Online Resource 5

1 SI-SSE-RV : Theoretical Analysis

Since SI-SSE-RV is a simple index based symmetric searchable encryption for conjunctive keyword search, we present theoretical analysis by comparing it with the other analogous SI-based conjunctive keyword SSE schemes. In addition, SI-SSE-RV offers search result verification and hence, we compare its result verification process with the existing verification enabled II-based SSE schemes.

1.1 Comparison with the existing SI-based schemes

We compare storage-computational overhead incurred by the proposed algorithms and the corresponding algorithms of the other schemes [1–4] in Table 1. Note that amongst the existing SSE schemes [1–16], only schemes [1–4] are based on SI search structure and hence, we consider these schemes for comparison.

1.1.1 Storage Overhead

We present storage overhead i.e. Ciphertext Size (CS) (excluding payload data) and Token Size (TS) in terms of size of elements from bilinear groups G_1, G_2, G_T.

From Table 1, we identify that with the asymptotic complexity $O(n)$, CS for the proposed SI-SSE-RV is same as that of the schemes [2, 4] whereas with exactly $(2|G_2|)$ elements, TS for SI-SSE-RV is constant with asymptotic complexity $O(1)$ as that in schemes [1–3].

1.1.2 Computational Overhead

We present computational overhead (excluding the cost of encryption/decryption of payload data) for the proposed *Encryption()*, *Token()* and *Search()* algorithms in terms of operations viz. modular multiplication (m), scalar multiplication (m'), exponentiation (e), and pairing (p) (Table 1). Note that the computational cost of a single modular multiplication operation is negligible as compared to a single pairing, exponentiation, or scalar multiplication operation over bilinear groups [17–19].

Therefore, from Table 1, we determine that the scheme [3] with n executions of two costly operations (i.e. pairing and exponentiation), offers computationally expensive *Encryption()* algorithm. In contrast, the proposed SI-SSE-RV and other schemes [1,2,4] seems more practical since they use either exponentiations or scalar multiplications for *Encryption()* algorithm. We note that with exact $(5m')$ operations, the proposed *Token()* algorithm has constant $O(1)$ computational

Table 1: Comparison of SI-SSE-RV with the existing SI-based SSE schemes

Schemes	Storage Overhead		Computational Overhead		
	CS	TS	Encryption()	Token()	Search()
Golle et al. 2004 [1]	$(2n+1)\|G_1\|$	$3\|G_1\|$	$2nm+(2n+1)e$	$2m+3e$	$2tp$
Ballard et al. 2005 [2]	$(n+1)\|G_1\|$	$2\|G_2\|$	$nm+(n+1)m'$	$3m'$	$2p$
Byun et al. 2006 [3]	$2\|G_1\|+n\|G_T\|$	$2\|G_1\|$	$m+(n+2)e+np$	$tm+3e$	$tm+2p$
Wang et al. 2008 [4]	$n\|G_1\|$	$n\|G_1\|$	$nm+ne$	$(t+n)m+ne$	np
SI-SSE-RV	$\mathbf{(n+3)\|G_1\|}$	$\mathbf{2\|G_2\|}$	$\mathbf{nm+(n+3)m'}$	$\mathbf{5m'}$	$\mathbf{2p}$

CS: Size of a ciphertext (excluding payload), **TS**: Size of a token, **n**: Number of keywords in ciphertext, **t**: Number of keywords in a query, $|\mathbf{G_1}|, |\mathbf{G_2}|, |\mathbf{G_T}|$: Size of an element from the groups G_1, G_2, or G_T, **m**: Modular multiplications, $\mathbf{m'}$: Scalar multiplications, **p**: Pairing, **e**: Exponentiation

complexity, which is more efficient than the corresponding algorithm of the schemes [3,4] having $O(t)$ token computational overhead. In addition, we determine that the proposed *Search()* algorithm with exact $(2p)$ operations offers constant i.e. $O(1)$ computational complexity. Such complexity is indeed more effective than the existing schemes [1,3] with $O(t)$ search overhead or scheme [4] with $O(n)$ search overhead.

From such analysis, we infer that the proposed SI-SSE-RV with the almost same storage-computational overhead as that in [2], offers an additional utility of search result verification besides conjunctive keyword search provided by [2].

1.2 Comparison with the existing II-based schemes

In this section, we first describe the generalized algorithms involved in the existing II-based schemes [20–29]. Subsequently, we discuss the storage-computational overhead incurred by the verification process of the proposed SI-SSE-RV as compared to the verification overhead for the existing II-based SSE schemes.

1.2.1 Algorithms for II-based SSE schemes

Any II-based verification enabled scheme includes the following fundamental algorithms.

1. **KeyGen(1^λ)** Executed by data owner.
   Taking security parameter λ, this algorithm outputs a master secret key $Key = \{Key_1, Key_2\}$.

2. **IndexGen(D, Key)** Executed by data owner.
   By preprocessing the input document collection $D = \{d_1, ..., d_m\}$, the algorithm first defines a keyword space for Inverted Index $\mathcal{KS}_{\mathcal{II}} = \{v_1, ..., v_V\}$. Utilizing standard encryption (symmetric/asymmetric), this algorithm encrypts each $d_i \in D$. From the available $\mathcal{KS}_{\mathcal{II}}$ and D, the algorithm prepares Inverted Index II that includes each keyword $v_i \in V$ along with a list $L_i = \{ID_{ij} | 1 \leq j \leq m\}$ where ID_{ij} is an identifier of document $d_j \in D$ containing keyword v_i. However, II may be represented either by table [23, 26, 28] or matrix [25] or link list [21, 26, 29] or vector [27] or tree [20, 22, 24] data structure. In addition, to support search result verification, the algorithm computes server-side verification component (VC_S) from the available (V, D). Different schemes use different data structures to compute VC_S. For example, the schemes [20, 21, 23–26] use merkle hash tree to build VC_S. On the other hand, [27] uses Authentication Tag (AT), [28] uses QSet- a special data structure, and [29] uses Accumulative Authentication tag (AAT) to compute VC_S.

3. **Token(W', Key)** Executed by data user.
   This algorithm takes as input a set of keywords $W' = \{v'_\ell | v'_\ell \in \mathcal{KS}_{\mathcal{II}}, 1 \leq \ell \leq V, V = |\mathcal{KS}_{\mathcal{II}}|\}$ and prepares a token $T = \{t_\ell | 1 \leq \ell \leq V\}$ using secret key Key. Note that $\ell = 1$ for single keyword search.

4. **Search(II, T)** Executed by storage server.
   This algorithm applies $t_j \in T$ on index II and extracts the associated list L_j for $1 \leq j \leq \ell$. It then sets a search result $SR = L_j$. Note that for $\ell \neq 1$, $SR = (L_1|L_2|...|L_j)$ where '|' denotes concatenation. Additionally, from the available VC_S, the algorithm builds a proof component PC. Depending upon the data structure used to build VC_S, the size and computational complexity of PC is different. The detailed comparative analysis for proof component is given in Table 4. Finally, the algorithm returns a result $Res = \{SR, PC\}$ to data user.

5. **Verify(Res, Key)** Executed by data user.
   By applying Key_2 on $Res = \{SR, PC\}$, this algorithm verifies the correctness of SR. For the correct result, the algorithm outputs '1' otherwise it outputs '0'.

With the following additional algorithms defined by Kamara et al. in [9], any II-based SSE scheme offers index update operation.

1. **AddTok(II, D, W'_D)** Executed by data owner.
   This algorithm takes as input a new document D to be inserted into the existing Inverted Index II along with a list of keywords associated with that document i.e. W'_D. Subsequently, it outputs an add token T_{add}.

2. **Add(II, T_{add})** Executed by storage server.
   With an available add token T_{add}, this algorithm updates Inverted Index II. On successful completion of add operation, it outputs '1' otherwise, outputs '0'.

3. **DelTok(II, D)** Executed by data owner.
   This algorithm takes as input an already available search index II and a chosen document D to be deleted. It then outputs a delete token T_{del}.

4. **Delete(II, T_{del})** Executed by storage server.
   With an available delete token T_{del}, this algorithm updates Index II. On successful completion of delete operation, it outputs '1' otherwise, outputs '0'.

1.2.2 Storage-Computational overhead during result verification

With Table 2, we present comparative analysis of the proposed SI-SSE-RV with the existing II-based SSE schemes [20–29]. The comparison is in terms of characteristics relevant to the result verification process. Such characteristics are - Computational Complexity (CC) for *IndexGen()* and *Verify()* algorithm at client side, CC to compute Proof Component (PC) at server side, Content of Search Result (SR), Size of PC ($|PC|$), Communication complexity (CM) incurred after execution of *Search()* algorithm. From Table 2, we remark the following:

- As discussed in Section 1.2.1, the *IndexGen()* algorithm computes inverted index II as well as server side verification component (VC_S). For all II-based schemes, inverted index is prepared by processing the given static data collection $D = \{d_1, d_2, ..., d_m\}$ and set of keywords V. Thus, the computational cost to prepare II would be $O(|D| \cdot V)$ for such schemes. However, the cost to compute VC_S depends on the structure used by the underlying scheme. More specifically, in the schemes [20, 21, 24–26] where VC_S is in terms of merkle tree, each keyword is represented by a separate leaf node where the hash of document identifiers containing that keyword is associated. In schemes [22, 23], each node of merkle tree represents a single alphabet and depth of the tree (d) is linear to the highest length word. Apparently, in all these tree-based schemes, the computational complexity (CC) for *IndexGen()* is $O(|D| \cdot V) + O(V \cdot d)$. On the other hand, in scheme [27], an Authentication Tag (AT) of λ-bit is prepared for each document (which is represented by V-bit vector) as VC_S where λ is a security parameter. Hence, CC for *IndexGen()* in [27] would be $O(|D| \cdot V) + O(|D| \cdot V \cdot \lambda)$. The scheme [28] builds a special data structure QSet by processing a set of keywords and its associated documents with the computational complexity $O(\sum |D_w|)$ where $|D_w|$ is the number of documents containing keyword $w \leq V$. Therefore, the overall CC for *IndexGen()* in [28] is $O(|D| \cdot V) + O(\sum |D_w|)$. The

Table 2: Comparison of SI-SSE-RV with the existing II-based SSE schemes

Scheme	CC for *IndexGen()/Encryption()*	Res		CC to compute PC	CC to execute *Verify()*	CM
		SR	|PC|			
II-based [20–24]	$O(\|D\| \cdot V) + O(V \cdot d)$	L_{ID}	$O(S + RT)$	$O(V \cdot d)$	$O(V \cdot d)$	$O(\|L_{ID}\|)$
[25, 26]	$O(\|D\| \cdot V) + O(V \cdot d)$	L_D	$O(\|Q\| \cdot d) + O(\|Q\| \cdot \|G\|)$	$O(\|Q\| \cdot d) + O(\|Q\|)$	$O(\|Q\| \cdot d) + O(\|Q\|)$	O(1)
[27]	$O(\|D\| \cdot V) + O(\|D\| \cdot V \cdot \lambda)$	L_D	$O(\|D\| \cdot \lambda)$	$O(\|D\| \cdot V \cdot \lambda)$	$O(\|D\| \cdot V)$	O(1)
[28]	$O(\|D\| \cdot V) + O(\sum_{1 \leq w \leq V} \|D_w\|)$	L_D	$O(\|Q\| \cdot \|MAC\|)$	$O(\sum_{w \in Q} \|D_w\|)$	$O(\sum_{w \in Q} \|D_w\|)$	$O(1)$
[29]	$O(\|D\| \cdot V) + O(\sum_{1 < w < V} (\|D_w\| \cdot b))$	L_D	$O(\|L_D\| \cdot V)$	$O(\|L_D\| \cdot (\sum (\|D_w\| \cdot b))$	$O(\|L_D\| \cdot \sum (\|D_w\| \cdot b))$	$O(1)$
SI-based **SI-SSE-RV**	$O(n)$	L_D	$O(\|L_D\| \cdot \|G_1\|)$	$O(\|L_D\|)$	$O(\|L_D\|)$	$O(1)$

Res: Query Result, **SR:** Search Result, **PC:** Proof Component, $|\mathbf{PC}|$: Size of Proof Component, **CC:** Computational Complexity, **CM:** Communication complexity, $|\mathbf{D}|$: Size of document collection, **V:** Total number of keywords in II, **d:** Depth of the tree, **S** : Subtree from the original verification tree, **RT:** Hash value of the root node, $\mathbf{L_{ID}}$: List of identifiers for documents satisfying query, $(\mathbf{L_D}, |\mathbf{L_D}|)$: List of encrypted documents satisfying query and its size, $|\mathbf{Q}|$: Number of keywords in query Q, λ : Security Parameter, $|\mathbf{D_w}|$: Number of documents containing keyword $w \leq V$, **b** : Number of blocks per document, **n**: Number of keyword fields used in SI, $\mathbf{G_1}$: Size of an element of bilinear group G_1

scheme [29] computes VC_S by generating Accumulative Authentication tag (AAT) for each keyword. To generate AAT, it divides each document containing that keyword into b blocks where each block of size V-bit. Therefore the CC for *IndexGen()* in [29] would be $O(|D| \cdot V) + O(\sum_{1 \leq w \leq V}(|D_w| \cdot b))$. Contrast to the aforesaid II-based SSE schemes, in the proposed SI-SSE-RV, a simple index is prepared during *Encryption()* algorithm with the computational overhead of $O(n)$ for n keyword fields in each ciphertext. In practice, $n << V$, and thus *Encryption()* algorithm is much more efficient than *IndexGen()* algorithm. However, a single execution of *Encryption()* algorithm prepares a single SI whereas a single execution of *IndexGen()* constructs an II for the entire data collection D.

- A Query Result (Res) for all verification enabled schemes includes two parts: Search Result (SR) and Proof Component (PC). For schemes [20–24] supporting a single-keyword search, SR includes L_{ID}=$\{ID_i | 1 \leq i \leq D_w\}$, ID_i is identifier for document D_i and $|D_w|$ is the number of documents containing the requested keyword w. With such Res, a user is required to communicate with server again to get actual documents D_i hence, the Communication Complexity (CM) in such schemes is $O(|L_{ID}|)$. On the other hand, in multi-keyword (especially conjunctive search) schemes, [25–29], $Res = L_D$ where L_D is the list of encrypted documents satisfying query. Therefore, a single-round of communication is required in these schemes. However, in schemes [25, 26, 28, 29], the set intersection operation at server leaks keyword-documents relation to server. Contrast to this, SI-SSE-RV extracts L_D from the set of encrypted documents without any set-intersection operation and thus avoids leakage of information to server.

- A proof component PC in tree-based mechanisms [20–24] include a subtree S of the original tree and the root node RT. Such subtree is computed by traversing the nodes in path from each leaf node to root node and hence CC to compute PC would be $O(V \cdot d)$ and size of $|PC|$ is $O(S + RT)$. The result verification in all these schemes includes full tree reconstruction and hence CC for *Verify()* algorithm would be $O(d \cdot V)$. On the other hand, in the other tree-based schemes [25, 26], the PC is computed by traversing the nodes on the path from the leafs representing the queried keyword to root. Additionally, PC includes group elements for all queried keywords and hence size and computational cost for PC is $(O(|Q| \cdot d) + O(|Q| \cdot |G|))$ and $(O(|Q| \cdot d) + O(|Q|))$. Correspondingly, the computational overhead for *Verify()* is also $(O(|Q| \cdot d) + O(|Q|))$. In the authentication tag based mechanism [27], PC is computed by preparing an authentication tag (of size λ-bit) for each encrypted document (V-bit vector) available on server and hence the $|PC|$=$O(|D| \cdot \lambda)$ and its computational cost is $O(|D| \cdot V \cdot \lambda)$. Accordingly, its verification cost is $O(|D| \cdot V)$. In the QSet based mechanism [28], PC includes MAC digest for each keyword in query. Such digest is computed by processing the number of documents containing that keyword and hence the size and computational cost for PC is $(O(|Q| \cdot |MAC|))$ and $(O(\sum_{w \in Q} |D_w|))$ respectively. The corresponding verification cost is also $(O(\sum_{w \in Q} |D_w|))$. In scheme [29], PC includes accumulative authentication tag (AAT) (of V-bit) for each document matching with the queried keyword and hence $|PC|$=$O(|L_D| \cdot V)$. However, such AAT would be prepared by processing such documents as b-bit blocks and hence computational overhead for PC would be $O(|L_D| \cdot (\sum(|D_w| \cdot b))$. The verification cost would also be the same. Contrast to all the aforesaid II-based schemes, PC in the proposed SI-SSE-RV, includes exactly $(2|G_1|)$ components. Such PC is associated with each returned document, and hence, the overall size of PC would be $(|L_D| \cdot |G_1|)$. Such size of PC and its computational cost would be much better than the existing schemes [20–27] where computational overhead depends on V or d. In addition, the cost to verify the result is also linear to $|L_D|$. Moreover, though the scheme [28] is computationally efficient (with overhead linear to $|Q|$), it leaks each keyword-document(s) relation to server by executing set-intersection operation. In addition, the computational cost in [29] is considerably more than SI-SSE-RV.

1.2.3 Remarks

Besides the aforementioned computational efficiency, we enlist the following advantages offered by the proposed SI-SSE-RV:

- **Support dynamic data collection** In the proposed SI-SSE-RV, a separate simple index I is prepared for each document. The construction of I does not require knowledge of entire data collection. Each document is independent of the other documents stored on SS. Any insertion or deletion or modification of a document does not affect the stored

data collection. Apparently, SI-SSE-RV supports dynamic data collection and is more suitable for the applications where future data insertion is required very often.

- **Support variable values for keywords** In SI-SSE-RV, an Index I is a list of encrypted keywords where each keyword is treated as 'KeywordName=Value'. The name of each keyword is pre-decided whereas variable values for the same keyword can be set in the Index prepared for different documents. Supporting variable values for keywords is indeed essential for applications where values of keywords are not known apriori and depend on the associated document. For such applications, SI-SSE-RV is more suitable than any of the existing II-based schemes.

- **Support conjunctive keyword search** With SI-SSE-RV, we offer conjunctive keyword search operation. Though the existing II-based schemes [20–24] provide multi-keyword search, they do not support conjunction among keywords. On the other hand, the schemes [25, 26, 28] offer conjunctive keyword search. However, with set-intersection operation (discussed as follows), these schemes reveal much information to server.
  Practically, in any II-based scheme, a conjunctive keyword search can be defined in either way : (i) For V keyword values, 2^V combination of keywords are required to be stored on server. Subsequently, on conjunctive query request, server performs search across all 2^V combinations, (ii) If conjunctive query contains t number of keywords, server extracts a separate set of document identifiers for each keyword and subsequently performs set-intersection operation among the extracted sets. The first solution incurs $O(2^V)$ storage overhead on server whereas the second solution reveals the keyword-document(s) relationship to server.
  With SI-SSE-RV, we do not store a separate list of keywords onto the server and hence, no direct value of any keyword is revealed to the server. Thus, the proposed *Search()* algorithm offers conjunctive keyword search without any additional storage overhead on server and without any leakage of keyword-document(s) relation to server. Consequently, SI-SSE-RV is best suited for applications where conjunctive search query is the most common query type.

References

[1] Philippe Golle, Jessica Staddon, and Brent Waters. Secure conjunctive keyword search over encrypted data. In *Applied Cryptography and Network Security*, pages 31–45. Springer, 2004.

[2] Lucas Ballard, Seny Kamara, and Fabian Monrose. Achieving efficient conjunctive keyword searches over encrypted data. In *Information and Communications Security*, pages 414–426. Springer, 2005.

[3] Jin Wook Byun, Dong Hoon Lee, and Jongin Lim. Efficient conjunctive keyword search on encrypted data storage system. In *European Public Key Infrastructure Workshop*, pages 184–196. Springer, 2006.

[4] Peishun Wang, Huaxiong Wang, and Josef Pieprzyk. Keyword field-free conjunctive keyword searches on encrypted data and extension for dynamic groups. In *Cryptology and Network Security*, pages 178–195. Springer, 2008.

[5] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 44–55. IEEE, 2000.

[6] Eu-Jin Goh et al. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

[7] Kaoru Kurosawa and Yasuhiro Ohtaki. Uc-secure searchable symmetric encryption. In *International Conference on Financial Cryptography and Data Security*, pages 285–298. Springer, 2012.

[8] Reza Curtmola and Ostrovsky Garay, Kamara Seny. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 79–88. ACM, 2006.

[9] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 965–976. ACM, 2012.

[10] Seny Kamara and Charalampos Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security*, pages 258–274. Springer, 2013.

[11] Peter Van Liesdonk, Saeed Sedghi, Jeroen Doumen, Pieter Hartel, and Willem Jonker. Computationally efficient searchable symmetric encryption. In *Secure data management*, pages 87–100. Springer, 2010.

[12] Haiping Huang, Jianpeng Du, Hui Wang, and Ruchuan Wang. A multi-keyword multi-user searchable encryption scheme based on cloud storage. In *Trustcom/BigDataSE/I SPA, 2016 IEEE*, pages 1937–1943. IEEE, 2016.

[13] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology–CRYPTO 2013*, pages 353–373. Springer, 2013.

[14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Network and Distributed System Security Symposium (NDSS14)*, 2014.

[15] Tarik Moataz and Abdullatif Shikfa. Boolean symmetric searchable encryption. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 265–276. ACM, 2013.

[16] Yunling Wang, Jianfeng Wang, Shi-Feng Sun, Joseph K Liu, Willy Susilo, and Xiaofeng Chen. Towards multi-user searchable encryption supporting boolean query and fast decryption. In *International Conference on Provable Security*, pages 24–38. Springer, 2017.

[17] Nam-Hun Koo, Gook-Hwa Jo, Chang-Hoon Kim, and Soon-Hak Kwon. On the computational cost of pairing and ecc scalar multiplication. *The Journal of Korean Institute of Communications and Information Sciences*, 36(1C):14–21, 2011.

[18] Dustin Moody, Rene Peralta, Ray Perlner, Andrew Regenscheid, Allen Roginsky, and Lily Chen. Report on pairing-based cryptography. *Journal of research of the National Institute of Standards and Technology*, 120:11, 2015.

[19] Joppe W Bos, Craig Costello, and Michael Naehrig. Exponentiating in pairing groups. In *International Conference on Selected Areas in Cryptography*, pages 438–455. Springer, 2013.

[20] Jianfeng Wang, Xiaofeng Chen, Hua Ma, Qiang Tang, Jin Li, and Hui Zhu. A verifiable fuzzy keyword search scheme over encrypted data. *Journal of Internet Services and Information Security (JISIS)*, 2:49–58, 2012.

[21] HweeHwa Pang and Kyriakos Mouratidis. Authenticating the query results of text search engines. *Proceedings of the VLDB Endowment*, 1(1):126–137, 2008.

[22] Qi Chai and Guang Gong. Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In *Communications (ICC), 2012 IEEE International Conference on*, pages 917–922. IEEE, 2012.

[23] Zachary A Kissel and Jie Wang. Verifiable phrase search over encrypted data secure against a semi-honest-but-curious adversary. In *Distributed Computing Systems Workshops (ICDCSW), 2013 IEEE 33rd International Conference on*, pages 126–131. IEEE, 2013.

[24] Wenhai Sun, Bing Wang, Ning Cao, Ming Li, Wenjing Lou, Y Thomas Hou, and Hui Li. Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. *IEEE Transactions on Parallel and Distributed Systems*, 25(11):3025–3035, 2014.

[25] Wenhai Sun, Xuefeng Liu, Wenjing Lou, Y Thomas Hou, and Hui Li. Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 2110–2118. IEEE, 2015.

[26] Yuxi Li, Fucai Zhou, Yuhai Qin, Muqing Lin, and Zifeng Xu. Integrity-verifiable conjunctive keyword searchable encryption in cloud storage. *International Journal of Information Security*, 17(5):549–568, 2018.

[27] Zhiguo Wan and Robert H Deng. Vpsearch: Achieving verifiability for privacy-preserving multi-keyword search over encrypted cloud data. *IEEE Transactions on Dependable and Secure Computing*, 2016.

[28] Xiuxiu Jiang, Jia Yu, Jingbo Yan, and Rong Hao. Enabling efficient and verifiable multi-keyword ranked search over encrypted cloud data. *Information Sciences*, 403:22–41, 2017.

[29] Xinrui Ge, Jia Yu, Hanlin Zhang, Chengyu Hu, Zengpeng Li, Zhan Qin, and Rong Hao. Towards achieving keyword search over dynamic encrypted cloud data with symmetric-key based verification. *IEEE Transactions on Dependable and Secure Computing*, 2019.