LAMA: Leakage Abuse Attacks Against Microsoft Always Encrypted

Ryan Seah[1,2], Daren Khu[2], Alexander Hoover[3] and Ruth Ng[2]
[1]*McGill University, Montréal, Canada*
[2]*DSO National Laboratories, Singapore*
[3]*University of Chicago, Chicago, U.S.A.*

Keywords: Microsoft SQL, Leakage Abuse Attack, Database Management, Encrypted Search, Attack, Cryptography.

Abstract: Always Encrypted (AE) is a Microsoft SQL Server feature that allows clients to encrypt sensitive data inside client applications and ensures that the sensitive data is hidden from untrusted servers and database administrators. AE offers two column-encryption options: deterministic encryption (DET) and randomized encryption (RND). In this paper, we explore the security implications of using AE with both DET and RND encryption modes by running Leakage Abuse Attacks (LAAs) against the system. We demonstrate how an adversary could extract the necessary data to run a frequency analysis LAA against DET-encrypted columns and an LAA for Order-Revealing Encryption against RND-encrypted columns. We run our attacks using real-world datasets encrypted in a full-scale AE instancer and demonstrate that a snooping server can recovers over 95% of the rows in 8 out of 15 DET-encrypted columns, and 10 out of 15 RND-encrypted columns.

1 INTRODUCTION

When outsourcing databases to an untrusted server, users face a utility-security trade-off. Minimizing bandwidth and processing time while querying could compromise data security and privacy.

Previous research has explored using weakened cryptographic techniques such as deterministic encryption for equality queries (Popa et al., 2011) and order-preserving/order-revealing encryption for range queries (Agrawal et al., 2004). These methods, known as *property preserving encryption (PPE)* primitives, enable servers to handle queries without decrypting data or involving clients. However, weakening encryption raises security concerns.

Studies (Cash et al., 2015; Naveed et al., 2015; Bindschaedler et al., 2017; Grubbs et al., 2017) have identified vulnerabilities in such systems and conducted simulations demonstrating the reconstruction of significant portions of the underlying database. These attacks, known as leakage abuse attacks (LAAs), exploit prior knowledge of the value distribution in the database to infer plaintext values in the encrypted database.

This study, along with other similar LAAs, (Islam et al., 2012; Zhang et al., 2016; Pouliot and Wright, 2016; Bost and Fouque, 2017; Anzala-Yamajako et al., 2019; Ning et al., 2021) showcases how an adversary, such as a database administrator, can reconstruct databases encrypted with PPE. While many LAAs have been proposed and simulated, there is a lack of prior work testing these attacks on operational systems. This paper fills that gap by demonstrating these attacks on Microsoft's Always Encrypted (AE) (Antonopoulos et al., 2020) system, examining the cryptographic schemes (DET and RND) for potential leakage and exploiting them using LAAs. [1]

While the algorithms for LAAs have been previously discussed, our study is the first to:

- Identify leakage specific to AE implementation, particularly in its use of the indexing data structure for the RND scheme. This enables the application of an LAA that would not be feasible on RND in general, such as in CryptDB.
- Conduct a full-scale demonstrations of two LAAs against an instance of AE, illustrating their applicability in real-world scenarios, particularly when accessed by the server (database administrator).

This analysis presents the first examination of AE using LAAs that we are aware of, highlighting its cryptographic shortcomings within practical environments.

[1]Prior to submission, the authors informed Microsoft about these vulnerabilities.

Seah, R., Khu, D., Hoover, A. and Ng, R.
LAMA: Leakage Abuse Attacks Against Microsoft Always Encrypted.
DOI: 10.5220/0012714800003767
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 21st International Conference on Security and Cryptography (SECRYPT 2024)*, pages 628-633
ISBN: 978-989-758-709-2; ISSN: 2184-7711
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

1.1 Threat Model

Our study adopts a “smash-and-grab” setting, where an adversary gains full system access at a single point, retrieving an information snapshot. This adversary is weaker than both the "honest-but-curious" observer and the "malicious server" capable of protocol deviations. Despite this, we achieve significant attacks on AE. Additionally, we provide the adversary with auxiliary information about server data distribution, like a plaintext column assumed similar to the encrypted one. This aligns with prior research assumptions (Cash et al., 2015; Naveed et al., 2015; Bindschaedler et al., 2017; Grubbs et al., 2017), capturing the adversary’s ability to exploit publicly available or previously leaked data.

2 PRELIMINARIES

2.1 Database Console Commands (DBCC)

Microsoft SQL Server Database Console Commands (DBCC) are utilized to manage the SQL Server database, performing maintenance, informational, validation, and miscellaneous tasks. These commands are stored in plaintext, making them visible to the untrusted SQL Server Database administrator.

2.2 Always Encrypted

AE, a feature of Microsoft SQL Server, enables encryption of sensitive data columns on the client side before storage on an untrusted server. AE allows querying without server-side decryption or downloading the entire encrypted column to the client. Antonopoulos et al. (Antonopoulos et al., 2020) describe AE supporting two encryption types: deterministic (DET) and randomized (RND). For simplicity, we will assume a single encrypted column with r rows in AE, expressed as $(p_1,\ldots,p_r)$ unencrypted and $(c_1,\ldots,c_r)$ encrypted.

2.2.1 DET

Columns encrypted with DET facilitate point-selects, enabling users to search for rows with specific plaintext values. This encryption method employs deterministic AES-CBC with a value-dependent SHA hash as the initialization vector (IV), ensuring that if $p_i = p_j$, then $c_i = c_j$. Consequently, a server can identify rows matching p_i when provided with c_i. This DET encryption resembles that of CryptDB (Popa et al., 2011) but lacks “onion encryption”, making it susceptible to the Frequency Analysis LAA demonstrated against CryptDB by Naveed, Kamara, & Wright (Naveed et al., 2015).

2.2.2 RND

Similar to CryptDB, AE allows columns to employ the randomized encryption scheme RND, which removes all identifying information about p_i, such as equality patterns. This prevents LAAs from exploiting ciphertext properties but hampers server-assisted query processing. However, Antonopoulos et al. detail the construction of a range index enabling enclave-assisted equality, range, and pattern matching queries on RND columns (Antonopoulos et al., 2020).

This index facilitates server determination of column values falling within specified ranges by maintaining a B+-tree on the server with pointers to c_i values ordered according to their corresponding p_i values, i.e. if $p_i < p_j$, then c_i would appear ahead of c_j . Consequently, although no leakage can be extracted from ciphertexts, a linear ordering of plaintexts can be inferred from the B+-tree, similar to the leakage of Order-Revealing Encryption (ORE) (Grubbs et al., 2017).

It’s noteworthy that Antonopoulos et al. discuss a variant of AE on SQL Server 2016, predating the addition of secure enclaves in 2019 (Antonopoulos et al., 2020), while our LAAs were conducted on the current version of AE with secure enclaves.

2.3 Leakage Abuse Attacks (LAAs)

The (unencrypted) column values are assumed to originate from the domain $V = v_1,\ldots,v_N$, with an assumed ordering such that $v_1 < \cdots < v_N$. We presume the adversary possesses knowledge of the rough distribution of plaintexts across the domain, represented as a probability distribution $(a_1,\ldots,a_N)$, where $a_i = \Pr_{j \leftarrow\$ 1,\ldots,r}[p_j = v_i]$. This auxiliary data could be sourced from online repositories, public datasets, or data breaches. Our LAAs could be extended to multiple AE columns, attacking each effectively, potentially yielding higher success rates due to cross-column correlations (Bindschaedler et al., 2018). AE offers two column encryption forms, each catering to specific SQL query classes. We will now delineate the LAAs applicable to each encryption type.

2.3.1 Frequency Analysis LAA

This attack targets DET-encrypted columns, where the adversary utilizes observed frequencies of deterministically encrypted ciphertexts to infer corre-

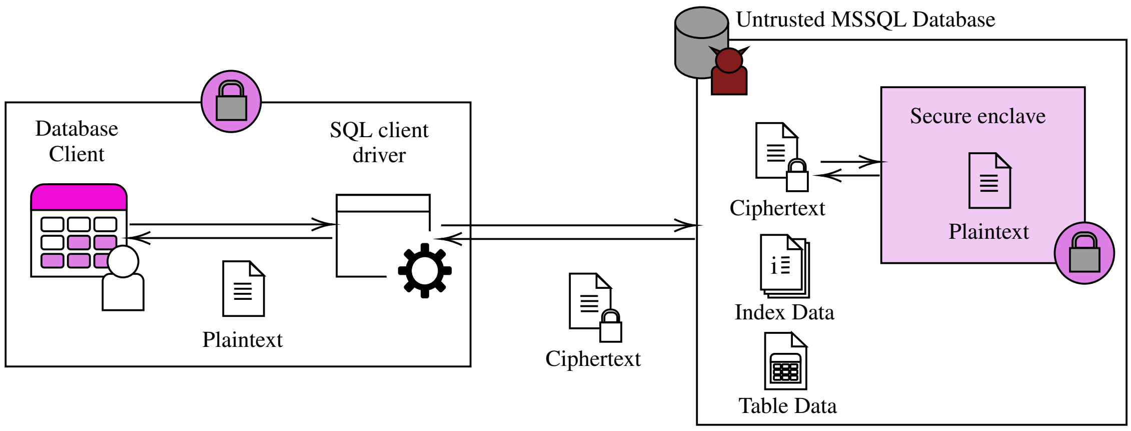

Figure 1: The system diagram shows how a Transact-SQL statement from a client is parsed by a SQL database. If the statement involves encrypted data, client drivers send column encryption keys to secure enclaves. These enclaves process the statement, protecting the keys from exposure. Index and Table data organize database files, accessible outside the enclaves.

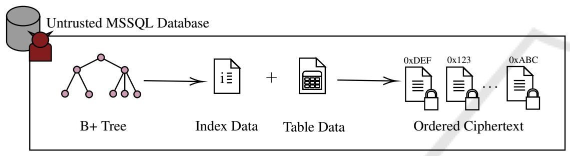

Figure 2: Using the Database Console Command (DBCC), an untrusted database administrator can retrieve ciphered columns ordered by plaintext values for RND encryption..

sponding plaintexts using auxiliary data. Specifically, $(c_1,\ldots,c_r)$ adopts at most N distinct ciphertext values, ordered lexicographically and frequency-counted. Subsequently, the adversary guesses that the i^{th} most frequent ciphertext decrypts to the i^{th} highest a_i probability. Patterson and Lacharité demonstrated this as the optimal strategy for the adversary in our context (Lacharité and Paterson, 2015).

2.3.2 Order-Revealing Encryption LAA

This attack can be applied to RND-encrypted columns. The intuition behind the attack is the following. Firstly, we estimate the number of c_i which decrypt to each plaintext value using the auxiliary data (i.e. assume that v_i occurs $r \cdot a_i$ times among $(c_1,\ldots,c_r)$). Then, we assign plaintexts values to the ciphertexts in the proportions matching our estimates, in accordance to the leaked ordering (e.g. assign the $\lceil r \cdot a_1 \rfloor$ lowest-valued ciphertexts, as leaked in the B+ tree, to v_1, the smallest valued plaintext). We note that even though these queries are performed with enclaves enabled, the attack is still possible as long as the the B+-tree index is constructed on the server.

3 ATTACK SETUP

3.1 Experimental Setup

We deploy a server on Azure Windows Server 2019 Datacenter edition with SQL Server 2019 (MSSQL), while the client runs on a separate computer. Figure 1 illustrates the system diagram, and Table 1 summarizes the conducted experiments. Each experiment is duplicated, with plaintext data encrypted using DET in the first iteration and RND in the second. In the RND experiment, columns are indexed to enable database computation queries like JOIN and GROUP.

3.2 Datasets Used

The experiments utilize datasets from Ohio voters, Florida voters, and the Healthcare Cost and Utilization Project's (HCUP) National Inpatient Sample (NIS) from 2018 and 2019. The Ohio and Florida datasets are sourced from publicly available voter registries [2], while the HCUP data is a selection of columns from HCUP-NIS [3]. The methods employed in this study comply with the Data Use Agreement provided by HCUP and do not enable individual re-identification in the HCUP data.

In Table 1, the first three columns detail the columns used in the 15 experiments. For each experiment, ciphertext and auxiliary data are derived from a million randomly selected rows of the full dataset. In

[2]https://www6.ohiosos.gov/ords/f?p=VOTERFTP:STWD:::\#stwdVtrFiles and http://69.64.83.144/~fl/download/20220331/

[3]For more information about HCUP data and how to obtain data, see http://www.hcup-us.ahrq.gov.

Table 1: Summary of experiments conducted. Fifteen experiment setups were used, with 1 million rows each of auxiliary and ciphertext data columns. For reference, we indicate the domain size (# of distinct values) in each ciphertext column. The success of the frequency analysis LAA against DET-encryption and ORE attack against RND-encryption is reported in terms of v-scores and r-scores (see Section 4) for each experiment setup.

Experiment	**Datasets**		**Domain Size**	**DET**		**RND**
Column Attribute	Auxiliary Data	Ciphertext Data	# distinct values in in ciphertext col.	v-score (%)	r-score (%)	r-score (%)
First Name	Florida	Ohio	54,477	0	9	5
Last Name	Florida	Ohio	143,012	0	2	0
Indicator of Sex	Florida	HCUP-19	5	**100**	**100**	**98**
Race	Florida	HCUP-19	8	33	66	**92**
Indicator of Sex (HCUP)	HCUP-18	HCUP-19	5	60	**99**	**99**
Race (HCUP)	HCUP-18	HCUP-19	8	**100**	**100**	**99**
Age	HCUP-18	HCUP-19	93	34	32	74
Neonatal age ind.	HCUP-18	HCUP-19	3	**100**	**100**	**98**
Admission month	HCUP-18	HCUP-19	13	77	76	**98**
Adm. on weekend	HCUP-18	HCUP-19	3	**100**	**100**	**99**
Died during hosp.	HCUP-18	HCUP-19	4	**100**	**100**	**99**
DRG on discharge	HCUP-18	HCUP-19	763	4	26	52
MDC on discharge	HCUP-18	HCUP-19	26	77	**98**	**98**
IDC-10-CM Diag.	HCUP-18	HCUP-19	13,189	0	22	35
Days to I10_PR1	HCUP-18	HCUP-19	135	21	**99**	**98**

the Florida - HCUP-2019 attacks, HCUP rows with undocumented values are excluded to ensure the attack's success can be accurately determined. Negative values are treated as "NULL" values, except in HCUP-HCUP experiments where they are retained uniquely for comparison purposes.

3.3 Ciphertext Data Extraction and Attack

We demonstrate that the attacks can be run by an adversary (e.g. the database administrator) whose access is limited to the encrypted data on the server.

The data is extracted as follows,

- DET: The entire encrypted database is downloaded for the attack
- RND: To determine the ciphertext order, the Table data is first used to identify all the Index Data (i.e. when `PageType` is 2). Next, we determine the ordering of the Index Data using columns `PrevPagePID` or `NextPagePID`. Finally, the ordered ciphertexts are obtained by querying the `PagePID` in order. This process is illustrated in Figure 2. Note that this is possible as an untrusted database administrator on the server can inspect Table data and Index data through the Database Console Command (DBCC).

LAAs is then applied, frequency analysis on DET and the ORE attack on RND.

4 RESULTS

The success of our attacks is meausured with two metrics, the *row-score* (r-score) and the *value-score* (v-score).

The r-score captures how much of the SQL data the adversary was able to accurately reconstruct. It is the percentage of rows in the AE-encrypted column which the LAA correctly inferred. For both RND and DET-encrypted columns, let $(p'_1, \ldots, p'_r)$ be the plaintext values output by the respective LAA (i.e. the adversary guesses that c_i decrypts to p'_i for each i). Then the r-score is given by

$$\underset{i \leftarrow\$ \{1,\ldots,r\}}{\Pr}[p'_i = p_i]. \tag{1}$$

In DET-encrypted columns, assessing the success of frequency analysis may involve comparing the number of distinct values that were correctly decrypted. This comparison is quantified using the v-score, defined as the fraction of distinct values correctly inferred by frequency analysis relative to the column's domain. Formally, let V represent the domain of the column, then the v-score is given by:

$$\underset{v \leftarrow\$ V, i \leftarrow\$ \{1,\ldots,r\}}{\Pr}[p_i = p'_i | p_i = v] \tag{2}$$

The v-score and r-score are generally correlated but can differ significantly in some cases, especially when real-world data follows a Zipfian distribution. This distribution tends to prioritize accuracy for higher frequency ciphertexts, skewing the r-score but not the v-score. Depending on the attack scenario, either metric may provide a more accurate representation of the adversary's success. It's worth noting that the v-score is irrelevant for the ORE attack since ciphertexts cannot be grouped by equality pattern.

The experiment results, detailed in Table 1, indicate that frequency analysis successfully recovers over 95% of rows in 8 out of 15 cases when deterministic encryption is employed. Additionally, the ORE attack retrieves more than 95% of rows in 10 out of 15 cases when randomized encryption is utilized. These findings underscore the inadequacy of Microsoft AE's encryption in our attacker model, particularly when encrypting small-domain column data.

Interestingly, the ORE attack outperforms frequency analysis in most cases, contrary to expectations given the traditional strength of randomized encryption (e.g., in CryptDB (Popa et al., 2011)). This highlights the significance of minimizing leakage in indexing data structures to preserve the integrity of high-security column encryption.

4.1 Impact and Mitigation

Our attack highlights the limited security provided by AE's encrypted column feature, especially for small-domain columns adhering to known distributions, against persistent eavesdropping adversaries. This vulnerability poses significant concerns, particularly in the context of outsourcing sensitive healthcare data, where any curious eavesdropper (e.g., an employee at the Cloud service provider) could reconstruct sensitive patient information using publicly available data.

To mitigate this threat, several approaches can be considered. As an immediate measure, AE could restrict range searches on RND columns and eliminate the B+-tree indices supporting them. This would restore RND columns to a level of security similar to that of CryptDB (Popa et al., 2011), which aligns with the intended security level for RND in AE's scheme. Without these indices, the ORE attack described earlier would be infeasible, encouraging users to store sensitive data using RND columns. Similar recommendations were proposed by the CryptDB authors (Popa et al., 2015).

However, we do not recommend this as a long-term solution. Not only does it reduce client functionality and fail to resolve the issue with the DET attack, recent research has shown that similar LAAs may still be possible on RND columns using cross-column correlations (Bindschaedler et al., 2018).

A more reliable long-term solution would be to integrate Structured Encryption (StE) into the AE cryptosystem (Chase and Kamara, 2010). StE, a generalization of Searchable Encryption (Song et al., 2000), offers a broad range of cryptographic schemes targeting the precise problem of minimizing leakage when searching encrypted data. Simple StE implementations for point-selects or range searches, with significantly lower leakage than DET or RND, have been deployed at scale, e.g. MongoDB's "Queryable Encryption"[4] and Stealth's SSE scheme (Ishai et al., 2016). In an StE system, the frequency of some plaintext p_i is revealed only when queried by the client, and ciphertexts are not linearly ordered. Thus, the RND attack does not apply, and the DET attack only succeeds after the client exhaustively queries the column, which may be an unrealistic assumption for large-domain columns.

This solution is substantially stronger than AE, where both attacks can be executed without a single query being made by the client (Chase and Kamara, 2010). Additionally, in a system with multiple columns, the attacks as presented become impossible since the StE system does not differentiate between queries to different columns. Therefore, StE is an attractive and commercially viable alternative for defending against these attacks.

For use-cases requiring even higher levels of security, heavier cryptographic techniques such as Volume-Hiding multimaps (Kamara and Moataz, 2019; Patel et al., 2019) or Oblivious RAM (Goldreich, 1987) may be employed to completely suppress leakage.

5 CONCLUSION

In this paper, we explore the possibility of running known LAAs against leakage from Microsoft AE. We demonstrate that an untrusted database administrator at the server can extract sufficient leakage from the encryption scheme to be able to reconstruct significant proportions of encrypted columns. These results indicate that AE's security guarantees on DET and RND might not be as robust as expected by the average user. Additionally, these findings raise concerns about the trade-off between utility and security in AE when using these encryption modes. Future real-world systems should be careful to understand

[4]https://www.mongodb.com/products/queryable-encryption

and make clear the security offered by their services, and to be aware that certain tools may be unfit for storing sensitive data.

REFERENCES

Agrawal, R., Kiernan, J., Srikant, R., and Xu, Y. (2004). Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563–574.

Antonopoulos, P., Arasu, A., Singh, K. D., Eguro, K., Gupta, N., Jain, R., Kaushik, R., Kodavalla, H., Kossmann, D., Ogg, N., et al. (2020). Azure sql database always encrypted. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1511–1525.

Anzala-Yamajako, A., Bernard, O., Giraud, M., and Lafourcade, P. (2019). No such thing as a small leak: Leakage-abuse attacks against symmetric searchable encryption. In *International Conference on E-Business and Telecommunications*, pages 253–277. Springer.

Bindschaedler, V., Grubbs, P., Cash, D., Ristenpart, T., and Shmatikov, V. (2017). The tao of inference in privacy-protected databases. Cryptology ePrint Archive, Report 2017/1078. https://eprint.iacr.org/2017/1078.

Bindschaedler, V., Grubbs, P., Cash, D., Ristenpart, T., and Shmatikov, V. (2018). The tao of inference in privacy-protected databases. *Proc. VLDB Endow.*, 11(11):1715–1728.

Bost, R. and Fouque, P.-A. (2017). Thwarting leakage abuse attacks against searchable encryption – A formal approach and applications to database padding. Cryptology ePrint Archive, Report 2017/1060. https://eprint.iacr.org/2017/1060.

Cash, D., Grubbs, P., Perry, J., and Ristenpart, T. (2015). Leakage-abuse attacks against searchable encryption. In Ray, I., Li, N., and Kruegel, C., editors, *ACM CCS 2015*, pages 668–679. ACM Press.

Chase, M. and Kamara, S. (2010). Structured encryption and controlled disclosure. In Abe, M., editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 577–594. Springer, Heidelberg.

Goldreich, O. (1987). Towards a theory of software protection and simulation by oblivious RAMs. In Aho, A., editor, *19th ACM STOC*, pages 182–194. ACM Press.

Grubbs, P., Sekniqi, K., Bindschaedler, V., Naveed, M., and Ristenpart, T. (2017). Leakage-abuse attacks against order-revealing encryption. In *2017 IEEE Symposium on Security and Privacy*, pages 655–672. IEEE Computer Society Press.

Ishai, Y., Kushilevitz, E., Lu, S., and Ostrovsky, R. (2016). Private large-scale databases with distributed searchable symmetric encryption. In Sako, K., editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 90–107. Springer, Heidelberg.

Islam, M. S., Kuzu, M., and Kantarcioglu, M. (2012). Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS 2012*. The Internet Society.

Kamara, S. and Moataz, T. (2019). Computationally volume-hiding structured encryption. In Ishai, Y. and Rijmen, V., editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 183–213. Springer, Heidelberg.

Lacharité, M.-S. and Paterson, K. G. (2015). A note on the optimality of frequency analysis vs. ℓ_p-optimization. Cryptology ePrint Archive, Report 2015/1158. https://eprint.iacr.org/2015/1158.

Naveed, M., Kamara, S., and Wright, C. V. (2015). Inference attacks on property-preserving encrypted databases. In Ray, I., Li, N., and Kruegel, C., editors, *ACM CCS 2015*, pages 644–655. ACM Press.

Ning, J., Huang, X., Poh, G. S., Yuan, J., Li, Y., Weng, J., and Deng, R. H. (2021). Leap: Leakage-abuse attack on efficiently deployable, efficiently searchable encryption with partially known dataset. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2307–2320.

Patel, S., Persiano, G., Yeo, K., and Yung, M. (2019). Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In Cavallaro, L., Kinder, J., Wang, X., and Katz, J., editors, *ACM CCS 2019*, pages 79–93. ACM Press.

Popa, R. A., Redfield, C. M., Zeldovich, N., and Balakrishnan, H. (2011). Cryptdb: Protecting confidentiality with encrypted query processing. In *Proceedings of the twenty-third ACM symposium on operating systems principles*, pages 85–100.

Popa, R. A., Zeldovich, N., and Balakrishnan, H. (2015). Guidelines for using the CryptDB system securely. Cryptology ePrint Archive, Report 2015/979. https://eprint.iacr.org/2015/979.

Pouliot, D. and Wright, C. V. (2016). The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In Weippl, E. R., Katzenbeisser, S., Kruegel, C., Myers, A. C., and Halevi, S., editors, *ACM CCS 2016*, pages 1341–1352. ACM Press.

Song, D. X., Wagner, D., and Perrig, A. (2000). Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55. IEEE Computer Society Press.

Zhang, Y., Katz, J., and Papamanthou, C. (2016). All your queries are belong to us: The power of file-injection attacks on searchable encryption. In Holz, T. and Savage, S., editors, *USENIX Security 2016*, pages 707–720. USENIX Association.