Online Resource 6

1 Performance Evaluation and Analysis

1.1 Experimental Setup

To conduct experiments, we use client-server architecture. As client, we use a local 32-bit, 2.10 GHz Pentium Core 2 Duo CPU with Windows 7 machine. As server, we employ an Amazon EC2 T2 (t2.micro) instance [1] of 64-bit, 3.3 GHz Intel Xeon processor with Microsoft Windows Server 2012. To implement *Encryption()*, *Token()* and *Search()* algorithms, we use Java Pairing Based Cryptographic (JPBC) library [2]. From JPBC Library, we utilize Type A pairing $G_1 \times G_2 \rightarrow G_T$ that is based on Elliptic Curve $E(F_q)$: $y^2 = x^3 + x$ where we set $G_1 = G_2$. Here, G_1 is subgroup of $E(F_q)$, and G_T is subgroup of $E(F_q)^2$ where q is a large prime number and order of G_1 i.e. $p = 160$ bits with the base field $q = 512$ bits.

In addition, we consider a real-world dataset, namely Enron email dataset [3]. We randomly choose emails from this dataset and use them as data collection (D). We next define a keyword space $\mathcal{KS}_{\mathcal{SI}}$ of $n = 100$ keyword fields (relevant to the email messages) to evaluate SI-based SSE schemes. Additionally, by considering each keyword field of $\mathcal{KS}_{\mathcal{SI}}$ as a set V_i wherein $|V_i| = 2^p$ for large p, we define a keyword space for II-based schemes as $\mathcal{KS}_{\mathcal{II}} = V = \sum_{i=1}^{n} |V_i| = (n \cdot 2^p) >>> n$. However, for implementation purpose, we select the moderate values for different simulation parameters listed in Table 1. With such parameters, we run experiments multiple times and consider their average values as results.

1.2 Result Analysis - SI based SSE schemes

Initially, we run $SK \leftarrow Setup()$ algorithm with security parameter λ of $p = 160$-bit (order of bilinear group). We further execute the proposed *Encryption()*, *Token()*, and *Search()* algorithms as well as the corresponding algorithms of the other SI-based conjunctive searchable schemes [4–7] and evaluate their results as follows:

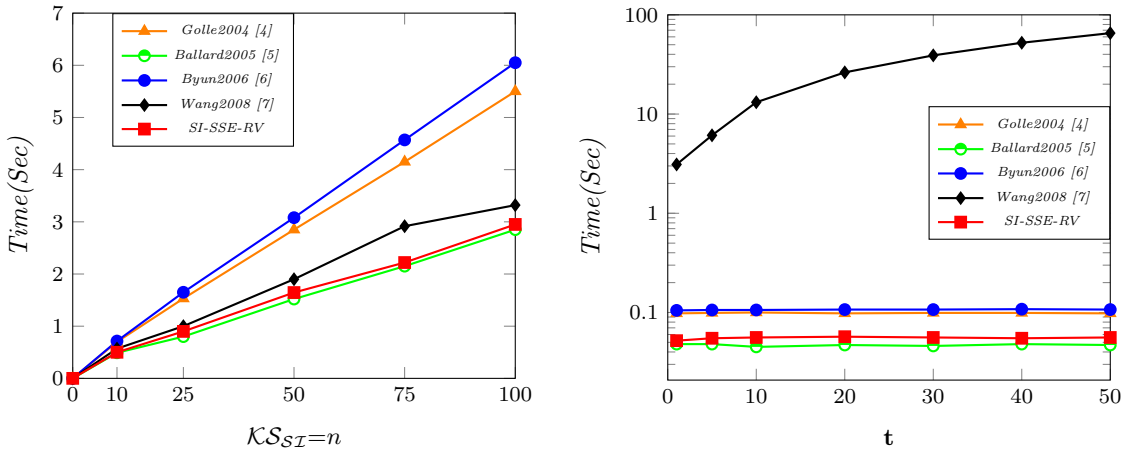
1.2.1 Encryption() algorithm

- We choose a random email D_i from data collection D and assign a random ID_i. For each $w_j \in \mathcal{KS}_{\mathcal{SI}}$, we choose a random value v_j and prepare a set $W = \{w_j = v_j | 1 \leq j \leq n\}$. We then run the proposed Encryption$(D_i(ID_i), W, SK)$. Note that to compute D'_i, we use AES algorithm (for 128 bit key) from javax.crypto package. We then run *Encryption()* algorithm of different SI-base SSE schemes [4–7]. We generates simulation results for different values of n (Table 1) and present them in Figure 1(a).

Table 1: Simulation Parameters

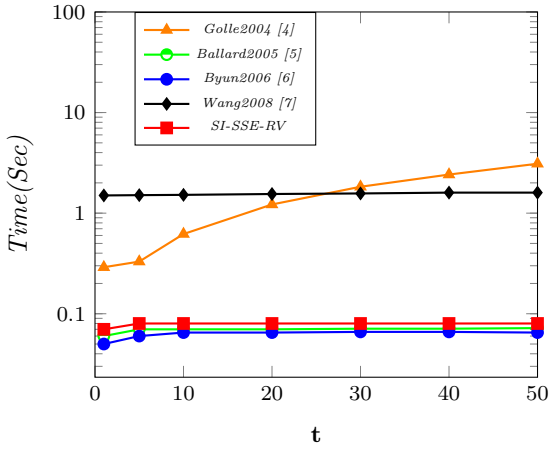
Parameters	**Values**
$\mathcal{KS}_{\mathcal{SI}}$=$n$	$\{10, 25, 50, 75, 100\}$
$\mathcal{KS}_{\mathcal{II}}$=$V$	$\{2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9, 2^{10}\}$
$\lvert D\rvert$	$\{100, 200, 300, 400, 500\}$
t	$\{10, 20, 30, 40, 50\}$
$\lvert L_D\rvert$	$\{500, 1000, 1500, 2000, 2500\}$

$\mathbf{n}$: Number of keyword fields in SI, $\mathbf{V}$: Number of keywords in II, $|\mathbf{D}|$: Number of documents in document collection, $\mathbf{t}$: Number of keywords in query, $|\mathbf{L_D}|$: Number of documents returned from the server as a query result



(a) Encryption()

(b) Token()



(c) Search()

Figure 1: Simulation results for SI-based SSE schemes

- From results, we observe that the execution time of the proposed *Encryption()* is linearly increasing with n. However, it is much more faster than the corresponding algorithms of schemes [4, 6, 7].

1.2.2 Token() Algorithm

- To evaluate performance of *Token()* algorithm, we select conjunctive queries Q' with different values of t (Table 1). As a large number of keywords in conjunction makes a query complex and impractical, we select comparatively small values for t and present the results in Figure 1(b).

- From results, we note that the execution time for the proposed *Token()* algorithm is constant regardless of the number of keywords in a query (t). With such optimal execution time, the proposed algorithm is more efficient than the *Token()* algorithm of scheme [7] where execution time is linear to t.

1.2.3 Search() Algorithm

- To evaluate performance of *Search()* algorithm, we select ciphertexts with $n = 100$, apply tokens with different t and present results in Figure 1(c). Note that the result presented in Figure 1 (c) shows the execution time of *Search()* over a single ciphertext at a time.

- From results, we identify that the execution time of the proposed *Search()* algorithm is almost constant and independent of t. The results show that SI-SSE-RV performs conjunctive search with much less computational time as compared to existing conjunctive search schemes [7] having search time linear to n.

1.3 Result Analysis - II based SSE schemes

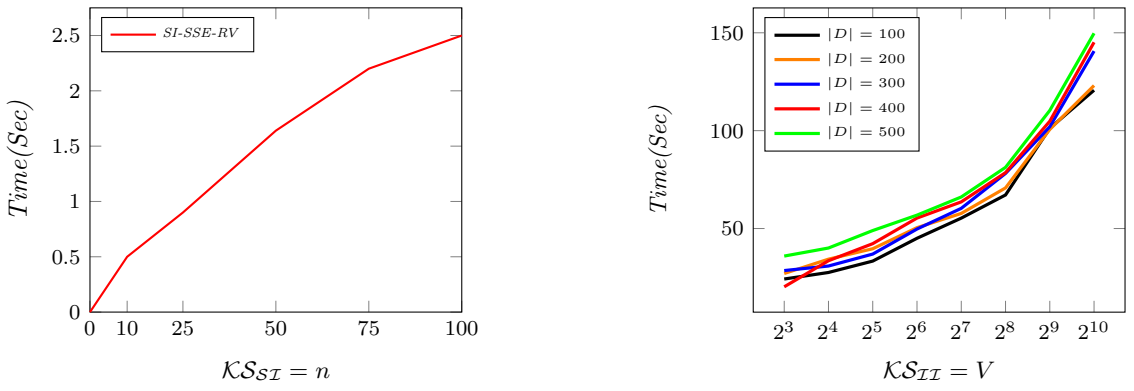
We evaluate the proposed verification process (that includes overhead incurred during Index preparation, Proof Component computation, and Result verification) by comparing *Encryption()*, *Search()* and *Verify()* algorithms of SI-SSE-RV with *IndexGen()*, *Search()* and *Verify()* algorithms (as discussed in **Online Resource 5**) of II-based scheme respectively. Note that different II-based schemes use different data structures to build II and VC_S. Thus, to simplify the comparison, we choose the first dynamic SSE scheme [8] and evaluate the performance of the proposed algorithms. To simulate *IndexGen()* algorithm, we implement *Enc()* algorithm of [8]. Additionally, within this algorithm, we add the steps of implementing merkle hash tree to build a verification tree VT. To simulate search process of II-based scheme, we implement *Search()* algorithm of scheme [8]. However, for simulating *Verify()* algorithm, we implement the steps to traverse already built merkle hash tree VT. In addition, to evaluate index update operation, we implement *AddTok()*, *Add()* and *DelToken()*, *Del()* algorithms of scheme [8].

1.3.1 Index Preparation

The overhead involved in the construction of Simple Index (SI) by *Encryption()* algorithm and Inverted Index (II) by *IndexGen()* algorithm is shown in Figure 2(a) and Figure 2(b) respectively. From Figure 2(a), we determine that an SI could be prepared in time linear to n. However, a single execution of *Encryption()* algorithm constructs an SI for a single document. On the other hand, from Figure 2(b), it is realized that the *IndexGen()* algorithm suffers from $O(|D| \cdot V)$ computational overhead (that could be much higher than $O(n)$) in order to prepare an II. However, one-time execution of *IndexGen()* algorithm constructs a single II that is common for entire data collection. From this fact, we infer that a small sized document collection with a large number of keyword values i.e. $V >> n$, the proposed *Encryption()* algorithm indeed performs better than *IndexGen()* algorithm. On the other hand, for a large data collection with a few fixed keyword values, the *IndexGen()* offers better result than the proposed *Encryption()* algorithm.

1.3.2 Proof Component Computation

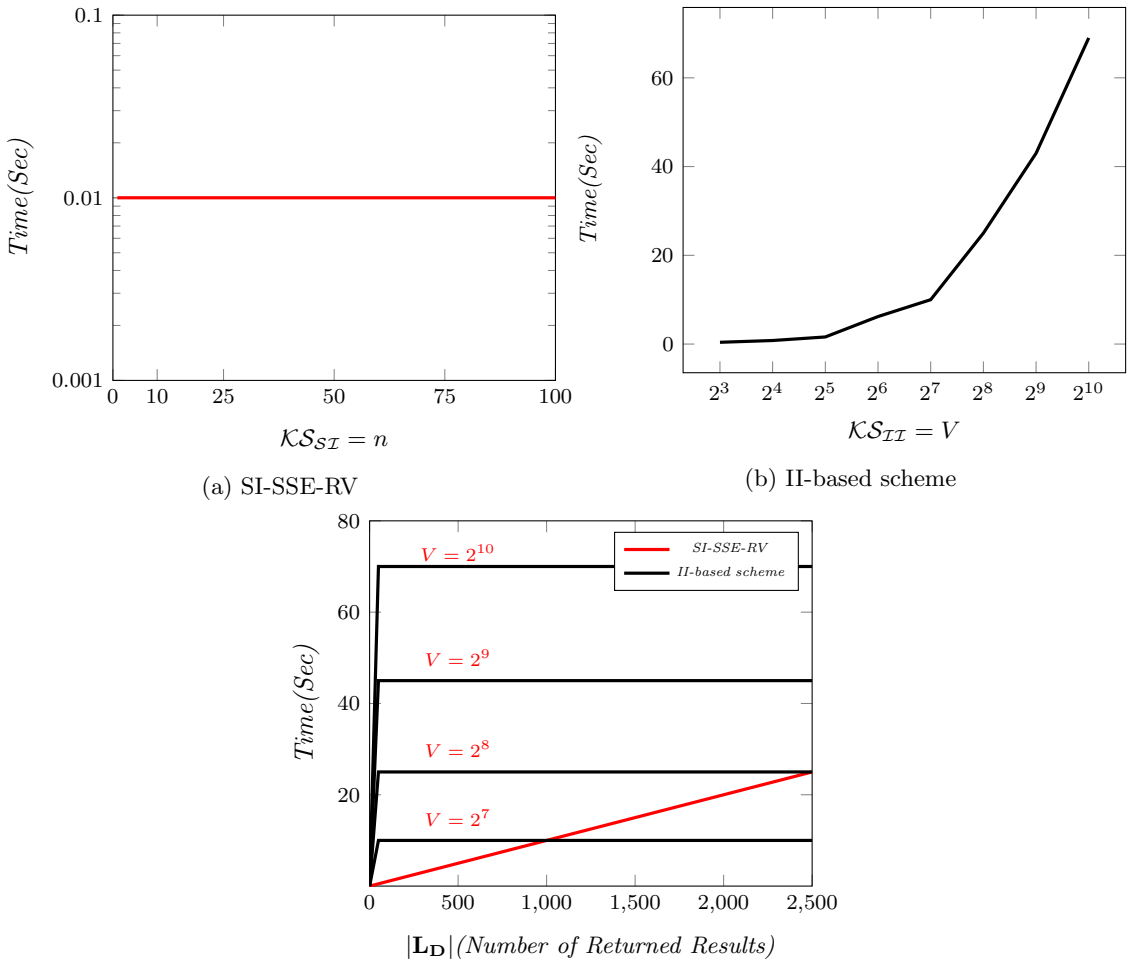
As discussed earlier, in all verification enabled schemes, the server after performing search, computes a poof component from the available VC_S. Though different schemes utilize different data structures for VC_S, their execution time overhead would be different. Here, we consider merkle tree-based VC_S and show that how the number of keywords in II (i.e. V)



(a) SI-SSE-RV

(b) II-based scheme

Figure 2: Simulation Results for Index Generation



(a) SI-SSE-RV

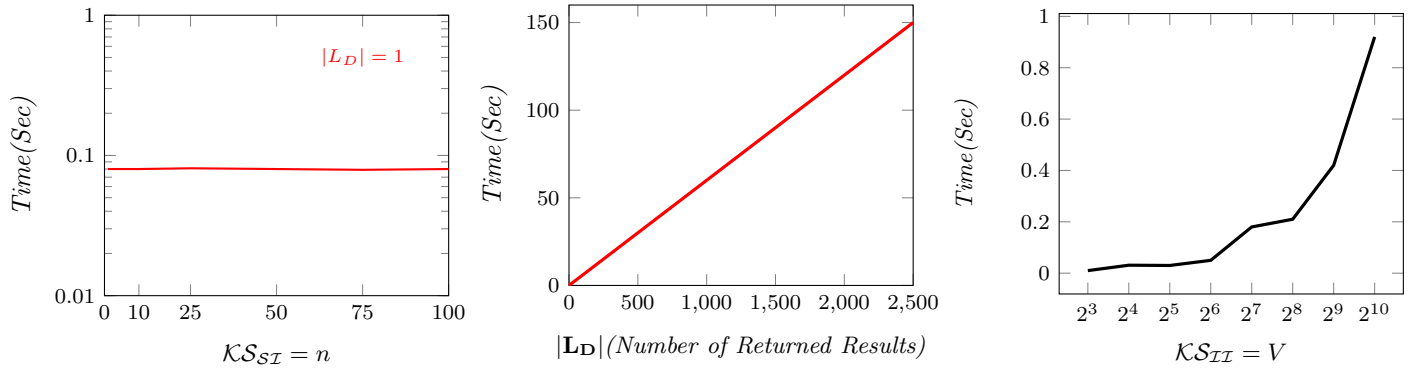
(b) II-based scheme

(c) SI-SSE-RV vs. II-based scheme

Figure 3: Simulation Results for Proof Component Generation

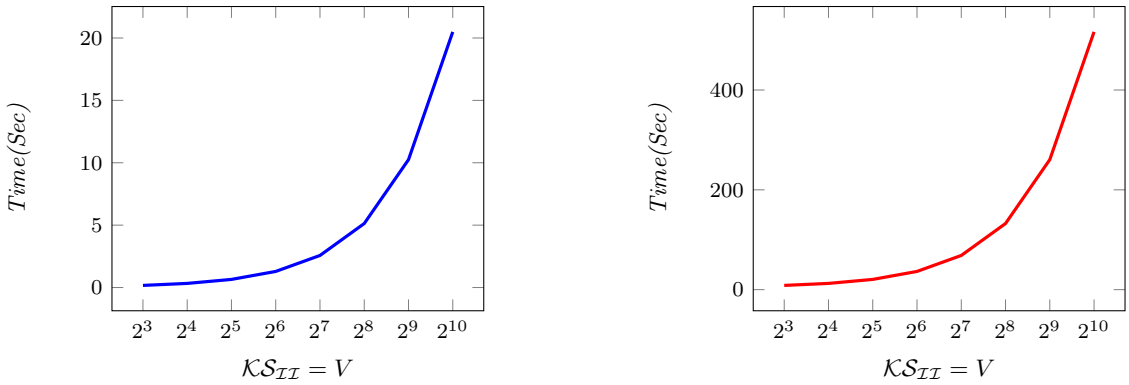
would adversely affects the proof component computation. With Figure 3(a), Figure 3(b) and Figure 3(c), we demonstrate the execution time overhead incurred by the proposed *Search()* algorithm and the *Search()* algorithm of II-based scheme respectively. Note that we consider only proof component computation overhead from the *Search()* algorithm of the underlined schemes.

From results in Figure 3(a), we observe that the execution overhead during computation of PC in the proposed SI-SSE-RV remains constant regardless of the number of keyword fields (n) in SI. On the other hand, from results in Figure 3(b), we determine that in II-based schemes (especially merkle tree-based schemes), computation of PC would be adversely affected by the number of keywords in II. This implies that the II-based schemes would take time significantly higher than the proposed scheme. Additionally, from results in Figure 3(c), we remark that the increasing number of returned results $|L_D|$ would affect the performance of the PC computation cost of the proposed SI-SSE-RV. However, it is worth noting that even though $|L_D|$ is small, the II-based scheme would take considerably large execution time. For example, for $|L_D| = 500$, the proposed SI-SSE-RV would take about 5sec whereas II-based scheme would take about 10s (for $V = 2^7$) or 70s (for $V = 2^{10}$). Thus, we claim that the proposed scheme would be much more better than the II-based scheme in case of $V >>> n$ and $|L_D| << |D|$. On the other hand, for a large static data collection and a small set of keywords, the existing II-based schemes are computationally sound.



(a) SI-SSE-RV (b) SI-SSE-RV (c) II-based scheme

Figure 4: Simulation Results for Verify() algorithm



(a) AddTok() algorithm (b) Add() algorithm

Figure 5: Simulation Results for Index update (Add a document) algorithms of II-based scheme

1.3.3 Result Verification

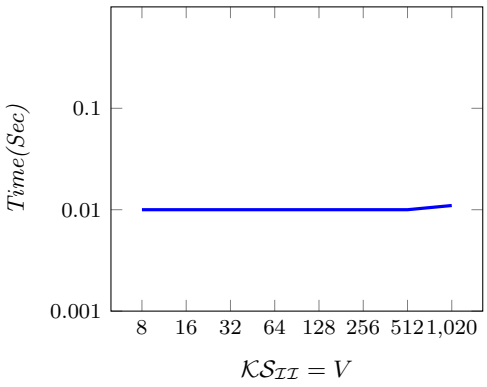
The proposed *Verify()* algorithm uses exactly 2 pairing operations to verify the correctness of a single result. On the other hand, the *Verify()* algorithm of II-based scheme (with merkle tree as VC_S) involves tree reconstruction from the returned proof component (that includes subtree computed from the leaf nodes presenting the requested keywords and the information of the other keywords not involved in query along with the root node value). Ultimately, the verification cost in II-based schemes depends on the total number of keywords in II i.e. V. However, with a single execution of *Verify()* algorithm of II-based scheme, the correctness of search result (comprising multiple document identifiers) could be checked.

The overhead involved in execution of *Verify()* algorithm of SI-SSE-RV for a single result and for multiple results are shown in Figure 4(a) and Figure 4(b) respectively. From these results, it can be observed that though the computational overhead for the proposed *Verify()* algorithm is constant (for a single result $|L_D| = 1$), it increases linearly with the number of returned results ($|L_D|$).

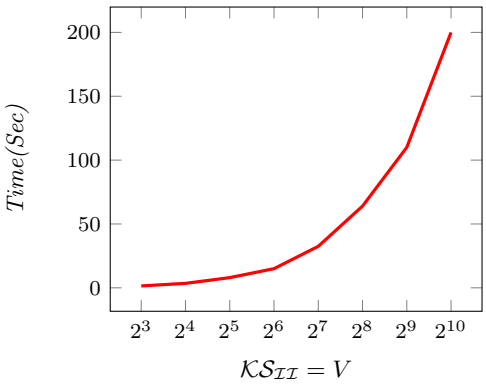
We additionally present simulation results for *Verify()* algorithm of II-based scheme in Figure 4(c). From results, we determine that the size of $\mathcal{KS}_{\mathcal{II}} = V$ affects the computational overhead of such *Verify()* algorithm and thus could be effective only for small V.

1.3.4 Index Update

There are 4 algorithms concerning update of inverted index viz. (i) *AddTok()*, (ii) *Add()*, (iii) *DelTok()*, (iv) *Del()* (as discussed in **Online Resource 5**). Since all these algorithms use $\mathcal{KS}_{\mathcal{II}}$, their computational cost depends on the size of



(a) DelTok() algorithm



(b) Del() algorithm

Figure 6: Simulation Results for Index update (Delete a document) algorithms of II-based scheme

$\mathcal{KS}_{\mathcal{II}}$ that is V.

We show the execution overhead for *AddTok()* and *Add()* algorithm (to insert a single document) in Figure 5(a) and Figure 5(b) respectively. From results, we observe that the size of II that is V, affects the execution of both of these algorithms. Since insertion of a single file into the existing document collection (at server) is highly expensive (Figure 5(b)), the II-based methods are not feasible for applications employing dynamic data collection.

The execution overhead incurred during *DelTok()* and *Del()* algorithm (to delete a single document from the stored collection) is given in Figure 6(a) and Figure 6(b) respectively. From Figure 6(a), we determine that the computational overhead for preparation of delete token is constant regardless of number of keywords associated with the document to be deleted. However, actual deletion of a document updates original index, and thus the computational overhead for *Del()* algorithm depends on V keywords of $\mathcal{KS}_{\mathcal{II}}$. On the other hand, SI-SSE-RV supports dynamic data update (insertion or deletion of a document) without any index update operation at server.

References

[1] Amazon ec2 aws. `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html`.

[2] Angelo De Caro and Vincenzo Iovino. jpbc: Java pairing based cryptography. In *Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011*, pages 850–855, Kerkyra, Corfu, Greece, June 28 - July 1, 2011. IEEE.

[3] William W Cohen. Enron email dataset. `https://www.cs.cmu.edu/~enron/`, 2009.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.