Contents

Chapter 1

Introduction

Cloud storage is a data storage model in which the digital data is stored remotely on servers owned by a third-party hosting company. It is a cheap alternative to local data storage and management, and has gain much popularity in industry. However, security becomes a real concern in this setting, as an evil man can extract useful information from the server itself or decipher database queries in some way.

As a solution, many encrypted database (EDB) systems have been proposed. CryptoDB [29] in particular, is known as a EDB on relational databases which can handle SQL style queries. It is built on property preserving encryption schemes such as deterministic and order-preserving encryption.

However, it has been proven that property preserving encryption schemes are not sufficient to protect the database system. The property preserved by the encryption scheme can often leak enough information for the attacker to recover something from the EDB. Furthermore, the queries made by the client to the server can be used to generate statistical attack.

In this project, we aim to address the two security concerns listed above.

#TODO: roadmap to the chapters (storage security on itself then query security)

Chapter 2

Literature on Property Preserving Encryption Schemes

For traditional encryption schemes, security is based on indistinguishability (IND) of ciphertexts [19] or equivalently semantic security [18]. For schemes that are secure in this setting, the adversary should learn no information from seeing a ciphertext. However, this also means that one cannot process encrypted data efficiently. For instance, keyword search can only be achieved in linear time [9, 30].

Schemes allowing for better processing of encrypted data are devised but the ciphertext often possess some additional properties, thus leaking information about the underlying plaintext. Those schemes usually do not satisfy the usual indistinguishability notion of security because the additional properties can be exploited to generate attacks. In that case, one needs to understand what is the maximum level of security those schemes can offer. For our interest, we will focus on a specific encryption scheme of the type, known as deterministic encryption.

2.1 Deterministic Encryption

In CRYPTO 2007, Bellare et al. presented a construction of deterministic encryption (DE) scheme based on (randomized) public-key encryption schemes, and defined security notion for their scheme in the random oracle model [4]. In the follow-up papers, definitional equivalences of security notion without random oracles are studied [5, 8].

In this section, we will describe the key ideas in the construction, and analyse security of the scheme under the security notion defined in the original papers.

Key generation	Encryption(pk, x)	Decryption(pk, sk, y)
1 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow_{\$} \mathcal{K}(1^n)$	1 : $\omega \leftarrow H(\mathsf{pk} \| x)$	1 : $x \leftarrow \mathcal{D}(\mathsf{sk}, y)$
	2 : $y \leftarrow \mathcal{E}(\mathsf{pk}, x; \omega)$	2 : $\omega \leftarrow H(\mathsf{pk} \| x)$
	3 : **return** y	3 : **if** $\mathcal{E}(\mathsf{pk}, x; \omega) = y$ **return** x
		4 : **return** $\perp$

Figure 2.1: Deterministic encryption based on hashing

2.1.1 Construction based on Hash function

Let $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ be any randomized public-key encryption scheme, and $H(\cdot)$ a hash function. The idea of deterministic encryption is that instead of instead of letting the encryption algorithm $\mathcal{E}$ to determine its randomness, we will use the hash function $H(\cdot)$ to flip the coins deterministically.

For the construction to work, of course we demand $H(\mathsf{pk} \| x) \in \mathrm{Coin}_{\mathsf{pk}}(|x|)$, where $\mathrm{Coin}_{\mathsf{pk}}$ denotes the set of coins of $\mathcal{E}(\mathsf{pk}, x)$. Intuitively, this scheme is secure because for an adversary to recover the message, he effectively has to know the underlying randomness used by the encryption algorithm, but for a secure hash function the adversary cannot do so.

2.1.2 Generalization of Previous Construction

The construction above can be generalised. Instead of hashing, we can use trapdoor permutation to generate the coins. We shall first define trapdoor permutation formally.

Definition 1 (Trapdoor permutation) *A trapdoor function is a family of functions that is easy to compute but hard to invert. However, if given some additional information, known as the 'trapdoor', the inversion can be computed efficiently.*
Formally, let $F = \{f_k : D_k \to R_k\}(k \in K)$ be a collection of one-way functions, then F is a trapdoor function if the following holds:

- *There exists a probabilistic polynomial time (PPT) sampling algorithm Gen such that $Gen(1^n) = (k, t_k)$ and $t_k \in \{0,1\}^*$ satisfies that $|t_k|$ is polynomial in length. Each t_k is called the trapdoor corresponding to k.*
- *Given input k, there exist a PPT algorithm that outputs $x \in D_k$.*
- *For any $k \in K$, there exist a PPT algorithm that computes f_k correctly.*
- *For any $k \in K$, there exist a PPT algorithm A such that $y = A(k, f_k(x), t_k)$, and $f_k(y) = f_k(x)$. That is, the function can be inverted efficiently with the trapdoor.*

Key generation	Encryption(pk, x)	Decryption(pk, sk, y)
1 : $(\phi, \tau) \leftarrow_{\$} \mathcal{G}(1^n)$	1 : $(\phi, \bar{\mathsf{pk}}, p) \leftarrow \mathsf{pk}$	1 : $(\tau, \bar{\mathsf{sk}}) \leftarrow \mathsf{sk}$
2 : $s \leftarrow_{\$} \{0,1\}^n$	2 : $y \leftarrow F(\phi, x)$	2 : $y \leftarrow \mathcal{D}(\bar{\mathsf{sk}}, c)$
3 : $(\bar{\mathsf{pk}}, \bar{\mathsf{sk}}) \leftarrow_{\$} \mathcal{K}(1^n)$	3 : $\omega \leftarrow GetCoins(F, \phi, x, s)$	3 : $x \leftarrow \bar{F}(\tau, y)$
4 : $\mathsf{pk} \leftarrow (\phi, \bar{\mathsf{pk}}, s)$	4 : $c \leftarrow \mathcal{E}(\mathsf{pk}, y; \omega)$	4 : **return** x
5 : $\mathsf{sk} \leftarrow (\tau, \bar{\mathsf{sk}})$	5 : **return** c	
6 : **return** $(\mathsf{pk}, \mathsf{sk})$		

Figure 2.2: Deterministic encryption based on trapdoor permutations

- *For any $k \in K$, without the trapdoor t_k, the adversary of any PPT adversary*

$$\mathsf{Adv}^{\text{trapdoor}}_{\mathcal{A},Trapdoor}(n) = \Pr\left[\begin{array}{c}(t, t_k) \leftarrow Gen(1^n), x \leftarrow D_k, y \leftarrow f_k(x), \\ z \leftarrow \mathcal{A}(1^n, k, y) : f_k(x) = f_k(y)\end{array}\right]$$

  is negligible.

For ease of notation, we write trapdoor permutation as $(G, F, \bar{F})$, where G is the key generation algorithm, F is the trapdoor permutation, and $\bar{F}$ is the inverse. Further, we write $F^n(\cdot)$ to denote $F(F(\cdots F(\cdot)))$, that is F is applied to the input n times. Finally, we denote $GetCoins(F, \phi, \cdot, \cdot)$ to be a pseudo-random generator based on one-way function F and its public key ϕ. Such construction can be found in the [7, 34, 17].

The generalised deterministic encryption scheme has a very similar construction to the case where hash function is used. The main differences are: (1) instead of encrypting the plaintext straight away, the trapdoor permutation is applied to the plaintext before using the probabilistic encryption algorithm $\mathcal{E}$, and (2) the trapdoor function is used to generate the randomness for the encryption scheme.

2.1.3 Usefulness of Deterministic Encryption

There are numerous searchable encryption schemes with strong security guarantees include [9, 20, 1, 3, 11, 10] in the public-key setting, and [30, 16, 13] in the symmetric-key setting. However, all the schemes require linear search time, which is not ideal for large databases.

On the other hand, researchers have proposed sub-linear time searchable encryption in [28, 2, 21, 15, 23, 24, 26, 22, 12, 32]. However, security of these schemes are usually not analysed, which means that they can be vulnerable to various type of attacks. Deterministic encryption is one of the first schemes that is efficient in encrypting database and making queries, with provable security guarantees. In particular, for deterministic encryption, searching for equality can be done in log-time with binary search [33], or log-log-time with more advanced Van Emde Boas tree [31]. In our research, we will focus mainly on the database application of deterministic encryption.

Experiment $\mathrm{EXP}_{\mathrm{I}}^{\mathrm{IND}}(n)$

1 : $st \leftarrow I_c(1^n)$
2 : $(m_0, m_1) \leftarrow I_m(st)$
3 : $b \leftarrow_{\$} \{0, 1\}$
4 : $c \leftarrow \mathrm{Encryption}(pk, m_b)$
5 : $b^{'} \leftarrow I_g(\mathsf{pk}, c, st)$

Figure 2.3: IND game for deterministic encryption

2.1.4 Security of Deterministic Encryption

SECURITY NOTION IN THE ORIGINAL PAPER. As the scheme is deterministic, it is impossible to meet classic IND-CPA security [19]. In the original papers for deterministic encryption [4, 5, 5], the authors presented a set of security notions based on the traditional semantic security and IND security with a less powerful adversary, and proved equivalence of the notions. For our interest, we will present the IND adversary.

An IND adversary is a triple $I = (I_c, I_m, I_g)$ of PPT algorithms. I_c is an algorithm that, given the security parameter as the input, generates a state that will be used by I_m. I_m then outputs two plaintexts (m_0, m_1) given the state passed on by I_c. The challenger picks a bit b randomly from $0, 1$ and encrypts m_b. The encrypted message is then sent back to the adversary. Finally, the adversary guesses the bit b by running I_g with public key, the encryption of m_b, and the state generated by I_c.

We say that a scheme is IND secure, if for any IND-adversary I, the probability that I guesses the bit b right is only negligibly higher than half. The adversary presented here has a few key differences to the traditional IND-adversary:

- The algorithms I_c and I_m accessed by the adversary have no access to the public key.
- The message space has high min-entropy [25].

Both of the assumptions on the power of the adversary are very strong. In practice, one can hardly expect the adversary to gain access to the public key only at the guessing stage. In the same vein, in many applications, the message space in fact has very low entropy. Hence, security in this notion does not necessarily imply security in practice.

SECURITY NOTION OF IND-DCPA. Alternatively, IND-DCPA (indistinguishability under distinct chosen-plaintext attack) is defined in [6]. As deterministic encryption will always leak plaintext equality, the idea of IND-DCPA is that the adversary should only make distinct queries to the oracle.

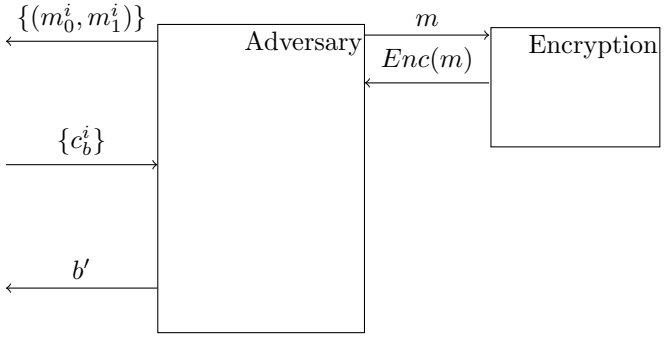

Figure 2.4: Cryptographic game of IND-DCPA

In the game, the adversary queries the challenger pairs of messages $(m_0^1, m_1^1), (m_0^2, m_1^2), \cdots, (m_0^q, m_1^q)$, where $m_b^1, \cdots, m_b^q$ are all distinct for $b \in \{0, 1\}$. The challenger then randomly picks $b \in \{0, 1\}$, encrypts all m_b^i for $i \in \{1, \cdots, q\}$ and sends them back to the adversary. The adversary, with the help of the encryption oracle, has to guess the bit b. He returns $b^{'}$ as his guess and he wins if $b = b^{'}$.

Just like the security notion in the original paper, assumption of IND-DCPA is hard to meet in practice. In particular, for databases, there is no reason why the same plaintext cannot be encrypted twice.

2.2 Attack on Database Encrypted using Deterministic Encryption

Deterministic encryption has fairly strong security guarantees if the assumptions of the security notion are met. However, for database applications, the assumptions are impossible to achieve. In [27], Naveed et al. have proposed frequency attack on databases encrypted using deterministic encryption, with auxiliary information on the distribution of the plaintext. They have tested their attack on National Inpatient Sample (NIS) database of the Healthcare Cost and Utilization Project (HCUP). In the experiment, they are able to recover almost all information of the database.

2.2.1 Notation

We denote $c = (c_0, c_1, \cdots, c_n)$ to be the list of entries for a column in the database. We write $Hist(c)$ to be the function that generates the histogram of c. On input c_i, $Hist(c)$ will output the frequency of c_i inside c. Further, we write $vSort(\cdot)$ to be the function that sorts a histogram in decreasing order of frequency. So $vSort(Hist(c))[0]$ will be the element in c that has the

highest frequency. Finally, we define $Rank_\psi(c_i)$ to be the rank of c_i in the sorted histogram ψ. That is, if c_i is the most frequent ciphertext in c, $Rank_{vSort(Hist(c))}(c_i) = 0$.

2.2.2 The attack

The attack used in the paper against deterministically encrypted databases is the most basic and famous attack, known as frequency attack. The attack relies on auxiliary information on the plaintext. Let c be the target of attack, and z be some auxiliary dataset. The attack can be written as follows:

Attack(z, c)

1 : compute $\psi \leftarrow vSort(Hist(c))$
2 : compute $\pi \leftarrow vSort(Hist(z))$
3 : $output A : C_k \rightarrow M_k$ such that
$$A(c) = \pi[Rank_\psi(c)]$$

The attack does no more than just assigning the most frequent plaintext of the auxiliary dataset to the most frequent ciphertext in the target database. But for statistical databases, the distributions usually do not change, so the attack has a good chance of recovering the plaintexts.

On a side note, additional datasets may be very easy to obtain in some cases. In the paper that the attack described is based on, the auxiliary data is the Texas Inpatient Public Use Data File (PUDF), which is publicly available online. Usage of the database only requires the user to sign a data use agreement, but there is no reason to assume that an evil adversary will keep his promise.

#TODO: OPE, full homomorphic encryption

2.3 Discussion

In this chapter, we have shown that deterministic encryption is very useful in encrypting databases. However, for such application, the assumptions of the security notion introduced in [4, 8, 5] cannot be met, so attacks such as frequency attack [27] can be deployed to exploit the cryptosystem.

As the security notions defined in the original paper are not useful against statistical attacks, we wish to construct new security notions that have security guarantee against such attacks. Furthermore, we want to find schemes that are efficient in encryption and database queries, and prove their security in the new security notion.

Chapter 3

Efficient Searchable Databases

In this chapter, we will first formally define relational databases used in our constructions, and then give three constructions based on padding. In the next chapter, we will prove that two of the constructions are secure under our definition of security, and the third one is secure in one definition but not secure in the other.

3.1 Relational Database

A relational database is a collection of tables organized based on the relational model proposed in [14]. Each row in the table represents an entity (e.g. a student or a transaction). Each column on the other hand, corresponds to an attribute (e.g. age or transaction fee). For each attribute, we call the set of all possible values *attribute space*, and denote it by C_i, where i is the index on the column.

For r a row of the database, we write $r||x$ to mean r append with column(s) x (Here, x is a row too.). We write (D, E) to mean append rows of E to D. Let Enc be some encryption scheme on one cell of the database, we abuse the notation to write $Enc(d_i) = Enc(d_{i,1})||Enc(d_{i,2})||\cdots||Enc(d_{i_m})$ if the public key $\mathsf{pk}$is obvious in the context.

3.2 Encryption Scheme on Relational Database

Just like usual encryption schemes, we define encryption scheme on relational database to be a triple (Kg, Enc, Dec), where Kg is the key generation function, Enc is the (deterministic) encryption function and Dec is the decryption function. We do not define how searching should work on the database here because that depends on the underlying data structure deployed by the database system and the searching algorithm used. In any case, as long as our database

is sane in size (larger than the trivial one by a constant factor), time complexity of searching will not be affected for any efficient searching algorithm.

As a standard correctness requirement, we demand for any database d and valid $(\mathsf{pk}, \mathsf{sk})$ pair generated by Kg, $d = Dec(Enc(d, \mathsf{pk}), \mathsf{sk})$.

3.3 Constructions of Encryption Schemes on Relational Database

All three constructions presented here are based on deterministic encryption in [4, 5, 8]. We denote the deterministic encryption scheme used in our construction as $(\overline{Kg_1}, \overline{Enc_1}, \overline{Dec_1})$, where $\overline{Kg}, \overline{Enc}, \overline{Dec}$ are the key generation, encryption and decryption functions respectively. Here we are not specifying which deterministic encryption to use. The only requirement is that the scheme is deterministic in nature, and satisfies the privacy (PRIV) notion of security defined in the papers mentioned above. For the constructions, we also rely on a probabilistic encryption scheme which we shall call it $(\overline{Kg_2}, \overline{Enc_2}, \overline{Dec_2})$,

As deterministic encryption reveals frequency, the idea is to pad the encrypted database in a way such that the frequencies of all possible values of the attributes are indistinguishable.

Without loss of generality, we will assume that our database has only one column. Multi-column database can be viewed as a single column database with attributes being combination of the attributes in the original database.

3.3.1 Full Padding Scheme

The most trivial way to hide frequency would be to pad all other possible values of the attribute (recall that we assume that our database has only one column) given an actual row to the database. An auxiliary column is used to mark if the row is real or fake.

Define $C = C_1 \times C_2 \times \cdots C_m$ to be the attribute space of the database. Let Then the full padding scheme `Padding1` can be written as `Padding1` $= (Kg, Enc, Dec)$, where $Kg = \overline{Kg_1}$, and encryption and decryption algorithms are specified below:

Enc(m)

```
1 : for x ∈ C
2 :   if x = m
3 :     D ← (D, (Enc_1(m)||Enc_2(True)))
4 :   else
5 :     D ← (D, (Enc_1(x)||Enc_2(FALSE))
```

Dec(D)

```
1 : E = ()
2 : (d_1, d_2, ··· , d_m) ← D
3 : for i = 1, ··· , m
4 :   c||x ← d_i
5 :   if Dec_2(x) = True
6 :     E ← (E, Dec(c))
7 : return E
```

This scheme is NOT efficient by any means. In particular, for each actual row, we need to append $|C| = |C_1 \times C_2 \times \cdots \times C_m| = \prod_i |C_i|$ dummy rows. That is not practical at all. However, for theoretical purposes, this scheme is relatively easy to understand and its nice properties enables for simple proofs of security.

3.3.2 Fixed Padding Scheme 1

To work around the inefficiency in the full padding scheme, we want to have some scheme that have a short and fixed padding. This is impossible with a deterministic padding scheme because the padded rows themselves will reveal frequency, so we must do it probabilistically.

Notation Let $\Pi_0 : C \to [0, 1]$ be the probability mass function of the input attributes. Without loss of generality, we can enumerate elements of C as $\{1, \cdots, k\}$, then $\Pi_0(i)$ is the frequency of occurrence of the i-th attribute.

Let p be the number of additional insertions we make per real entry, and $\Pi_1, \Pi_2, \cdots, \Pi_p$ be the corresponding probability mass functions for each additional insertion which we will specify later, then the following condition is necessary for the scheme to be secure:

$$\frac{1}{p+1}(\sum_{i=0}^{p} \Pi_i(j)) = \frac{1}{k} \qquad \forall j \in \{1, \cdots, k\}, \tag{3.1}$$

constrained to $\sum_j \Pi_i(j) = 1$ for all $i \in \{1, \cdots, p\}$. In plain words, we want all attributes to appear the same number of times in expectation. In the equation, Π_0 and k are fixed. We aim to find a lower bound on p. It is clear that Π_i's are probability mass functions so they must be non-negative, so we have

$$\begin{aligned}
&\frac{1}{p+1}(\sum_{i=0}^{p} \Pi_i(j)) \geq \frac{1}{p+1}\Pi_0(j) \qquad \forall j \in \{1, \cdots, k\} \\
\implies &\frac{1}{k} \geq \frac{1}{p+1}\Pi_0(j) \qquad \forall j \in \{1, \cdots, k\} \\
\implies &p \geq k \cdot \Pi_0(j) - 1 \qquad \forall j \in \{1, \cdots, k\} \\
\implies &p \geq k \cdot \max_j \Pi_0(j) - 1
\end{aligned} \tag{3.2}$$

This agrees with intuition. If $\max_j \Pi_0(j) = \frac{1}{k}$, then $p \geq 0$, in which case we can encrypt without any padding. On the other hand, if $\max_j \Pi_0(j) = 1$, then $p \geq k - 1$, which is essentially the full padding scheme introduced earlier.

Choose p and Π_i For the first fixed padding scheme, we propose to choose the smallest p possible, and $\Pi_i(j)$ to be the average of the remainder in equation 3.1:

$$p = \lceil k \cdot \max_j \Pi_0(j) - 1 \rceil, \tag{3.3}$$

$$\Pi_i(j) = \frac{1}{p}\left(\frac{p+1}{k} - \Pi_0(j)\right) \qquad \forall i \in \{1, \cdots, p\},\ j \in \{1, \cdots, k\}. \tag{3.4}$$

To illustrate this scheme in practice, we consider the following example.

Example 1 *Let* $\Pi_0(1) = \frac{2}{3},\ \Pi_0(2) = \frac{1}{6},\ \Pi_0(3) = \frac{1}{6}$, *then*

$$\begin{aligned} p &= \lceil k \cdot \max_j \Pi_0(j) - 1 \rceil \\ &= \lceil 3 \cdot \frac{2}{3} - 1 \rceil \\ &= 1, \\ \Pi_1(1) &= \frac{1}{1}\left(\frac{1+1}{3} - \Pi_0(1)\right) \\ &= \frac{2}{3} - \frac{2}{3} \\ &= 0, \\ \Pi_1(2) = \Pi_1(3) &= \frac{1}{1}\left(\frac{1+1}{3} - \frac{1}{6}\right) \\ &= \frac{1}{6}. \end{aligned}$$

Variation to the scheme As we can see, the number of paddings we have to apply depends exclusively on the attribute of the highest frequency (or min-entropy from information-theoretic point of view). This value can be reduced by splitting that attribute into many pieces (recursively if necessary). For example, if we want to split attribute 1 in example 1 into two pieces, we can do the following:

1. Re-define $\Pi_0(1) = \frac{1}{3},\ \Pi_0(2) = \frac{1}{6},\ \Pi_0(3) = \frac{1}{6},\ \Pi_0(4) = \frac{1}{3}$.
2. Compute p, and $\Pi_1, \cdots, \Pi_p$ (In this case $p = 1$ still, but for the interesting cases, p will be smaller.).
3. For attribute 2 and 3, use the original encryption scheme.
4. For attribute 1, rename it to attribute 1 or attribute 4 with probability 50% each, and use the original encryption scheme to encrypt it.

5. To decrypt, attribute 2 and 3 are straight forward. For attribute 1 and 4, the user only has to bookkeep the fact that attribute is a variant of attribute 1 to decrypt correctly.

Bibliography

[1] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. *Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions*, pages 205–222. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[2] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 563–574, New York, NY, USA, 2004. ACM.

[3] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. *Public Key Encryption with Keyword Search Revisited*, pages 1249–1259. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[4] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. *Deterministic and Efficiently Searchable Encryption*, pages 535–552. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[5] Mihir Bellare, Marc Fischlin, Adam O'Neill, and Thomas Ristenpart. *Deterministic Encryption: Definitional Equivalences and Constructions without Random Oracles*, pages 360–378. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[6] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Authenticated encryption in ssh: Provably fixing the ssh binary packet protocol. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 1–11, New York, NY, USA, 2002. ACM.

[7] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.

[8] Alexandra Boldyreva, Serge Fehr, and Adam O'Neill. *On Notions of Security for Deterministic Encryption, and Efficient Constructions without Random Oracles*, pages 335–359. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[9] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. *Public Key Encryption with Keyword Search*, pages 506–522. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[10] Dan Boneh and Brent Waters. *Conjunctive, Subset, and Range Queries on Encrypted Data*, pages 535–554. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[11] Xavier Boyen and Brent Waters. *Anonymous Hierarchical Identity-Based Encryption (Without Random Oracles)*, pages 290–307. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[12] R. Brinkman, L. Feng, J. Doumen, P. H. Hartel, and W. Jonker. Efficient tree search in encrypted data. *Information Systems Security*, 13(3):14–21, 2004.

[13] Yan-Cheng Chang and Michael Mitzenmacher. *Privacy Preserving Keyword Searches on Remote Encrypted Data*, pages 442–455. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[14] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.

[15] Ernesto Damiani, S. De Capitani Vimercati, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Balancing confidentiality and efficiency in untrusted relational dbmss. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, CCS '03, pages 93–102, New York, NY, USA, 2003. ACM.

[16] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. `http://eprint.iacr.org/2003/216/`.

[17] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, STOC '89, pages 25–32, New York, NY, USA, 1989. ACM.

[18] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 365–377, New York, NY, USA, 1982. ACM.

[19] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270 – 299, 1984.

[20] Philippe Golle, Jessica Staddon, and Brent Waters. *Secure Conjunctive Keyword Search over Encrypted Data*, pages 31–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[21] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, pages 216–227, New York, NY, USA, 2002. ACM.

[22] Hakan Hacıgümüş, Bala Iyer, and Sharad Mehrotra. *Efficient Execution of Aggregation Queries over Encrypted Relational Databases*, pages 125–136. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[23] Bijit Hore, Sharad Mehrotra, and Gene Tsudik. A privacy-preserving index for range queries. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 720–731. VLDB Endowment, 2004.

[24] Bala Iyer, Sharad Mehrotra, Einar Mykletun, Gene Tsudik, and Yonghua Wu. *A Framework for Efficient Storage Security in RDBMS*, pages 147–164. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[25] R. Koenig, R. Renner, and C. Schaffner. The operational meaning of min- and max-entropy. *ArXiv e-prints*, July 2008.

[26] Jun Li and Edward R. Omiecinski. *Efficiency and Security Trade-Off in Supporting Range Queries on Encrypted Databases*, pages 69–83. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[27] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 644–655, New York, NY, USA, 2015. ACM.

[28] Gultekin Ozsoyoglu, David A. Singer, and Sun S. Chung. *Anti-Tamper Databases*, pages 133–146. Springer US, Boston, MA, 2004.

[29] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: Protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 85–100, New York, NY, USA, 2011. ACM.

[30] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, SP '00, pages 44–, Washington, DC, USA, 2000. IEEE Computer Society.

[31] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6(3):80 – 82, 1977.

[32] Hui Wang and Laks V. S. Lakshmanan. Efficient secure query evaluation over encrypted xml databases. In *Proceedings of the 32Nd International Conference on Very Large Data Bases*, VLDB '06, pages 127–138. VLDB Endowment, 2006.

[33] Louis F. Williams, Jr. A modification to the half-interval search (binary search) method. In *Proceedings of the 14th Annual Southeast Regional Conference*, ACM-SE 14, pages 95–101, New York, NY, USA, 1976. ACM.

[34] A. C. Yao. Theory and application of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 80–91, Nov 1982.