

File-Injection Attacks on Searchable Encryption, Based on Binomial Structures

Master's Thesis

Tjard J. Langhout

File-Injection Attacks on Searchable Encryption, Based on Binomial Structures

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Tjard J. Langhout

born in Amsterdam, The Netherlands



Cyber Security Research Group
Department of Intelligent Systems
Faculty EEMCS, Delft University of Technology
Delft, The Netherlands
www.ewi.tudelft.nl



Secura B.V.
Herikerbergweg 15
Amsterdam, the Netherlands
www.secura.com

File-Injection Attacks on Searchable Encryption, Based on Binomial Structures

Author: Tjard J. Langhout
Student id: 4670124

Abstract

One distinguishable feature of file-inject attacks on searchable encryption schemes is the 100% query recovery rate, i.e., confirming the corresponding keyword for each query. The main efficiency consideration of file-injection attacks is the number of injected files. In the work of Zhang *et al.* (USENIX 2016), $\log_2 |K|$ injected files are required, each of which contains $|K|/2$ keywords for the keyword set K . Based on the construction of the uniform (s, n) -set, Wang *et al.* need fewer injected files when considering the threshold countermeasure. In this work, we propose a new attack that further reduces the number of injected files where Wang *et al.* need up to 38% more injections to achieve the same results. The attack is based on an increment (s, n) -set, which is also defined in this paper.

Thesis Committee:

Chair:	Prof. G. Smaragdakis, Faculty EEMCS, TU Delft
Daily supervisor:	Dr. K. Liang, Faculty EEMCS, TU Delft
Co-daily supervisor:	H. Chen, M.Sc., Faculty EEMCS, TU Delft
Committee member:	Dr. J.E.A.P. Decouchant, Faculty EEMCS, TU Delft
Company supervisor:	T. Tervoort, M.Sc., Secura

Preface

The past months have been an incredible journey. Initially, I was uncertain about my direction and the potential success of my project, but it soon evolved into an enriching pursuit, where I specialized in my subject and confidently developed improvements on previous works. Publishing a paper, especially within the timeline of this thesis, was beyond my wildest dreams. I am eagerly looking forward to present this work at ESORICS 2024.

This journey would not have been possible without the guidance and support of my exceptional supervisors. They were always ready to help, candid in their feedback, and committed to fostering improvement. I would like to extend my heartfelt thanks to Huanhuan Chen for his time and effort in guiding me and co-authoring the paper. I am also grateful to Kaitai Liang for his guidance throughout the project and his valuable feedback on my report, which continually enhanced my work. Finally, I would like to express my appreciation to Georgios Smaragdakis. Despite his busy schedule, he consistently provided feedback and was always available for very pleasant discussions and updates.

I am very grateful to Secura for providing me with the opportunity to conduct my thesis as a graduation intern. I could not have had a better supervisor than Tom Tervoort, who was very prepared to help with setting up testing environments and brainstorm through the challenges thrown with me.

Tjard J. Langhout
Delft, the Netherlands
June 26th, 2024

Contents

Preface	iii
Contents	v
List of Figures	vii
List of Tables	vii
1 Introduction	1
1.1 Introduction	1
2 Background	5
2.1 SE-Schemes	5
2.1.1 SE Basics	5
2.1.2 L1: Query-revealed occurrence pattern	7
2.1.3 L2: Fully-revealed occurrence pattern	7
2.1.4 L3: Fully-revealed occurrence pattern with keyword order	7
2.1.5 L4: Full plaintext under deterministic word-substitution cipher	8
2.1.6 Countermeasures	8
2.2 SE-Attacks	9
2.2.1 Models	9
2.2.2 Overview	12
3 Related Works	15
3.1 Notation	15
3.2 Binary-Search Attack	15
3.2.1 Mechanism of Operation	15
3.2.2 Construction Methodology	15
3.2.3 Binary-Attack with Threshold	16
3.2.4 Examples	16
3.3 Finite Set Theory Attack	18

3.3.1	Mechanism of Operation	19
3.3.2	Constructing of a uniform (s, n) -set of a finite set K	19
3.3.3	Examples	21
4	Design and Develop	25
4.1	SE Model	25
4.2	Attack Model	25
4.3	Research Questions	26
4.3.1	$RQ_{1.1}$: (s, n) -set Construction	27
4.3.2	$RQ_{1.2}$: File Interrelation	31
4.3.3	$RQ_{1.3}$: Space Efficiency	34
4.3.4	RQ_1 : File-Injection Attack	38
5	Experiments and Results	41
5.1	Experimental setup	41
5.2	Performance on the Enron Dataset	41
5.3	Performance under a Threshold of 200	42
5.4	Performance under Different Thresholds	43
6	Countermeasure & Mitigation	45
6.1	File-injection Attacks on SE Schemes with Keyword Padding	45
6.1.1	Calculating the Effects	45
6.1.2	Visualising the Effects	46
6.2	Adopted Binomial-Attack	50
6.2.1	Removing the (n, n) -Set	51
6.2.2	Results after the Mitigation	51
7	Discussion	53
7.1	Discussion	53
7.2	Future Work	54
8	Conclusion	55
8.1	Conclusion	55
	Bibliography	57
A	Paper	61

List of Figures

4.1	Technical framework of existing attacks.	26
4.2	Re-organized technical framework for clear quality dispersion.	27
5.1	Performance of different injection attacks under a threshold of $T=200$	42
5.2	Performance of file-injection attacks under different thresholds.	43
6.1	Candidate set size per query, $T=200$	47
6.2	Extra injection size per query, $T=200$	47
6.3	Candidate set size per query, $T=100$	47
6.4	Extra injection size per query, $T=100$	47
6.5	Candidate set size per query, $T=300$	48
6.6	Extra injection size per query, $T=300$	48
6.7	Candidate set sizes per query when padding is applied, $T=200$	49
6.8	Candidate set sizes per query that is not in the target set when padding is applied, under different thresholds, where the target set is a subset of the full keyword universe, for the standard Binomial-attack.	50
6.9	Extra injection sizes per query that is not in the target set when padding is applied, under different thresholds, where the target set is a subset of the full keyword universe, for the standard Binomial-attack.	50
6.10	Candidate set sizes per query that is not in the target set when padding is applied, under different thresholds, where the target set is a subset of the full keyword universe, for the adopted Binomial-attack.	52
6.11	Extra injection sizes per query that is not in the target set when padding is applied, under different thresholds, where the target set is a subset of the full keyword universe, for the adopted Binomial-attack.	52

List of Tables

2.1	Plaintext Searchable Scheme	6
2.2	Searchable Encryption Scheme	7
2.3	Threshold countermeasure with a threshold of $T=3$. Only the files with three or less keywords get accepted by the scheme.	8
2.4	Padding countermeasure example for the response on a query for k_1 . Besides the corresponding files, also additional files are sent back that do not match the query keyword.	9
2.5	Obfuscation countermeasure example for the response on a query for k_1 . On the contrary to padding, less files are sent back. The response does not contain all the matching files to the query.	9
2.6	File-injection in an SE-scheme	10
2.7	Different kinds of leakage patterns.	11
2.8	An overview of previous SE-attacks.	13
3.1	Summary of notations used for our-and the previous-attacks.	16
3.2	An example of the binary-attack, with a keyword universe of 8.	17
3.3	An example of the binary-attack, with a keyword universe of 16 and a threshold of 4.	18
3.4	The results of Table 3.3 displayed in a more compressed manner.	18
3.5	An example of recovering 10 keywords with the uniform (3,5)-set.	20
3.6	The number of the keywords in the keyword universe K_i corresponding to the relation between s_i and n_i when $T=200$	21
3.7	An example of recovering 23 keywords with threshold $T=7$ by the uniform (7,8)-set.	22
3.8	An example of recovering 16 keywords with threshold $T=5$ by the uniform (4,5)-set and (3,4)-set.	23
4.1	An example of all the variants possible from a positional pattern (P.P.) for $n = 7$ and $r = 3$, and how they would be injected.	28
4.2	An example of the construct of the injected files for $x_4r \leq T \leq x_5r$, $n = 7$ and $r = 3$. Where up till the last positional pattern (P.P.) any positional patterns can be used to make unique combinations and the last positional pattern consists out of consecutive files (P.P. 1). If $T = x_i r$, the threshold ends precisely in between positional patterns. If $x_i r < T < x_{i+1} r$, P.P. 1 should be used from $x_i r$ onward. All columns in P.P. 1 can identify $\frac{n}{r}$ keywords.	29
4.3	Construction of the last $T \pmod r$ columns of an (s, n) -set under the two different possible scenarios, guaranteeing $\frac{n}{r}$ identifiable keywords per column. . .	30

4.4	The binary search structure of the injected files in the Binary-attack for 5 050 keywords, with a threshold of 200.	32
4.5	The (s, n) -sets structure of the injected files in the FST-attack for 5 050 keywords, with a threshold of 200.	33
4.6	The updated (s, n) -set structure of the injected files in the for 5 050 keywords, with a threshold of 200.	34
4.7	The increment $[r, n]$ -set structure of the injected files for 5 050 keywords, with a threshold of 200.	35
4.8	An example of recovering 23 keywords with threshold $T=7$ by an increment $[3, 6]$ -set, which is divided into 3 blocks. Keywords in the 1st, 2nd, and 3rd block can be recovered by 1, 2, and 3 returned files, respectively.	37
6.1	Distribution of an Increment $[3, 6]$ -set, without $(6, 6)$ -set, $T=7$	51

Chapter 1

Introduction

1.1 Introduction

Ensuring exclusive data access remains a paramount concern, often necessitating external cloud servers due to limited user storage capacity. To enable efficient data searches, these servers must implement search-over-plaintext methods for speed and efficacy.

Song *et al.* [19] were pioneers in proposing a cryptographic scheme tailored to address the challenge of searching encrypted data, particularly enabling controlled and concealed keyword searches. This general searchable encryption (SE) framework entails the storage of an index and database on the server. Each keyword within a file undergoes independent encryption, alongside the encryption of the file as a whole. Retrieval of files containing specific keywords involves the user generating a token by encrypting the desired keyword, which is then matched against all encrypted keywords stored on the server. Upon a match, the entire encrypted file is returned to the user for decryption. Since the introduction of this foundational scheme, numerous researchers have proposed diverse variants of SE schemes [2, 3, 5, 10, 13, 16, 20]. These schemes offer varying levels of file and keyword privacy, with the ORAM scheme emerging as the most secure, effectively concealing access pattern leakage [13]. However, schemes with minimal leakage patterns tend to be computationally intensive and impractical. Alternatively, other proposed schemes, while computationally less burdensome, permit a marginally higher degree of leakage. Cash *et al.* [4] categorized these schemes into distinct leakage levels: L1, L2, L3, and L4, each revealing different degrees of information about keyword occurrences. Subsequent studies have demonstrated the potential exploitation of even minimal leakage to extract significant information from databases, emphasizing the critical role of prior knowledge in facilitating successful attacks [1, 9, 11, 14]. Recovery of keywords involves retrieving the keyword associated with the queried token, representing an encrypted keyword of a file.

Attacks on SE schemes may manifest as either passive or active. Passive attacks entail the observation of leakage patterns to construct keyword-query matches [7, 11, 15, 18]. These attacks refrain from interfering with protocols and leverage preexisting knowledge to execute their strategies. Passive attacks typically target weaker schemes exhibiting higher leakage levels (L2-L4) and often necessitate external or prior knowledge for execution.

Conversely, active attacks involve servers injecting files into a user’s database to glean insights. Injection attacks leverage either file access patterns or volume patterns [1, 4, 17, 21, 22, 23]. Active attacks, typically assuming L1 leakage or less, necessitate minimal prior knowledge, contrasting with the requirements of passive attacks. Successful recovery of keywords in active attacks is consistently achieved with 100% accuracy, with the performance metric being the number of injections required for a successful attack.

Cash *et al.* [4] were among the first to introduce an active attack wherein the server sends files to the client, subsequently encrypted and stored by the client. These attacks typically assume L2-L3 leakage, akin to passive attacks. Attackers construct files of their choosing and transmit them to users, compelling the application of the scheme to the received file, thereby enabling observation of ciphertext by the server. Zhang *et al.* [23] categorized such attacks as file-injection attacks and introduced the Binary-attack, premised on L1 leakage and injecting half of the keyword universe per injection, akin to binary search methodologies.

Countermeasures such as thresholds and padding are implemented to impede the success of attacks. Thresholds impose limits on the number of keywords a file can contain, while padding obscures actual results by introducing additional files alongside queried files. Wang *et al.* [21] proposed an alternative approach to injection attacks based on finite set theory, offering superior performance compared to previous methods. This approach, known as the FST-attack, necessitates fewer injections than the Binary-attack under certain conditions, leveraging so-called (s, n) -sets to enhance attack efficacy. Despite these advancements, the FST-attack’s reliance on singular s values for all identified keywords represents a notable limitation.

Our contributions. In this research, we introduce a novel file-injection attack on SE schemes surpassing all prior works. Termed the Binomial-attack, our approach leverages binomial structures for calculation. It involves a self-defined Increment $[r, n]$ -set comprising multiple uniform (s, n) -sets, resulting in a reduction of injected files compared to existing standards like the FST-attack [21]. The (s, n) -sets utilized in the Increment $[r, n]$ -set are constructed in a new way to allow using just a portion of an (s, n) -set. The Increment $[r, n]$ -set begins with the lowest frequency per keyword and incrementally increases frequency with each successive combination, thereby minimizing space occupancy per keyword and reducing the need for injected files. Notably, our attack outperforms previous methods, even under threshold conditions, requiring substantially fewer injections for successful execution. To get to these results a main research question was set, including sub-research question to be able to answer the main question.

RQ₁ How should injected files be constructed to optimise the number of identifiable keywords, given the presence of file access pattern leakage?

RQ_{1.1} Is there a more optimal method for constructing injection files compared to using (s, n) -sets?

RQ_{1.2} How can the interrelatedness of injected files be enhanced beyond the current methods?

RQ_{1.3} How can maximum keyword turnover per space be achieved across all injected files?

Organization of the Paper. The remainder of this paper is organized as follows. Chapter 2 provides a comprehensive description of Searchable Encryption (SE) schemes and SE-attacks. It elucidates the various categories of leakage and discusses countermeasures that can be employed to make the schemes more resilient to attacks. Furthermore, the models of SE-attacks are explained alongside an overview of previous attacks. Chapter 3 presents a brief overview of previously mentioned attacks, with an in-depth discussion of the Binary and FST-attacks due to their relevance to the attack model proposed in this paper. Chapter 4 details the SE-model and attacker model pertinent to the proposed attack in this report and traverses through the design and development of the new attack, addressing the different sub-research questions and integrating the findings to address the main research question. The proposed Binomial-attack is here introduced. Chapter 5 continues on the Binomial-attack demonstrating its applicability under various thresholds and dataset sizes, and its performance compared to previous file-injection attacks. Chapter 6 discusses countermeasures and mitigation strategies for this attack. In Chapter 7.1, the results are debated, and topics not covered within the paper are examined. Finally, the paper is summarized in Chapter 8.1.

Chapter 2

Background

This chapter covers the background of SE-schemes and SE-attacks, providing an overview of existing attacks and the models they operate under.

2.1 SE-Schemes

Searchable encryption (SE) schemes enable clients to encrypt data stored on an untrusted server while still allowing searches through the encrypted files. This section explains the general structure of such schemes and the types of leakage they may incur.

Cash et al. [4] introduced leakage levels L1-L4 where L1 leaks the least information and L4 the most, according to the Curtmola et al. [6] security notion. Moreover, starting from Sect. 2.1.2.

2.1.1 SE Basics

The basic SE scheme involves a client and an untrusted server, where the client encrypts files and keywords separately before storing them on the server. The server maintains these encrypted files and keywords in a database and an inverted index, respectively. When the client wishes to search for files containing a specific keyword, it sends a query with the encrypted keyword to the server. The server then retrieves and returns the matching encrypted files to the client. In the absence of encryption, the server would have the capability to read all the files in its database storage, as they are stored in plaintext. For a visual representation of the information transactions in such a scheme, refer to Fig. 2.1.

Client		Server	
		Operations	Inverted Index
Query(a)	→	a	a : F ₁ , F ₂
		Search(a)	→
		{F ₁ , F ₂ }	←
		↓	Database
		Search({F ₁ , F ₂ })	→
		{a, b, c}, {a, d, e}	←
{a, b, c}, {a, d, e}	←	Response({a, b, c}, {a, d, e})	F ₃ : b, e, f

Table 2.1: Plaintext Searchable Scheme

Definition 1 (Searchable Encryption) An index-based searchable encryption scheme comprises eight algorithms: *KeyGen*, *Encrypt*, *Upload*, *Store*, *Trapdoor*, *Query*, *Search*, *Decrypt*.

1. $\text{KeyGen}(1^\lambda) \rightarrow K$: Generates the secret key K from the security parameter λ .
2. $\text{Encrypt}(F, K) \rightarrow (F_c, W_c)$: Encrypts the file F and its keywords using the secret key K , returning a tuple of the ciphertext of the file and the set of ciphertexts of the keywords.
3. $\text{Upload}(F_c, W_c)$: Sends the encrypted file and keywords to the server for storage.
4. $\text{Store}(F_{c,i}, W_{c,i}) \rightarrow (F_{c,1}, \dots, F_{c,i}, W_{c,1}, \dots, W_{c,i})$: The server stores the encrypted file and keywords in its database F and inverted index I .
5. $\text{Trapdoor}(w, K) \rightarrow t$: Encrypts the keyword to produce a trapdoor t .
6. $\text{Query}(t) \rightarrow F_{c,i}$: The client sends the trapdoor to the server, which responds with the files $F_{c,i}$ containing the trapdoor, where $F_i \in F$.
7. $\text{Search}(t) \rightarrow F_{c,i}$: The server searches its index for the trapdoor and returns the corresponding files to the client.
8. $\text{Decrypt}(F_c, K) \rightarrow F$: The client decrypts the file using the secret key K .

By applying searchable encryption to the previously outlined scheme in Fig. 2.1, we introduce a new set of operations and information transactions. This modification ensures that the server cannot access the keywords stored in the inverted index and database. For a detailed illustration of the new scheme, refer to Fig. 2.2.

Client		Server	
		Operations	Inverted Index
Trapdoor(a) = k_1			$k_1 : F_1, F_2$
Query(k_1)	→	k_1	...
		Search(k_1)	→ ...
		$\{F_1, F_2\}$	← $k_6 : F_3$
		↓	Database
		Search($\{F_1, F_2\}$)	→ $F_1 : k_1, k_2, k_3$
		$\{\{k_1, k_2, k_3\}, \{k_1, k_4, k_5\}\}$	← $F_2 : k_1, k_4, k_5$
$\{\{k_1, k_2, k_3\}, \{k_1, k_4, k_5\}\}$	←	Response($\{k_1, k_2, k_3\}, \{k_1, k_4, k_5\}\}$)	$F_3 : k_2, k_5, k_6$
Decrypt($\{\{k_1, k_2, k_3\}, \{k_1, k_4, k_5\}\}$)			
	↓		
$\{\{a, b, c\}, \{a, d, e\}\}$			

Table 2.2: Searchable Encryption Scheme

2.1.2 L1: Query-revealed occurrence pattern

L1 leakage initially only leaks basic size information, such as the total length of the document. Once a query is performed it also leaks the access pattern of the query, exposing the identifiers of files containing the queried keyword w . From the keyword set W of a file F , a sequence of queries q_1, q_2, \dots, q_Q reveals a sequence of sets

$$\{i : q_1 \in W_i\}, \{i : q_2 \in W_i\}, \dots, \{i : q_Q \in W_i\}.$$

The scheme may scramble the indices i at random to obscure the direct plaintext-ciphertext relation. To conclude, for every query the scheme leaks the identifiers of all the files containing the keyword of the query.

2.1.3 L2: Fully-revealed occurrence pattern

L2 leakage discloses the occurrence pattern for every term prior to any performed queries. However, the file order per keywords is still at random. If files together contain the terms $\cup_i W_i = \{w_1, \dots, w_N\}$, then the scheme leaks the collection of sets

$$\{i : w_1 \in W_i\}, \dots, \{i : w_N \in W_i\}.$$

2.1.4 L3: Fully-revealed occurrence pattern with keyword order

L3 leakage reveals the same as L2, plus the keyword order. Initially, it already fully exposes the occurrence pattern, in the order of their first appearance. The number of occurrences of a keyword within a single document remains hidden. For all keywords w_1, \dots, w_N , the profile leaks the sets

$$\{(i, j) : w_i[j] = w_1\}, \dots, \{(i, j) : w_i[j] = w_N\},$$

where every set contains all the pairs (i, j) with i being the file and j the j -th term of the first keyword.

2.1.5 L4: Full plaintext under deterministic word-substitution cipher

L4 leakage does not conceal any information, exposing all previous mentioned details including the count of each keyword per file.

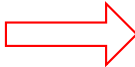
2.1.6 Countermeasures

There are three main countermeasure techniques: thresholds, padding, and obfuscation. Each method addresses specific types of attacks and aims to enhance the security of searchable encryption schemes.

1. **Thresholds:** This technique limits the size of files, preventing them from becoming excessively large. It is particularly effective against active attacks that inject large files to maximize data extraction from a minimal number of files. Refer to Table 2.3 for a visual example.
2. **Padding:** In schemes utilizing padding, the server returns a larger set of results than those that match the query. This set includes true positives as well as false positives, thereby obscuring the actual query response. After decryption, the user can easily filter out the false positives. Padding effectively mitigates leakage by increasing the difficulty for attackers to distinguish between genuine and decoy responses. See Table 2.4.
3. **Obfuscation:** Unlike padding, obfuscation reduces the response set by withholding some true positive files, resulting in a smaller, more ambiguous set of returned files. While this helps obscure the true query results, it may lead to incomplete data retrieval for the user, potentially affecting usability. See Table 2.5.

Each of these countermeasures adds a layer of complexity to the attacker's task, thereby enhancing the overall security of the SE-scheme. However, they also introduce trade-offs in terms of efficiency and usability, which must be carefully balanced.

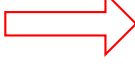
F_1	k_1	k_2	k_3	k_4
F_2	k_2	k_3	k_5	
F_3	k_1	k_7		
F_4	k_4	k_8	k_9	
F_5	k_{11}	k_{12}	k_{13}	k_{14}



F_2	k_2	k_3	k_5
F_3	k_1	k_7	
F_4	k_4	k_8	k_9

Table 2.3: Threshold countermeasure with a threshold of $T=3$. Only the files with three or less keywords get accepted by the scheme.

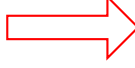
F_1	k_1	k_2	k_3	k_4
F_2	k_2	k_3	k_5	k_6
F_3	k_1	k_3	k_7	k_8
F_4	k_4	k_8	k_9	k_{10}
F_5	k_{11}	k_{12}	k_{13}	k_{14}



F_1	k_1	k_2	k_3	k_4
F_3	k_1	k_3	k_7	k_8
F_2	k_2	k_3	k_5	k_6
F_5	k_{11}	k_{12}	k_{13}	k_{14}

Table 2.4: Padding countermeasure example for the response on a query for k_1 . Besides the corresponding files, also additional files are sent back that do not match the query keyword.

F_1	k_1	k_2	k_3	k_4
F_2	k_2	k_3	k_5	k_6
F_3	k_1	k_3	k_7	k_8
F_4	k_4	k_8	k_9	k_{10}
F_5	k_{11}	k_{12}	k_{13}	k_{14}



F_1	k_1	k_2	k_3	k_4
F_3	k_1	k_3	k_7	k_8

Table 2.5: Obfuscation countermeasure example for the response on a query for k_1 . On the contrary to padding, less files are sent back. The response does not contain all the matching files to the query.

2.2 SE-Attacks

Over the past decades, numerous SE-attacks have been proposed under various models. This section provides an overview of these models and the corresponding attacks, categorizing them based on their specific characteristics and methodologies.

2.2.1 Models

Attack models encompass the different settings and scenarios in which an attack operates. These models can be categorized based on the following criteria: behaviour, type, leakage patterns, objective, pre-knowledge, query frequency selectivity, and adversary.

Behaviour

An attacker can be either honest-but-curious or malicious. An honest-but-curious attacker adheres to the protocol without performing any additional operations but attempts to glean information from the traffic and protocols it observes. This behavior is classified as a *Passive* attack. In contrast, a malicious attacker actively seeks to learn information about the data it handles, going beyond the prescribed protocol by performing extra steps and operations. This behavior is categorized as an *Active* attack.

Attack Types

Attacks can be based on three kinds of data: sampled data, auxiliary data, and chosen data.

- **Sampled data:** Attacks utilizing sampled data require a sample from the target dataset, where the sample's distribution closely matches that of the entire dataset. These attacks are known as *inference* attacks.
- **Auxiliary data:** Auxiliary data refers to data that has been leaked independently of the scheme or extracted from public resources. Attacks based on such data are termed *leakage abuse* attacks.
- **Chosen data:** Chosen data is data created by the attacker. When an attacker can select and generate their own data, the attack is called an *injection* attack.

An extension of the outline of the SE-scheme is provided to illustrate how file injection attacks can be executed (see Fig. 2.6). In this example, note that the inverted index and the database themselves are not encrypted; only the values within the inverted index and database are encrypted. This is solely done for explanation purposes, as normally these components would be encrypted as well. When a file is sent to the client, the client automatically encrypts the message and stores it on the server. If the client subsequently queries the keyword contained in the injected file, the server can detect the access to the injected file and thereby infer the keyword that was searched.

Client		Server	
		Operations	Inverted Index
$\{f\}$	\leftarrow	Inject($\{f\}$)	$k_1 : F_1, F_2$
Encrypt($\{f\}$)			\dots
\downarrow			\dots
Upload($\{k_6\}, \{k_6\}$)	\rightarrow	Store($\{k_6\}, \{k_6\}$)	$k_6 : F_3, F_4$
Query(k_6)	\rightarrow	k_6	
		Search(k_6)	\rightarrow
		$\{F_3, F_4\}$	\leftarrow
		\downarrow	
		Search($\{F_3, F_4\}$)	\rightarrow
		$\{\{k_2, k_5, k_6\}, \{k_6\}\}$	\leftarrow
$\{\{k_2, k_5, k_6\}, \{k_6\}\}$	\leftarrow	Response($\{\{k_2, k_5, k_6\}, \{k_6\}\}$)	$F_1 : k_1, k_2, k_3$
Decrypt($\{\{k_2, k_5, k_6\}, \{k_6\}\}$)			$F_2 : k_1, k_4, k_5$
\downarrow			$F_3 : k_2, k_5, k_6$
$\{\{b, e, f\}, \{f\}\}$			$F_4 : k_6$

Table 2.6: File-injection in an SE-scheme

Leakage Patterns

An SE-scheme can leak a variety of information, which falls into different categories. *Atomic* leakage reveals information about individual matching documents. Table 2.7 provides an overview of the different leakage categories along with clarifications.

Leakage Patterns			Leaks
Atomic	Volumetric	Volume (<i>vol</i>)	Returned documents size
	Access	File ID (<i>fid</i>)	Accessed files
		Co-occurrence (<i>co-oc</i>)	Keyword co-occurrence
Non-Atomic	Volumetric	Total Volume (<i>tvol</i>)	Returned total size
	Access	Response length (<i>rlp</i>)	Number of matches
		Response difference (<i>rdp</i>)	Query response change
	Search (<i>sp</i>)		Queries keyword

Table 2.7: Different kinds of leakage patterns.

Objective

The objective of an attack may encompass the recovery of either the query-keyword pair associated with queries transmitted to the server or the plaintext content of files stored on the server. These objectives are categorized as *Query Recovery* and *Data Recovery*, respectively.

Pre-Knowledge

Attackers may possess various forms of prior knowledge, including sampled data, known queries, and known files.

- **Sampled data:** Data that shares a similar distribution with the targeted dataset.
- **Known queries:** Queries for which the attacker already knows the corresponding keyword matches.
- **files:** Data files from the targeted dataset that are already familiar to the attacker.

Query Frequency Selectivity

In many attacks, a query set is essential for calculating recovery accuracy or ratio. However, real-world datasets often lack existing query logs, necessitating their creation. The query frequency denotes how frequently each keyword is queried.

- **High selectivity:** Most attacks exhibit high selectivity, focusing on querying the most frequently occurring keywords in the dataset.

- **Low selectivity:** Conversely, attacks with low selectivity target keywords that appear infrequently in the dataset.
- **Pseudo-low selectivity:** This approach involves selecting keywords that have a relatively low overall occurrence in the dataset (typically 10-13 times).

Adversary Type

An attacker can either be a *persistent* or a *snapshot* adversary. A persistent adversary only has access to the response data of a query. Conversely, a snapshot adversary extends their intrusion to encompass the transcript of interactions between the client and the server.

2.2.2 Overview

Table 2.8 provides an overview of previous attacks, characterized using the topics outlined in Section 2.2.1. Passive attacks often rely on substantial leakage and may involve unrealistic scenarios, such as high query frequency selectivity. In contrast, active attacks do not require pre-existing knowledge and are unaffected by query selectivity. The objective of all attacks is query recovery, hence it is not mentioned further in the overview. The behaviour is mentioned as either passive or active (P/A). The type as interference/leakage abuse/injection (Inf./Leak./Inj.). The column Query Frequency is abbreviated to Q.F..

Attack	Behav.	Type	Leakage	Pre-know.	Q. F.
Frequency analysis[11]	P.	Inf.	<i>sp</i>	Sample	High
Graph matching[18]	P.	Inf.	<i>co-oc</i>	Sample	High
IKK [9]	P.	Leak.	<i>co-oc</i>	Sample	High
SAP[15]	P.	Leak.	<i>co-oc</i> & <i>rdp</i>	Sample	High
Count[4]	P.	Leak.	<i>co-oc</i> & <i>rlp</i>	Files	High
Refined score[7]	P.	Leak.	<i>co-oc</i>	Sample & Queries	-
LEAP[14]	P.	Leak.	<i>co-oc</i>	Files	-
Subgraph _{VL} [1]	P.	Leak.	<i>vol</i> & <i>rlp</i>	Queries	Pseudo-low
Decoding[1]	A.	Inj.	<i>tvol</i> & <i>rdp</i>	Files	-
Binary search[23]	A.	Inj.	<i>fid</i>	-	-
Finite set theory[21]	A.	Inj.	<i>fid</i>	-	-
Single-round[17]	A.	Inj.	<i>rlp</i>	-	-
Multi-round[17]	A.	Inj.	<i>rlp</i>	-	-
BVA[22]	A.	Inj.	<i>rdp</i>	-	-
BVMA[22]	A.	Inj.	<i>vol</i>	-	-

Table 2.8: An overview of previous SE-attacks.

Chapter 3

Related Works

Two attacks that perform under the same circumstances as the attack of this report are summarised and displayed.

3.1 Notation

We present Table 3.1, which lists the notations used in both previous attacks and our own approach, accompanied by their respective explanations.

3.2 Binary-Search Attack

The Binary-attack was pioneering in proposing an active file-injection attack on an SE-scheme, leveraging file access pattern leakage. This attack involves injecting files into the database and observing the response files to discern the searched keyword. Each keyword corresponds uniquely to a combination of injected files, and the keyword queried can be inferred from the files returned in the response.

3.2.1 Mechanism of Operation

The base attack employs a standard non-adaptive version of binary search to identify user-query keywords. Following file injection, each query returns a distinct combination of injected files. The specific combination of returned files exposes the queried keyword, as each file combination corresponds uniquely to a single keyword. Remarkably, the attack can discern a keyword universe of K using merely $\lceil \log |K| \rceil$ injected files.

3.2.2 Construction Methodology

To determine the number of files needed for injection, we simply calculate the logarithm of the keyword universe size. The algorithm for constructing these files looks as follows:

A visual example illustrating the construction of the attack is provided in Section 3.2.4.

Notations	Definitions
m	keyword size
k_i	Keyword with identifier i
\mathbb{K}	Keyword universe $\mathbb{K} = \{k_1, k_2, \dots, k_m\}$
F_i	File with identifier i
T	Threshold
$ K $	Keyword universe size
Col_i	i th column of the injected file
K_i	Subset of the keyword universe $K_i \in \mathbb{K}$
w	Number of keyword subsets, for Binary
M_i	Combination of injected files that not contain d_i
l	Number of files that contain the keyword k
s	Number of injected files needed to contain all the keywords
n	Number of injected files
r	Keyword frequency in the files
x_i	i th building block for new (s, n) -set construction
n_left	Number of cells left in a specific column of all the injected files together
$P.P.$	Positional Pattern (see Sect. 4.3.1 for definition)

Table 3.1: Summary of notations used for our-and the previous-attacks.

Algorithm 1 Binary-attack

Input: Keyword set $\mathbb{K} = \{k_1, k_2, \dots, k_m\}$

Output: $\mathbf{F} = \{F_1, \dots, F_{\log |K|}\}$.

- 1: **for** $i = 1, \dots, \log |K|$ **do**
 - 2: Generate a file F_i that contains exactly the keywords in \mathbb{K} whose i th bit is 1.
 - 3: **end for**
-

3.2.3 Binary-Attack with Threshold

When the SE-scheme incorporates a threshold, the attack must adapt to ensure it does not surpass this threshold length. It achieves this by generating $\lceil |K|/T \rceil$ subsets, with each containing T keywords. These subsets are then injected as files into the dataset. Subsequently, the attack pairs the subsets into twos and executes the original Binary-attack. The algorithm for this Advanced Binary-attack is outlined as follows:

3.2.4 Examples

For a visual representation of the construct of the injected files and a better understanding of the attack, both with and without a threshold, some examples are given.

Algorithm 2 Advanced Binary-attack**Input:** Keyword set $\mathbb{K} = \{k_1, k_2, \dots, k_m\}$ **Output:** $\mathbf{F} = \{F_1, \dots, F_w, \mathbf{F}_1, \dots, \mathbf{F}_{w/2}\}$.

- 1: Partition the universe into $w = \lceil |\mathbb{K}|/T \rceil$ subsets K_1, \dots, K_w of T keywords each.
- 2: **for** $i = 1, 2, \dots, w$ **do**
- 3: Generate F_i containing every keyword $k \in K_i$.
- 4: **end for**
- 5: **for** $i = 1, 2, \dots, w/2$ **do**
- 6: $\mathbf{F}_i \leftarrow \text{Binary-attack}(K_{2i-1} \cup K_{2i})$
- 7: **end for**

Examples illustrating the construction of the injected files, both with and without a threshold, to provide a visual representation and enhance understanding of the attack, are now given.

Example 1 As an example of recovering 8 keywords $\{k_1, k_2, \dots, k_8\}$, we compute Alg. 1 for $K = 8$. The corresponding injected files are shown in Table 3.2. Keywords are assigned into files: F_1, F_2, F_3 , where 1 denotes the presence of the corresponding keyword and 0 indicates its absence. For example, if files F_1 and F_2 are returned after a query to a token t , we know the corresponding keyword to t is k_2 .

Files	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8
F_1	1	1	1	1	0	0	0	0
F_2	1	1	0	0	1	1	0	0
F_3	1	0	1	0	1	0	1	0

Table 3.2: An example of the binary-attack, with a keyword universe of 8.

Example 2 As an example of recovering 16 keywords $\{k_1, k_2, \dots, k_{16}\}$ with threshold 5, we compute Alg. 2 for $K = 8$. The corresponding injected files are shown in Table 3.3. Keywords are assigned into files: F_1, \dots, F_3 , where 1 denotes the presence of the corresponding keyword and 0 indicates its absence. For example, if files F_1 and F_5 are returned after a query to a token t , we know the corresponding keyword to t is k_2 .

The subsequent sections of the report will explicitly detail the keywords contained in each file and their respective order. Table 3.4 mirrors the contents of Table 3.3, albeit presented in a different format.

Files	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	k_{11}	k_{12}	k_{13}	k_{14}	k_{15}	k_{16}
F_1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
F_2	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
F_3	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
F_4	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
F_5	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0
F_6	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0
F_7	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0
F_8	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0

Table 3.3: An example of the binary-attack, with a keyword universe of 16 and a threshold of 4.

Files	Col_1	Col_2	Col_3	Col_4
F_1	k_1	k_2	k_3	k_4
F_2	k_5	k_6	k_7	k_8
F_3	k_9	k_{10}	k_{11}	k_{12}
F_4	k_{13}	k_{14}	k_{15}	k_{16}
F_5	k_1	k_2	k_5	k_6
F_6	k_1	k_3	k_5	k_7
F_7	k_9	k_{10}	k_{13}	k_{14}
F_8	k_9	k_{11}	k_{13}	k_{15}

Table 3.4: The results of Table 3.3 displayed in a more compressed manner.

3.3 Finite Set Theory Attack

In this section, we review the definition of a uniform (s, n) -set, how the Finite Set Theory (FST)-attack works, how it can be constructed, and its correctness [21]. Afterwards, its implementation for injections with thresholds are shown, and with it, its implications. The method to construct a uniform (s, n) -set of a finite set is presented by Liu and Cao [12].

The FST-attack is an attack designed for SE-schemes that utilizes thresholds on its files. It is an improvement from the Binary-attack as long as the size of the threshold T is larger than 9 and the size of the keyword universe is larger than $2T$.

3.3.1 Mechanism of Operation

The FST-attack uses uniform (s, n) -set to determine which files will represent which keywords and in which order every keyword should be injected in every file. The uniform (s, n) -set is defined as below.

Definition 2 (Uniform (s, n) -set [21]) *Let a set $\mathbb{K} = \{k_1, k_2, \dots, k_m\}$ and the subsets $K_1, K_2, \dots, K_n \subset \mathbb{K}$ be called a uniform (s, n) -set of \mathbb{K} ($m \geq \binom{n}{s-1}$) if the following three conditions are satisfied:*

- $|K_1| = |K_2| = \dots = |K_n|$;
- For any s subsets $K_{i_1}, \dots, K_{i_s} \in \{K_1, \dots, K_n\}$, there is $\bigcup_{j=1}^s K_{i_j} = \mathbb{K}$;
- For any $s-1$ subsets $K_{i_1}, \dots, K_{i_{s-1}} \in \{K_1, \dots, K_n\}$ there is $\bigcup_{j=1}^{s-1} K_{i_j} = \mathbb{K} \setminus \{k_i\}$.

Where n denotes the number of injected files, $|K_i|$ the size of each injected file and m the keyword universe.

3.3.2 Constructing of a uniform (s, n) -set of a finite set K .

We show an example in Table 3.5. The keywords are recovered according to the injected file response of a query.

Suppose there will be n injections for the uniform (s, n) -set of K . The files (F_1, F_2, \dots, F_n) are divide into $\tilde{m} = \binom{n}{s-1}$ groups. Each group contains $s-1$ F_i s. Denotes as:

$$M_1 = F_{1_1} \cup \dots \cup F_{1_{s-1}},$$

$$M_2 = F_{2_1} \cup \dots \cup F_{2_{s-1}},$$

...

$$M_{\tilde{m}} = F_{\tilde{m}_1} \cup \dots \cup F_{\tilde{m}_{s-1}};$$

here, $M_i \neq M_j (1 \leq i \neq j \leq \tilde{m})$ and

$$F_{k_l} \in \{F_1, \dots, F_n\} (1 \leq k \leq \tilde{m}, 1 \leq l \leq s-1).$$

For $1 \leq i \leq \tilde{m}$, let

$$k_i \notin M_i \text{ and } k_i \in M_j \text{ for all } j \neq i.$$

For $\tilde{m} < i \leq m$, let

$$k_i \in F_1 \cap F_2 \cap \dots \cap F_n.$$

The construction works by including k_i in each set F_j where F_j is not one of the sets making up M_i .

Example 3 Constructing a uniform $(3, 5)$ -set for keyword set $\mathbb{K} = \{k_1, k_2, \dots, k_{10}\}$. Let the 5 subset as $(F_1, F_2, F_3, F_4, F_5)$ and we denote

$$M_1 = F_1 \cup F_2, M_2 = F_1 \cup F_3, M_3 = F_1 \cup F_4$$

$$M_4 = F_1 \cup F_5, M_5 = F_2 \cup F_3, M_6 = F_2 \cup F_4$$

$$M_7 = F_2 \cup F_5, M_8 = F_3 \cup F_4, M_9 = F_3 \cup F_5$$

$$M_{10} = F_4 \cup F_5$$

Let

$$k_1 \notin M_1, k_2 \notin M_2, k_3 \notin M_3$$

$$k_4 \notin M_4, k_5 \notin M_5, k_6 \notin M_6$$

$$k_7 \notin M_7, k_8 \notin M_8, k_9 \notin M_9$$

$$k_{10} \notin M_{10}$$

Then F_1, F_2, F_3, F_4 and F_5 can be deduced as shown in Table 3.5.

Files	(3, 5)-set					
	Col ₁	Col ₂	Col ₃	Col ₄	Col ₅	Col ₆
F_1	k_5	k_6	k_7	k_8	k_9	k_{10}
F_2	k_2	k_3	k_4	k_8	k_9	k_{10}
F_3	k_1	k_3	k_4	k_6	k_7	k_{10}
F_4	k_1	k_2	k_4	k_5	k_7	k_9
F_5	k_1	k_2	k_3	k_5	k_6	k_8

Table 3.5: An example of recovering 10 keywords with the uniform $(3, 5)$ -set.

A uniform (s, n) -set for a finite set with size m has the following properties, when we choose $m = \binom{n}{s-1}$.

Lemma 1 ([21]) Let (F_1, F_2, \dots, F_n) be a uniform (s, n) -set, then we have

- *Size.* The size of each file F_i is $|F_i| = \binom{n-1}{s-1}$ for $1 \leq i \leq n$.
- *Intersection.* Let $r = n - s + 1$, then the size of the intersection of arbitrary r files is only 1: $|\cap_{j=1}^r F_{i_j}| = 1$.

Based on the uniform (s, n) -set, Wang *et al.*[21] presents a file-injection attack to SE. We assume the keyword set $\mathbb{K} = \{k_1, k_2, \dots, k_m\}$.

Their basic attack is to first construct a uniform (s, n) -set $\{A_1, A_2, \dots, A_n\}$ based on the technique presented by Liu and Cao [12] for the keyword set \mathbb{K} such that $\binom{n}{s-1} \geq m^1$, and then generate a file set of size n : $\{F_1, F_2, \dots, F_n\}$, where the file F_i contains the same keyword in the A_i for $1 \leq i \leq n$. Those files are then injected into the SE scheme, and the attack recovers the keyword corresponding to a token by the returned $n - s + 1$ files. The correctness of the basic attack is guaranteed by Lemma 1, i.e., there only exists one keyword in the intersection of $n - s + 1$ files.

When the threshold countermeasure is taken into consideration, that is the number of keywords in each file should be smaller than a threshold T , they proposed an advanced file-injection attack, aiming at obtaining a minimum n , the number of files that should be injected. Towards this goal, they choose the minimum n such that

$$\begin{cases} \binom{n-1}{s-1} \leq T \\ \binom{n}{s-1} \geq m. \end{cases} \quad (3.1)$$

Moreover, they present look-up tables to determine the optimal s and n corresponding to the threshold T and the number of keywords in different intervals. the look-up table for a threshold of 200 is displayed in Table 3.6.

The range of the number of keywords in K_i	The range of n_i	Relation between s_i and n_i
$1330 < K_i \leq 20100$	$53 \leq n_i \leq 201$	$s_i = n_i - 1$
$495 < K_i \leq 1330$	$16 \leq n_i \leq 21$	$s_i = n_i - 2$
$252 < K_i \leq 495$	$n_i = 11, 12$	$s_i = n_i - 3$
$ K_i \leq 252$	$n_i \leq 10$	$s_i < n_i$

Table 3.6: The number of the keywords in the keyword universe K_i corresponding to the relation between s_i and n_i when $T=200$.

3.3.3 Examples

Two examples of different attack scenarios are provided to enhance understanding of its operation.

Example 4 As an example of recovering 23 keywords $\{k_1, k_2, \dots, k_{23}\}$ with threshold 7, we solve the Eq. (3.1) for $T = 7$ and $m = 23$ and then get a minimum $n = 8$ and $s = n - 1 = 7$. The corresponding injected files according to a uniform $(7, 8)$ -set are shown in Table 3.7.

¹It means the maximal number of keywords in the uniform (s, n) -set is greater than the keyword size m .

Note that some parts in files $\{F_1, \dots, F_6\}$ are left as blank since the number of keywords in these files has reached 23. Each keyword can be matched by $n - s + 1 = 2$ returned files. For example, if files F_7 and F_8 are returned after a query to a token t , we know the corresponding keyword to t is k_1 .

Files	(7, 8)-set						
	Col ₁	Col ₂	Col ₃	Col ₄	Col ₅	Col ₆	Col ₇
F_1	k_{22}	k_{23}					
F_2	k_{16}	k_{17}	k_{18}	k_{19}	k_{20}	k_{21}	
F_3	k_{11}	k_{12}	k_{13}	k_{14}	k_{15}	k_{21}	
F_4	k_7	k_8	k_9	k_{10}	k_{15}	k_{20}	
F_5	k_4	k_5	k_6	k_{10}	k_{14}	k_{19}	
F_6	k_2	k_3	k_6	k_9	k_{13}	k_{18}	
F_7	k_1	k_3	k_5	k_8	k_{12}	k_{17}	k_{23}
F_8	k_1	k_2	k_4	k_7	k_{11}	k_{16}	k_{22}

Table 3.7: An example of recovering 23 keywords with threshold $T=7$ by the uniform (7, 8)-set.

Example 5 As an example of recovering 16 keywords $\{k_1, k_2, \dots, k_{16}\}$ with threshold 5, we split the keyword universe into two subsets of sizes 10 and 6, as the number of files needed for this is greater with a single set of 16 keywords. Next we solve the Eq. (3.1) for $T = 5$, $m = 10$ and for $T = 5$, $m = 6$. Then we get a minimum $n = 5$ and $s = n - 1 = 4$ for the set of size 10 and $n = 4$ and $s = n - 1 = 3$ for the set of size 6. The corresponding injected files according to the uniform (4, 5)-set and (3, 4)-set are shown in Table 3.8. Note that some columns in files $\{Col_4, Col_5\}$ are left as blank as the (s, n) -sets have run out of possible combinations. Each keyword can be matched by $n - s + 1 = 2$ returned files. For example, if files F_4 and F_5 are returned after a query to a token t , we know the corresponding keyword to t is k_1 .

Files	(4,5)-set & (3,4)-set				
	<i>Col₁</i>	<i>Col₂</i>	<i>Col₃</i>	<i>Col₄</i>	<i>Col₅</i>
F_1	k_7	k_8	k_9	k_{10}	
F_2	k_4	k_5	k_6	k_{10}	
F_3	k_2	k_3	k_6	k_9	
F_4	k_1	k_3	k_5	k_8	
F_5	k_1	k_2	k_4	k_9	
F_6	k_{14}	k_{15}	k_{16}		
F_7	k_{12}	k_{13}	k_{16}		
F_8	k_{11}	k_{13}	k_{15}		
F_9	k_{11}	k_{12}	k_{14}		

Table 3.8: An example of recovering 16 keywords with threshold $T=5$ by the uniform (4,5)-set and (3,4)-set.

These examples also elucidate our primary motivation: a single uniform (s,n) -set within the keyword set, or multiple sets stacked together, may not optimize the effectiveness of a file injection attack. In other words, the number of injected files n may not be optimal due to the possibility that the number of keywords in injected files falls significantly short of reaching the threshold.

Chapter 4

Design and Develop

In an ideal scenario, a searchable encryption scheme would neither leak information during setup and operations nor incur any performance overhead. However, there must be a trade-off between security and efficiency. It is necessary to determine which type of leakage is acceptable to maintain adequate performance of the scheme. This chapter will first discuss the assumed leakage and attack models for the attack proposed in this report. Subsequently, the logical steps for designing our new attack will be visualized and elaborated upon through the use of sub-research questions.

4.1 SE Model

The SE model of the SE-scheme encompasses all settings and information that are leaked during the setup and operation of the SE-scheme's protocol. This model is assumed to function as an email inbox. The scheme adheres to the L1 leakage level, as described in Sect. 2.1.2, where only the file identifiers are exposed for each query performed. This paper assumes a persistent leakage model, where both queries and responses can be observed over time. The query frequency selectivity is not of importance for this particular leakage model, as Section 4.2 will detail an active attack model.

4.2 Attack Model

The attacker in this scenario is a malicious server capable of performing file injections on the client's database, thereby classifying the attack as an active attack. The attack will rely exclusively on the file identifiers from the query responses to gather information. The objective is to recover the underlying keywords of the queries performed by the client. Although no pre-existing knowledge is required for the attack, having some pre-knowledge could be advantageous for constructing a relevant keyword set, ensuring that the injected keywords align with the database's topic.

We have developed a technical framework that outlines two existing file access leakage injection attacks, as well as our new attack. The framework details the types of keyword identification combinations employed by each attack, along with the methods used to con-

struct injected files (building blocks) and their respective orientations relative to each other. This framework is illustrated in Fig. 4.1.

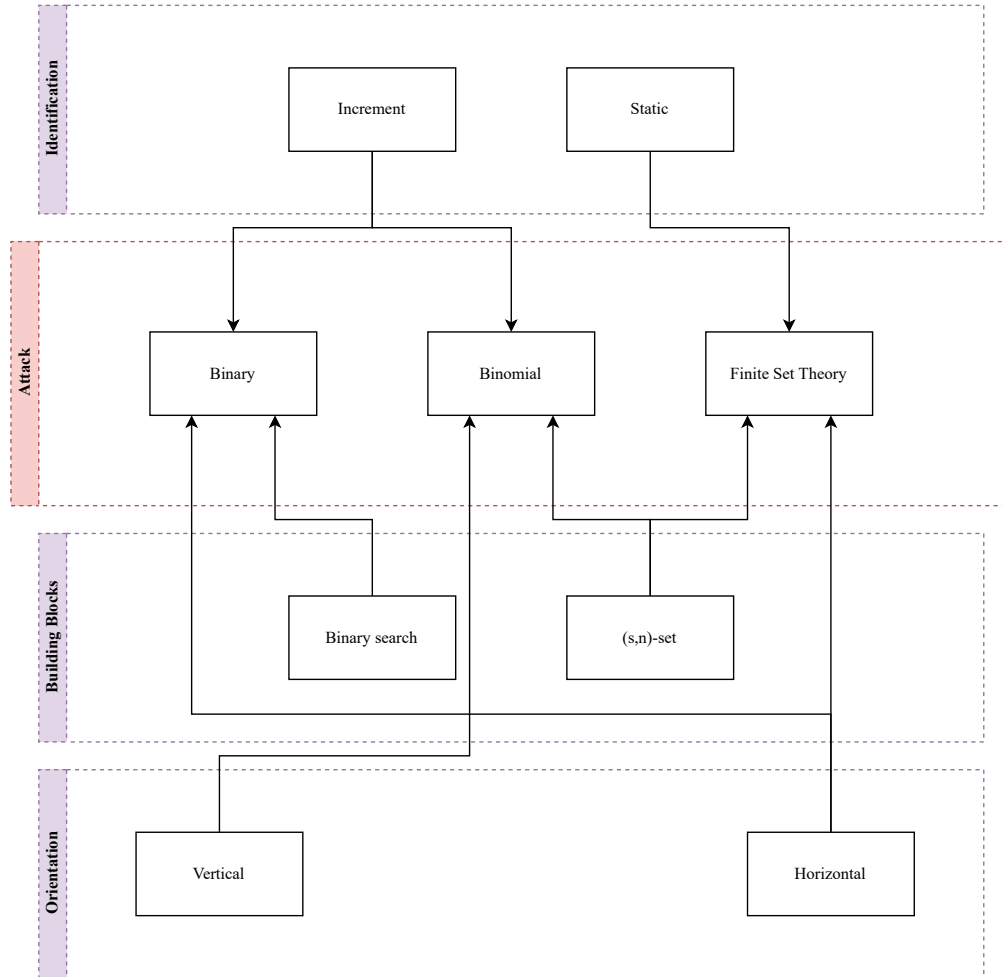


Figure 4.1: Technical framework of existing attacks.

If the technical framework is observed closely it can be found that, besides all attacks having overlapping qualities, every attack also has a unique quality. To make this more obvious/clear, the cells of Fig. 4.1 are re-organized in Fig. 4.2. The attributes of the Binomial-attack will be revealed through the investigation of the research questions.

4.3 Research Questions

This section presents findings on how injected files should be constructed to achieve optimal keyword identifiability in an SE-scheme that leaks its file access pattern. To address this

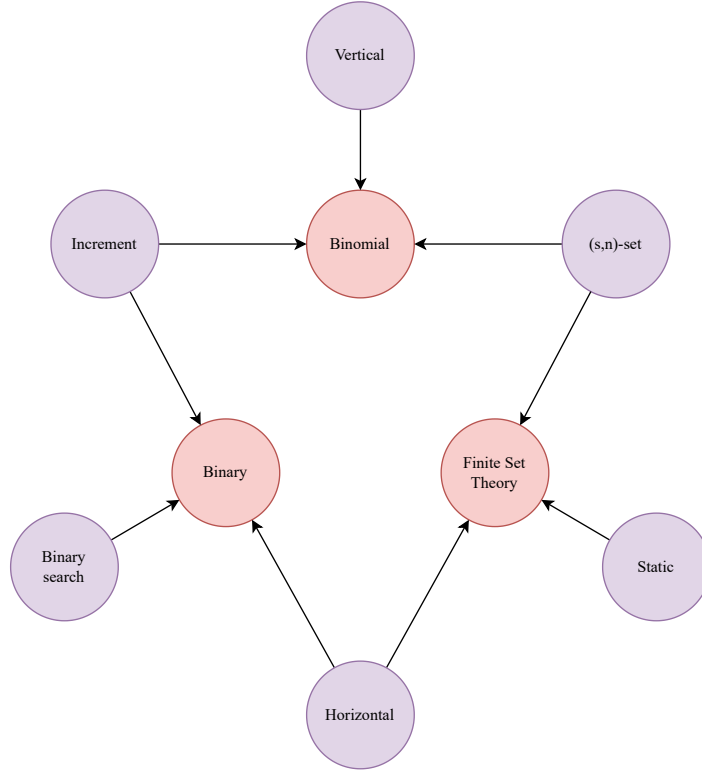


Figure 4.2: Re-organized technical framework for clear quality dispersion.

issue, three sub-research questions are answered first. The synthesis of these sub-questions will provide the answer to our main research question.

4.3.1 $RQ_{1.1}$: (s, n) -set Construction

Is there a more optimal method for constructing injection files compared to using (s, n) -sets?

A complete uniform (s, n) -set generates all possible combinations among the used files with a frequency of $r = n - s + 1$, ensuring an optimal approach for keyword identification. However, the construction of these (s, n) -sets still allows for significant improvement. Specifically, the (s, n) -set can only be utilized in its entirety; no subsets can be made due to the set's structure within the files. This issue is clearly demonstrated in Tables 3.5 and 3.7. If the (s, n) -sets are interrupted at any column, the number of identifiable keywords decreases significantly more than the proportion of columns lost.

To address this issue and enhance the flexibility of working with (s, n) -sets, we introduce Positional Patterns (P.P.) and construct the files based on these patterns. Both the original and new methods result in complete sets containing the same number of identifi-

able keywords. However, the (s, n) -set based on P.P. can identify the maximum number of keywords, even when only half of the set is utilized in the files.

New construction method of (s, n) -sets

To establish the integrity of our new (s, n) -set construct, we initially define a positional pattern as a distinct relative arrangement of files used to identify a keyword. Each positional pattern offers up to n variations before repeating itself. If all variants of a positional pattern are employed, they occupy at most $n \cdot r$ positions across r columns.

Consider the simplest positional pattern where files are consecutive. This pattern generates variants as illustrated in Table 4.1.

Files	P.P.	Variants							Injected structure		
		v_1	v_2	v_3	v_4	v_5	v_6	v_7	col_1	col_2	col_3
F_1	x	k_1					k_6	k_7	k_1	k_7	k_6
F_2	x	k_1	k_2					k_7	k_1	k_7	k_2
F_3	x	k_1	k_2	k_3					k_1	k_3	k_2
F_4			k_2	k_3	k_4				k_4	k_3	k_2
F_5				k_3	k_4	k_5			k_4	k_3	k_5
F_6					k_4	k_5	k_6		k_4	k_6	k_5
F_7						k_5	k_6	k_7	k_7	k_6	k_5

Table 4.1: An example of all the variants possible from a positional pattern (P.P.) for $n = 7$ and $r = 3$, and how they would be injected.

Given that each positional pattern occupies r columns (with exceptions noted for trivial edge cases, discussed later), patterns can be inserted into the files until the threshold restricts usage to fewer than r columns. For the remaining columns, we consistently employ positional pattern (P.P. 1) from Table 4.1. Preceding columns in front of these columns can be filled with complete positional patterns, utilizing all variants of the pattern across r columns. See Table 4.2 for a visual clarification.

Proof

For our proof, we start by assuming $T = x_i \cdot r + 1$ and generalize from there. In the case of $T = x_i \cdot r + 1$, one column is left for the new positional pattern. In this column, there are three possible scenarios, where n_left stands for the number of free spots in the column available for keyword combinations:

1. $n_left > r$: inject the variant starting from on the first available spot.

Files		P.P. 3			P.P. 4			P.P. 5			P.P. 1			
		<i>col</i> ₇	<i>col</i> ₈	<i>col</i> ₉	<i>col</i> ₁₀	<i>col</i> ₁₁	<i>col</i> ₁₂	<i>col</i> ₁₃	<i>col</i> ₁₄	<i>col</i> ₁₅	<i>col</i> ₁₆	<i>col</i> ₁₇	<i>col</i> ₁₈	
<i>F</i> ₁	...	<i>k</i> ₁₅	<i>k</i> ₁₈	<i>k</i> ₂₁	<i>k</i> ₂₂	<i>k</i> ₂₅	<i>k</i> ₂₈	<i>k</i> ₂₉	<i>k</i> ₃₁	<i>k</i> ₃₅	<i>k</i> ₁	<i>k</i> ₇	<i>k</i> ₆	...
<i>F</i> ₂	...	<i>k</i> ₁₅	<i>k</i> ₁₆	<i>k</i> ₁₉	<i>k</i> ₂₂	<i>k</i> ₂₃	<i>k</i> ₂₆	<i>k</i> ₂₉	<i>k</i> ₃₀	<i>k</i> ₃₂	<i>k</i> ₁	<i>k</i> ₇	<i>k</i> ₂	...
<i>F</i> ₃	...	<i>k</i> ₁₆	<i>k</i> ₁₇	<i>k</i> ₂₀	<i>k</i> ₂₃	<i>k</i> ₂₄	<i>k</i> ₂₇	<i>k</i> ₃₀	<i>k</i> ₃₁	<i>k</i> ₃₃	<i>k</i> ₁	<i>k</i> ₃	<i>k</i> ₂	...
<i>F</i> ₄	...	<i>k</i> ₁₇	<i>k</i> ₁₈	<i>k</i> ₂₁	<i>k</i> ₂₄	<i>k</i> ₂₅	<i>k</i> ₂₈	<i>k</i> ₃₁	<i>k</i> ₃₂	<i>k</i> ₃₄	<i>k</i> ₄	<i>k</i> ₃	<i>k</i> ₂	...
<i>F</i> ₅	...	<i>k</i> ₁₅	<i>k</i> ₁₈	<i>k</i> ₁₉	<i>k</i> ₂₂	<i>k</i> ₂₅	<i>k</i> ₂₆	<i>k</i> ₃₂	<i>k</i> ₃₃	<i>k</i> ₃₅	<i>k</i> ₄	<i>k</i> ₃	<i>k</i> ₅	...
<i>F</i> ₆	...	<i>k</i> ₁₆	<i>k</i> ₁₉	<i>k</i> ₂₀	<i>k</i> ₂₃	<i>k</i> ₂₆	<i>k</i> ₂₇	<i>k</i> ₂₉	<i>k</i> ₃₃	<i>k</i> ₃₄	<i>k</i> ₄	<i>k</i> ₆	<i>k</i> ₅	...
<i>F</i> ₇	...	<i>k</i> ₁₇	<i>k</i> ₂₀	<i>k</i> ₂₁	<i>k</i> ₂₄	<i>k</i> ₂₇	<i>k</i> ₂₈	<i>k</i> ₃₀	<i>k</i> ₃₄	<i>k</i> ₃₅	<i>k</i> ₇	<i>k</i> ₆	<i>k</i> ₅	...

$T =$	x_1r	x_2r	x_3r	x_4r	$x_4r + 1$	$x_4r + 2$	x_5r
-------	--------	--------	--------	--------	------------	------------	--------

Table 4.2: An example of the construct of the injected files for $x_4r \leq T \leq x_5r$, $n = 7$ and $r = 3$. Where up till the last positional pattern (P.P.) any positional patterns can be used to make unique combinations and the last positional pattern consists out of consecutive files (P.P. 1). If $T = x_i r$, the threshold ends precisely in between positional patterns. If $x_i r < T < x_{i+1} r$, P.P. 1 should be used from $x_i r$ onward. All columns in P.P. 1 can identify $\frac{n}{r}$ keywords.

2. $n_left < r$: inject the variant starting from on the first available spot and continue at the beginning of the next column.
3. $n_left = r$: n/r is a an integer, inject the last variant to fully occupy the column.

The first scenario will ultimately progress to either the second or third scenario, both of which are illustrated in Table 4.3. In both scenarios, the column will eventually contain $\frac{n}{r}$ combinations. Let's now consider $T = x_i \cdot r + 2$, following from the previous scenarios which concluded with either scenario 2 or 3.

- In the case of scenario 2: n and r are relatively prime. The variants will not repeat themselves for $n \cdot r$ spaces, equivalent to $\frac{n \cdot r}{n} = r$ columns. This means that variants will not repeat until $T = x_i \cdot r + r = x_{i+1} \cdot r$.
- In the case of scenario 3: $\frac{n}{r}$ is an integer, which makes n/r variants of the positional pattern. Repeating the same steps from the $T = x_i \cdot r + 1$ scenario, starting from a different unused position point, will not lead to repetition until all variants are utilized, totaling n variants. Together, these variants occupy $n \cdot r$ spaces, equivalent to $\frac{n \cdot r}{n} = r$ columns.

Repeating this for the next T will eventually lead to $T = x_i \cdot r + r$, after which the reasoning can be started from $T = x_{i+1} \cdot r + 1$ again.

P.P.1						P.P.1					
Files		col ₁₆	col ₁₇	col ₁₈		Files		col ₁₆	col ₁₇	col ₁₈	
F_1	...	k_1	k_7	k_6	...	F_1	...	k_1	k_5	k_6	...
F_2	...	k_1	k_7	k_2	...	F_2	...	k_1	k_2	k_6	...
F_3	...	k_1	k_3	k_2	...	F_3	...	k_1	k_2	k_3	...
F_4	...	k_4	k_3	k_2	...	F_4	...	k_4	k_2	k_3	...
F_5	...	k_4	k_3	k_5	...	F_5	...	k_4	k_5	k_3	...
F_6	...	k_4	k_6	k_5	...	F_6	...	k_4	k_5	k_6	...
F_7	...	k_7	k_6	k_5	...						
		↓	↓	↓	↓			↓	↓	↓	↓
$T =$		x_4r		$x_4r + 2$		$T =$		x_4r		$x_4r + 2$	
			$x_4r + 1$		x_5r				$x_4r + 1$		x_5r
(a) Second scenario						(b) Third scenario					

Table 4.3: Construction of the last $T \pmod r$ columns of an (s, n) -set under the two different possible scenarios, guaranteeing $\frac{n}{r}$ identifiable keywords per column.

This concludes that for any T , the described positional pattern P.P.1 can consistently identify $\frac{n}{r}$ keywords within a single column. By consistently using this positional pattern in the last $T \pmod r$ columns, we guarantee the identification of $\frac{n}{r}$ keywords per column.

edge case

if n and r are co-prime ($\gcd(r, n) = 1$) all positional patterns have n variants and take r columns. If $\gcd(r, n) = cp$, where $cp > 1$, there is at least 1 positional pattern that has less than n variants. namely $v = n/cp$ variants. If $cp/2$ is an integer, there is also a positional variant which has $v = n/(cp/2)$ variants, and so on. These take $\frac{v \cdot r}{n}$ columns per pattern. All other positional patterns have n variants and hence r columns. Note $\frac{v \cdot r}{n}$ still holds for $cp = 1$. The fact that there can be one or two positional patterns with less than r columns does not pose any trouble as P.P. 1 is always used as last positional pattern. Any positional pattern can be used for the construct of the files as long as the last pattern is the P.P. 1 pattern, to guarantee $\frac{n}{r}$ keywords per column.

Conclusion

To conclude, with the new construct of the (s, n) -set, every column can identify $\binom{n}{r}$ keywords. Meaning, if x columns of the (s, n) -set are to be used, $\lfloor x \cdot \binom{n}{r} \rfloor$ keywords can be identified. On the contrary to the old construction method, where most of the keywords will be lost due to its wide spread placement of its keyword identifiers.

4.3.2 $RQ_{1.2}$: File Interrelation

How can the interrelatedness of injected files be enhanced beyond the current methods?

Both the Binary- and FST-attack combine file sets sequentially to achieve the desired number of identifiable keywords. This stacking approach results in files from different sets lacking matching keywords between them, leading to a loss of possible combinations and an increased total number of injected files required. The re-implementation of the Binary- and FST-attack on the Enron dataset [8] demonstrates how these attacks stack their sets in the injected files, as visualized in Tables 4.4 and 4.5. This stacking is necessary since traditional (s, n) -sets cannot exceed the threshold and must be fully utilized.

However, with the new construction method introduced in Section 4.3.1, this limitation is overcome. The (s, n) -set can now be terminated at any desired column without losing keywords from the remaining portion of the (s, n) -set. This enhancement ensures no combination loss between injected files, thereby reducing the need for additional files. Table 4.6 illustrates how the construction of injected files is now vertical rather than horizontal, utilizing all files collectively. The parameter n in the (s, n) -set now consistently represents the total number of files to be injected.

Files	<i>Col</i> ₁	<i>Col</i> ₂	...	<i>Col</i> ₆₅	<i>Col</i> ₆₆	...	<i>Col</i> ₉₉	<i>Col</i> ₁₀₀	...	<i>Col</i> ₂₀₀
<i>F</i> ₁	<i>k</i> ₁	<i>k</i> ₂	...	<i>k</i> ₆₅	<i>k</i> ₆₆	...	<i>k</i> ₉₉	<i>k</i> ₁₀₀	...	<i>k</i> ₂₀₀
...
<i>F</i> ₂₆	<i>k</i> ₅₀₀₁	<i>k</i> ₅₀₀₂	...							
<i>F</i> ₂₇	<i>k</i> ₁	<i>k</i> ₂	...	<i>k</i> ₂₅₇	<i>k</i> ₂₅₈	...	<i>k</i> ₃₈₉			
...			
<i>F</i> ₃₄	<i>k</i> ₁	<i>k</i> ₃	...	<i>k</i> ₂₅₇	<i>k</i> ₂₅₉	...				
...			
<i>F</i> ₁₁₅	<i>k</i> ₄₂₇₃	<i>k</i> ₄₂₇₄	...	<i>k</i> ₄₅₂₉	<i>k</i> ₄₅₃₀	...	<i>k</i> ₄₆₆₁			
...			
<i>F</i> ₁₂₂	<i>k</i> ₄₂₇₃	<i>k</i> ₄₂₇₅	...	<i>k</i> ₄₅₂₉	<i>k</i> ₄₅₃₁	...				
<i>F</i> ₁₂₃	<i>k</i> ₄₆₆₂	<i>k</i> ₄₆₆₃	...	<i>k</i> ₄₉₁₈	<i>k</i> ₄₉₁₉	...	<i>k</i> ₅₀₅₀			
...			
<i>F</i> ₁₃₀	<i>k</i> ₄₆₆₂	<i>k</i> ₄₆₆₄	...	<i>k</i> ₄₉₁₈	<i>k</i> ₄₉₂₀	...				

Table 4.4: The binary search structure of the injected files in the Binary-attack for 5 050 keywords, with a threshold of 200.

Files	(19, 21)-sets & (18, 20)-set									
	Col_1	Col_2	\dots	Col_{65}	Col_{66}	\dots	Col_{190}	Col_{191}	\dots	Col_{200}
F_1	k_{1141}	k_{1142}	\dots	k_{1205}	k_{1206}	\dots	k_{1330}			
\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots			
F_{21}	k_1	k_2	\dots	k_{65}	k_{66}	\dots	k_{190}			
F_{22}	k_{2471}	k_{2472}	\dots	k_{2535}	k_{2536}	\dots	k_{2660}			
\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots			
F_{42}	k_{1331}	k_{1332}	\dots	k_{1395}	k_{1396}	\dots	k_{1520}			
F_{43}	k_{3801}	k_{3802}	\dots	k_{3865}	k_{3866}	\dots	k_{3990}			
\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots			
F_{63}	k_{2661}	k_{2662}	\dots	k_{2725}	k_{2726}	\dots	k_{2850}			
F_{64}	k_{4960}	k_{4961}	\dots	k_{5024}	k_{5025}	\dots				
\dots	\dots	\dots	\dots	\dots	\dots	\dots				
F_{83}	k_{3991}	k_{3992}	\dots	k_{4055}	k_{4056}	\dots				

Table 4.5: The (s, n) -sets structure of the injected files in the FST-attack for 5 050 keywords, with a threshold of 200.

(74,76)-set										
Files	Col ₁	Col ₂	...	Col ₆₅	Col ₆₆	...	Col ₁₉₀	Col ₁₉₁	...	Col ₂₀₀
F_1	k_1	k_{74}	...	k_{1649}	k_{1672}	...	k_{4789}	k_{4799}	...	k_{5042}
...
F_{21}	k_{18}	k_{20}	...	k_{1617}	k_{1669}	...	k_{4808}	k_{4809}	...	k_{5049}
F_{22}	k_{19}	k_{21}	...	k_{1618}	k_{1670}	...	k_{4809}	k_{4829}	...	k_{5049}
...
F_{42}	k_{39}	k_{41}	...	k_{1615}	k_{1638}	...	k_{4829}	k_{4830}	...	
F_{43}	k_{40}	k_{42}	...	k_{1616}	k_{1639}	...	k_{4830}	k_{4831}	...	
...	
...	
F_{76}	k_{73}	k_{75}	...	k_{1671}	k_{1672}	...	k_{4798}	k_{4863}	...	
F_{77}										
...										
F_{83}										

Table 4.6: The updated (s, n) -set structure of the injected files in the for 5 050 keywords, with a threshold of 200.

Conclusion

Utilizing the newly defined construction method of the (s, n) -set ($RQ_{1.1}$), the (s, n) -sets can be merged into a single comprehensive (s, n) -set. This integration enhances the potential for combinations and optimizes the utilization of available columns. Unlike the horizontal buildup typical of the Binary- and FST-attack methods, a vertical approach demonstrates efficiency by requiring fewer files to achieve the same number of keywords.

4.3.3 $RQ_{1.3}$: Space Efficiency

How can maximum keyword turnover per space be achieved across all injected files?

Keyword identification logically takes the least space when it is represented by as few files as possible, the minimum being one. However, such scenarios offer limited unique combinations since each file can only be used once. Generally speaking, within the bounds of realistic datasets, a higher frequency for keyword identification allows for more combinations in the files. For instance, a $(60, 65)$ -set can generate more combinations than a

(63, 65)-set, despite the latter requiring less space per keyword. As seen in $RQ_{1.2}$, a vertical construction of the (s, n) -set in the files yields the most possible combinations, compared to other (s, n) -sets with the same frequency.

To optimize turnover per file space and enhance interrelation between files, we introduce our own incremental $[r, n]$ -set. This set comprises multiple (s, n) -sets, constructed using the updated method defined in $RQ_{1.1}$. See Table 4.7 for a visual representation.

Files	Increment $[3, 65]$ -set									
	(65, 65)-set	(64, 65)-set			(63, 65)-set					
	Col_1	Col_2	\dots	Col_{65}	Col_{66}	\dots	Col_{190}	Col_{191}	\dots	Col_{200}
F_1	k_1	k_{66}	\dots	k_{2115}	k_{2146}	\dots	k_{4811}	k_{4832}	\dots	k_{5049}
\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots
F_{21}	k_{21}	k_{84}	\dots	k_{2135}	k_{2163}	\dots	k_{4830}	k_{4831}	\dots	
F_{22}	k_{22}	k_{85}	\dots	k_{2136}	k_{2164}	\dots	k_{4831}	k_{4832}	\dots	
\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots	
F_{42}	k_{42}	k_{105}	\dots	k_{2091}	k_{2184}	\dots	k_{4851}	k_{4852}	\dots	
F_{43}	k_{43}	k_{106}	\dots	k_{2092}	k_{2185}	\dots	k_{4852}	k_{4853}	\dots	
\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots	\dots	
F_{65}	k_{65}	k_{128}	\dots	k_{2114}	k_{2207}	\dots	k_{4831}	k_{4937}	\dots	
F_{66}										
\dots										
F_{83}										

Table 4.7: The increment $[r, n]$ -set structure of the injected files for 5 050 keywords, with a threshold of 200.

Increment $[r, n]$ -Set

The main idea of the increment $[r, n]$ -set is to optimize the available space in the injected files, which are defined as follows.

Definition 3 (Increment $[r, n]$ -set) Let A be a set, then the subsets

$A_1, A_2, \dots, A_n \subset A$ are called an increment $[r, n]$ -set of A if the following conditions are satisfied:

- $|A_1| = |A_2| = \dots = |A_{n-r+1}|;$

- Elements in A_1, A_2, \dots, A_n are separated into r blocks such that the i -th ($1 \leq i \leq r$) block of A_1, A_2, \dots, A_n forms a uniform $(n - i + 1, n)$ -set of the union set of the i -th block of A_1, A_2, \dots, A_n .

Recall that for a uniform (s, n) -set, a keyword can be uniquely recovered by $n - s + 1$ returned files. Therefore, the keywords in the i -th block of an increment $[r, n]$ -set are determined by $n - (n - i + 1) + 1 = i$ files, since the i -th block is a uniform $(n - i + 1, n)$ -set by definition. Therefore, the keywords in the first block can be represented by one file, the keywords in the second block by two files, and so forth. This concept is referred to as an *increment*.

We denote the i -th block of A_j by A_j^i for $1 \leq j \leq n$, then we get the following corollary which follows from Lemma 1.

Corollary 1 *If (A_1, A_2, \dots, A_n) is an increment $[r, n]$ -set of A , then we have*

$$|A_j^i| = \binom{n-1}{n-i},$$

for $1 \leq i \leq r, 1 \leq j \leq n$.

Therefore, we know that the size of A_j is $|A_j| = \sum_{i=1}^r \binom{n-1}{n-i}$ for $1 \leq j \leq n$. The main idea of our basic file-injection attack is to construct an increment $[r, n]$ -set, instead of complete independent (s, n) -sets spread over different chunks of files, like FST does. We are aiming at reducing the total number of injected files n to as few files as possible. Keywords are recovered according to the different combinations of returned files (details are present in Section [4.3.3]).

Example 6 *We give an example of an increment $[r, n]$ -set of the keyword set $\{k_1, k_2, \dots, k_{23}\}$ with threshold seven for a comparison to the example in Table 3.7. We compute r and the minimum n such that*

$$\begin{cases} \sum_{i=1}^r \binom{n-1}{n-i} \leq 7 \\ \sum_{i=1}^r \binom{n}{n-i} \geq 23, \end{cases} \quad (4.1)$$

and then we get $r = 3$ and $n = 6$. The increment $[3, 6]$ -set of the aimed keyword set is shown in Table 4.8. Compared to Example 4, it reduces the number of injected files from 8 to 6! Every space in these files is filled with keywords, while still controlling the total number of keywords within the threshold. The construct of the increment $[3, 6]$ -set in Table 4.8 only has a uniform $(6, 6)$ - and $(5, 6)$ -set. The $(4, 6)$ -set is disrupted by the threshold. In the next second we show how a ratio of an (s, n) -set can identify the same ratio of keywords, compared to a complete (s, n) -set.

A trick to find the optimal n to Eq. 4.1 is to try r for values in $\{1, 2, \dots\}$ in a row, and a general technique to construct an increment $[r, n]$ -set for m keywords with threshold T is presented in Sect. 4.3.3.

Files	(6,6)-set	(5,6)-set					(4,6)-set
	Col ₁	Col ₂	Col ₃	Col ₄	Col ₅	Col ₆	Col ₇
F_1	k_1	k_7	k_{10}	k_{13}	k_{15}	k_{19}	k_{22}
F_2	k_2	k_7	k_{11}	k_{14}	k_{16}	k_{20}	k_{22}
F_3	k_3	k_8	k_{11}	k_{13}	k_{17}	k_{21}	k_{22}
F_4	k_4	k_8	k_{12}	k_{14}	k_{18}	k_{19}	k_{23}
F_5	k_5	k_9	k_{12}	k_{15}	k_{17}	k_{20}	k_{23}
F_6	k_6	k_9	k_{10}	k_{16}	k_{18}	k_{21}	k_{23}

Table 4.8: An example of recovering 23 keywords with threshold $T=7$ by an increment $[3,6]$ -set, which is divided into 3 blocks. Keywords in the 1st, 2nd, and 3rd block can be recovered by 1, 2, and 3 returned files, respectively.

Construction of Increment $[r,n]$ -Set

In this section, we present a way to the construction of increment $[r,n]$ -set of a finite set, which uses the method of constructing uniform (s,n) -set proposed in Sect. 4.3.1 as a sub-routine.

Given as input the size of the keyword m and threshold of the number of keywords in a file T , we aim to construct an increment $[r,n]$ -set of the keyword set with the minimum n such that (1) the size of each file should not be greater than the threshold T , and (2) the maximal number of keywords that those files can recover is at least m . To maximize the recovery ability under condition (1), our overall idea is to construct r uniform $(n-i+1, n)$ -sets by the technique shown in Sect. 3.3 for $1 \leq i \leq r$ and return the first T columns as the aimed set. Then by Lemma 1, we know the first $r-1$ blocks take $\sum_{i=1}^{r-1} \binom{n-1}{n-i}$ columns and can recover $\sum_{i=1}^{r-1} \binom{n}{n-i}$ keywords in total. The last block takes the rest $T - \sum_{i=1}^{r-1} \binom{n-1}{n-i}$ columns and allows to recover $\lfloor n/r \cdot [T - \sum_{i=1}^{r-1} \binom{n-1}{n-i}] \rfloor$ keywords. Then the condition (2) is equal to

$$\sum_{i=1}^{r-1} \binom{n}{n-i} + \left\lfloor \frac{n}{r} \cdot \left[T - \sum_{i=1}^{r-1} \binom{n-1}{n-i} \right] \right\rfloor \geq m. \quad (4.2)$$

We proceed in the discussion of r starting from 1 to T . For each r , we record all the possible n to the Inequality 4.2, with the minimum one as the optimal solution. For simplicity of exposition, we denote $NK(r,n)$ as the left part of the above inequality. The whole process of constructing an increment $[r,n]$ -set is present in Algorithm 3.

Going back to Example 4, we compute the Inequality 4.2 to get $candidate = [(2,7), (3,6), (4,7)]$. Then we know the optimal increment $[r,n]$ -set is $r=3$, and $n=6$.

Algorithm 3 Construction of increment $[r, n]$ -set

Input: Number of keywords m , threshold T

Output: An increment $[r, n]$ -set of the keyword set $\{k_1, k_2, \dots, k_m\}$

- 1: Initialize an empty candidate set: $candidate \leftarrow []$
 - 2: **for** $r = 1$ **to** T **do**
 - 3: Solve n from $NK(r, n) \geq T$ and denote the minimum n as n_0
 - 4: Append (r, n_0) to $candidate$
 - 5: $r = r + 1$
 - 6: **end for**
 - 7: Find (r, n_0) with the minimum n_0 and corresponding r from $candidate$
 - 8: **for** $i = 1$ **to** r **do**
 - 9: Construct a uniform $(n_0 - i + 1, n_0)$ -set of keywords with index from $\sum_{j=1}^{i-1} \binom{n_0-1}{n_0-j}$ to $\sum_{j=1}^i \binom{n_0-1}{n_0-j}$ by the technique proposed in Sect. 4.3.1
 - 10: **end for**
 - 11: **Output** the first T columns of the created files
-

Conclusion

The increment $[r, n]$ -set optimizes its space consumption by initiating keyword identification with a frequency of one and incrementally increasing this frequency once no additional combinations can be made. The FST-attack requires 83 injection files to identify the entire Enron dataset, whereas the increment $[r, n]$ -set requires only 65 injection files. This efficiency is achieved by constructing the (s, n) -sets vertically rather than horizontally ($RQ_{1.2}$), utilizing the new construction method for the (s, n) -sets ($RQ_{1.1}$).

4.3.4 RQ_1 : File-Injection Attack

How should injected files be constructed to optimise the number of identifiable keywords, given the presence of file access pattern leakage?

In this section, we present our new file-injection attack to searchable encryption schemes. It is based on our new definition of a subset family of a finite set, called *increment $[r, n]$ -set*. Our main technique is to construct an increment $[r, n]$ -set of the keyword set. Compared to the uniform (s, n) -set used in [21], the increment $[r, n]$ -set enables us to put more keywords in the injected files, thus significantly reducing the number of injected files. Our attack is based on binomial calculations to fit in as many keywords as possible. The attack can be seen as multiple (s, n) -sets next to each other, however the formatting of these (s, n) -sets are different compared to the ones from FST. It outperforms both the basic and advanced FST-attack at all times, in terms of number of injections needed for the same result.

Binomial-Attack

Given the keyword universe $\mathbb{K} = \{k_1, k_2, \dots, k_m\}$ and the threshold T as the maximal number of keywords in each file, we present our file injection attack in Algorithm 4, which is based on the increment $[r, n]$ -set of the \mathbb{K} .

Algorithm 4 Binomial-attack

Input: Keyword set $\mathbb{K} = \{k_1, k_2, \dots, k_m\}$, threshold T , a query token t

Output: Keyword corresponding to the token t

- 1: Generate an increment $[r, n]$ -set A_1, A_2, \dots, A_n of \mathbb{K} with threshold T by Algorithm 3
 - 2: **for** $j = 1$ **to** n **do**
 - 3: Let a file D_i contain the same keywords as A_i
 - 4: **end for**
 - 5: Inject files $\{F_1, F_2, \dots, F_n\}$ into the SE scheme
 - 6: **return** the corresponding keyword to t according to the returned i files ($1 \leq i \leq n$)
-

Now that the structure of the attack is understood, we can proceed to calculate the required number of injections to achieve the desired number of identifiable keywords. There are multiple formulas to calculate the required number of injections. The appropriate formula to utilize depends on at which $(n - r + 1, n)$ -set the threshold will limit the attack from injecting more combinations.

Deciding the Number of Injections

The attack always starts with $r = 1$ and progresses incrementally from there on forth. At some point within an $(n - r + 1, n)$ -set, the threshold will limit the number of keywords it can inject. Refer to Table 4.8 for a visual representation.

By Eq. 4.2 we know the number of keywords an $(n - r + 1, n)$ -set can utilize for a certain threshold, under a specific r . When the threshold is reached in the $(n - 1, n)$ -set, where $r = 2$ the equation can be written in term of n like the following:

$$F_2(K, T) = \frac{2K}{T + 1} \quad (4.3)$$

Similarly to the $(n - 2, n)$ -set, where $r = 3$, the equation becomes:

$$F_3(K, T) = \frac{-(3 + 2T) + \sqrt{(3 + 2T)^2 + 24K}}{2} \quad (4.4)$$

The formulas $F_4(K, T)$ and beyond are only of relevance when the threshold is a significant portion of the number of keywords that need to be injected.

Deciding the Injection Formulas

The next step involves determining the appropriate utilization of each formula for different scenarios.

To determine the appropriate value for r in the increment $[r, n]$ -set, we must assess whether the threshold allows for additional keywords in the files following a uniform $(n - r + 1, n)$ -set. This evaluation must be conducted for each r , commencing at $r = 2$. By Lemma 1 we know the first two blocks utilize a total of n columns. Therefore if $T > n$, the $(n - 2, n)$ -set can also be used. However, the value of n remains unknown at this stage.

To address this uncertainty, we substitute $n = T$ into F_2 . This yields the threshold at which both the $(n - r + 1, n)$ -sets in the increment $[2, n]$ -set become uniform and precisely meet the threshold. If the keyword universe exceeds this value, the attack will require more than T injections. Conversely, if the keyword universe falls below this value, fewer than T injections are required. Consequently, there will be residual space in the injected files for (at least) the $(n - 2, n)$ -set. The minimum value of K to only be able to build up to an Increment $[2, n]$ -set is outlined as follows:

$$Min_{F_2}(T) = \frac{1}{2}T^2 + \frac{1}{2}T \quad (4.5)$$

F_2 should be applied when the outcome of $Min_{F_2}(T) \leq K$. Alternatively, the formula of Min_{F_3} determines whether F_3 or F_4 should be utilized. Following the same procedures as before, we get:

$$Min_{F_3}(T) = \frac{2+T}{3} \cdot \frac{1+\sqrt{8T-7}}{2} + \frac{T-1}{3} \quad (4.6)$$

These formulas already hold an improvement over FST, since FST had a lookup table with overlapping values and no clear points to choose from. Using our previous example 4, we see $Min_{F_2}(7) > 23$ and $Min_{F_3}(7) < 23$. This means we need to use F_3 , which results in $F_3(23, 7) = 6$ files.

Chapter 5

Experiments and Results

This chapter will detail the setup and execution of the experiments, and demonstrate through their results how our attack is superior over previous file-injection attacks based on file access pattern leakage.

5.1 Experimental setup

The performance of the attack is stated by the number of injected files necessary to be able to identify every keyword in the dataset, based on the injected files returned in the response of a query. Logically, fewer files is always better as it costs less effort and most importantly, the chances for detection are slimmer.

To conduct the experiments we have first looked at previous studies and how they define their performance. In previous studies, the Enron dataset [8] served as a benchmark for attack performance. The Enron dataset has 30 109 files containing together 5 050 unique keywords. The average file contains 90 keywords. Only 3% of the files in the Enron body contain more than 200 keywords, making it a good choice for setting the threshold. The trade-off of setting a limit of $T=200$ keywords per file results in losing a few files from honest clients to protect them from server exposure, requiring the server to inject many more files to achieve the same results.

After covering the Enron dataset, the threshold remains constant at $T=200$, but experiments are conducted on datasets of varying sizes, containing up to 20 000 unique keywords. The significant performance difference becomes increasingly evident during this phase.

Following the investigation of different dataset sizes with a threshold of 200, various thresholds are implemented to examine the relationship between performance and threshold values. These experiments aim to demonstrate that the Binomial attack consistently outperforms other methods, irrespective of dataset size or threshold.

5.2 Performance on the Enron Dataset

Performing the attack on the Enron dataset results in the following steps:

1. $Min_{F_2}(200) = 20\,100$, which is greater than the keyword universe ($K=5\,050$), hence we have to try a higher Min_{F_x} .
2. $Min_{F_3}(200) = 2\,034$, which is smaller than the keyword universe, hence we will continue with $F_3(K, T)$
3. $F_3(5050, 200) = 65$

Hence, when applied to the Enron dataset, our attack requires only 65 files to cover all the keywords. In contrast, the Binary-attack necessitates a staggering 130 files, twice as many as our method, while the FST-attack requires 83 files to achieve the same coverage. Thus, in this real dataset scenario, the FST-attack requires 28% more injections than the Binomial-attack.

5.3 Performance under a Threshold of 200

When the threshold remains consistent, but the dataset varies, the Binomial-attack consistently outperforms the FST-attack with at least one injected file, regardless of the dataset size. See Fig. 5.1 for the performance differences. With a threshold of 200, the most substantial disparity occurs in datasets ranging from 7 200 to 7 400 keywords, where the FST-attack requires 33 more injections compared to the Binomial-attack. This represents a 38% increase in injections needed by the FST-attack for equivalent results. Furthermore, in scenarios where the dataset size exceeds 20 000 keywords, the FST-attack regresses to its initial performance level on top of where it was left, necessitating additional injections beyond those required at 20 000 keywords. This phenomenon will be visually evident in the upcoming section in Fig. 5.2.”

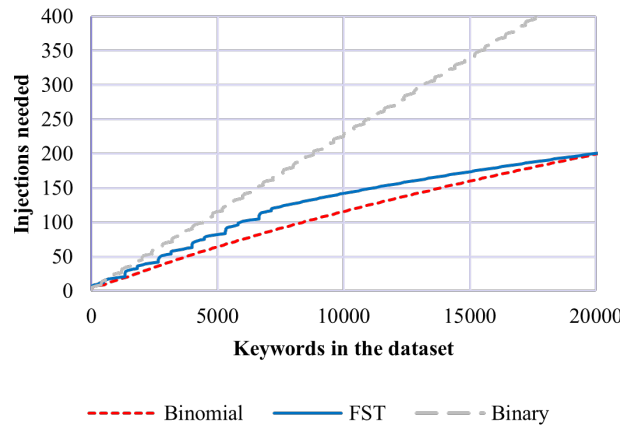


Figure 5.1: Performance of different injection attacks under a threshold of $T=200$.

5.4 Performance under Different Thresholds

To provide a comprehensive overview of the differences between the attacks, Fig. 5.2 illustrates the comparative performance of the Binary-, FST-, and Binomial-attack across the thresholds $T = \{100, 300\}$, with datasets ranging up to 20 000 keywords.

The results consistently demonstrate the superiority of the Binomial-attack over the FST-attack across various thresholds and dataset sizes.

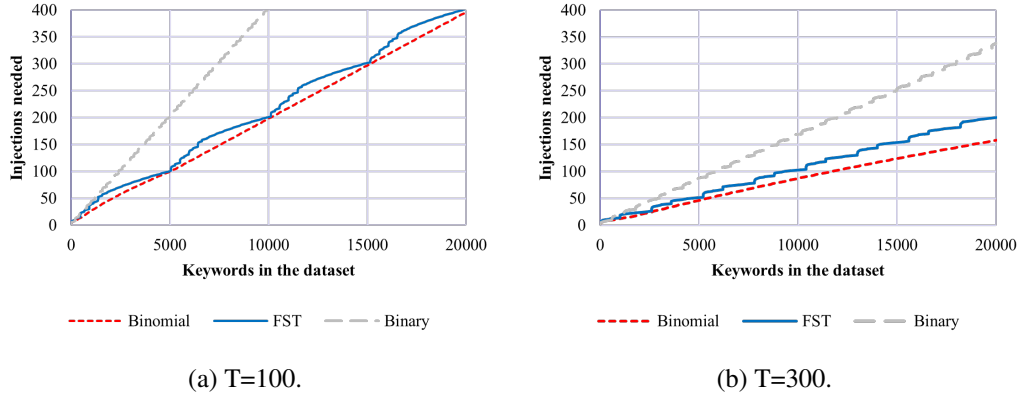


Figure 5.2: Performance of file-injection attacks under different thresholds.

Performance Difference Explanation, FST- and Binomial-attack

The difference in performance of the base attack follows from the different s values used for certain keyword universes. FST takes the s value for which it can make the most combinations, under the condition that all possible combinations are utilized. This is due to their construction method of the set, where not every column can identify $\frac{n}{r}$ keywords, hence they need the set to be complete of all combinations. This is a huge restriction. If more keywords can be identified by making subsets of the same keyword universe, the attack will do so, as long as the subsets all consist out of complete (s, n) -sets. An example from the FST-paper [21] (p. 8) is given where the Enron dataset (5050 keywords) under a threshold of 200, can be identified with a single (s, n) -set with $n = 101$ and $s = 100$. However, if subsets are made, multiple complete (s, n) -sets can be utilized which need less injections combined: four subsets with $n = 21$ and $s = 19$. The FST-attack does this for every keyword universe, using the best complete (s, n) -set combinations.

The Binomial-attack has proven that the (s, n) -sets do not require to be complete to still be able to identify a portion of the keywords, i.e. the ratio of the complete set being used can identify the same ratio of keywords, compared to what the complete (s, n) -set could identify. Since this is possible, it is always the best choice to start with an s as close to n as possible and continue with a lower s once all the combinations are made. This is the equivalent to start identifying a keyword with a single file and increasing the number of files a keyword is identified with once no more unique combinations are left to make.

These differences explain why there is such a difference in performance in these attacks. When they are far apart, The FST-attack uses multiple subsets with high frequencies to

identify keywords, which is not space efficient. The Binomial always uses subsets with as low as possible frequencies, where different subsets never have the same frequency. FST does have multiple subsets with the same frequency in a lot of occasions. Once the FST attacks starts straightening out, it is solely using the frequency of two files per keyword, however there are still multiple subsets. Once it gets to the closest point possible to the binomial attack it has exactly one complete (s, n) -set with a frequency of two files per keyword. Binomial still performs better in this scenario as it has also a subset of frequency one, which can identify more keywords in the same space as a frequency of two.

Chapter 6

Countermeasure & Mitigation

This chapter will cover the padding countermeasure of the SE-scheme. First the countermeasure is explained, after which the effects and its calculations are given. Once the consequences of this countermeasure are clear, a mitigation is proposed called the adopted Binomial-attack.

6.1 File-injection Attacks on SE Schemes with Keyword Padding

Keyword padding serves as a countermeasure within the SE scheme aimed at obscuring query results by returning more files than necessary. In addition to the files containing the queried keyword, the scheme also includes random files from the dataset in its response. In this section, we delve into the consequences of padding and compare the implications between the FST- and Binomial-attack methodologies. Previous studies, such as the FST- and Binary-attack, explored this topic assuming a file dataset of 30 109 files and a keyword universe of 5 050 keywords. The scheme adopts a threshold of 200, and on average, a query yields matches on 560 files, with an additional 60% of random files included (336 files). Section 6.1.1 delves into the quantitative effects of padding, while Section 6.1.2 presents a visual exploration of these effects.

6.1.1 Calculating the Effects

To assess the impact, three key steps are necessary. Firstly, we must determine the average number of additional injected files returned as a consequence of their selection for padding. Subsequently, we can proceed to determine the average number of keyword combinations we can generate. These combinations represent distinct file arrangements utilized for the unique identification of a single keyword, collectively referred to as the candidate set for a query. Finally, the last step entails re-executing the attack on the candidate set to pinpoint the specific keyword utilized.

Injected Files from Padding

To calculate the average number of injected files chosen during padding, we can utilize the hypergeometric distribution function. Our population size is $30\,109 - 560 = 29\,549$, since the matched files for the query can not be chosen for the padding. The number of successes will be $F3(5050, 200) = 64.8 \approx 65$ files, minus the average injected file response, leaves $65 - 3 = 62$ successes. The sample size is 336. We can calculate the probabilities for all possible numbers of successes in the sample and then multiply each probability by the corresponding number of successes. The results are then summed to determine the average number of injected files (p) chosen in the padding:

$$p = \sum_{n=1}^{62} \frac{\binom{62}{n} \binom{29549-62}{336-n}}{\binom{29549}{336}} \quad (6.1)$$

Average Candidate Set Size

The average candidate set size is determined by three key factors associated with each $(n - r + 1, n)$ -set used to identify keywords. The first factor considers the number of possible combinations within the given $(n - r + 1, n)$ -set when $r + p$ injected files are returned. The second factor accounts for the ratio of combinations utilized in that $(n - r + 1, n)$ -set compared to its total possible combinations. The third factor represents the ratio of identifiable keywords in the $(n - r + 1, n)$ -set to the total number of identifiable keywords. Multiplying these three factors together yields the average candidate set size per $(n - r + 1, n)$ -set. Summing the results across all $(n - r + 1, n)$ -sets provides the overall average candidate set size:

$$\sum_{r=1}^R \binom{r+p}{r} \cdot \frac{|K_r|^2}{\binom{n}{r} \cdot |K_R|} \quad (6.2)$$

Number of Extra Injections Needed

A straightforward method to determine the number of extra injections required is to analyze on a per-query basis. By considering the average candidate set size per query, we can execute our attack specifically for that particular candidate set to recover the searched keyword. While this approach is not optimal, it suffices for comparison purposes with the FST-attack.

6.1.2 Visualising the Effects

This section will demonstrate the effects of padding on both FST and the Binomial-attack. While the Binomial-attack may not always appear significantly better based solely on the average candidate set size per query, it's important to consider that FST is generally less efficient, requiring more injections to cover the same candidate set. Here, we present the results for the schemes with a threshold of $T = \{100, 200, 300\}$.

Targeting the Whole Dataset

In Fig. 6.1, we see the average sizes of candidate sets for different dataset sizes. The corresponding number of extra injections required for the candidate sets is illustrated in Fig. 6.2. While there is a small dataset size range where the Binomial-attack requires one more injection than the FST-attack, FST generally performs worse for all other dataset sizes.

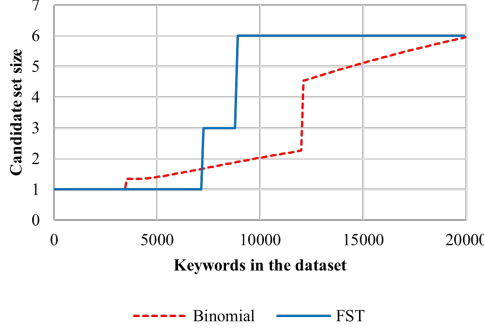


Figure 6.1: Candidate set size per query,
T=200.

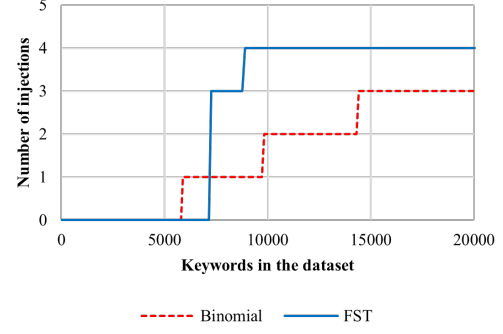


Figure 6.2: Extra injection size per query,
T=200.

Fig. 6.3 demonstrates that the candidate set for the Binomial-attack frequently exceeds that of the FST-attack. However, this discrepancy is mitigated by the fact that the Binomial-attack requires fewer injections to achieve the same candidate set size. Consequently, the Binomial-attack never requires more injections for these candidate sets than the FST-attack does for its own candidate set, as illustrated in Fig. 6.4.

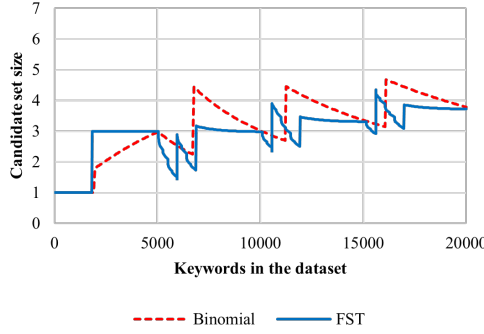


Figure 6.3: Candidate set size per query,
T=100.

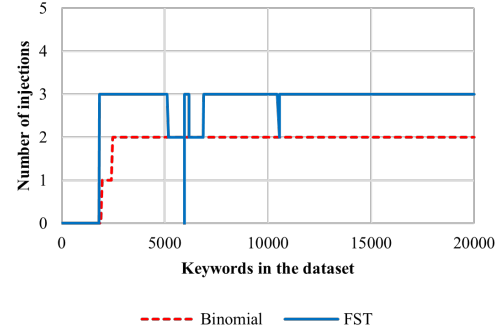


Figure 6.4: Extra injection size per query,
T=100.

With a threshold of 300, the Binomial-attack maintains an average candidate set slightly above 1, as shown in Fig. 6.5. However, this does not impact the average number of injections, as depicted in Fig. 6.6. The Binomial-attack only requires more additional injections on average between dataset sizes of 13,600 and 18,300. Beyond this range, the

FST-attack consistently requires more additional injections, whereas before this dataset size, both attacks require the same number of additional injections.

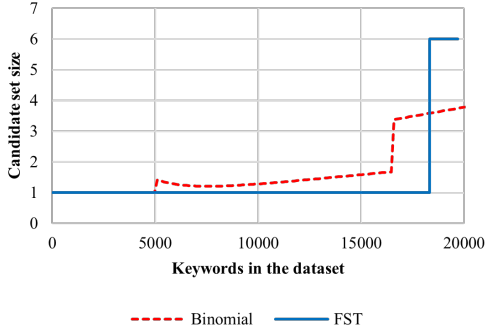


Figure 6.5: Candidate set size per query, T=300.

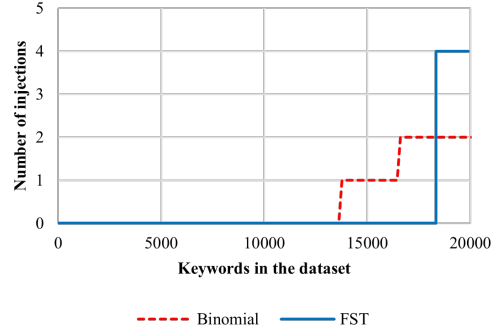


Figure 6.6: Extra injection size per query, T=300.

Divergent Behaviors Explained

When looking at the consequences of padding, the different differences can be explained with the frequencies the attack uses and the number of injections that are needed for the same results. The number of extra injected files that are returned due to the padding is similar or higher for the FST-attack compared to the Binomial-attack, as it needs more injections for the same result, hence the odds of injected files being chosen in the padding is greater for FST. However, since the FST-attack uses more files for the same results, this also means the spread of a keyword combination is greater compared to the spread of the Binomial-attack. Since there is a greater spread, the candidate set is slimmer when the number of injected files in the padding is the same for both the FST- and Binomial-attack. This spread can become so inefficient that (on average) an injected file in the padding will have no influence in the candidate size. As can be seen in clarification of Fig. 6.7, where the Binomial attack immediately peaks when the first extra injected file from the padding is introduced. The reason FST's line is always on an integer, is due to the fact it always uses a frequency of two for it's subsets once the line moves away from its candidate set size of one. The Binomial-attack has three different frequencies in it's construct at that point which gives an average of all the three frequencies. The reason FST's second vertical peak is before the Binomial attack's vertical peak is due to the fact the Binomial-attack can identify more keywords with less injections. The peaks are the moment the average injected files in the padding becomes two ($n = 134$), but the Binomial-attack can identify more keywords with that number of injections, hence it is further in the graph.

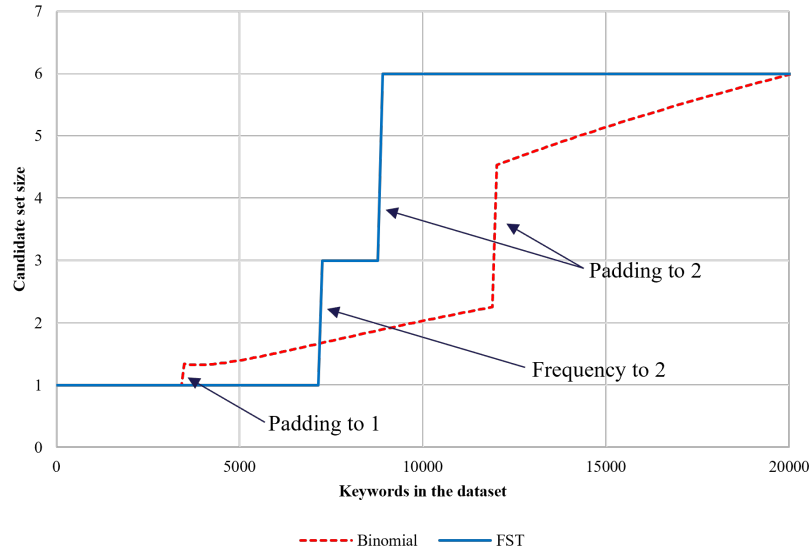


Figure 6.7: Candidate set sizes per query when padding is applied, $T=200$.

Targeting a Subset of the Dataset

When targeting a subset of the keyword universe, fewer injections are required to cover the target set, benefiting both attacks. However, not every query relates to a keyword in the target set. When combined with padding, this may not pose an issue if we assume a consistent average number of injected files in the padding. For instance, if two injected files are returned and the average padding injection is also two, it suggests a search for a keyword not in the target set. However, if a return of two injected files could also indicate a search for a keyword occurring once or twice, all searches become candidate sets. While these candidate sets may not contain actual keywords from the target set, distinguishing beforehand is impossible. The only option is to re-perform the attack on the candidate set.

In this scenario, our attack performs notably worse. This is because the Binomial-attack initiates with an (n, n) -set. FST does not follow this approach, resulting in fewer potential combinations when all preceding $(n - r + 1, n)$ -sets are included in the candidate set, see Fig. 6.8. Fig. 6.9 illustrates the number of extra injections required when searching for a keyword that is not in the target set.

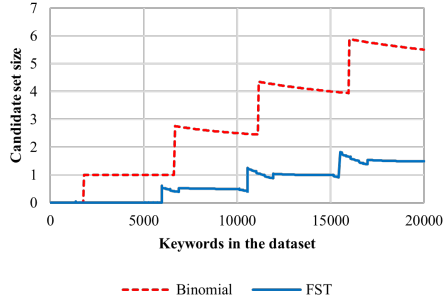
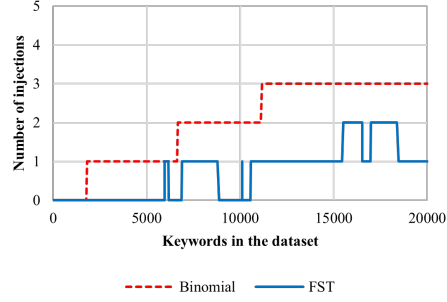
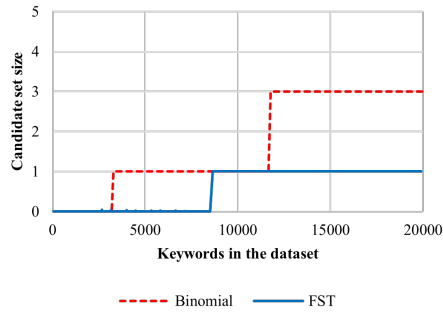
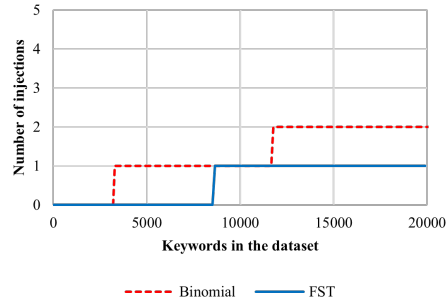
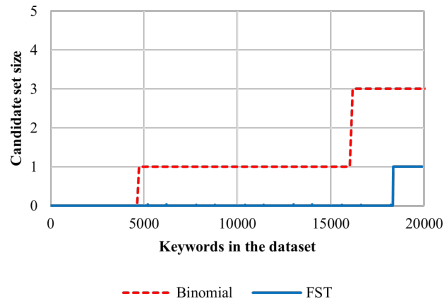
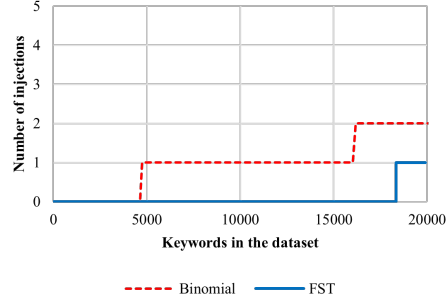

 (a) $T=100$.

 (a) $T=100$.

 (b) $T=200$.

 (b) $T=200$.

 (c) $T=300$.

 (c) $T=300$.

Figure 6.8: Candidate set sizes per query that is not in the target set when padding is applied, under different thresholds, where the target set is a subset of the full keyword universe, for the standard Binomial-attack.

Figure 6.9: Extra injection sizes per query that is not in the target set when padding is applied, under different thresholds, where the target set is a subset of the full keyword universe, for the standard Binomial-attack.

6.2 Adopted Binomial-Attack

When the target set is a subset of the dataset, searches for keywords outside the target set result in additional candidate sets. To mitigate the size of these extra candidate sets, adjustments to the attack methodology are necessary. This section outlines the modifications

required to minimize candidate size while maintaining effectiveness. Despite the trade-off, the attack consistently requires fewer initial injections than FST.

6.2.1 Removing the (n, n) -Set

In the Binomial-attack, the lowest value for r is always one. While this minimizes the space occupied in injected files, it also leads to greater overlap with keywords spread across multiple injected files. Conversely, higher values of r in the $(n - r + 1, n)$ -sets for all keywords result in smaller candidate sets per query. To reduce the size of candidate sets, keywords should not be identified with only one injected file, meaning the attack starts from $(n - 1, n)$ instead of (n, n) . This frees up space that can be allocated to a different $(n - r + 1, n)$ -set.

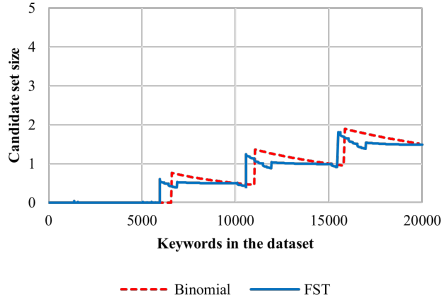
6.2.2 Results after the Mitigation

The number of identifiable keywords decreases by either $\frac{n}{2}$ or $\frac{2n}{3}$, depending on which $(n - r + 1, n)$ -set the attack terminates due to the threshold. Refer to Table 6.1 for a visual representation of this transformation.

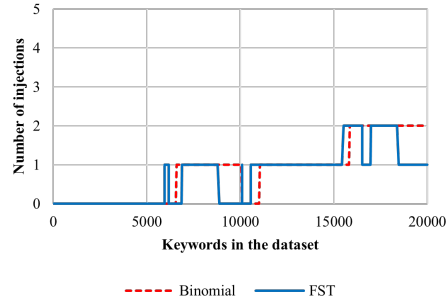
In Fig. 6.10 and 6.11, the difference in candidate sets and extra injections required between the FST- and adopted Binomial-attack are illustrated. Especially Fig. 6.10(a) shows a clear transformation from its previous performance in Fig. 6.8(a). Depending on the threshold, FST consistently requires an equal or greater number of injections to recover candidate sets.

Files	(5, 6)-set					(4, 6)-set	
	Col ₁	Col ₂	Col ₃	Col ₄	Col ₅	Col ₆	Col ₇
F_1	k_1	k_4	k_7	k_9	k_{13}	k_{16}	k_{19}
F_2	k_1	k_5	k_8	k_{10}	k_{14}	k_{16}	k_{18}
F_3	k_2	k_5	k_7	k_{11}	k_{15}	k_{16}	k_{18}
F_4	k_2	k_6	k_8	k_{12}	k_{13}	k_{17}	k_{18}
F_5	k_3	k_6	k_9	k_{11}	k_{14}	k_{17}	k_{19}
F_6	k_3	k_4	k_{10}	k_{12}	k_{15}	k_{17}	k_{19}

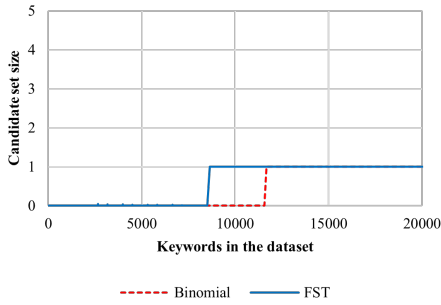
Table 6.1: Distribution of an Increment $[3, 6]$ -set, without $(6, 6)$ -set, $T=7$.



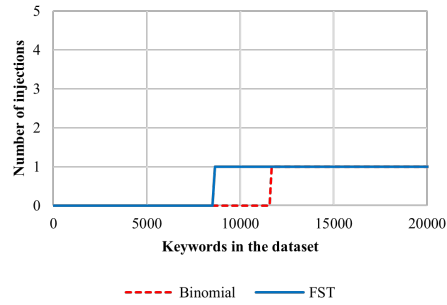
(a) $T=100$.



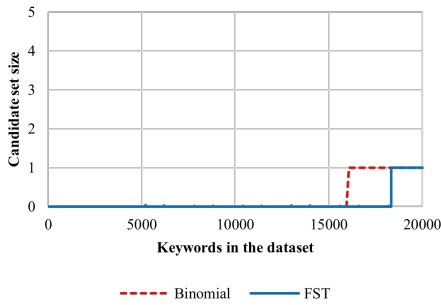
(a) $T=100$.



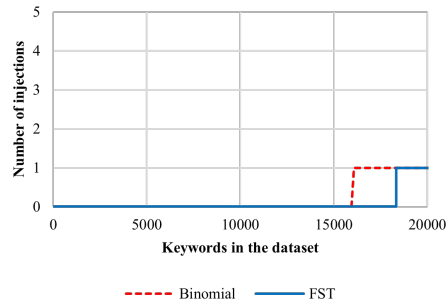
(b) $T=200$.



(b) $T=200$.



(c) $T=300$.



(c) $T=300$.

Figure 6.10: Candidate set sizes per query that is not in the target set when padding is applied, under different thresholds, where the target set is a subset of the full keyword universe, for the adopted Binomial-attack.

Figure 6.11: Extra injection sizes per query that is not in the target set when padding is applied, under different thresholds, where the target set is a subset of the full keyword universe, for the adopted Binomial-attack.

Chapter 7

Discussion

7.1 Discussion

The scenario regarding the probabilities of injected files being selected for padding purposes was adopted from previous studies. This decision was made to facilitate comparisons between the Binomial-attack and prior injection attack methodologies. Specifically, the scenario involved a dataset consisting of 30 109 files, 5 050 keywords, and an average of 560 hits per query. This scenario was scaled to encompass various sizes of keyword datasets, allowing for comparative analyses across different scales. It is worth noting that while the keyword dataset size was varied, other parameters remained unchanged. In real-world scenarios, adjustments to these parameters would be warranted, as changes in the keyword dataset size may significantly impact the number of hits per query. However, for the purpose of comparing our attack methodology against the FST, the scenarios were retained in their original form. These scenarios solely serve as a means to illustrate the effectiveness of our proposed attack in comparison to existing methodologies, particularly FST.

When considering the scenario where the entire dataset is targeted, it becomes necessary to possess knowledge of the entire keyword universe. While previous studies have made stronger assumptions than this [1], we regard the assumption of possessing knowledge of the entire keyword universe as unrealistic. However, a target set that constitutes a subset of the entire dataset is highly plausible. This target set does not necessarily need to be constructed from leaked data; rather, it can consist of self-assembled keywords that are deemed interesting by the attacker.

In addition to padding, there exist other countermeasures aimed at increasing the difficulty of attacks. One such countermeasure involves the creation of clusters of keywords, as described in [11]. When a search query is initiated for one of the keywords within a cluster, all files containing keywords from the same cluster are returned. This approach not only obscures the specific keyword being searched for, but also introduces ambiguity regarding the association of injected files with specific keywords. Due to the potential for multiple combinations of keywords within the returned files, the attacker may be compelled to employ higher $(n - r + 1, n)$ -sets, necessitating a greater number of injected files. It is important to note, however, that this countermeasure assumes a static keyword universe and

may require modification to accommodate dynamic searchable encryption scenarios.

Despite its theoretical appeal, searchable encryption has yet to achieve widespread adoption in practical applications and can vary significantly in its configurations, including the implementation of countermeasures. Consequently, predicting the exact characteristics of a searchable encryption scheme in practice remains challenging. Nevertheless, there is value in speculating on the potential implications of different settings and attempting to assess the scheme's security under various conditions, even if these scenarios remain largely theoretical at present. This makes it harder to determine how big the safety issues of the schemes are.

7.2 Future Work

The additional injections required to neutralize candidate sets are primarily utilized to compare the attack against FST. However, the method itself is far from optimal. As presented in this paper and the FST paper, each keyword necessitates multiple additional injections. This approach may result in a greater number of injections than initially required for the attack. A more efficient strategy involves combining candidate sets and reusing earlier injections, thereby reducing the overall number of additional injections required. However, the optimal method for achieving this remains to be determined.

This attack is an active attack that makes no use of leakage apart from the returned injected files. In contrast, other attacks combine active and passive methods [22]. If Binary- or FST-attack methods are employed, they could be enhanced by incorporating the Binomial-attack. Revisiting these attacks may reveal potential improvements.

Chapter 8

Conclusion

8.1 Conclusion

In conclusion, our research has introduced an innovative approach to SE attacks by combining advanced techniques from existing literature with novel methodologies developed during this study. Previous techniques have demonstrated sub-optimal performance, primarily due to their lack of optimization in file injection strategies. These methods fail to establish optimal correlations between injected files and overly focus on either maintaining a consistent number of files for keyword identification or rigidly adhering to previous methodologies when adaptations are necessary.

Our proposed attack methodology is enhanced through three critical factors: Building Blocks, File Interrelation, and Space Efficiency. The Binary-attack, not originally designed for thresholds, has been poorly adapted, retaining its initial building blocks without substantial improvements. Conversely, the FST-attack, specifically designed for SE-schemes with thresholds, neglects the space-efficient characteristics inherent to the Binary-attack. Both approaches fail to optimize file interrelations among injected files.

By establishing a new (s, n) -set structure that offers greater flexibility than the traditional (s, n) -set construction, our attack achieves optimal file interrelation among all injected files. This optimization leads to improved space efficiency in keyword identification, ensuring that keywords are identified with the minimum number of files at all times.

The Binomial-attack represents a significant advancement over these active attack methods that utilize the leakage of file access patterns. It maximizes the storage of keywords within a limited number of injected files by employing an Increment $[r, n]$ -set to identify keywords. This approach iterates through all possible combinations of an $(n - r + 1, n)$ -set starting from $r = 1$, progressing with $r = r + 1$ until no additional space is available in the files. The adopted Binomial-attack starts at $r = 2$ to decrease the candidate set size for a query when the SE scheme uses padding as a countermeasure.

Our findings demonstrate that, regardless of the presence or absence of a threshold, the Binomial-attack consistently outperforms both the Binary- and FST-attack methods. However, when padding is introduced, there are specific threshold and dataset size combinations where FST requires fewer additional injections on average. It remains uncertain whether

CHAPTER 8. CONCLUSION

this advantage would persist with the implementation of a more efficient keyword recovery method.

Bibliography

- [1] Laura Blackstone, Seny Kamara, and Tarik Moataz. Revisiting leakage abuse attacks. In *NDSS 2020*. The Internet Society, 2020. doi: 10.14722/ndss.2020.23103.
- [2] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *CCS 2017*, pages 1465–1482. ACM, 2017. doi: 10.1145/3133956.3133980.
- [3] David Cash and Stefano Tessaro. The locality of searchable symmetric encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 351–368. Springer, 2014. doi: 10.1007/978-3-642-55220-5_20.
- [4] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *CCS 2015*, page 668–679. Association for Computing Machinery, 2015. ISBN 9781450338325. doi: 10.1145/2810103.2813700.
- [5] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. New constructions for forward and backward private symmetric searchable encryption. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *CCS 2018*, pages 1038–1055. ACM, 2018. doi: 10.1145/3243734.3243833.
- [6] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. *J. Comput. Secur.*, 19(5):895–934, 2011. doi: 10.3233/JCS-2011-0426. URL <https://doi.org/10.3233/JCS-2011-0426>.
- [7] Marc Damie, Florian Hahn, and Andreas Peter. A highly accurate query-recovery attack against searchable encryption using non-indexed documents. In Michael D. Bailey and Rachel Greenstadt, editors, *USENIX 2021*, pages 143–160. USENIX Association, 2021.
- [8] Enron Corporation. Enron email dataset, 2004. URL <http://www.cs.cmu.edu/~enron/>.

- [9] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS 2012*. The Internet Society, 2012.
- [10] Seny Kamara and Tarik Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 94–124, 2017. doi: 10.1007/978-3-319-56617-7_4.
- [11] Chang Liu, Liehuang Zhu, Mingzhong Wang, and Yu an Tan. Search pattern leakage in searchable encryption: Attacks and new construction. *Information Sciences*, 265: 176–188, 2014. ISSN 0020-0255. doi: 10.1016/j.ins.2013.11.021.
- [12] R Liu and Z F Cao. In *Two new methods of distributive management of cryptographic key*, pages 10–14. J. Commun., 8, 1987.
- [13] Muhammad Naveed. The fallacy of composition of oblivious ram and searchable encryption. *IACR Cryptol. ePrint Arch.*, 2015:668, 2015. URL <https://api.semanticscholar.org/CorpusID:11042885>.
- [14] Jianting Ning, Xinyi Huang, Geong Sen Poh, Jiaming Yuan, Yingjiu Li, Jian Weng, and Robert H. Deng. LEAP: leakage-abuse attack on efficiently deployable, efficiently searchable encryption with partially known dataset. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS 2021*, pages 2307–2320. ACM, 2021. doi: 10.1145/3460120.3484540.
- [15] Simon Oya and Florian Kerschbaum. Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption. In Michael D. Bailey and Rachel Greenstadt, editors, *USENIX 2021*, pages 127–142. USENIX Association, 2021.
- [16] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Symmetric searchable encryption with sharing and unsharing. In Javier López, Jianying Zhou, and Miguel Soriano, editors, *ESORICS 2018, Part II*, volume 11099 of *LNCS*, pages 207–227. Springer, 2018. doi: 10.1007/978-3-319-98989-1_11.
- [17] Rishabh Poddar, Stephanie Wang, Jianan Lu, and Raluca Ada Popa. Practical volume-based attacks on encrypted databases. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020*, pages 354–369. IEEE, 2020. doi: 10.1109/EUROSP48549.2020.00030.
- [18] David Pouliot and Charles V. Wright. The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *CCS 2016*, pages 1341–1352. ACM, 2016. doi: 10.1145/2976749.2978401.
- [19] Dawn Xiaodong Song, David A. Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55. IEEE Computer Society, 2000. doi: 10.1109/SECPRI.2000.848445.

- [20] Shifeng Sun, Xingliang Yuan, Joseph K. Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, and Surya Nepal. Practical backward-secure searchable encryption from symmetric puncturable encryption. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *CCS 2018*, pages 763–780. ACM, 2018. doi: 10.1145/3243734.3243782.
- [21] Gaoli Wang, Zhenfu Cao, and Xiaolei Dong. Improved file-injection attacks on searchable encryption using finite set theory. *Comput. J.*, 64(8):1264–1276, 2021. doi: 10.1093/COMJNL/BXAA161.
- [22] Xianglong Zhang, Wei Wang, Peng Xu, Laurence T. Yang, and Kaitai Liang. High recovery with fewer injections: Practical binary volumetric injection attacks against dynamic searchable encryption. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 5953–5970. USENIX Association, 2023.
- [23] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 707–720. USENIX Association, 2016.

Appendix A

Paper

File-Injection Attacks on Searchable Encryption, Based on Binomial Structures

Tjard Jan Langhout, Huanhuan Chen, and Kaitai Liang

Delft University of Technology, Delft, The Netherlands

tjardlanghout@gmail.com

{h.chen-2,kaitai.liang}@tudelft.nl

Abstract. One distinguishable feature of file-inject attacks on searchable encryption schemes is the 100% query recovery rate, i.e., confirming the corresponding keyword for each query. The main efficiency consideration of file-injection attacks is the number of injected files. In the work of Zhang *et al.* (USENIX 2016), $\lceil \log_2 |K| \rceil$ injected files are required, each of which contains $|K|/2$ keywords for the keyword set K . Based on the construction of the uniform (s, n) -set, Wang *et al.* need fewer injected files when considering the threshold countermeasure. In this work, we propose a new attack that further reduces the number of injected files where Wang *et al.* need up to 38% more injections to achieve the same results. The attack is based on an increment (s, n) -set, which is also defined in this paper.

Keywords: Searchable encryption · File-injection attack · Binomial · Increment (s, n) -set

1 Introduction

Ensuring exclusive data access remains a paramount concern, often necessitating external cloud servers due to limited user storage capacity. To enable efficient data searches, these servers must implement search-over-plaintext methods for speed and efficacy.

Song *et al.* [19] were pioneers in proposing a cryptographic scheme tailored to address the challenge of searching encrypted data, particularly enabling controlled and concealed keyword searches. This general searchable encryption (SE) framework entails the storage of an index and database on the server. Each keyword within a file undergoes independent encryption, alongside the encryption of the file as a whole. Retrieval of files containing specific keywords involves the user generating a token by encrypting the desired keyword, which is then matched against all encrypted keywords stored on the server. Upon a match, the entire encrypted file is returned to the user for decryption. Since the introduction of this foundational scheme, numerous researchers have proposed diverse variants of SE schemes [2, 4, 5, 9, 13, 16, 20]. These schemes offer varying levels of file and keyword privacy, with the ORAM scheme emerging as the most secure, effectively concealing access pattern leakage [13]. However, schemes with minimal

leakage patterns tend to be computationally intensive and impractical. Alternatively, other proposed schemes, while computationally less burdensome, permit a marginally higher degree of leakage. Cash *et al.* [3] categorized these schemes into distinct leakage levels: L1, L2, L3, and L4, each revealing different degrees of information about keyword occurrences. Subsequent studies have demonstrated the potential exploitation of even minimal leakage to extract significant information from databases, emphasizing the critical role of prior knowledge in facilitating successful attacks [1, 8, 11, 14]. Recovery of keywords involves retrieving the keyword associated with the queried token, representing an encrypted keyword of a file.

Attacks on SE schemes may manifest as either passive or active. Passive attacks entail the observation of leakage patterns to construct keyword-query matches [6, 11, 15, 18]. These attacks refrain from interfering with protocols and leverage preexisting knowledge to execute their strategies. Passive attacks typically target weaker schemes exhibiting higher leakage levels (L2-L4) and often necessitate external or prior knowledge for execution. Conversely, active attacks involve servers injecting files into a user’s database to glean insights. Injection attacks leverage either file access patterns or volume patterns [1, 3, 17, 21–23]. This paper will be based on file-injections with the use of file access pattern leakage. Active attacks, typically assuming L1 leakage or less, necessitate minimal prior knowledge, contrasting with the requirements of passive attacks. Successful recovery of keywords in active attacks is consistently achieved with 100% accuracy, with the performance metric being the number of injections required for a successful attack.

Cash *et al.* [3] were among the first to introduce an active attack wherein the server sends files to the client, subsequently encrypted and stored by the client. These attacks typically assume L2-L3 leakage, akin to passive attacks. Attackers construct files of their choosing and transmit them to users, compelling the application of the scheme to the received file, thereby enabling observation of ciphertext by the server. Zhang *et al.* [23] categorized such attacks as file-injection attacks and introduced the Binary-attack, premised on L1 leakage and injecting half of the keyword universe per injection, akin to binary search methodologies.

Countermeasures such as thresholds and padding are implemented to impede the success of attacks. Thresholds impose limits on the number of keywords a file can contain, while padding obscures actual results by introducing additional files alongside queried files. Wang *et al.* [21] proposed an alternative approach to injection attacks based on finite set theory, offering superior performance compared to previous methods. This approach, known as the FST-attack, necessitates fewer injections than the Binary-attack under certain conditions, leveraging so-called (s, n) -sets to enhance attack efficacy. Despite these advancements, the FST-attack’s reliance on complete (s, n) -sets for all identified keywords represent a notable limitation.

Organization of the paper. The organization of the rest of the paper goes as follows. In Chapter 2, the description of the SSE scheme, file-injection, thresholds, and the latest state-of-the-art full file-injection attack on SSE schemes are

given. In Chapter 3, our Binomial-attack is explained in detail, together with how it can easily be applied under any threshold and dataset size, and the performances of our attack compared to previous file-injection attacks are visualised. In Chapter 4, we show the consequences of padding on our attack and compare these with the consequences on the FST-attack. In Chapter 5, a mitigation is proposed to perform better under a scheme that uses padding, with minimal trade-off. In Chapter 6 and 7 the results and untouched topics of the paper are debated. Finally, the paper is summarized in Chapter 8.

2 Preliminaries

2.1 Searchable Encryption

An *searchable encryption* (SE) scheme has three algorithms: encryption, search, and update (only for dynamic) algorithms.

The encryption algorithm takes as input a set of files $F = \{F_1, \dots, F_n\}$ and a secret key from the data owner, and outputs the encrypted files. These ciphertexts are then stored on the cloud server. The search algorithm takes a secret key and a keyword k as input, and outputs a query(token) t , which allows the cloud server to search the files that contain the corresponding keyword k among the encrypted files. The data owner can then decrypt the returned documents from the server and identify all related files to the keyword k . The update algorithm only applies to the dynamic SE schemes, which outputs updated files, given a secret key and a set of files.

2.2 File Injection Attack

One of the goals of the attacker is called query recovery. The attack attempts to recover the underlying keywords to queries, which threatens query privacy and file privacy. We focus on file-injection attacks in this paper.

Instead of passive attacks, the attacker in file-injection attacks is active by sending to the data owner some proper documents, which are then encrypted by the latter and also stored on the cloud according to the SE schemes. As an example, one can inject files to a user by sending designed emails in the email system. The attacker then observes the returned files, especially its own injected files, corresponding to the queries through the search algorithm. According to the returned (previously injected) files, the attacker can achieve the goal of query recovery.

The first file-injection attack is proposed by Cash *et al.* [3] and further improved by Zhang *et al.* [23]. We show an example of the binary-search attack in Table 1 that injects $\log_2(|K|)$ files and achieves a 100% query recovery, where $|K|$ is the size of the keyword set. In this example, if returned files corresponding to a query t are F_1 and F_3 , then we know its underlying keyword is k_2 . Analogously, other keywords can also be matched according to different combinations of returned files.

Files	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8
F_1	1	1	1	1	0	0	0	0
F_2	1	0	1	0	1	0	1	0
F_3	1	1	0	0	1	1	0	0

Table 1: An example of the binary-attack file with a keyword universe of 8. Keywords are assigned into files: F_1 , F_2 , F_3 , where 1 denotes the presence of the corresponding keyword and 0 indicates its absence.

In this work, our file-injection attack is based on the same assumption in [3, 23] that the attacker knows the file access pattern (i.e., knowing the returned files according to queries) and also can identify the files on the cloud corresponding to its injected files. One distinguishable feature of file-inject attacks is the 100% query recovery rate, so we evaluate the efficiency of such attacks from the number of injected files.

Wang *et al.* [21] further improved the work [23] to deal with the countermeasures of a threshold of a maximal number of keywords in each file.

2.3 FST-Attack

In this section, we review the definition of a uniform (s, n) -set and how the FST-attack works [21], based on the uniform (s, n) -set. The method to construct a uniform (s, n) -set of a finite set is presented by Liu and Cao [12].

Definition 1 (Uniform (s, n) -set [21]). Let a set $A = \{d_1, d_2, \dots, d_m\}$ and the subsets $A_1, A_2, \dots, A_n \subset A$ be called a uniform (s, n) -set of A ($m \geq \binom{n}{s-1}$) if the following three conditions are satisfied:

- $|A_1| = |A_2| = \dots = |A_n|$;
- For any s subsets $A_{i_1}, \dots, A_{i_s} \in \{A_1, \dots, A_n\}$, there is $\bigcup_{j=1}^s A_{i_j} = A$;
- For any $s-1$ subsets $A_{i_1}, \dots, A_{i_{s-1}} \in \{A_1, \dots, A_n\}$ there is $\bigcup_{j=1}^{s-1} A_{i_j} = A \setminus \{d_i\}$.

Where n denotes the number of injected files, $|A_i|$ the size of each injected file and m the keyword universe.

A uniform (s, n) -set for a finite set with size m has the following properties, when we choose $m = \binom{n}{s-1}$.

Lemma 1 ([21]). Let (A_1, A_2, \dots, A_n) be a uniform (s, n) -set, then we have

- Size. The size of each file A_i is $|A_i| = \binom{n-1}{s-1}$ for $1 \leq i \leq n$.

- *Intersection.* Let $r = n - s + 1$, then the size of the intersection of arbitrary r files is only 1: $|\cap_{j=1}^r A_{i_j}| = 1$.

Based on the uniform (s, n) -set, Wang *et al.* [21] presents a file-injection attack to SE. We assume the keyword set $\mathbb{K} = \{k_1, k_2, \dots, k_m\}$.

Their basic attack is to first construct a uniform (s, n) -set $\{A_1, A_2, \dots, A_n\}$ based on the technique presented by Liu and Cao [12] for the keyword set \mathbb{K} such that $\binom{n}{s-1} \geq m^1$, and then generate a file set of size n : $\{D_1, D_2, \dots, D_n\}$, where the file D_i contains the same keyword in the A_i for $1 \leq i \leq n$. Those files are then injected into the SE scheme, and the attack recovers the keyword corresponding to a token by the returned $n - s + 1$ files. The correctness of the basic attack is guaranteed by Lemma 1, i.e., there only exists one keyword in the intersection of $n - s + 1$ files.

When the threshold countermeasure is taken into consideration, that is the number of keywords in each file should be smaller than a threshold T , they proposed an advanced file-injection attack, aiming at obtaining a minimum n , the number of files that should be injected. Towards this goal, they choose the minimum n such that

$$\begin{cases} \binom{n-1}{s-1} \leq T \\ \binom{n}{s-1} \geq m. \end{cases} \quad (1)$$

Moreover, they present look-up tables to determine the optimal s and n corresponding to the threshold T and the number of keywords in different intervals.

Example 1. As an example of recovering 23 keywords $\{k_1, k_2, \dots, k_{23}\}$ with threshold 7, we solve the Eq. (1) for $T = 7$ and $m = 23$ and then get a minimum $n = 8$ and $s = n - 1 = 7$. The corresponding injected files according to a uniform $(7, 8)$ -set are shown in Table 2. Note that some parts in files $\{D_1, \dots, D_8\}$ are left as blank since the number of keywords in these files has reached 23. Each keyword can be matched by $n - s + 1 = 2$ returned files. For example, if files D_1 and D_2 are returned after a query to a token t , we know the corresponding keyword to t is k_1 .

This example also explains our major motivation: a single uniform (s, n) -set of the keyword set may not maximize the ability of a file injection attack, or in other words, the number of injected files n is not optimal. The reason is the number of keywords in injected files may be far from reaching the threshold.

3 A New File-injection Attack

In this chapter, we present our new file-injection attack to searchable encryption schemes. It is based on our new definition of a subset family of a finite set, called

¹ It means the maximal number of keywords in the uniform (s, n) -set is greater than the keyword size m .

(7, 8)-set							
Files	Col ₁	Col ₂	Col ₃	Col ₄	Col ₅	Col ₆	Col ₇
F_1	k_{22}	k_{23}					
F_2	k_{16}	k_{17}	k_{18}	k_{19}	k_{20}	k_{21}	
F_3	k_{11}	k_{12}	k_{13}	k_{14}	k_{15}	k_{21}	
F_4	k_7	k_8	k_9	k_{10}	k_{15}	k_{20}	
F_5	k_4	k_5	k_6	k_{10}	k_{14}	k_{19}	
F_6	k_2	k_3	k_6	k_9	k_{13}	k_{18}	
F_7	k_1	k_3	k_5	k_8	k_{12}	k_{17}	k_{23}
F_8	k_1	k_2	k_4	k_7	k_{11}	k_{16}	k_{22}

Table 2: An example of recovering 23 keywords with threshold $T=7$ by the uniform (7, 8)-set.

increment $[r, n]$ -set. Our main technique is to construct an increment $[r, n]$ -set of the keyword set. Compared to the uniform (s, n) -set used in [21], the increment $[r, n]$ -set enables us to put more keywords in the injected files, thus significantly reducing the number of injected files.

3.1 Increment $[r, n]$ -Set

The main idea of the increment $[r, n]$ -set is to optimize the available space in the injected files, which are defined as follows.

Definition 2 (Increment $[r, n]$ -set). *Let A be a set, then the subsets $A_1, A_2, \dots, A_n \subset A$ are called an increment $[r, n]$ -set of A if the following conditions are satisfied:*

- $|A_1| = |A_2| = \dots = |A_{n-r+1}|$;
- *Elements in A_1, A_2, \dots, A_n are separated into r blocks such that the i -th ($1 \leq i \leq r$) block of A_1, A_2, \dots, A_n forms a uniform $(n - i + 1, n)$ -set of the union set of the i -th block of A_1, A_2, \dots, A_n .*

An (s, n) -set is here a single block and the increment $[r, n]$ -set consists out of multiple (s, n) -sets (blocks), where the r is increased per block. Recall that for a uniform (s, n) -set, a keyword can be uniquely recovered by $r = n - s + 1$ returned files. Therefore, the keywords in the i -th block of an increment $[r, n]$ -set are determined by $n - (n - i + 1) + 1 = i$ files, since the i -th block is a uniform $(n - i + 1, n)$ -set by definition. That is, the keywords in the 1st block can be represented by 1 file, the keywords in the 2nd block by 2 files, and so on. This is what we call an *increment*. We refer to Table 3 for a visual example.

We denote the i -th block of A_j by A_j^i for $1 \leq j \leq n$, then we get the following corollary which follows from Lemma 1.

Corollary 1. *If (A_1, A_2, \dots, A_n) is an increment $[r, n]$ -set of A , then we have*

$$|A_j^i| = \binom{n-1}{n-i},$$

for $1 \leq i \leq r, 1 \leq j \leq n$.

Therefore, we know that the size of A_j is $|A_j| = \sum_{i=1}^s \binom{n-1}{n-i}$ for $1 \leq j \leq n$. The main idea of our basic file-injection attack is to construct an increment $[r, n]$ -set, instead of complete independent (s, n) -sets spread over different chunks of files, like FST does. We are aiming at reducing the total number of injected files n to as few files as possible. Keywords are recovered according to the different combinations of returned files (details are present in Section [3.2]).

Example 2. We give an example of an increment $[r, n]$ -set of the keyword set $\{k_1, k_2, \dots, k_{23}\}$ with threshold seven for a comparison to the example in Table 2. We compute r and the minimum n such that

$$\begin{cases} \sum_{i=1}^r \binom{n-1}{n-i} \leq 7 \\ \sum_{i=1}^r \binom{n}{n-i} \geq 23, \end{cases} \quad (2)$$

and then we get $r = 3$ and $n = 6$. The increment $[3, 6]$ -set of the aimed keyword set is shown in Table 3. Compared to Example 1, it reduces the number of injected files from 8 to 6! Every space in these files is filled with keywords, while still controlling the total number of keywords within the threshold.

	(6, 6)-set		(5, 6)-set				(4, 6)-set
Files	Col ₁	Col ₂	Col ₃	Col ₄	Col ₅	Col ₆	Col ₇
F_1	k_1	k_7	k_{10}	k_{13}	k_{15}	k_{19}	k_{22}
F_2	k_2	k_7	k_{11}	k_{14}	k_{16}	k_{20}	k_{22}
F_3	k_3	k_8	k_{11}	k_{13}	k_{17}	k_{21}	k_{22}
F_4	k_4	k_8	k_{12}	k_{14}	k_{18}	k_{19}	k_{23}
F_5	k_5	k_9	k_{12}	k_{15}	k_{17}	k_{20}	k_{23}
F_6	k_6	k_9	k_{10}	k_{16}	k_{18}	k_{21}	k_{23}

Table 3: An example of recovering 23 keywords with threshold $T=7$ by an increment $[3, 6]$ -set, which is divided into 3 blocks. Keywords in the 1st, 2nd, and 3rd block can be recovered by 1, 2, and 3 returned files, respectively.

3.2 Construction of Increment $[r, n]$ -Set

In this section, we present a way to the construction of increment $[r, n]$ -set of a finite set, which uses the method of constructing uniform (s, n) -set as a subroutine (we also provide a new construction method of a uniform (s, n) -set in the full version [10]).

Given as input the size of the keyword m and threshold of the number of keywords in a file T , we aim to construct an increment $[r, n]$ -set of the keyword set with the minimum n such that (1) the size of each file should not be greater than the threshold T , and (2) the maximal number of keywords that those files can recover is at least m . To maximize the recovery ability under condition (1), our overall idea is to construct r uniform $(n - i + 1, n)$ -sets. for $1 \leq i \leq r$ and return the first T columns as the aimed set. Then by Lemma 1, we know the first $r - 1$ blocks take $\sum_{i=1}^{r-1} \binom{n-1}{n-i}$ columns and can recover $\sum_{i=1}^{r-1} \binom{n}{n-i}$ keywords in total. The last block takes the rest $T - \sum_{i=1}^{r-1} \binom{n-1}{n-i}$ columns and allows to recover $\lfloor n/r \cdot [T - \sum_{i=1}^{r-1} \binom{n-1}{n-i}] \rfloor$ keywords. Then the condition (2) is equal to

$$\sum_{i=1}^{r-1} \binom{n}{n-i} + \left\lfloor \frac{n}{r} \cdot \left[T - \sum_{i=1}^{r-1} \binom{n-1}{n-i} \right] \right\rfloor \geq m. \quad (3)$$

We proceed in the discussion of r starting from 1 to T . For each r , we record all the possible n to the Inequality 3, with the minimum one as the optimal solution. For simplicity of exposition, we denote $NK(r, n)$ as the left part of the above inequality. The whole process of constructing an increment $[r, n]$ -set is present in Algorithm 1.

Algorithm 1 Construction of increment $[r, n]$ -set

Input: Number of keywords m , threshold T

Output: An increment $[r, n]$ -set of the keyword set $\{k_1, k_2, \dots, k_m\}$

- 1: Initialize an empty candidate set: $candidate \leftarrow []$
 - 2: **for** $r = 1$ **to** T **do**
 - 3: Solve n from $NK(r, n) \geq T$ and denote the minimum n as n_0
 - 4: Append (r, n_0) to $candidate$
 - 5: $r = r + 1$
 - 6: **end for**
 - 7: Find (r, n_0) with the minimum n_0 and corresponding r from $candidate$
 - 8: **for** $i = 1$ **to** r **do**
 - 9: Construct a uniform $(n_0 - i + 1, n_0)$ -set of keywords with index from $\sum_{j=1}^{i-1} \binom{n_0-1}{n_0-j}$ to $\sum_{j=1}^i \binom{n_0-1}{n_0-j}$ by the technique proposed in [10].
 - 10: **end for**
 - 11: **Output** the first T columns of the created files
-

Going back to Example 1, we compute the Inequality 3 to get $candidate = [(2, 7), (3, 6), (4, 7)]$. Then we know the optimal increment $[r, n]$ -set is $r = 3$, and $n = 6$.

3.3 Binomial-Attack

Given the keyword universe $\mathbb{K} = \{k_1, k_2, \dots, k_m\}$ and the threshold T as the maximal number of keywords in each file, we present our file injection attack in Algorithm 2, which is based on the increment $[r, n]$ -set of the \mathbb{K} .

Algorithm 2 Binomial-attack

Input: Keyword set $\mathbb{K} = \{k_1, k_2, \dots, k_m\}$, threshold T , a query token t

Output: Keyword corresponding to the token t

- 1: Generate an increment $[r, n]$ -set A_1, A_2, \dots, A_n of \mathbb{K} with threshold T by Algorithm 1
 - 2: **for** $j = 1$ **to** n **do**
 - 3: Let a file D_i contain the same keywords as A_i
 - 4: **end for**
 - 5: Inject files $\{F_1, F_2, \dots, F_n\}$ into the SE scheme
 - 6: **return** the corresponding keyword to t according to the returned i files ($1 \leq i \leq r$)
-

Now that the structure of the attack is understood, we can proceed to calculate the required number of injections to achieve the desired number of identifiable keywords. There are multiple formulas to calculate the required number of injections. The appropriate formula to utilize depends on at which $(n - r + 1, n)$ -set the threshold will limit the attack from injecting more combinations.

Deciding the Number of Injections. The attack always starts with $r = 1$ and progresses incrementally from there on forth. At some point within an $(n - r + 1, n)$ -set, the threshold will limit the number of keywords it can inject. Refer to Table 3 for a visual representation.

By Eq. 3 we know the number of keywords an $(n - r + 1, n)$ -set can utilize for a certain threshold, under a specific r . When the threshold is reached in the $(n - 1, n)$ -set, where $r = 2$ the equation can be written in terms of n like the following:

$$F_2(K, T) = \frac{2K}{T + 1} \quad (4)$$

Similarly to the $(n - 2, n)$ -set, where $r = 3$, the equation becomes:

$$F_3(K, T) = \frac{-(3 + 2T) + \sqrt{(3 + 2T)^2 + 24K}}{2} \quad (5)$$

The formulas $F_4(K, T)$ and beyond are only of relevance when the threshold is a significant portion of the number of keywords that need to be injected.

Deciding the Injection Formulas. The next step involves determining the appropriate utilization of each formula for different scenarios.

To determine the appropriate value for r in the increment $[r, n]$ -set, we must assess whether the threshold allows for additional keywords in the files following a uniform $(n - r + 1, n)$ -set. This evaluation must be conducted for each r , commencing at $r = 2$. By Lemma 1 we know the first two blocks utilize a total of n columns. Therefore if $T > n$, the $(n - 2, n)$ -set can also be used. However, the value of n remains unknown at this stage. To address this uncertainty, we substitute $n = T$ into F_2 . This yields the threshold at which both the $(n - r + 1, n)$ -sets in the increment $[2, n]$ -set become uniform and precisely meet the threshold. If the keyword universe exceeds this value, the attack will require more than T injections. Conversely, if the keyword universe falls below this value, fewer than T injections are required. Consequently, there will be residual space in the injected files for (at least) the $(n - 2, n)$ -set. The minimum value of K to only be able to build up to an Increment $[2, n]$ -set is outlined as follows:

$$Min_{F_2}(T) = \frac{1}{2}T^2 + \frac{1}{2}T \quad (6)$$

F_2 should be applied when the outcome of $Min_{F_2}(T) \leq K$. Alternatively, the formula of Min_{F_3} determines whether F_3 or F_4 should be utilized. Following the same procedures as before, we get:

$$Min_{F_3}(T) = \frac{2+T}{3} \cdot \frac{1+\sqrt{8T-7}}{2} + \frac{T-1}{3} \quad (7)$$

These formulas already hold an improvement over FST, since FST had a lookup table with overlapping values and no clear points to choose from. Using our previous example 1, we see $Min_{F_2}(7) > 23$ and $Min_{F_3}(7) < 23$. This means we need to use F_3 , which results in $F_3(23, 7) = 6$ files.

3.4 Performance under Different Thresholds

The results consistently demonstrate the superiority of the Binomial-attack over the FST-attack across various thresholds and dataset sizes.

The Binomial-attack consistently outperforms the FST-attack with at least one injected file, regardless of the dataset size. With a threshold of 200, the most substantial disparity occurs in datasets ranging from 7 200 to 7 400 keywords, where the FST-attack requires 33 more injections compared to the Binomial-attack. This represents a 38% increase in injections needed by the FST-attack for equivalent results. In previous studies, the Enron dataset [7] served as a benchmark for attack performance. When applied to the Enron dataset, the FST-attack requires 83 files to cover the entire keyword universe, whereas the Binomial-attack accomplishes this with only 65 files. Thus, in this real dataset scenario, the FST-attack necessitates 28% more injections than the Binomial-attack.

To provide a comprehensive overview of these differences, Fig. 1 illustrates the comparative performance of the Binary-, FST-, and Binomial-attack across various thresholds, with datasets ranging up to 20 000 keywords.

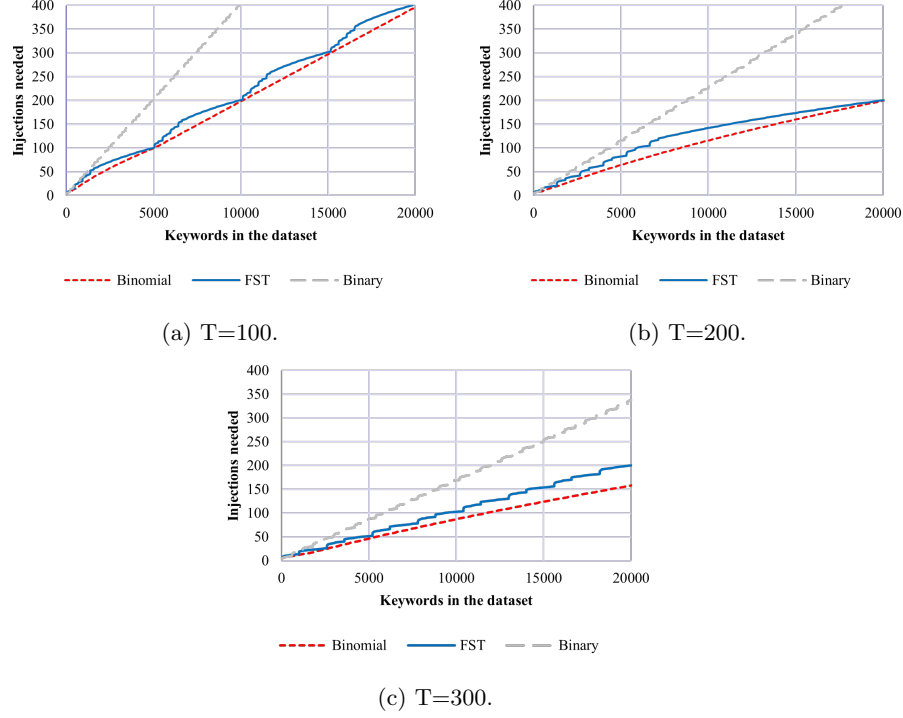


Fig. 1: Performance of different injection attacks under different thresholds.

4 File-injection Attacks on SE Schemes with Keyword Padding

Keyword padding serves as a countermeasure within the SE scheme aimed at obscuring query results by returning more files than necessary. In addition to the files containing the queried keyword, the scheme also includes random files from the dataset in its response. In this chapter, we delve into the consequences of padding and compare the implications between the FST- and Binomial-attack methodologies. Previous studies, such as the FST- and Binary-attack, explored this topic assuming a file dataset of 30 109 files and a keyword universe of 5 050 keywords. The scheme adopts a threshold of 200, and on average, a query yields matches on 560 files, with an additional 60% of random files included (336 files). Section 4.1 delves into the quantitative effects of padding, while Section 4.2 presents a visual exploration of these effects.

4.1 Calculating the Effects

To assess the impact, three key steps are necessary. Firstly, we must determine the average number of additional injected files returned as a consequence of

their selection for padding. Subsequently, we can proceed to determine the average number of keyword combinations we can generate. These combinations represent distinct file arrangements utilized for the unique identification of a single keyword, collectively referred to as the candidate set for a query. Finally, the last step entails re-executing the attack on the candidate set to pinpoint the specific keyword utilized.

Injected Files from Padding. To calculate the average number of injected files chosen during padding, we can utilize the hypergeometric distribution function. Our population size is $30\,109 - 560 = 29\,549$, since the matched files for the query can not be chosen for the padding. The number of successes will be $F3(5\,050, 200) = 64.8 \approx 65$ files, minus the average injected file response, leaves $65 - 3 = 62$ successes. The sample size is 336. We can calculate the probabilities for all possible numbers of successes in the sample and then multiply each probability by the corresponding number of successes. The results are then summed to determine the average number of injected files (p) chosen in the padding:

$$p = \sum_{n=1}^{62} \frac{\binom{62}{n} \binom{29549-62}{336-n}}{\binom{29549}{336}} \quad (8)$$

Average Candidate Set Size. The average candidate set size is determined by three key factors associated with each $(n-r+1, n)$ -set used to identify keywords. The first factor considers the number of possible combinations within the given $(n-r+1, n)$ -set when $r+p$ injected files are returned. The second factor accounts for the ratio of combinations utilized in that $(n-r+1, n)$ -set compared to its total possible combinations. The third factor represents the ratio of identifiable keywords in the $(n-r+1, n)$ -set to the total number of identifiable keywords. Multiplying these three factors together yields the average candidate set size per $(n-r+1, n)$ -set. Summing the results across all $(n-r+1, n)$ -sets provides the overall average candidate set size:

$$\sum_{r=1}^R \binom{r+p}{r} \cdot \frac{|K_r|^2}{\binom{n}{r} \cdot |K_R|} \quad (9)$$

Number of Extra Injections Needed. A straightforward method to determine the number of extra injections required is to analyze on a per-query basis. By considering the average candidate set size per query, we can execute our attack specifically for that particular candidate set to recover the searched keyword. While this approach is not optimal, it suffices for comparison purposes with the FST-attack.

4.2 Visualising the Effects

This section will demonstrate the effects of padding on both FST and the Binomial-attack. While the Binomial-attack may not always appear significantly

better based solely on the average candidate set size per query, it's important to consider that FST is generally less efficient, requiring more injections to cover the same candidate set. Here, we present the results for a scheme with a threshold of 200. Results for different thresholds are available in Appendix A.

Targeting the Whole Dataset. In Fig. 2, we see the average sizes of candidate sets for different dataset sizes. The corresponding number of extra injections required for the candidate sets is illustrated in Fig. 3. While there is a small dataset size range where the Binomial-attack requires one more injection than the FST-attack, FST generally performs worse for all other dataset sizes.

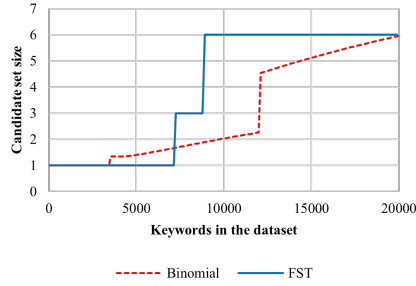


Fig. 2: Candidate set size per query, $T=200$.

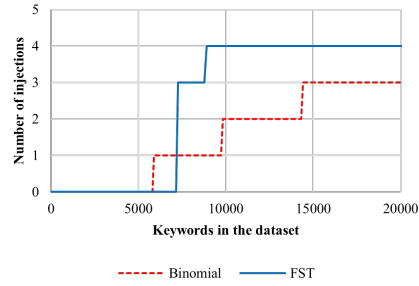


Fig. 3: Extra injection size per query, $T=200$.

Targeting a Subset of the Dataset. When targeting a subset of the keyword universe, fewer injections are required to cover the target set, benefiting both attacks. However, not every query relates to a keyword in the target set. When combined with padding, this may not pose an issue if we assume a consistent average number of injected files in the padding. For instance, if two injected files are returned and the average padding injection is also two, it suggests a search for a keyword not in the target set. However, if a return of two injected files could also indicate a search for a keyword occurring once or twice, all searches become candidate sets. While these candidate sets may not contain actual keywords from the target set, distinguishing beforehand is impossible. The only option is to re-perform the attack on the candidate set.

In this scenario, our attack performs notably worse. This is because the Binomial-attack initiates with an (n, n) -set. FST does not follow this approach, resulting in fewer potential combinations when all preceding $(n - r + 1, n)$ -sets are included in the candidate set. Figure 4 illustrates the number of extra injections required when searching for a keyword that is not in the target set.

5 Adopted Binomial-Attack

When the target set is a subset of the dataset, searches for keywords outside the target set result in additional candidate sets. To mitigate the size of these extra candidate sets, adjustments to the attack methodology are necessary. This chapter outlines the modifications required to minimize candidate size while maintaining effectiveness. Despite the trade-off, the attack consistently requires fewer initial injections than FST.

5.1 Removing the (n, n) -Set

In the Binomial-attack, the lowest value for r is always one. While this minimizes the space occupied in injected files, it also leads to greater overlap with keywords spread across multiple injected files. Conversely, higher values of r in the $(n - r + 1, n)$ -sets for all keywords result in smaller candidate sets per query. To reduce the size of candidate sets, keywords should not be identified with only one injected file, meaning the attack starts from $(n - 1, n)$ instead of (n, n) . This frees up space that can be allocated to a different $(n - r + 1, n)$ -set.

5.2 Results after the Mitigation

The number of identifiable keywords decreases by either $\frac{n}{2}$ or $\frac{2n}{3}$, depending on which $(n - r + 1, n)$ -set the attack terminates due to the threshold. Refer to Table 4 for a visual representation of this transformation.

In Fig. 5, the difference in extra injections required between the FST- and adopted Binomial-attack is illustrated. FST consistently requires an equal or greater number of injections to recover candidate sets.

Files	(5, 6)-set					(4, 6)-set	
	Col ₁	Col ₂	Col ₃	Col ₄	Col ₅	Col ₆	Col ₇
F_1	k_1	k_4	k_7	k_9	k_{13}	k_{16}	k_{19}
F_2	k_1	k_5	k_8	k_{10}	k_{14}	k_{16}	k_{18}
F_3	k_2	k_5	k_7	k_{11}	k_{15}	k_{16}	k_{18}
F_4	k_2	k_6	k_8	k_{12}	k_{13}	k_{17}	k_{18}
F_5	k_3	k_6	k_9	k_{11}	k_{14}	k_{17}	k_{19}
F_6	k_3	k_4	k_{10}	k_{12}	k_{15}	k_{17}	k_{19}

Table 4: Distribution of an Increment $[3, 6]$ -set, without $(6, 6)$ -set, $T=7$.

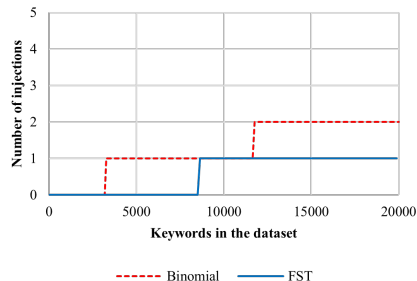


Fig. 4: Extra injection sizes per query that is not in the target set, $T=200$.

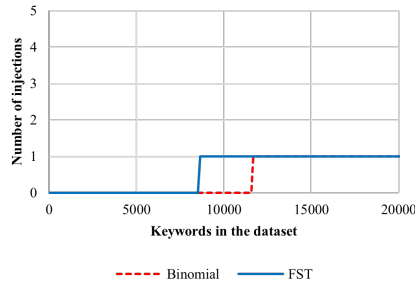


Fig. 5: Extra injection sizes per query that is not in the target set, $T=200$, for the adopted attack.

6 Discussion

In addition to padding, there exist other countermeasures aimed at increasing the difficulty of attacks. One such countermeasure involves the creation of clusters of keywords, as described in [11]. When a search query is initiated for one of the keywords within a cluster, all files containing keywords from the same cluster are returned. This approach not only obscures the specific keyword being searched for, but also introduces ambiguity regarding the association of injected files with specific keywords. Due to the potential for multiple combinations of keywords within the returned files, the attacker may be compelled to employ higher $(n - r + 1, n)$ -sets, necessitating a greater number of injected files. It is important to note, that this countermeasure assumes a static keyword universe and may require modification to accommodate dynamic searchable encryption scenarios.

Despite its theoretical appeal, searchable encryption has yet to achieve widespread adoption in practical applications and can vary significantly in its configurations, including the implementation of countermeasures. Consequently, predicting the exact characteristics of a searchable encryption scheme in practice remains challenging. Nevertheless, there is value in speculating on the potential implications of different settings and attempting to assess the scheme's security under various conditions, even if these scenarios remain largely theoretical at present. This makes it harder to determine how big the safety issues of the schemes are.

7 Future work

The additional injections required to neutralize candidate sets are primarily utilized to compare the attack against FST. However, the method itself is far from optimal. As presented in this paper and the FST paper, each keyword necessitates multiple additional injections. This approach may result in a greater number of injections than initially required for the attack. A more efficient strategy

involves combining candidate sets and reusing earlier injections, thereby reducing the overall number of additional injections required. However, the optimal method for achieving this remains to be determined.

This attack is an active attack that makes no use of leakage apart from the returned injected files. In contrast, other attacks combine active and passive methods [22]. If Binary- or FST-attack methods are employed, they could be enhanced by incorporating the Binomial-attack. Revisiting these attacks may reveal potential improvements. We also note that further exploration into fields such as coding theory and combinatorics using our increment $[r, n]$ -set could yield relevant connections and contributions and vice-versa.

8 Conclusion

The Binomial-attack represents a significant advancement over existing active attack methods. It maximizes the storage of keywords within a limited number of injected files by employing an Increment $[r, n]$ -set to identify keywords. This approach iterates through all possible combinations of an $(n-r+1, n)$ -set starting from $r = 1$, progressing with $r = r + 1$ until no additional space is available in the files. The adopted Binomial-attack starts at $r = 2$ to decrease the candidate set size for a query when the SE scheme uses padding as a countermeasure.

Our findings demonstrate that, regardless of the presence or absence of a threshold, the Binomial-attack consistently outperforms both the Binary- and FST-attack methods. However, when padding is introduced, there are specific threshold and dataset size combinations where FST requires fewer additional injections on average. It remains uncertain whether this advantage would persist with the implementation of a more efficient keyword recovery method.

Acknowledgment

This work was partly supported by the European Union’s Horizon Europe Research and Innovation Program under Grant No. 101073920 (TENSOR), No. 101070052 (TANGO) and No. 101070627 (REWIRE).

References

1. Blackstone, L., Kamara, S., Moataz, T.: Revisiting leakage abuse attacks. In: NDSS 2020. The Internet Society (2020). <https://doi.org/10.14722/ndss.2020.23103>
2. Bost, R., Minaud, B., Ohrimenko, O.: Forward and backward private searchable encryption from constrained cryptographic primitives. In: Thuraisingham, B., Evans, D., Malkin, T., Xu, D. (eds.) CCS 2017. pp. 1465–1482. ACM (2017). <https://doi.org/10.1145/3133956.3133980>
3. Cash, D., Grubbs, P., Perry, J., Ristenpart, T.: Leakage-abuse attacks against searchable encryption. In: CCS 2015. p. 668–679. Association for Computing Machinery (2015). <https://doi.org/10.1145/2810103.2813700>

4. Cash, D., Tessaro, S.: The locality of searchable symmetric encryption. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 351–368. Springer (2014). https://doi.org/10.1007/978-3-642-55220-5_20
5. Chamani, J.G., Papadopoulos, D., Papamanthou, C., Jalili, R.: New constructions for forward and backward private symmetric searchable encryption. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) CCS 2018. pp. 1038–1055. ACM (2018). <https://doi.org/10.1145/3243734.3243833>
6. Damie, M., Hahn, F., Peter, A.: A highly accurate query-recovery attack against searchable encryption using non-indexed documents. In: Bailey, M.D., Greenstadt, R. (eds.) USENIX 2021. pp. 143–160. USENIX Association (2021)
7. Enron Corporation: Enron email dataset (2004), <http://www.cs.cmu.edu/~enron/>
8. Islam, M.S., Kuzu, M., Kantarcioglu, M.: Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In: NDSS 2012. The Internet Society (2012)
9. Kamara, S., Moataz, T.: Boolean searchable symmetric encryption with worst-case sub-linear complexity. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part III. LNCS, vol. 10212, pp. 94–124 (2017). https://doi.org/10.1007/978-3-319-56617-7_4
10. Langhout, T., Chen, H., Liang, K.: File-injection attacks on searchable encryption, bases on binomial structures. IACR Cryptol. ePrint Arch. (2024), <https://eprint.iacr.org/2024/1000>
11. Liu, C., Zhu, L., Wang, M., an Tan, Y.: Search pattern leakage in searchable encryption: Attacks and new construction. Information Sciences **265**, 176–188 (2014). <https://doi.org/10.1016/j.ins.2013.11.021>
12. Liu, R., Cao, Z.F.: Two new methods of distributive management of cryptographic key. pp. 10–14. J. Commun., 8 (1987)
13. Naveed, M.: The fallacy of composition of oblivious ram and searchable encryption. IACR Cryptol. ePrint Arch. **2015**, 668 (2015), <https://api.semanticscholar.org/CorpusID:11042885>
14. Ning, J., Huang, X., Poh, G.S., Yuan, J., Li, Y., Weng, J., Deng, R.H.: LEAP: leakage-abuse attack on efficiently deployable, efficiently searchable encryption with partially known dataset. In: Kim, Y., Kim, J., Vigna, G., Shi, E. (eds.) CCS 2021. pp. 2307–2320. ACM (2021). <https://doi.org/10.1145/3460120.3484540>
15. Oya, S., Kerschbaum, F.: Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption. In: Bailey, M.D., Greenstadt, R. (eds.) USENIX 2021. pp. 127–142. USENIX Association (2021)
16. Patel, S., Persiano, G., Yeo, K.: Symmetric searchable encryption with sharing and unsharing. In: López, J., Zhou, J., Soriano, M. (eds.) ESORICS 2018, Part II. LNCS, vol. 11099, pp. 207–227. Springer (2018). https://doi.org/10.1007/978-3-319-98989-1_11
17. Poddar, R., Wang, S., Lu, J., Popa, R.A.: Practical volume-based attacks on encrypted databases. In: IEEE European Symposium on Security and Privacy, EuroS&P 2020. pp. 354–369. IEEE (2020). <https://doi.org/10.1109/EUROSP48549.2020.00030>
18. Pouliot, D., Wright, C.V.: The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) CCS 2016. pp. 1341–1352. ACM (2016). <https://doi.org/10.1145/2976749.2978401>
19. Song, D.X., Wagner, D.A., Perrig, A.: Practical techniques for searches on encrypted data. In: 2000 IEEE Symposium on Security and Privacy. pp. 44–55. IEEE Computer Society (2000). <https://doi.org/10.1109/SECPRI.2000.848445>

20. Sun, S., Yuan, X., Liu, J.K., Steinfeld, R., Sakzad, A., Vo, V., Nepal, S.: Practical backward-secure searchable encryption from symmetric puncturable encryption. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) CCS 2018. pp. 763–780. ACM (2018). <https://doi.org/10.1145/3243734.3243782>
21. Wang, G., Cao, Z., Dong, X.: Improved file-injection attacks on searchable encryption using finite set theory. *Comput. J.* **64**(8), 1264–1276 (2021). <https://doi.org/10.1093/COMJNL/BXAA161>
22. Zhang, X., Wang, W., Xu, P., Yang, L.T., Liang, K.: High recovery with fewer injections: Practical binary volumetric injection attacks against dynamic searchable encryption. In: Calandrino, J.A., Troncoso, C. (eds.) USENIX Security 2023. pp. 5953–5970. USENIX Association (2023)
23. Zhang, Y., Katz, J., Papamanthou, C.: All your queries are belong to us: The power of file-injection attacks on searchable encryption. In: Holz, T., Savage, S. (eds.) USENIX Security 2016. pp. 707–720. USENIX Association (2016)

A Performance Comparison under Different Scenarios

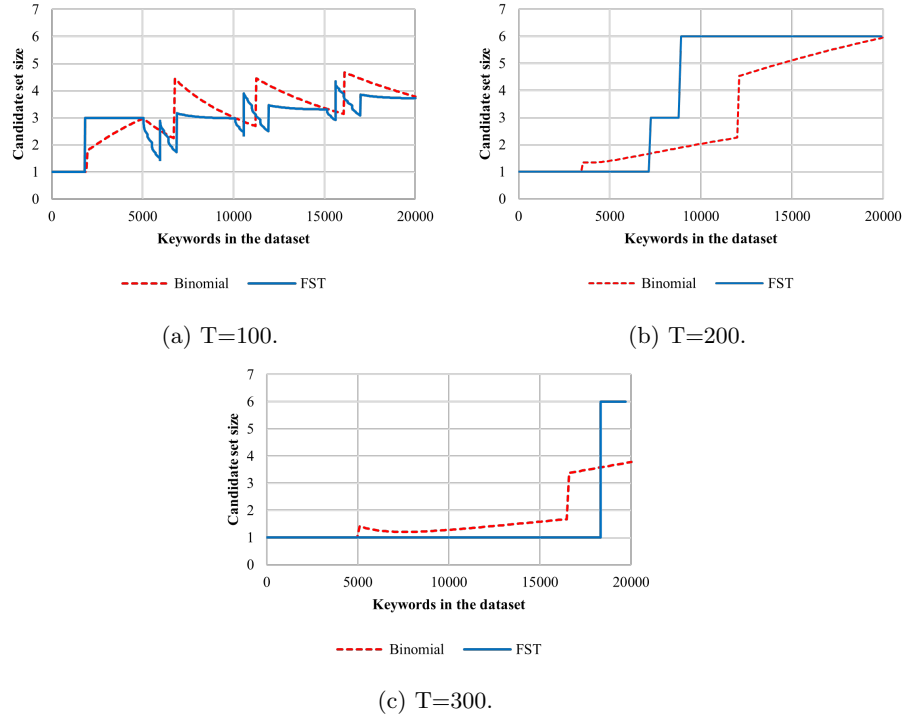


Fig. 6: Candidate set sizes per query when padding is applied, under different thresholds, where the target set is the full keyword universe, for the standard Binomial-attack.

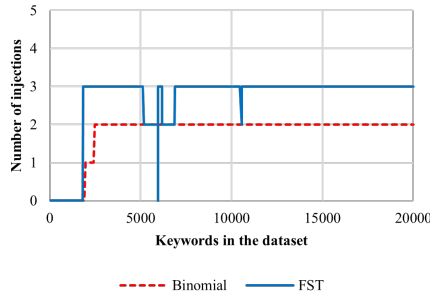
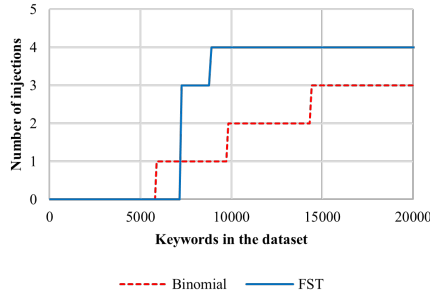
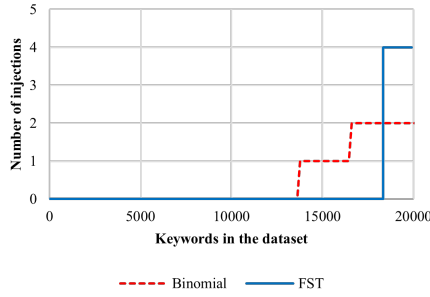
(a) $T=100$.(b) $T=200$.(c) $T=300$.

Fig. 7: Extra injection sizes per query when padding is applied, under different thresholds, where the target set is the full keyword universe, for the standard Binomial-attack.

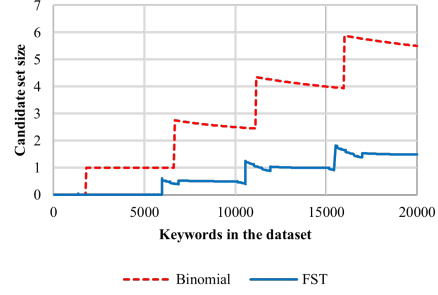
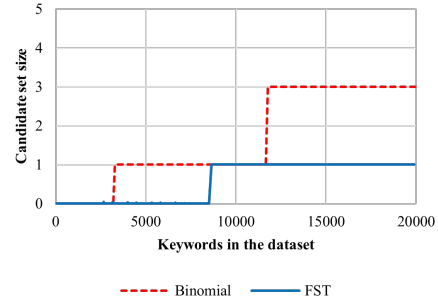
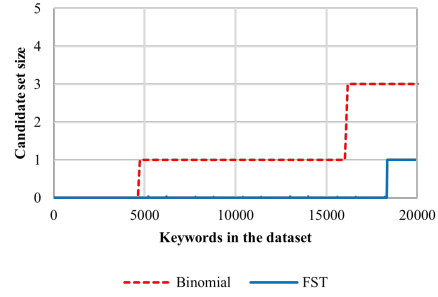
(a) $T=100$.(b) $T=200$.(c) $T=300$.

Fig. 8: Candidate set sizes per query that is not in the target set when padding is applied, under different thresholds, where the target set is a subset of the full keyword universe, for the standard Binomial-attack.

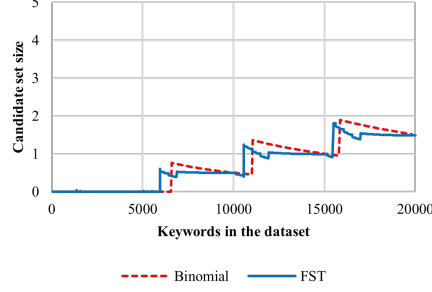
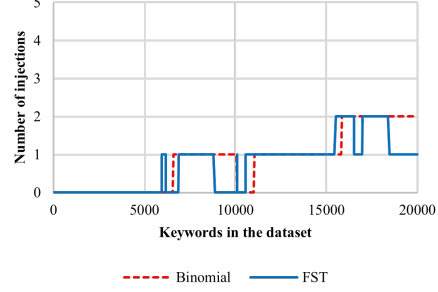
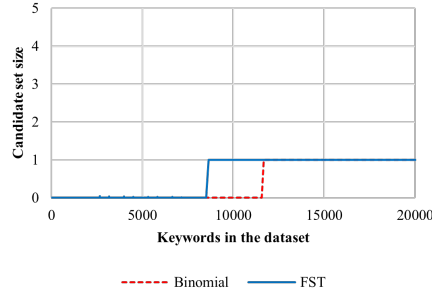
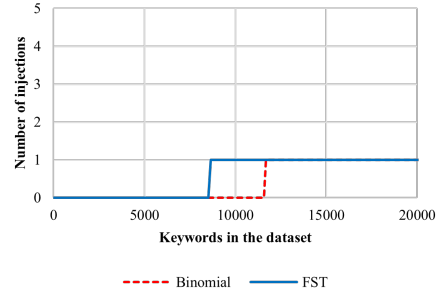
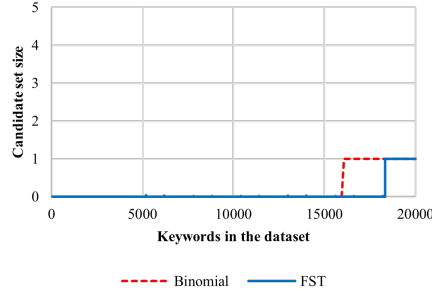
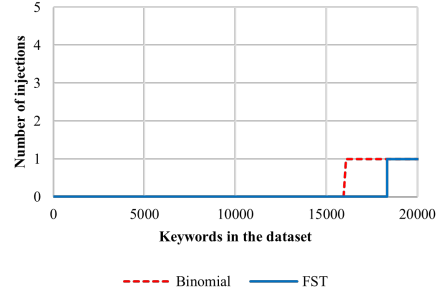
(a) $T=100$.(a) $T=100$.(b) $T=200$.(b) $T=200$.(c) $T=300$.(c) $T=300$.

Fig. 9: Candidate set sizes per query that is not in the target set when padding is applied, under different thresholds, where the target set is a subset of the full keyword universe, for the adopted Binomial-attack.

Fig. 10: Extra injection sizes per query that is not in the target set when padding is applied, under different thresholds, where the target set is a subset of the full keyword universe, for the adopted Binomial-attack.