}}(\mathcal{B}_2) + \mathrm{negl}(\lambda).$$

This proves part (ii).

Proof of part (iii). Consider a completed partnered session after erasure of r_1, m_1, r_2, m_2. After subsequent compromise of the client password, the corresponding server-side verifier file, or both, the previous key material still satisfies $Y = E_2 + s_2 D = r_2 D$ and $K = (r_1 + \beta)Y = (s_1 + \beta)Y - m_1 Y$. A solver $\mathcal{B}_{\mathrm{FS}}$ guesses the target session and relevant random-oracle query and uses a randomized CDH challenge $(G, A, B) = (G, aG, bG)$. It sets $E_2 = G$, $E_1 = A$, and $Y = B$, samples $s_2 \leftarrow_R \mathbb{Z}_p^*$, and, except for the negligible event $B = G$, defines $D = s_2^{-1}(B - G)$. Then $Y = E_2 + s_2 D$, and unique nonzero m_2, r_2 exist with $E_2 = -m_2 D$, $Y = r_2 D$, and $s_2 = r_2 + m_2$; conditioned on non-degeneracy, this has the honest server-transcript distribution. The reduction chooses the password, salt, α, and β, fixes the public bases consistently with $D = P + \alpha Q$, and samples s_1 subject to $s_1 G \neq A$. Thus $m_1 = a$, $r_1 = s_1 - a \neq 0$, $E_1 = m_1 E_2$, and $K = (s_1 + \beta)B - abG$. Consequently, a correct previous-session key-material query K^* yields $abG = (s_1 + \beta)B - K^*$. Hence

$$\mathrm{Adv}_{v1}^{\mathrm{fs}}(\mathcal{A}) \leq (q_e + q_s)(q_{H_1} + q_{H_2}) \cdot \mathrm{Adv}_{\mathbb{G}}^{\mathrm{CDH}}(\mathcal{B}_{\mathrm{FS}}) + \mathrm{negl}(\lambda).$$

This proves part (iii) and completes the proof of Theorem 1. ■

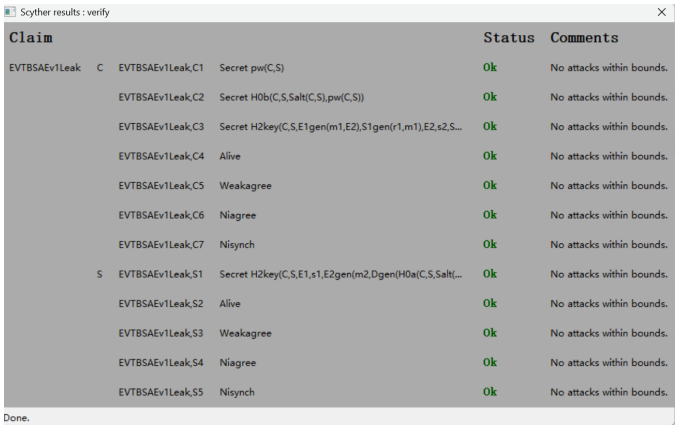


Fig. 4. Scyther verification results for E-VTBSAEv1.

C. Symbolic Verification with Scyther

We use Scyther v1.1.3 [28] to perform symbolic security verification for E-VTBSAEv1. Scyther is an automated protocol analysis tool based on the symbolic model, where protocol executions and security claims are described using the Security Protocol Description Language (SPDL). Scyther searches for potential attack traces under the Dolev–Yao adversary model [29], in which the adversary can fully control the public communication channel, including eavesdropping, replaying, modifying, and forging protocol messages. In our SPDL models, elliptic-curve operations and hash computations are abstracted as symbolic functions. To model server-side verifier-file leakage, the model explicitly gives the adversary the leaked server-side information $\{s, \alpha, D, V\}$, while keeping the client password $\mathrm{pw}(C, S)$ and the password-derived secret value β unknown.

We specify 12 security claims in the verification. In the client role C, we verify the secrecy of the client password, β, and the session key, together with `Alive`, `Weakagree`, `Niagree`, and `Nisynch`. In the server role S, we verify the secrecy of the session key and the same four authentication properties. Fig. 4 reports the Scyther verification results. Under this leakage model, Scyther reports `OK` for all specified secrecy and authentication claims, with no attacks found within the explored bounds.

VII. Protocol Comparison

In this section, we compare E-VTBSAEv1 and E-VTBSAEv2 with VTBSAE and related verifier-based PAKE protocols.

A. Functionality Comparison

Table II compares the considered verifier-based PAKE protocols in terms of their security properties and functional features. SRP-6a [15] avoids password-dependent hash-to-curve operations, but its standard construction is defined over a multiplicative group modulo a safe prime and does not directly

TABLE II
SECURITY AND FUNCTIONALITY COMPARISON OF VERIFIER-BASED PAKE PROTOCOLS

Property	Existing verifier-based PAKE baselines				Target	Enhanced Designs	
	SRP-6a	AuCPace	OPAQUE	SAE+	VTBSAE	E-VTBSAEv1	E-VTBSAEv2
Verifier-based / augmented PAKE	(✓)	(✓)	(✓)	(✓)	(✓)	(✓)	(✓)
Supports elliptic-curve instantiation	(×)	(✓)	(✓)	(✓)	(✓)	(✓)	(✓)
Avoids password-based H2C in online phase	(✓)	(×)	(×)	(×)	(✓)	(✓)	(✓)
Resists public-transcript offline dictionary attacks	(✓)	(✓)	(✓)	(✓)	(✓)	(✓)	Informal
Resists replay attacks	(✓)	(✓)	(✓)	(✓)	(✓)	(✓)	Informal
Resists server-compromise client-impersonation attacks	(✓)	(✓)	(✓)	(✓)	(×)	(✓)	Informal
Forward secrecy	(✓)	(✓)	(✓)	(✓)	Claimed	(✓)	Not separately proved

Note: "✓" and "×" indicate whether the property is supported. "Informal" indicates that the property is discussed only through informal analysis.

TABLE III
COMPUTATIONAL OVERHEAD COMPARISON OF RELATED PROTOCOLS

Protocol	Verifier Generation	Client	Server	Total AKE Computation
VTBPEKE	$1Exp + 1H$	$4Exp + 4H$	$4Exp + 3H$	$8Exp + 7H$
SPAKE2+	$1SM + 1H$	$5SM + 2PA + 6H$	$5SM + 2PA + 5H$	$10SM + 4PA + 11H$
AuCPace	$1SM + 1H$	$4SM + 7H$	$4SM + 5H$	$8SM + 12H$
OPAQUE	$3Exp + 1MTP + 1H$	$5Exp + 1MTP + 4H + 1I$	$4Exp + 7H$	$9Exp + 1MTP + 11H + 1I$
SAE+	$1SM + 1H$	$3SM + 1PA + 4H + 1I$	$3SM + 1PA + 3H + 1I$	$6SM + 2PA + 7H + 2I$
VTBSAE	$2SM + 1PA + 1H$	$4SM + 2PA + 3H$	$4SM + 3PA + 2H$	$8SM + 5PA + 5H$
E-VTBSAEv1	$2SM + 1PA + 1H$	$4SM + 2PA + 3H$	$5SM + 3PA + 2H + 1I$	$9SM + 5PA + 5H + 1I$
E-VTBSAEv2	$2SM + 1PA + 1H$	$4SM + 2PA + 4H + 1MM$	$5SM + 3PA + 3H$	$9SM + 5PA + 7H + 1MM$

Note: Exp: Exponentiation; SM: Scalar Multiplication; PA: Point Addition; H: Hash; I: Inversion; MM: Modular Multiplication; MTP: Map-To-Point.

support an elliptic-curve instantiation. In contrast, AuCPace, OPAQUE, and SAE+ support elliptic-curve implementations but involve password-dependent H2C operations in the online phase. VTBSAE combines elliptic-curve support with an online H2C-free structure; however, it remains vulnerable to the server-compromise client-impersonation attacks identified in this paper. E-VTBSAEv1 and E-VTBSAEv2 address this vulnerability while preserving both the verifier-based architecture and the online H2C-free elliptic-curve structure.

B. Communication and Storage Overhead

We compare the online AKE communication overhead and the server-side storage overhead. Let ℓ_{ID}, ℓ_s, $\ell_{\mathbb{Z}}$, $\ell_{\mathbb{G}}$, and ℓ_σ denote the bit-lengths of an identity, a salt, a scalar, an encoded elliptic-curve point, and an authentication tag, respectively. In the implementation setting used in this paper, we set $\ell_{\mathrm{ID}} = 160$, $\ell_s = 256$, $\ell_{\mathbb{Z}} = 256$, $\ell_{\mathbb{G}} = 264$, $\ell_\sigma = 128$. Here, $\ell_{\mathbb{G}} = 264$ follows the conventional SEC 1 compressed-point encoding over a 256-bit prime field [30].

VTBSAE, E-VTBSAEv1, and E-VTBSAEv2 share equal online AKE transcript, consisting of $C, s_1, S, s, E_2, s_2, E_1, \sigma$ and τ. Accordingly, their communication overhead is $2\ell_{\mathrm{ID}} + \ell_s + 2\ell_{\mathbb{Z}} + 2\ell_{\mathbb{G}} + 2\ell_\sigma = 1872$ bits. The server stores the verifier file (C, s, α, V), which results in a server-side storage overhead of $\ell_{\mathrm{ID}} + \ell_s + \ell_{\mathbb{Z}} + \ell_{\mathbb{G}} = 936$ bits.

In E-VTBSAEv1, the inverse m_2^{-1} is derived by the server from its ephemeral scalar m_2, whereas in E-VTBSAEv2, the challenge value c is derived from public session messages. Neither value is included in the protocol transcript or the stored verifier file. Hence, under the same implementation setting, E-VTBSAEv1 and E-VTBSAEv2 have the same communication and storage overheads as VTBSAE.

C. Efficiency Comparison

We evaluate the computational efficiency of E-VTBSAEv1 and E-VTBSAEv2 in terms of theoretical operation counts and practical runtime measurements.

Table III compares the computational overheads of the related protocols in terms of verifier generation, client-side key derivation, server-side key derivation, and the overall AKE stage. Relative to VTBSAE, E-VTBSAEv1 incurs one additional server-side modular inversion and one additional server-side elliptic-curve scalar multiplication for deriving $m_2^{-1}E_1$. E-VTBSAEv2 requires one challenge-hash evaluation at each party, one client-side modular multiplication for computing $c\beta$, and one additional server-side elliptic-curve scalar multiplication for computing cV. These additional operations are local and do not alter the online message flow.

To compare the practical runtime efficiency of VTBSAE, E-VTBSAEv1, and E-VTBSAEv2, we consider two client–server experimental settings: (i) a Raspberry Pi 4B server with an AutoDL client and (ii) a Raspberry Pi 4B server with a Raspberry Pi 4B client. The Raspberry Pi 4B is equipped with a Broadcom BCM2711 processor at 1.5 GHz, 8 GB RAM, and Ubuntu 20.04, whereas the AutoDL platform uses an Intel Xeon Platinum 8352V processor at 2.10 GHz, 32 vCPUs, 60 GB RAM, and Ubuntu 22.04. All protocols were implemented in Python. For secp256r1 and secp384r1, we used the ECC module of PyCryptodome, whereas sm2p256v1

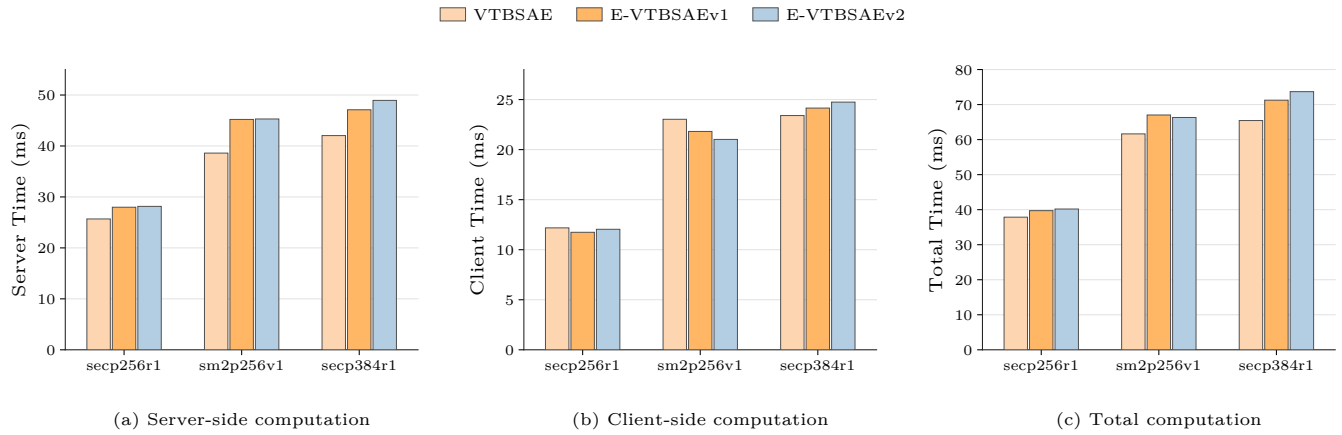


Fig. 5. Runtime of complete protocol execution with a Raspberry Pi 4B server and an AutoDL client.

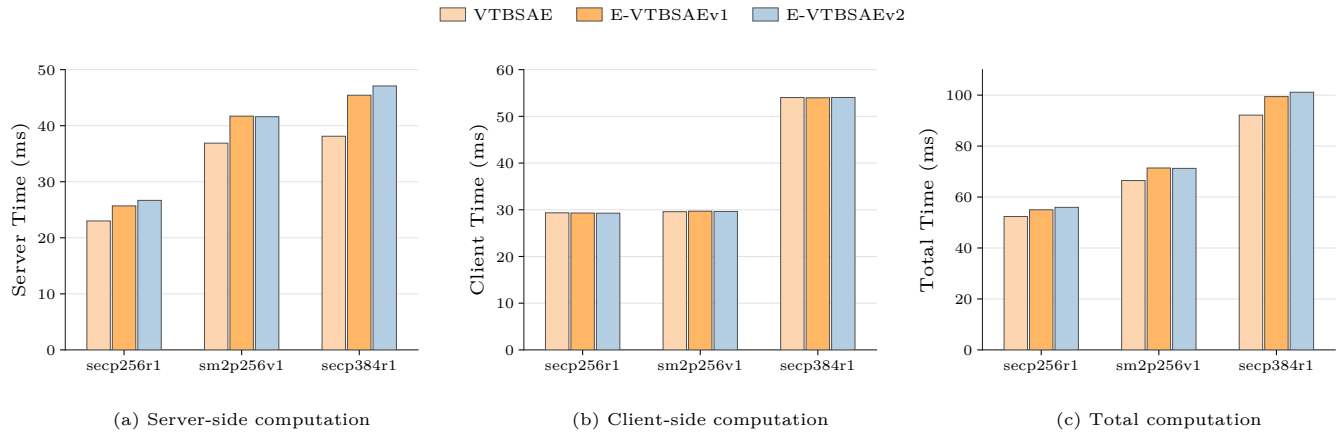


Fig. 6. Runtime of complete protocol execution with a Raspberry Pi 4B server and a Raspberry Pi 4B client.

was implemented using pure-Python short-Weierstrass elliptic-curve arithmetic. For each protocol and elliptic curve, we performed five experimental rounds, each consisting of 1000 complete protocol executions, and report the average runtime over the five rounds. These settings represent two representative IIoT deployment scenarios: heterogeneous communication between a resource-constrained industrial-device server and a more powerful management-terminal client, and lightweight deployment in which both endpoints are resource-constrained.

Figs. 5 and 6 compare the client-side runtime, server-side runtime, and total computation runtime of VTBSAE, E-VTBSAEv1, and E-VTBSAEv2 over the secp256r1, sm2p256v1, and secp384r1 elliptic curves under the two experimental settings. For the Raspberry Pi 4B server–AutoDL client configuration, E-VTBSAEv1 increases the total computation runtime by 4.91%–8.88% compared with VTBSAE, while E-VTBSAEv2 incurs an increase of 6.15%–12.60%. For the Raspberry Pi 4B server–Raspberry Pi 4B client configuration, the corresponding runtime increases are 5.05%–7.88% and 6.86%–9.75% for E-VTBSAEv1 and E-VTBSAEv2, respectively. These results indicate that E-VTBSAEv1 maintains runtime performance comparable to that of VTBSAE across the evaluated configurations.

VIII. Conclusion

We investigated the security of VTBSAE, a verifier-based password-authenticated key exchange protocol for Industrial Internet of Things applications, and identified a server-compromise client-impersonation vulnerability caused by verifier cancellation in the linear server-side key computation. To address this vulnerability, we proposed E-VTBSAEv1 while preserving the verifier-based architecture and online H2C-free structure, and also discussed E-VTBSAEv2 as a challenge-based variant. We established the AKE security and forward secrecy of E-VTBSAEv1 in the random-oracle model under the CDH assumption, formally analyzed resistance to server-compromise client-impersonation attacks after server-side verifier-file leakage, and further verified its security properties using Scyther. Experimental results on devices with different computing capabilities show that E-VTBSAEv1 maintains runtime efficiency comparable to that of the original protocol. Although E-VTBSAEv1 is suitable for resource-constrained IIoT devices, more efficient elliptic-curve implementations and curve choices may be further explored for scenarios requiring higher runtime efficiency.

REFERENCES

[1] J. Srinivas, A. K. Das, X. Li, M. K. Khan, and M. Jo, "Designing anonymous signature-based authenticated key exchange scheme for Internet of Things-enabled smart grid systems," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4425–4436, 2021.

[2] K. Mahmood, M. N. Fatima, S. Shamshad, Z. Ghaffar, A. K. Das, and M. J. F. Alenazi, "A cost-effective key agreement encryption protocol for securing IIoT-enabled WSN communication," *IEEE Internet of Things Journal*, vol. 12, no. 5, pp. 5185–5193, 2025.

[3] H. M. S. Badar, K. Mahmood, W. Akram, Z. Ghaffar, M. Umar, and A. K. Das, "Secure authentication protocol for home area network in smart grid-based smart cities," *Computers and Electrical Engineering*, vol. 108, Art. no. 108721, 2023.

[4] T. Tariq, W.-C. Kuo, K. Mahmood, S. Shamshad, A. K. Das, and M. J. F. Alenazi, "A lightweight authentication protocol for RFID-assisted supply chain management system," *IEEE Internet of Things Journal*, vol. 12, no. 9, pp. 12845–12852, 2025.

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

[6] V. Boyko, P. D. MacKenzie, and S. Patel, "Provably secure password-authenticated key exchange using Diffie–Hellman," in *Advances in Cryptology – EUROCRYPT 2000*. Berlin, Germany: Springer, 2000, pp. 156–171.

[7] O. Goldreich and Y. Lindell, "Session-key generation using human passwords only," in *Advances in Cryptology – CRYPTO 2001*. Berlin, Germany: Springer, 2001, pp. 408–432.

[8] J. Katz, R. Ostrovsky, and M. Yung, "Efficient password-authenticated key exchange using human-memorable passwords," in *Advances in Cryptology – EUROCRYPT 2001*. Berlin, Germany: Springer, 2001, pp. 475–494.

[9] S. M. Bellovin and M. Merritt, "Encrypted key exchange: Password-based protocols secure against dictionary attacks," in *Proc. IEEE Computer Society Symposium on Research in Security and Privacy*, 1992, pp. 72–84.

[10] D. P. Jablon, "Strong password-only authenticated key exchange," *SIGCOMM Computer Communication Review*, vol. 26, no. 5, pp. 5–26, 1996.

[11] F. Hao and P. Ryan, "J-PAKE: Authenticated key exchange without PKI," in *Transactions on Computational Science XI*. Berlin, Germany: Springer, 2010, pp. 192–206.

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

[13] M. Qi and W. Hu, "Provably secure asymmetric PAKE protocol for protecting IoT access," *IEEE Internet of Things Journal*, vol. 11, no. 4, pp. 7071–7078, 2024.

[14] T. Wu, "The secure remote password protocol," in *Proc. Network and Distributed System Security Symposium*, 1998.

[15] D. Taylor, T. Wu, N. Mavrogiannopoulos, and T. Perrin, "Using the Secure Remote Password (SRP) protocol for TLS authentication," RFC 5054, Nov. 2007.

[16] V. Shoup, "Security analysis of SPAKE2+," in *Theory of Cryptography*. Cham, Switzerland: Springer, 2020, pp. 31–60.

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

[18] S. Jarecki, H. Krawczyk, and J. Xu, "OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks," in *Advances in Cryptology – EUROCRYPT 2018*, Lecture Notes in Computer Science, vol. 10822. Cham, Switzerland: Springer, 2018, pp. 456–486.

[19] Y. Gu, S. Jarecki, and H. Krawczyk, "KHAPE: Asymmetric PAKE from key-hiding key exchange," in *Advances in Cryptology – CRYPTO 2021*. Cham, Switzerland: Springer, 2021, pp. 701–730.

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

[21] A. Faz-Hernandez, S. Scott, N. Sullivan, R. S. Wahby, and C. A. Wood, "Hashing to elliptic curves," RFC 9380, Aug. 2023.

[22] M. Vanhoef and E. Ronen, "Dragonblood: Analyzing the Dragonfly handshake of WPA3 and EAP-pwd," in *Proc. IEEE Symposium on Security and Privacy*, 2020, pp. 517–533.

[23] D. Pointcheval and G. Wang, "VTBPEKE: Verifier-based two-basis password exponential key exchange," in *Proc. ACM Asia Conference on Computer and Communications Security*, 2017, pp. 301–312.

[24] M. Qi, "VTBSAE: Provably secure verifier-based password authenticated key exchange for IIoT," *IEEE Internet of Things Journal*, 2026, doi: 10.1109/JIOT.2026.3685297.

[25] R. Canetti and H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels," in *Advances in Cryptology – EUROCRYPT 2001*. Berlin, Germany: Springer, 2001, pp. 453–474.

[26] E. Bresson, O. Chevassut, and D. Pointcheval, "Security proofs for an efficient password-based key exchange," in *Proc. ACM Conference on Computer and Communications Security*, 2003, pp. 241–250.

[27] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proc. ACM Conference on Computer and Communications Security*, 1993, pp. 62–73.

[28] C. Cremers, "The Scyther tool: Verification, falsification, and analysis of security protocols," in *Computer Aided Verification*. Berlin, Germany: Springer, 2008, pp. 414–418.

[29] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[30] Certicom Research, "Standards for Efficient Cryptography 1: Elliptic Curve Cryptography," SEC 1, Version 2.0, 2009.