Comments on "Server-Aided Public Key Authenticated Searchable Encryption With Constant Ciphertext and Constant Trapdoor"

Takeshi Yoshida[1] and Keita Emura*[1,2]

[1.]Kanazawa University, Japan.
[2.]AIST, Japan.

October 20, 2026

Abstract

Cheng and Meng (IEEE Transactions on Information Forensics and Security 2024) introduced server-aided public key authenticated encryption with keyword search (SA-PAEKS). In this short note, without a public-key infrastructure, we show that a cloud server (tester) can obtain keyword information from both a ciphertext and a trapdoor.

1 Introduction

Public key Encryption with Keyword Search (PEKS) [1] is vulnerable to Keyword Guessing Attacks (KGA), where an adversary with a trapdoor can obtain keyword information. To resist KGA, Huang and Li proposed Public key Authenticated Encryption with Keyword Search (PAEKS) [5], but it requires a receiver to generate trapdoors for all senders, i.e., the communication costs and searching time is linearly proportional to the number of senders. Li et al. [6] addressed this inefficiency with Public key Authenticated Encryption with ciphertext Update and Keyword Search (PAUKS), where a fully trusted administrator updates and uploads ciphertexts. To remove this trust assumption, Cheng and Meng [3] introduced Server-Aided PAEKS (SA-PAEKS), which adds a sender server and a receiver server. A sender i generates a ciphertext $\mathsf{Ct}_{i\to\mathsf{RS}}$, which can be interpreted as "from sender i to the receiver server RS." Similarly, a receiver j generates a trapdoor $\mathsf{Tr}_{j\to\mathsf{SS}}$, which means "from receiver j to the sender server SS." The sender server SS transforms $\mathsf{Ct}_{i\to\mathsf{RS}}$ into $\mathsf{Ct}_{\mathsf{SS}\to\mathsf{RS}}$, while the receiver server RS transforms $\mathsf{Tr}_{j\to\mathsf{SS}}$ into $\mathsf{Tr}_{\mathsf{RS}\to\mathsf{SS}}$. Through these transformation procedures, the information about the sender and receiver, namely i and j, is effectively removed. As a result, the search complexity no longer depends on the number of senders.

Contribution of this paper. In this short note, we observe that their security models do not explicitly specify assumptions regarding a public-key infrastructure (PKI). Without such assumptions, we show that a cloud server can generate a sender's key and a receiver's key, respectively, and run the encryption algorithm and the trapdoor algorithm using these keys and the challenge keyword.

We clarify that Cheng and Meng have restricted that the cloud server is NOT allowed to *collude* with any sender or receiver as follows:

*Corresponding Author: k-emura@se.kanazawa-u.ac.jp

(III. C. Threat Model in [3]): *The sender server is allowed to collude with any sender/receiver except for the challenge sender/receiver; the receiver server is also allowed to collude with any sender/receiver except for the challenge sender/receiver; the cloud server is not allowed to collude with any sender or receiver.*

Here, "collude" refers to the case where there exists a sender or receiver distinct from the cloud server, and the cloud server obtains the secret key of that sender or receiver. Actually, they also mentioned that the cloud server can determine whether the sender or receiver is colluding with the server as follows:

(V. A. Collusion Resistance in [3]): *The last component of the transformed ciphertext (resp. trapdoor) is exactly the public key of the sender (resp. receiver). Therefore, the cloud server can easily identify which transformed ciphertext (resp. transformed trapdoor) was derived from a ciphertext generated by a colluded sender (resp. receiver). To address this issue, the sender server (resp. receiver server) can randomize the transformed ciphertext (resp. trapdoor) prior to submitting it to the cloud server.*

These descriptions suggest that the case where the cloud server itself runs the key-generation algorithms is not captured in the security models. We also clarify that, in our attacks, the adversary does not deviate from the protocol description and adheres to the honest-but-curious model. One possible way to address this issue is to introduce a registration authority or a bulletin-board PKI, so that the sender server (resp. receiver server) rejects unregistered ciphertexts (resp. trapdoors) before invoking the transformation procedure.

Related Work. As an independent work, Yang et al. [7] presented an attack on the Cheng-Meng SA-PAEKS scheme. By using the cloud server's secret key, they showed that it is possible to check whether two ciphertexts correspond to the same keyword. Since their attack specifically targets the Cheng–Meng SA-PAEKS scheme, it is construction-specific, whereas our attack is more general, focusing on the underlying model and applying independently of particular constructions.

2 Definition of SA-PAEKS

Let $\mathsf{SA\text{-}PAEKS}(\mathsf{Setup}, \mathsf{KeyGen}_{\mathsf{Sender}}, \mathsf{KeyGen}_{\mathsf{Receiver}}, \mathsf{KeyGen}_{SS}, \mathsf{KeyGen}_{\mathsf{RS}}, \mathsf{KeyGen}_{\mathsf{CS}}, \mathsf{PAEKS}, \mathsf{Tran}_{\mathsf{Ct}}, \mathsf{Trapdoor}, \mathsf{Tran}_{\mathsf{Tr}}, \mathsf{Test})$ be a SA-PAEKS scheme defined as follows. All algorithms except Setup implicitly take the common parameter pp as input, even when it is not explicitly specified.

- $\mathsf{Setup}(1^\lambda)$: Given a security parameter $\lambda \in \mathbb{N}$ as input, the setup algorithm outputs the common parameter pp.
- $\mathsf{KeyGen}_{\mathsf{Sender}}(\mathsf{pp})$: Given pp as input, the sender key generation algorithm outputs $(\mathsf{pk}_{S,i}, \mathsf{sk}_{S,i})$.
- $\mathsf{KeyGen}_{\mathsf{Receiver}}(\mathsf{pp})$: Given pp as input, the receiver key generation algorithm outputs $(\mathsf{pk}_{R,j}, \mathsf{sk}_{R,j})$.
- $\mathsf{KeyGen}_{\mathsf{SS}}(\mathsf{pp})$: Given pp as input, the sender server key generation algorithm outputs $(\mathsf{pk}_{\mathsf{SS}}, \mathsf{sk}_{\mathsf{SS}})$.
- $\mathsf{KeyGen}_{\mathsf{RS}}(\mathsf{pp})$: Given pp as input, the receiver server key generation algorithm outputs $(\mathsf{pk}_{\mathsf{RS}}, \mathsf{sk}_{\mathsf{RS}})$.
- $\mathsf{KeyGen}_{\mathsf{CS}}(\mathsf{pp})$: Given pp as input, the cloud server key generation algorithm outputs $(\mathsf{pk}_{\mathsf{CS}}, \mathsf{sk}_{\mathsf{CS}})$.
- $\mathsf{PAEKS}(\mathsf{pp}, \mathsf{sk}_{S,i}, \mathsf{pk}_{\mathsf{RS}}, \mathsf{pk}_{\mathsf{CS}}, kw)$: Given $\mathsf{pp}, \mathsf{sk}_{S,i}, \mathsf{pk}_{\mathsf{RS}}, \mathsf{pk}_{\mathsf{CS}}$ and a keyword kw as input, the encryption algorithm outputs the ciphertext $\mathsf{Ct}_{i\to\mathsf{RS}}$.

- $\mathsf{Tran}_{\mathsf{Ct}}(\mathsf{pk}_{\mathsf{CS}}, \mathsf{sk}_{\mathsf{SS}}, \mathsf{pk}_{S,i}, \mathsf{Ct}_{i\to\mathsf{RS}})$: Given $\mathsf{pk}_{\mathsf{CS}}, \mathsf{sk}_{\mathsf{SS}}, \mathsf{pk}_{S,i}$ and $\mathsf{Ct}_{i\to\mathsf{RS}}$ as input, the ciphertext transformation algorithm outputs $\mathsf{Ct}_{\mathsf{SS}\to\mathsf{RS}}$.
- $\mathsf{Trapdoor}(\mathsf{pp}, \mathsf{sk}_{R,j}, \mathsf{pk}_{\mathsf{SS}}, \mathsf{pk}_{\mathsf{CS}}, kw')$: Given $\mathsf{pp}, \mathsf{sk}_{R,j}, \mathsf{pk}_{\mathsf{SS}}, \mathsf{pk}_{\mathsf{CS}}$ and a keyword to be searched kw' as input, the trapdoor algorithm outputs $\mathsf{Tr}_{j\to\mathsf{SS}}$.
- $\mathsf{Tran}_{\mathsf{Tr}}(\mathsf{pk}_{\mathsf{CS}}, \mathsf{sk}_{\mathsf{RS}}, \mathsf{pk}_{R,j}, \mathsf{Tr}_{j\to\mathsf{SS}})$: Given $\mathsf{pk}_{\mathsf{CS}}, \mathsf{sk}_{\mathsf{RS}}, \mathsf{pk}_{R,j}$ and $\mathsf{Tr}_{j\to\mathsf{SS}}$ as input, the trapdoor transformation algorithm outputs $\mathsf{Tr}_{\mathsf{RS}\to\mathsf{SS}}$.
- $\mathsf{Test}(\mathsf{sk}_{\mathsf{CS}}, \mathsf{Ct}_{\mathsf{SS}\to\mathsf{RS}}, \mathsf{Tr}_{\mathsf{RS}\to\mathsf{SS}})$: Given $\mathsf{sk}_{\mathsf{CS}}, \mathsf{Ct}_{\mathsf{SS}\to\mathsf{RS}}, \mathsf{Tr}_{\mathsf{RS}\to\mathsf{SS}}$, the test algorithm outputs 0 or 1.

In the $\mathsf{Tran}_{\mathsf{Ct}}$ (resp. $\mathsf{Tran}_{\mathsf{Tr}}$) algorithm, the sender's (resp. receiver's) public key is input. Here, we simply assume that a ciphertext (resp. trapdoor) contains the public key.

Ciphertext indistinguishability against cloud server (CI-CS) [3]. Let $\mathcal{A}$ be an adversary which plays a role of the cloud server and $\mathcal{C}$ be the challenger. $\mathcal{C}$ generates the public parameter by running $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$. $\mathcal{C}$ runs following algorithms.

- $(\mathsf{pk}_{S^*}, \mathsf{sk}_{S^*}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sender}}(\mathsf{pp})$, $(\mathsf{pk}_{R^*}, \mathsf{sk}_{R^*}) \leftarrow \mathsf{KeyGen}_{\mathsf{Receiver}}(\mathsf{pp})$, $(\mathsf{pk}_{\mathsf{SS}}, \mathsf{sk}_{\mathsf{SS}}) \leftarrow \mathsf{KeyGen}_{\mathsf{SS}}(\mathsf{pp})$, and $(\mathsf{pk}_{\mathsf{RS}}, \mathsf{sk}_{\mathsf{RS}}) \leftarrow \mathsf{KeyGen}_{\mathsf{RS}}(\mathsf{pp})$.

Then $\mathcal{C}$ sends $(\mathsf{pp}, \mathsf{pk}_{S^*}, \mathsf{pk}_{R^*}, \mathsf{pk}_{\mathsf{SS}}, \mathsf{pk}_{\mathsf{RS}})$ to $\mathcal{A}$. Then $\mathcal{A}$ runs $(\mathsf{pk}_{\mathsf{CS}}, \mathsf{sk}_{\mathsf{CS}}) \leftarrow \mathsf{KeyGen}_{\mathsf{CS}}(\mathsf{pp})$ and sends $\mathsf{pk}_{\mathsf{CS}}$ to $\mathcal{C}$. $\mathcal{A}$ is allowed to access the following oracles.

- $\mathcal{O}_{\mathsf{Ct}}$: Given kw, $\mathcal{C}$ returns a ciphertext $\mathsf{Ct}_{S^*\to\mathsf{RS}} \leftarrow \mathsf{PAEKS}(\mathsf{pp}, \mathsf{sk}_{S^*}, \mathsf{pk}_{\mathsf{RS}}, \mathsf{pk}_{\mathsf{CS}}, kw)$.
- $\mathcal{O}_{\mathsf{Tran}_{\mathsf{Ct}}}$: Given $\mathsf{Ct}_{S^*\to\mathsf{RS}}$, $\mathcal{C}$ returns the transformed ciphertext $\mathsf{Ct}_{\mathsf{SS}\to\mathsf{RS}} \leftarrow \mathsf{Tran}_{\mathsf{Ct}}(\mathsf{pk}_{\mathsf{CS}}, \mathsf{sk}_{\mathsf{SS}}, \mathsf{pk}_{S^*}, \mathsf{Ct}_{S^*\to\mathsf{RS}})$.
- $\mathcal{O}_{\mathsf{Tr}}$: Given kw, $\mathcal{C}$ returns a trapdoor $\mathsf{Tr}_{R^*\to\mathsf{SS}} \leftarrow \mathsf{Trapdoor}(\mathsf{pp}, \mathsf{sk}_{R^*}, \mathsf{pk}_{\mathsf{SS}}, \mathsf{pk}_{\mathsf{CS}}, kw)$.
- $\mathcal{O}_{\mathsf{Tran}_{\mathsf{Tr}}}$: Given $\mathsf{Tr}_{R^*\to\mathsf{SS}}$, $\mathcal{C}$ returns the transformed trapdoor $\mathsf{Tr}_{\mathsf{RS}\to\mathsf{SS}} \leftarrow \mathsf{Tran}_{\mathsf{Tr}}(\mathsf{pk}_{\mathsf{CS}}, \mathsf{sk}_{\mathsf{RS}}, \mathsf{pk}_{R^*}, \mathsf{Tr}_{R^*\to\mathsf{SS}})$.

In the challenge phase, $\mathcal{A}$ sends $\mathcal{C}$ two challenge keywords (kw_0^*, kw_1^*) which are not allowed to use for $\mathcal{O}_{\mathsf{Ct}}, \mathcal{O}_{\mathsf{Tr}}$. $\mathcal{C}$ randomly selects a challenge bit $b \xleftarrow{\$} \{0,1\}$ and runs $\mathsf{Ct}^*_{S^*\to\mathsf{RS}} \leftarrow \mathsf{PAEKS}(\mathsf{pp}, \mathsf{sk}_{S^*}, \mathsf{pk}_{\mathsf{RS}}, \mathsf{pk}_{\mathsf{CS}}, kw_b^*)$, sends it to $\mathcal{A}$. $\mathcal{A}$ is allowed to access to $\mathcal{O}_{\mathsf{Ct}}$, $\mathcal{O}_{\mathsf{Tran}_{\mathsf{Ct}}}$, $\mathcal{O}_{\mathsf{Tr}}$, $\mathcal{O}_{\mathsf{Tran}_{\mathsf{Tr}}}$. $\mathcal{A}$ outputs $b' \in \{0,1\}$ and wins if $b = b'$.

Trapdoor indistinguishability against cloud server (TI-CS) [3]. The setting and the accessible oracles are the same as those of CI-CS. In challenge phase, $\mathcal{A}$ sends $\mathcal{C}$ two challenge keywords (kw_0^*, kw_1^*) which are not allowed to use for $\mathcal{O}_{\mathsf{Ct}}$ and $\mathcal{O}_{\mathsf{Tr}}$. $\mathcal{C}$ randomly selects a challenge bit $b \xleftarrow{\$} \{0,1\}$ and runs $\mathsf{Tr}^*_{R^*\to\mathsf{SS}} \leftarrow \mathsf{Trapdoor}(\mathsf{pp}, \mathsf{sk}_{R^*}, \mathsf{pk}_{\mathsf{SS}}, \mathsf{pk}_{\mathsf{CS}}, kw_b^*)$, sends it to $\mathcal{A}$. $\mathcal{A}$ is allowed to access to $\mathcal{O}_{\mathsf{Ct}}$, $\mathcal{O}_{\mathsf{Tran}_{\mathsf{Ct}}}$,$\mathcal{O}_{\mathsf{Tr}}$, $\mathcal{O}_{\mathsf{Tran}_{\mathsf{Tr}}}$. $\mathcal{A}$ outputs $b' \in \{0,1\}$ and wins if $b = b'$.

In both models, we clarify that a ciphertext (resp. a trapdoor) submitted to $\mathcal{O}_{\mathsf{Tran}_{\mathsf{Ct}}}$ (resp. $\mathcal{O}_{\mathsf{Tran}_{\mathsf{Tr}}}$) is not explicitly restricted as the challenge sender S^* (resp. receiver R^*)-related one. Given the roles of the sender server (resp. receiver server), any ciphertext (resp. trapdoor) should be permitted to be sent to the corresponding transform oracle. If this is not the case, the system would have only a single sender and a single receiver, making the introduction of additional servers meaningless for achieving sender-independent search complexity.

3 Our Attacks

Through these transformation procedures, the information about the sender and receiver is effectively removed. Our attack specifically exploits these transformation procedures. In the attack, the cloud server generates a sender key and a receiver key on its own, and then runs the encryption and trapdoor generation algorithms using those keys. The resulting ciphertext and trapdoor are subsequently transformed in the same manner as in the original scheme. Since the transformation procedures eliminate the information about the original sender and receiver, the search algorithm functions correctly regardless of who the original sender and receiver were.

Our attack against CI-CS. $\mathcal{A}$ generates a trapdoor of the challenge keyword by running

- $(\mathsf{pk}_{R,j}, \mathsf{sk}_{R,j}) \leftarrow \mathsf{KeyGen}_{\mathsf{Receiver}}(\mathsf{pp})$
- $\mathsf{Tr}_{j\to\mathsf{SS}} \leftarrow \mathsf{Trapdoor}(\mathsf{pp}, \mathsf{sk}_{R,j}, \mathsf{pk}_{\mathsf{SS}}, \mathsf{pk}_{\mathsf{CS}}, kw_0^*)$

Then $\mathcal{A}$ sends $\mathsf{Tr}_{j\to\mathsf{SS}}$ and $\mathsf{pk}_{R,j}$ to $\mathcal{C}$ as $\mathcal{O}_{\mathsf{Tran}_{\mathsf{Tr}}}$ and $\mathcal{C}$ returns $\mathsf{Tr}_{\mathsf{RS}\to\mathsf{SS}} \leftarrow \mathsf{Tran}_{\mathsf{Tr}}(\mathsf{pk}_{\mathsf{CS}}, \mathsf{sk}_{\mathsf{RS}}, \mathsf{pk}_{R,j}, \mathsf{Tr}_{j\to\mathsf{SS}})$ to $\mathcal{A}$. $\mathcal{A}$ sends $\mathsf{Ct}^*_{S^*\to\mathsf{RS}}$ to $\mathcal{O}_{\mathsf{Tran}_{\mathsf{Ct}}}$ and receives $\mathsf{Ct}^*_{\mathsf{SS}\to\mathsf{RS}} \leftarrow \mathsf{Tran}_{\mathsf{Ct}}(\mathsf{pk}_{\mathsf{CS}}, \mathsf{sk}_{\mathsf{SS}}, \mathsf{pk}_{S^*}, \mathsf{Ct}^*_{S^*\to\mathsf{RS}})$. Finally, $\mathcal{A}$ outputs $b = 0$ if $\mathsf{Test}(\mathsf{sk}_{\mathsf{CS}}, \mathsf{Ct}^*_{\mathsf{SS}\to\mathsf{RS}}, \mathsf{Tr}_{\mathsf{RS}\to\mathsf{SS}}) = 1$ holds, otherwise, $\mathcal{A}$ outputs $b = 1$.

Our attack against TI-CS. $\mathcal{A}$ generates a ciphertext of the challenge keyword by running

- $(\mathsf{pk}_{S,i}, \mathsf{sk}_{S,i}) \leftarrow \mathsf{KeyGen}_{\mathsf{Sender}}(\mathsf{pp})$
- $\mathsf{Ct}_{i\to\mathsf{RS}} \leftarrow \mathsf{PAEKS}(\mathsf{pp}, \mathsf{sk}_{S,i}, \mathsf{pk}_{\mathsf{RS}}, \mathsf{pk}_{\mathsf{CS}}, kw_0^*)$

Then $\mathcal{A}$ sends $\mathsf{Ct}_{i\to\mathsf{RS}}$ and $\mathsf{pk}_{S,i}$ to $\mathcal{C}$ as $\mathcal{O}_{\mathsf{Tran}_{\mathsf{Ct}}}$ and $\mathcal{C}$ returns $\mathsf{Ct}_{\mathsf{SS}\to\mathsf{RS}} \leftarrow \mathsf{Tran}_{\mathsf{Ct}}(\mathsf{pk}_{\mathsf{CS}}, \mathsf{sk}_{\mathsf{SS}}, \mathsf{pk}_{S,i}, \mathsf{Ct}_{i\to\mathsf{RS}})$ to $\mathcal{A}$. $\mathcal{A}$ sends $\mathsf{Tr}^*_{R^*\to\mathsf{SS}}$ to $\mathcal{O}_{\mathsf{Tran}_{\mathsf{Tr}}}$ and receives $\mathsf{Tr}^*_{\mathsf{RS}\to\mathsf{SS}} \leftarrow \mathsf{Tran}_{\mathsf{Tr}}(\mathsf{pk}_{\mathsf{CS}}, \mathsf{sk}_{\mathsf{RS}}, \mathsf{pk}_{R^*}, \mathsf{Tr}^*_{R^*\to\mathsf{SS}})$. Finally, $\mathcal{A}$ outputs $b = 0$ if $\mathsf{Test}(\mathsf{sk}_{\mathsf{CS}}, \mathsf{Ct}_{\mathsf{SS}\to\mathsf{RS}}, \mathsf{Tr}^*_{\mathsf{RS}\to\mathsf{SS}}) = 1$ holds, otherwise, $\mathcal{A}$ outputs $b = 1$.

4 Discussion

Our attacks do not indicate an inherent vulnerability in the protocol flow of SA-PAEKS itself. Rather, they demonstrate that the CI-CS and TI-CS security models do not explicitly capture setup assumptions. In particular, the attacks are prevented if one assumes a bulletin-board PKI together with an honest key-generation procedure. Therefore, our contribution is to clarify the setup assumptions required for the intended security goals.

Cheng and Meng claimed to extend the PAUKS framework of Li et al. [6] by deploying two additional servers. Here, we explain that our attacks do not work against the Li et al. PAUKS scheme as follows. In PAUKS, as in PAEKS, a sender generates a ciphertext for a keyword using the sender's secret key and a receiver's public key. The receiver generates two types of trapdoors: one that depends on a sender, as in PAEKS, and another that is sender-independent, as in PEKS. The sender-related information in the ciphertext is removed by updating it with an update key, which is generated by a sender's public key and a receiver's secret key. After this update, the sender-independent ciphertext can be searched using the second type of trapdoor, as in PEKS, and the search complexity becomes independent of the number of senders. Since an adversary does not possess the challenge receiver's secret key, it cannot generate the update key. This implies that our attacks do not apply because if an adversary generates a receiver's secret key and generates an update key, the update key does not work against the challenge ciphertext. Since Cheng and Meng claimed to extend the PAUKS framework by deploying two additional servers, this observation suggests that the server-aided setting requires additional setup assumptions that should be explicitly

incorporated into the security model. While our attacks can be prevented by introducing a trusted registration authority or a bulletin-board PKI, SA-PAEKS was originally proposed to eliminate the trusted administrator required in PAUKS. Therefore, although additional setup assumptions address the issue, constructing SA-PAEKS that achieves the intended security guarantees while maintaining resistance to our attacks at the cryptographic level without trusted entities remains an open problem.

We note that Yang et al. [7] employed a different model, called the dual-server PAEKS model [2, 4], and modified the Cheng-Meng SA-PAEKS scheme accordingly. We do not attempt to verify whether the model of Yang et al. exhibits the same vulnerability, namely, that it becomes insecure if a server is allowed to generate both a sender's key and a receiver's key on its own. This remains an interesting direction for future work.

Acknowledgment: This work was supported by JSPS KAKENHI Grant Numbers JP25H01106 and JP26K02908.

References

[1] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *EUROCRYPT*, pages 506–522, 2004. https://doi.org/10.1007/978-3-540-24676-3_30.

[2] Biwen Chen, Libing Wu, Sherali Zeadally, and Debiao He. Dual-server public-key authenticated encryption with keyword search. *IEEE Transactions on Cloud Computing*, 10(1):322–333, 2022. https://doi.org/10.1109/TCC.2019.2945714.

[3] Leixiao Cheng and Fei Meng. Server-aided public key authenticated searchable encryption with constant ciphertext and constant trapdoor. *IEEE Transactions on Information Forensics and Security*, 19:1388–1400, 2024. https://doi.org/10.1109/TIFS.2023.3336160.

[4] Keita Emura. Generic construction of dual-server public key authenticated encryption with keyword search. *IET Information Security*, 2025(1), 2025. https://doi.org/10.1049/ise2/6610587.

[5] Qiong Huang and Hongbo Li. An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks. *Information Sciences*, 403:1–14, 2017. https://doi.org/10.1016/j.ins.2017.03.038.

[6] Hongbo Li, Qiong Huang, Jianye Huang, and Willy Susilo. Public-key authenticated encryption with keyword search supporting constant trapdoor generation and fast search. *IEEE Transactions on Information Forensics and Security*, 18:396–410, 2023. https://doi.org/10.1109/TIFS.2022.3224308.

[7] Wenjie Yang, Shujie Lin, Futai Zhang, and Anjia Yang. Dual-server public-key authenticated searchable encryption with constant trapdoor against inside keyword guessing attacks. *IEEE Transactions on Information Forensics and Security*, 21:5794–5807, 2026. https://doi.org/10.1109/TIFS.2026.3691603.