Assuring Certified Database Utility in Privacy-Preserving Database Fingerprinting

Mingyang Song[1], Zhongyun Hua[1,*], Yifeng Zheng[2], Tao Xiang[3], Guoai Xu[1], and Xingliang Yuan[4]

[1]*School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen*
[2]*Department of Electrical and Electronic Engineering, The Hong Kong Polytechnic University*
[3]*College of Computer Science, Chongqing University*
[4]*School of Computing and Information Systems, University of Melbourne*

Abstract

Fingerprinting techniques allow a database owner (DO) to embed unique identifiers within relational databases to trace unauthorized redistribution. To protect its interests, the DO often prioritizes maximizing fingerprint robustness, resulting in extensive modifications to the databases. However, excessive modifications may significantly degrade the databases' utility, making recipients hesitant to purchase databases that seem compromised when they cannot evaluate the maximum number of modified bits made during fingerprinting process. Current database fingerprinting techniques focus only on boosting fingerprint robustness, without providing recipients any mechanism to verify the degree of modifications.

This paper, for the first time, addresses the research gap in providing recipients the ability to verify the maximum number of modified bits in database fingerprinting. We introduce a fuzzy perturbation verification (FPV) protocol, which enables a verifier to assess the extent of modifications made to a bit-string by a prover while keeping the exact modification positions and original bit-string confidential. Using the FPV protocol, we propose UtiliClear, a novel database fingerprinting scheme that allows the recipient to specify and verify the modification degree within the fingerprinted database. We theoretically validate that UtiliClear enables recipients to verify the extent of modifications during the fingerprinting process while maintaining fingerprint robustness, database utility, and data privacy. To demonstrate its effectiveness, we evaluate UtiliClear's performance using large real-world datasets. The experimental results and analysis indicate that UtiliClear incurs modest overhead while preserving fingerprint robustness and database utility comparable to existing state-of-the-art schemes.

1 Introduction

Currently, many database owners (DOs) gather vast amounts of user data to build specialized relational databases (collections of data records sharing the same set of attributes [1]) across various fields, such as patient health records in medicine and consumer behavior data in business [2, 3]. These databases are sold to multiple recipients for profit [4, 5], with institutions and companies often purchasing them for specific purposes [6, 7], like predictive analytics in healthcare, targeted marketing strategies, or trend analysis in retail [8, 9]. For example, some institutions (e.g., Flatiron Health [10]) may gather healthcare data from patients to create comprehensive medical databases, and pharmaceutical companies purchase them to analyze patient demographics, drug interactions, and potential treatment outcomes [11, 12].

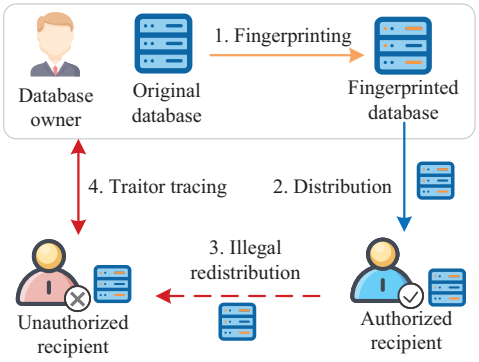


Figure 1: Illustration of database fingerprinting.

In the database distribution paradigm, dishonest recipients may resell the purchased database to earn illegal profits, adversely affecting the economic interests of the DO. The DO prioritizes measures to prevent illegal redistribution and to trace dishonest recipients [11, 13]. Fingerprinting is a widely studied technique that enables the DO to trace piracy in database distribution. As shown in Figure 1, the DO embeds a unique identifier (i.e., fingerprint) associated with each authorized recipient into the distributed database. If the authorized recipient illegally redistributes the fingerprinted database to unauthorized recipients, the DO can identify and trace the dishonest recipient by extracting the fingerprint from the redistributed database. Various database

*Corresponding author.

fingerprinting schemes [14–17] have been developed to prevent illegal database redistribution. Since recipients might attempt to damage or remove fingerprints to obscure their unauthorized activities, current research focuses on enhancing fingerprint robustness while preserving high utility of the database. Moreover, recent research by Ji *et al*. [18] integrates data privacy protection with fingerprinting into a unified mechanism. By leveraging the inherent randomness of fingerprinting, it achieves provable entry-level differential privacy (DP) for the fingerprinted database.

Research gap. All database fingerprinting schemes share the common objective of enabling the DO to identify dishonest recipients. In practice, the DO may apply extensive modifications to the database to enhance fingerprint robustness. However, recipients are primarily concerned with the database utility to support accurate data analysis and mining [19, 20]. Excessive modifications may significantly degrade the database's utility [18]. Without the ability to assess the degree of changes, recipients may hesitate to invest in databases that seem compromised in utility. Current database fingerprinting techniques primarily focus on strengthening fingerprint robustness but do not provide mechanisms for recipients to verify the maximum number of bits modified during the fingerprinting process. This highlights a significant research gap.

However, allowing the recipients to verify the extent of modifications while maintaining the fingerprint robustness presents a significant challenge. To prevent fingerprint from being damaged or removed, the modifications made to the database must remain confidential from the recipients, which hinders the modification extent verification. This leads us to pose the technique challenge:

- *How to enable a recipient to verify the extent of modifications made to the database during the fingerprinting process without degrading fingerprint robustness?*

A feasible but impractical solution. Zero-knowledge proof (ZKP) protocols [21–24] for arbitrary circuits have gained significant attention recently. These protocols allow one party with a secret input w to convince another party that $\mathcal{C}(w)$ is the correct output of a circuit $\mathcal{C}$ without revealing any additional information. A straightforward approach to address the aforementioned challenge is to apply an existing ZKP protocol to the fingerprint embedding circuit, which is configured to a fixed extent of modifications. The DO, who holds the secret input (i.e., the original database), interacts with the recipient through the ZKP protocol to execute the fingerprint embedding circuit. This interaction convinces the recipient that the output (i.e., the fingerprinted database) conforms to the specified fingerprinting circuit, ensuring that no intermediate information is leaked during the execution of ZKP protocol and thus maintaining the fingerprint robustness. However, ZKP is a resource-intensive cryptographic tool. For instance, as discussed in [21], using the state-of-the-art ZKP protocol on an Amazon EC2 instance (type m5.4xlarge with 16 vCPUs and 64GB of RAM) incurs 320 seconds of computation time, 4.2GB of communication overhead, and 400MB of memory cost to compute the multiplication of two 512×512 matrices over a 61-bit field. As a result, based on the experiments in [21], it is estimated that applying the ZKP protocol under the same hardware conditions to the fingerprint embedding algorithm in [18] would require approximately 1000 hours of computation time, 32TB of bandwidth cost, and 400MB of memory cost to process a 34 GB database containing about 230 million records, which is prohibitively expensive.

Our contributions. This paper introduces an efficient solution for verifying the extent of modifications in fingerprinted databases. We first design a fuzzy perturbation verification (FPV) protocol to address the trust issue in modifications when a prover sends a modified bit-string to a verifier. The FPV protocol enables the verifier to determine the maximum number of modifications within a bit-string. In this protocol, the prover submits a modified bit-string to the verifier, who then checks that the number of modifications does not exceed a predetermined threshold. The core concept of the FPV protocol is to map the bitwise differences between the original and modified bit-strings onto a codeword of an error-correcting code (ECC). During execution, the prover must demonstrate to the verifier that the embedded error codeword can be decoded within the ECC's error tolerance. Unlike traditional commitment schemes [25–28], which expose the original data to the verifier during the commitment open process, the FPV protocol maintains full-hiding. It ensures that the exact positions of the modifications and the original bit-string remain confidential to the verifier.

Using the FPV protocol, we introduce UtiliClear, a practical database fingerprinting scheme that enables recipients to specify and verify the extent of modifications. A database consists of records, each containing several ordered attributes. Considering the potential size of a database, UtiliClear divides each column of attribute values into equally sized groups, and processes the attribute values within each group separately. Additionally, each attribute value is typically divided into significant and insignificant bits (e.g., leading and trailing bits for an integer attribute). Since significant bits are crucial for maintaining data utility, fingerprints are typically embedded by modifying insignificant bits.

As a result, UtiliClear first separates the attributes values within each group to obtain a significant bit-string and an insignificant bit-string. For each group, it then employs a commitment scheme [25] to ensure that the significant bit-string remains unmodified, while utilizing the FPV protocol to validate that modifications to the insignificant bit-string fall within a specified range. The advanced database fingerprinting technique described in [18] is utilized for embedding and extracting fingerprints. In summary, this paper makes the following contributions:

- This paper is the first to explore the concept of modification extent verification in database fingerprinting. We content that providing recipients with the ability to verify the maximum number of bits modified during the fingerprinting process is important for assuring database utility.
- We introduce the FPV protocol that enables a verifier to assess the maximum number of modifications made to a bitstring by the prover. Building on this foundation, we develop UtiliClear, a novel database fingerprinting scheme, which allows recipients to specify and verify the extent of modifications within the fingerprinted databases.
- We theoretically analyze that UtiliClear allows recipients to verify the extent of modifications made during fingerprinting while preserving fingerprint robustness, database utility, and data privacy. To demonstrate UtiliClear's practicality, we conduct experiments to evaluate its performance using large real-world datasets.

Insights and limitations. This paper is the first to enable recipients to verify the extent of modifications in database fingerprinting, thereby ensuring the fairness between the DO and the recipient. We design a verification mechanism that is compatible with existing bit-level fingerprinting methods. When applied with an existing database fingerprinting method (e.g., the scheme in [18]), the verification mechanism does not compromise the fingerprint robustness, database utility, or data privacy. However, implementing this verification mechanism inevitably introduces additional overhead for both the recipient and the DO, potentially affecting the overall efficiency of the fingerprinting system.

2 Preliminaries

We present the preliminaries used in our scheme. Table 1 lists the important symbols in this paper.

2.1 Error-Correcting Code

An $(n,k,2t+1)$-ECC scheme [29] encodes a message $m \in \{0,1\}^k$ into a codeword $cw \in \{0,1\}^n$ containing redundant bits. The m can be recovered even if up to t bits of cw are corrupted. The functions of ECC are described as follows:

- $\mathrm{EC}_{n,t}(m) \to (cw)$: Given a message m, this function encodes m into a codeword cw of length n with the maximum error tolerance of t bits.
- $\mathrm{DC}_{n,t}(cw') \to (m)$: Given a codeword cw', the original message m can be reconstructed if no more than t bits are corrupted (i.e., $\mathrm{Dis}(cw,cw') \le t$).

2.2 Pedersen Commitment

Pedersen commitment [25] is a cryptographic tool that allows a committer to hide a value temporarily. Later, the value can be disclosed to a verifier and validated as untampered. The commitment scheme are described as follows:

- $\mathrm{PC.Setup}(\kappa) \to (pp)$: It chooses two primes p and q of κ bits, and randomly selects two generators g and h from a subgroup of $\mathbb{Z}_p^*$ as the public parameters $pp = \langle g,h,q,p \rangle$.
- $\mathrm{PC.Com}(x,r,pp) \to (c)$: Given a secret value x and a random integer $r \in_R \mathbb{Z}_q$, it outputs the commitment $c = g^x \cdot h^r$.
- $\mathrm{PC.Open}(x,r,pp,c) \to (0/1)$: Given a commitment c, a revealed value x, and a random integer r, this function outputs 1 if $c = g^x \cdot h^r$. Otherwise, it outputs 0.

The commitment scheme satisfies properties of hiding and binding. **Hiding:** Given a commitment $c = g^x \cdot h^r$, it is computationally difficult to determine the secret value x before it is revealed. **Binding:** Given a commitment $c = g^x \cdot h^r$, it is computationally difficult to find different values $x' \neq x$ and r' such that $c = g^{x'} \cdot h^{r'}$.

Table 1: Important Symbols in This Paper.

Symbol	Description
$\mathbf{R}$	The original database.
$\overline{\mathbf{R}}$	The fingerprinted database.
r.*key*	The primary key of a record r.
r[i]	The i-th attribute value of r.
r[i][j]	The j-th bit of r[i].
r[i][a,b]	The substring of r[i] from position a to b.
l_f	The bit length of fingerprint.
n_c	The number of columns in $\mathbf{R}$.
n_g	The number of groups in each column.
cw	The codeword generated by the recipient.
cw'	The codeword with bit errors embedded.
$\mathrm{a} \oplus \mathrm{b}$	The bitwise XOR operation on a and b.
$d \in_R \mathrm{D}$	Randomly select an element d from D.
$\mathrm{C_a} \circ \mathrm{C_b}$	The Hadamard product of $\mathrm{C_a}$ and $\mathrm{C_b}$.
$J(\frac{a}{p})$	The Jacobi symbol of a and p.
$\mathrm{Perm}(\mathrm{D},\delta)$	Permute elements in D using a seed δ.
$\mathrm{Dis}(\mathrm{a},\mathrm{b})$	The Hamming distance of strings a and b.
$\mathrm{S}(\cdot)$	A pseudorandom sequence generator.
$\mathrm{S}_i(\cdot)$	The i-th output of S.

2.3 Goldwasser-Micali Cryptosystem

The Goldwasser-Micali (GM) cryptosystem [30] is semantic secure asymmetric encryption scheme, which consists of the following three algorithms:

- $\mathrm{GM.KeyGen}(\kappa) \to (pk,sk)$: Randomly choose two primes p and q of κ bits, and compute $N = p \cdot q$. Generate an integer ξ such that $J(\frac{\xi}{p}) = J(\frac{\xi}{q}) = -1$. The secret key is $sk = (p,q)$, and the public key is $pk = (N,\xi)$.
- $\mathrm{GM.Enc}(m,pk) \to (c)$: Randomly choose a uniform $\gamma \in_R \mathbb{Z}_N^*$. If $m = 0$, compute $c = \gamma^2 \bmod N$; otherwise, compute $c = \xi \cdot \gamma^2 \bmod N$.

- GM.Dec$(c, sk) \rightarrow (m)$: If $J(\frac{c}{p}) = J(\frac{c}{q}) = 1$, then $m = 0$; otherwise, $m = 1$.

The GM encryption scheme satisfies the property of XOR homomorphism, meaning GM.Enc$(a \oplus b, pk) =$ GM.Enc$(a, pk) \circ$ GM.Enc(b, pk).

2.4 Permutation Function

Given an array D, a permutation function Perm maps D to another array D′ by rearranging the positions of the elements in D, i.e., Perm$(D, \delta) \rightarrow D'$ [31]. The seed $\delta \in_R \Delta$ is used to generate a mapping that determines how the elements of D are rearranged to produce D′. The space of possible mappings, Δ, has a size of $|D|!$, where $|D|$ represents the number of elements in D.

2.5 Digital Signature

Digital signature [32] is used for electronic authentication of digital information. A digital signature scheme Sig consists of the following three algorithms:

- Sig.KeyGen$(\kappa) \rightarrow (sk_{sig}, pk_{sig})$: Given the bit length κ of the key, this algorithm generates and outputs a public-private signature key pair $\langle pk_{sig}, sk_{sig} \rangle$.
- Sig.Sign$(sk_{sig}, x) \rightarrow (sign_x)$: Given a message x, this algorithm uses the secret key sk_{sig} to sign the message and outputs a signature $sign_x$.
- Sig.Verify$(pk_{sig}, sign_x, x) \rightarrow (0/1)$: It uses pk_{sig} to verify the validity of signature $sign_x$ for the message x.

3 Problem Formulation

3.1 System Model

The system model of UtiliClear involves a database owner (DO) and multiple recipients. The DO sells its database to various recipients for profit. To trace illegal redistribution, the DO embeds a unique fingerprint corresponding to each authorized recipient into the sold database. A authorized recipient purchases the database from the DO. To achieve accurate data analysis and mining, the recipient is primarily concerned with the extent of modifications to avoid paying heavily for utility reduced database.

3.2 Threat Model

In our system, both the DO and recipients are considered untrusted entities, with potential threats outlined as follows. A rational DO would not intentionally compromise the utility of its sold database. However, to ensure its profit, the DO may modify more bits than the recipient specified during fingerprint embedding, increasing fingerprint density to enhance robustness. These excessive modifications may significantly reduce the database's utility. A dishonest recipient may redistribute the database to unauthorized recipients to obtain illegal benefits.

3.3 Design Goals

We aim to design a database fingerprinting scheme, UtiliClear, with the following objectives. (i) **Modification extent verification:** It allows recipients to specify and verify the maximum number of modified bits during the fingerprinting process. (ii) **Fingerprint robustness, database utility, and data privacy preservation:** UtiliClear employs the advanced fingerprinting technique in [18] for embedding and extracting fingerprints. It supports modification extent verification while preserving fingerprint robustness, database utility, and data privacy, comparable to the employed fingerprinting technique.

3.4 Definitions

We provide formal definitions of the proposed FPV and the UtiliClear scheme.

Definition 1. *An (n,t)-*FPV *protocol consists of three algorithms:* FPV.Setup*,* FPV.Lock*, and* FPV.Verify*.*

- FPV.Setup$(\kappa) \rightarrow (pk, sk)$. *This algorithm is executed by the prover, taking the key's bit length κ as input, and outputs a public-private key pair $\langle pk, sk \rangle$.*
- FPV.Lock$(sk, pk, x) \rightarrow (y, vp)$. *This algorithm takes a bit string $x \in \{0,1\}^n$, along with the prover's secret key sk and public key pk, as inputs. It outputs a locked string y and verification parameters vp.*
- FPV.Verify$(y, x', vp, k, t) \rightarrow (1/0)$. *Given a modified string $x' \in \{0,1\}^n$, the verification parameters vp, the bit length k of the ECC's input message, the ECC's maximum error tolerance t, and the locked string y, it verifies whether the number of modified bits in x' is no more than t.*

An (n,t)-FPV protocol enables a verifier to assess the maximum number of modifications within a bit-string. In this protocol, the prover submits a modified bit-string to the verifier, who then checks that the modification number does not exceed a predetermined threshold. This verification process ensures that the exact positions of the modifications and the original bit-string remain confidential from the verifier.

Given a secret bit-string x of length n held by the prover, the FPV.Lock algorithm allows the prover and the verifier to collaboratively lock x before any modification. The prover can later modify x, resulting in a new bit-string x', which is then revealed to the verifier. The FPV.Verify algorithm enables the verifier to check whether the prover made no more than t bits of modification to x (i.e., Dis$(x, x') \leq t$) without revealing the specific bitwise differences between x and x'.

A secure FPV protocol should exhibit two key properties: fuzzy-binding and full-hiding. Fuzzy-binding ensures that the modified bit-string x' can pass verification only when $\mathrm{Dis}(x,x') \leq t$. Full-hiding ensures that the verifier cannot infer the exact differences between x and x' during the execution of FPV. We provide the formal presentation of a secure FPV protocol as Definition 2.

Definition 2. *An* (n,t)-FPV *(*FPV.Setup*,* FPV.Lock*, and* FPV.Verify*) is secure if it has the following properties:*

- 1) Fuzzy-binding: *For any adversary* $\mathcal{A}$ *(i.e., the prover):*

$$\Pr\left[\begin{array}{l}\mathrm{FPV.Setup}(\kappa)\rightarrow(pk,sk),\\ \mathcal{A}(pk,sk)\rightarrow(x,x',y'):\\ \mathrm{FPV.Lock}(sk,pk,x)\rightarrow(y,vp)\wedge\\ \mathrm{FPV.Verify}(y',x',vp,k,t)\rightarrow 1\wedge\\ \mathrm{Dis}(x,x')>t\end{array}\right]\leq \mathrm{negl}(\kappa).$$

- 2) Full-hiding: *Given the public key pk, and a modified string* x'*, no adversary (i.e., the verifier) can identify the differing positions between the modified string* x' *and the original string x, during the execution of* FPV.

Definition 3. *UtiliClear is a five-tuple of polynomial-time algorithms, defined as follows.*

- $\mathrm{UtiliClear.Setup}(\kappa)\rightarrow(pp,kp_o,kp_r)$. *It takes the key's bit length* κ *as input and generates the following outputs: public parameters pp and a public-private key pair* $kp_o=\langle pk,sk\rangle$ *for the DO, as well as a public-private signature key pair* $kp_r=\langle pk_{sig},sk_{sig}\rangle$ *for the recipient.*
- $\mathrm{UtiliClear.Preproc}(\mathbf{R},pp,kp_o,n_c,n_g,\mathrm{P})\rightarrow(\Psi,\Phi,\mathrm{OP},\mathrm{VP})$. *The algorithm takes the original database* $\mathbf{R}$*, public parameters pp, the DO's public-private key pair* kp_o*, the number of columns* n_c*, the number of groups* n_g*, and the positions of insignificant bits* $\mathrm{P}=\{s_i,e_i\}_{i=1}^{n_c}$ *as inputs. It outputs a commitment set* Ψ *and a verification parameter set* OP *for the significant bits, along with a locked string set* Φ *and a verification parameter set* VP *for the insignificant bits.*
- $\mathrm{UtiliClear.Fingembed}(\mathbf{R},kp_r,n_c,n_g,\mathrm{P},t,sk,ID)\rightarrow(\mathrm{f},\overline{\mathbf{R}})$. *The algorithm takes the original database* $\mathbf{R}$*, the recipient's signature key pair* kp_r*, the number of columns* n_c*, the number of groups* n_g*, the positions of insignificant bits* $\mathrm{P}=\{s_i,e_i\}_{i=1}^{n_c}$*, the number of modifiable bits in each group t, the DO's secret key sk, and the recipient's identity ID as inputs. It outputs the recipient's fingerprint* f *and fingerprinted database* $\overline{\mathbf{R}}$.
- $\mathrm{UtiliClear.Verify}(\overline{\mathbf{R}},pp,n_c,n_g,k,t,\mathrm{P},\Psi,\mathrm{OP},\Phi,\mathrm{VP})\rightarrow(0/1)$. *The algorithm takes the fingerprinted database* $\overline{\mathbf{R}}$*, the DO's public parameters pp, the number of columns* n_c*, the number of groups* n_g*, the bit length k of the ECC's input message, the number of modifiable bits in each group t, the positions of insignificant bits* $\mathrm{P}=\{s_i,e_i\}_{i=1}^{n_c}$*, the locked string set* Φ*, the verification parameter set* VP *for insignificant bits, as well as the commitment set* Ψ *and verification parameter set* OP *of significant bits as inputs. It outputs the verification result.*
- $\mathrm{UtiliClear.Fingextract}(\mathbf{R},\overline{\mathbf{R}},n_c,n_g,\mathrm{P},l_f,t,sk)\rightarrow(\mathrm{f}')$. *The algorithm takes the original database* $\mathbf{R}$*, fingerprinted database* $\overline{\mathbf{R}}$*, the number of columns* n_c*, the number of groups* n_g*, the positions of insignificant bits* $\mathrm{P}=\{s_i,e_i\}_{i=1}^{n_c}$*, the length of fingerprint* l_f*, the number of modifiable bits in each group t, and DO's secret key sk as inputs. It outputs the extracted fingerprint string* f$'$.

UtiliClear is designed to achieve modification extent verification, meaning that the recipient can verify whether the modifications during the fingerprinting process are within the specified limit (i.e., $t\cdot n_c\cdot n_g$). We formally define UtiliClear with modification extent verification in Definition 4.

Definition 4. UtiliClear *exhibits the property of modification extent verification, if for any adversary* $\mathcal{A}$ *(i.e., the DO):*

$$\Pr\left[\begin{array}{l}\mathrm{UtiliClear.Setup}(\kappa)\rightarrow(pp,kp_o,kp_r),\\ \mathcal{A}(kp_o,pk_{sig},pp)\rightarrow(\mathbf{R},\overline{\mathbf{R}},\Phi'):\\ \mathrm{Dis}(\mathbf{R},\overline{\mathbf{R}})>(t\cdot n_c\cdot n_g)\wedge\mathrm{UtiliClear.Pre}\\ \mathrm{proc}(\mathbf{R},pp,kp_o,n_c,n_g,\mathrm{P})\rightarrow(\Psi,\Phi,\mathrm{OP},\\ \mathrm{VP})\wedge\mathrm{UtiliClear.Verify}(\overline{\mathbf{R}},pp,n_c,n_g,k,t,\\ \mathrm{P},\Psi,\mathrm{OP},\Phi',\mathrm{VP})\rightarrow 1\end{array}\right]\leq\mathrm{negl}(\kappa).$$

4 Fuzzy Perturbation Verification Protocol

4.1 Technical Highlights

The core idea behind our FPV protocol is to transfer the bitwise differences between the modified bit-string x' and the original bit-string x onto a codeword $cw=\mathrm{EC}_{n,t}(m)$ of the ECC by constructing an embedded error codeword $cw'=x\oplus x'\oplus cw$. Here, $\mathrm{EC}_{n,t}(m)$ encodes a message m into a codeword cw of length n with the maximum error tolerance of t bits. By checking whether the cw' can be correctly decoded (i.e., whether $\mathrm{DC}_{n,t}(cw')$ equals to m), we can verify that at most t bits differ between x' and x. However, since the ECC decoding process can reveal the error positions, cw' cannot be disclosed to the verifier. Additionally, the construction of cw' should not be completed solely by the prover, as the prover may use a string with no more than t bit modifications to construct cw', while actually sending the verifier a bit-string x' that has more than t bit modifications. Therefore, a careful design is required to transfer the bitwise differences between x and x' onto the codeword cw.

We address these challenges by enabling the verifier and the prover to collaborate in constructing the embedded error codeword cw'. The FPV protocol operates in two phases: the locking phase, before bit-string modification, and the verification phase, after modification. In the locking phase, the

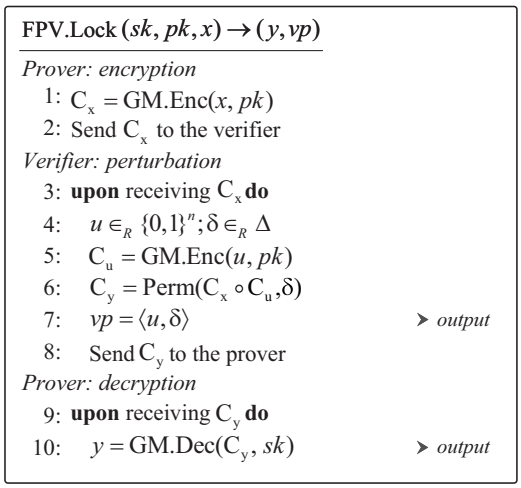


Figure 2: The locking algorithm (FPV.Lock) of the FPV protocol.

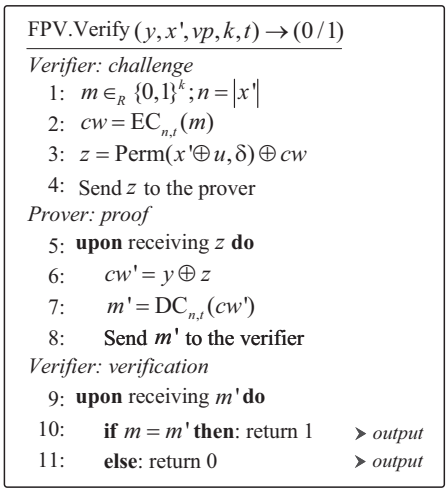


Figure 3: The verification algorithm (FPV.Verify) of the FPV protocol.

verifier and the prover work together to generate a locked bit-string by introducing perturbation specified by the verifier into the original bit-string. During this phase, the original bit-string remains confidential to the verifier, while the perturbation remains concealed from the prover. In the verification phase, after receiving the modified bit-string, the verifier applies the same perturbation and the codeword to the modified bit-string and sends the result to the prover. The prover can extract the embedded error codeword cw' only by combining the result with the locked bit-string to remove the perturbation and unmodified bits. This prevents the prover from using a bit-string with fewer modifications for validation while sending a bit-string with more modifications.

4.2 Detailed Construction

This section details the construction of the (n,t)-FPV, where n is the length of the input bit-string and t is the maximum number of modified bit. Both n and t are public parameters.

Setup. The prover preforms FPV.Setup by executing the GM.KeyGen algorithm of GM encryption with the input parameter κ, generating a public-private key pair $\langle pk, sk \rangle$.

Locking. The prover and the verifier collaborate to execute the FPV.Lock algorithm, as illustrated in Figure 2, which outlines the formal procedure for the locking process.

- Prover: encryption. The prover encrypts the bit-string x using the public key pk to obtain $C_x = \text{GM.Enc}(x, pk)$, and then sends the ciphertext C_x to the verifier.
- Verifier: perturbation. Upon receiving the ciphertext C_x, the verifier locks the string x by perturbing the ciphertext. Specifically, the verifier first selects a bit-string $u \in_R \{0,1\}^n$ and a seed $\delta \in_R \Delta$ for the permutation function. The verifier then encrypts u using the prover's public key pk to obtain $C_u = \text{GM.Enc}(u, pk)$, and computes the perturbation result $C_y = \text{Perm}(C_x \circ C_u, \delta)$. Finally, the verifier securely retains the verification parameters $vp = \langle u, \delta \rangle$ and sends the ciphertext C_y to the prover. Note that, to ensure the security of the perturbation process against the prover, u and δ are randomly selected by the verifier and are not reused in each execution.
- Prover: decryption. Upon receiving the ciphertext C_y, the prover decrypts it using the secret key sk to obtain the locked string $y = \text{GM.Dec}(C_y, sk) = \text{Perm}(x \oplus u, \delta)$.

Verification. The prover and the verifier collaborate to execute the FPV.Verify algorithm, as illustrated in Figure 3.

- Verifier: challenge. After receiving the modified bit-string x' from the prover, the verifier first selects a message $m \in_R \{0,1\}^k$ and encodes it to produce $cw = \text{EC}_{n,t}(m)$ using an ECC scheme with the maximum error tolerance of t bits. The FPV employs an ECC scheme that cannot verify whether decoding succeeds or not, and outputs a random result when the number of errors exceed the error tolerance. The verifier perturbs x' by following the same perturbation during the locking phase and XORs the result with the codeword cw to obtain $z = \text{Perm}(x' \oplus u, \delta) \oplus cw$. The verifier then sends z to the prover.
- Prover: proof. Upon receiving z, the prover obtains an embedded error codeword cw' by computing $cw' = y \oplus z$ to eliminate the perturbation and unmodified bits. The prover then decodes cw'.
- Verifier: verification. The verifier validates that at most t bits are modified in x' compared to x by checking if $m = m'$. If they match, the verification passes; otherwise, it indicates that the prover has modified more than t bits in x'.

It is likely that the verifier might attempt to infer the original bit-string by manipulating the perturbation process, leading

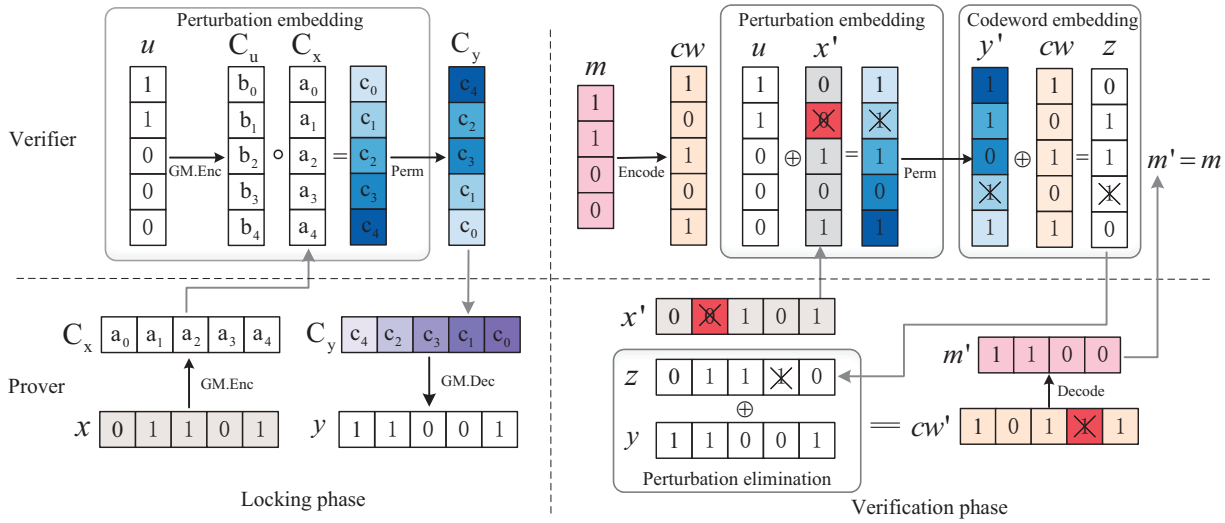


Figure 4: Example of a (5,1)-FPV protocol on a five-bit string with a single bit modification.

to the prover decoding it. The prover can counter this threat by comparing the statistical properties (e.g., the numbers of 1's and 0's) of the codeword with the original bit-string.

4.3 Illustration

Figure 4 provides an illustration of the FPV protocol applied to a five-bit string with a single bit modification. In the locking phase, to ensure that the verifier cannot obtain the original bit-string x, the prover encrypts x using its public key pk to generate the ciphertext $\mathrm{C_x} = \mathrm{GM.Enc}(x, pk)$, which is then sent to the verifier. After receiving $\mathrm{C_x}$, the verifier embeds perturbation into $\mathrm{C_x}$. To ensure the confidentiality of the perturbation, a random bit-string is embedded to $\mathrm{C_x}$ along with a permutation operation. Specifically, the verifier perturbs $\mathrm{C_x}$ by first randomly selecting a bit-string u and encrypting it as $\mathrm{C_u} = \mathrm{GM.Enc}(u, pk)$. Then, the verifier computes $\mathrm{C_x} \circ \mathrm{C_u}$, and shuffles the result to obtain $\mathrm{C_y} = \mathrm{Perm}(\mathrm{C_x} \circ \mathrm{C_u}, \delta)$. The verifier returns the perturbed result $\mathrm{C_y}$ to the prover. Due to the XOR homomorphism of GM encryption, the prover can compute $y = \mathrm{GM.Dec}(\mathrm{C_y}, sk) = \mathrm{Perm}(x \oplus u, \delta)$ using its secret key sk. During this phase, the bit-string x remains confidential to the verifier, while the perturbation (i.e., the random bit-string u and the permutation seed δ) remains concealed from the prover.

In the verification phase, upon receiving the modified string x', the verifier first selects a message m and encodes it to produce the codeword $cw = \mathrm{EC}_{n,t}(m)$ using an ECC scheme with a maximum error tolerance of t bits (one bit in this example). The verifier then performs XOR operation on x' with the string u and shuffles the result to obtain $y' = \mathrm{Perm}(x' \oplus u, \delta)$, following the same perturbation during the locking phase. Next, the verifier performs XOR operation on cw with y' to obtain $z = y' \oplus cw$, and sends z to the prover for the final construction of the embedded error codeword cw'. Upon receiving z, the prover transfers the bitwise differences between x and x' onto cw' by computing $cw' = y \oplus z = \mathrm{Perm}(x \oplus u, \delta) \oplus \mathrm{Perm}(x' \oplus u, \delta) \oplus cw$ and then decodes cw' to obtain $m' = \mathrm{DC}_{n,t}(cw')$. The prover sends the recovered m' to the verifier. If the verifier checks that $m' = m$, it confirms that the number of error bits in cw' is no more than t, further indicating that the difference between x' and x is no more then t bits. Otherwise, difference between x' and x exceeds t bits.

Note that the prover can only eliminate the perturbation and unmodified bits by performing XOR operation on z and y to derive the embedded error codeword cw'. This ensures that the original string x and the modified string x' cannot be changed by the prover. Additionally, since neither cw' nor x is exposed to the verifier, the concealment of the modification positions is maintained.

4.4 Correctness and Security Proof

Correctness. We first demonstrate that the bitwise differences between two bit-strings x and x' can be correctly transferred to cw'. Since the codeword cw' is ultimately constructed as $cw' = y \oplus z$, we can express cw' as follows:

$$\begin{aligned} cw' =& y \oplus z \\ =& \mathrm{GM.Dec}(\mathrm{Perm}(\mathrm{C_x} \circ \mathrm{C_u}, \delta), sk) \oplus \mathrm{Perm}(x' \oplus u, \delta) \oplus cw \\ =& \mathrm{GM.Dec}(\mathrm{Perm}(\mathrm{C_{x \oplus u}}, \delta), sk) \oplus \mathrm{Perm}(x' \oplus u, \delta) \oplus cw \\ =& \mathrm{Perm}(x \oplus u, \delta) \oplus \mathrm{Perm}(x' \oplus u, \delta) \oplus cw \\ =& \mathrm{Perm}(x, \delta) \oplus \mathrm{Perm}(x', \delta) \oplus cw. \end{aligned}$$

Since $\mathrm{Dis}(x, x') = \mathrm{Dis}(\mathrm{Perm}(x, \delta), \mathrm{Perm}(x', \delta))$, we have $\mathrm{Dis}(cw, cw') = \mathrm{Dis}(\mathrm{Perm}(x, \delta), \mathrm{Perm}(x', \delta)) = \mathrm{Dis}(x, x')$, demonstrating that the bitwise differences between x and x' are correctly transferred to cw'.

Therefore, if the prover successfully decodes m from cw', we can deduce that the prover has modified at most t bits in cw' (i.e., $\mathrm{Dis}(cw, cw') \leq t$), which implies $\mathrm{Dis}(x, x') \leq t$. Otherwise, if the prover cannot decode m from cw', it indicates that more than t bits have been modified. This allows the verifier to check the extent of modifications, thus proving the correctness of the FPV protocol.

Security. We demonstrate the security of the proposed FPV by proving its fuzzy-binding and full-hiding properties.

Theorem 1. *The* FPV *is secure if the GM encryption is semantically secure.*

Proof. We prove the security of FPV by separately demonstrating its fuzzy-binding and full-hiding properties.

Fuzzy-binding: Given two bit-strings x and x' satisfying $\mathrm{Dis}(x, x') > t$ and $\mathrm{FPV.Lock}(x, sk, pk) \rightarrow (y, vp)$, an adversary $\mathcal{A}$ (i.e., the prover) attempts to generate a forged string y' such that $\mathrm{FPV.Verify}(y', x', vp, k, t)$ outputs 1.

During the execution of the FPV protocol, the prover can obtain information, including x, x', y, and z. To validate the verification, the adversary must obtain an embedded error codeword cw' such that $\mathrm{Dis}(cw', cw) \leq t$. Since only z includes the correct codeword cw, the adversary can only obtain the codeword cw' from z.

In the FPV protocol, the verifier introduces perturbation (denoted as $\mathrm{Perm}(x \oplus u, \delta)$ to the bit-sting x (or x') by XORing it with a random bit-string u and then permuting the result using a permutation function determined by a seed δ. To validate the verification, the adversary must obtain u and δ, and compute a string $y' = \mathrm{Perm}(x^* \oplus u, \delta)$ using a forged x^* satisfying $\mathrm{Dis}(x^*, x') \leq t$. The adversary can then generate a valid codeword $cw' = y' \oplus z$, satisfying $\mathrm{Dis}(cw', cw) \leq t$. However, since u and δ are randomly chosen by the verifier, the perturbation is secure against the prover, which is proved in Appendix B.1. Therefore, the adversary cannot forge y' and successfully pass the verification. As a result, the proposed FPV satisfies the fuzzy-binding property.

Full-hiding: During the execution of the FPV protocol, the public key pk of the prover, a modified bit-string x', and the ciphertext $\mathrm{C_x}$ of the original bit-string x are available to the verifier, and it attempts to obtain the differing positions between x and x'. However, since the GM encryption is semantic security, the adversary cannot obtain the plaintext x. Thus, the adversary cannot obtain the differing positions by comparing x' and x, which demonstrates the full-hiding property of the proposed FPV. As a result, we can conclude that the FPV protocol is secure as it satisfies Definition 2.

□

5 Design of UtiliClear

The FPV protocol enables the recipient to verify the extent of modifications made by the DO to the database. Initially, before obtaining the recipient's fingerprint, the DO and the recipient interact to execute the PFV.Lock algorithm, allowing the recipient to lock the original database. Subsequently, the recipient sends a randomly selected ζ, its identity ID, and the signature $sign_{ID\|\zeta}$ of $ID \parallel \zeta$ to the DO. This process allows the DO to embed the fingerprint $\mathrm{f} = \mathrm{H}_2(ID \parallel \zeta)$ into the database securely, where $\mathrm{H}_2 : \{0,1\}^* \rightarrow \{0,1\}^{l_f}$. The signature $sign_{ID\|\zeta}$ ensures that only the specific fingerprint can be used to trace the recipient, making any attempt to embed other information before obtaining ζ and ID meaningless. After obtaining the fingerprinted database, the recipient can interact with the DO to execute the FPV.Verify algorithm to check the extent of modifications in the fingerprinted database.

However, a large-scale database may contain a substantial number of records, and processing each column of attribute values in a single execution of the FPV protocol may exceed the memory limits of individual devices. To address this limitation, we divide each column of attribute values into equally sized groups and process each group separately. Furthermore, considering that leading bits are typically crucial for maintaining data utility, bit-level database fingerprinting schemes often divide attribute values into significant and insignificant bits. Fingerprints are then embedded by modifying the insignificant bits [16, 18, 33]. Thus, UtiliClear further divides the attributes values within each group into significant bits and insignificant bits, and employs distinct strategies for verification. For the significant bits, UtiliClear employs the Pedersen commitment scheme [25] to effectively verify that these bits remain unmodified in the fingerprinted database. The modifiable insignificant bits are validated using the FPV protocol.

Specifically, the DO and the recipient first preprocess the database through several interactions. For each column of attribute values, the DO begins by organizing them into groups, categorizing the attribute values within each group into significant bits and insignificant bits. For each group, the DO commits to the significant bits and transmits these commitments to the recipient. For the insignificant bits, the DO and the recipient collaboratively execute the FPV.Lock algorithm, enabling the recipient to lock these bits. After preprocessing, the recipient selects a random string ζ, signs $ID \parallel \zeta$, and sends ζ, ID, and the signature to the DO. With this information, the DO computes the fingerprint $\mathrm{f} = \mathrm{H}_2(ID \parallel \zeta)$, embeds it into the database, and sends the fingerprinted database to the recipient. Upon receiving the fingerprinted database, the recipient can verify each group of significant bits using the commitment open algorithm PC.Open and assess the extent of modifications to each group of insignificant bits using the FPV.Verify algorithm.

Our protocol allows the recipient to validate the input data from the DO after the locking phase, as the DO can only remove recipient-implemented perturbation to pass verification using the locked data. The DO has no incentive to modify the original data before the locking phase, as the recipient

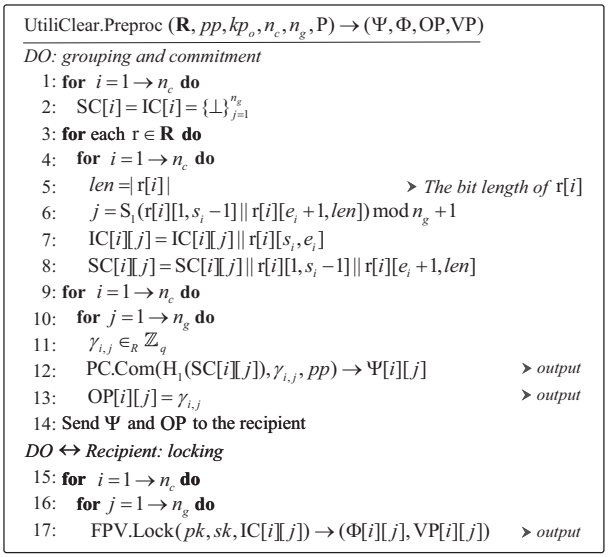


Figure 5: The preprocessing algorithm (UtiliClear.Preproc) of UtiliClear.

provides its fingerprint after this phase. Pre-locking modifications cannot embed fingerprint information and would degrade database utility, offering no benefit to the DO.

5.1 Construction of UtiliClear

Setup. The DO executes the PC.Setup and FPV.Setup algorithms using the same input parameter κ, generating the public parameters $pp = \langle g, h, p, q \rangle$ and a public-private key pair $kp_o = \langle pk, sk \rangle$. The DO sends the public parameters pp and the public key pk to the recipient. The recipient executes the Sig.KeyGen algorithm to generate a public-private signature key pair $kp_r = \langle pk_{sig}, sk_{sig} \rangle$. The recipient keeps sk_{sig} confidential and sends pk_{sig} to the DO.

Preprocessing. The recipient can specify some parameters for the purchased database, including the number of groups n_g, the number of modifiable bits t in each group, and the positions of insignificant bits $\mathrm{P} = \{s_i, e_i\}_{i=1}^{n_c}$. Based on these parameters, the DO interacts with the recipient to preprocess the database, as formally described in Figure 5.

- DO: grouping. For each attribute value $\mathrm{r}[i]$ ($\mathrm{r} \in \mathbf{R}$, $1 \le i \le n_c$), the DO first computes its group order as $j = \mathrm{S}_1(\mathrm{r}[i][1, s_i - 1] \,\|\, \mathrm{r}[i][e_i + 1, len]) \bmod n_g + 1$, where $len = |\mathrm{r}[i]|$ represents the bit length of $\mathrm{r}[i]$. Subsequently, the DO appends the significant bits $\mathrm{r}[i][1, s_i - 1] \,\|\, \mathrm{r}[i][e_i + 1, len]$ to the string $\mathrm{SC}[i][j]$ and the insignificant bits $\mathrm{r}[i][s_i, e_i]$ to $\mathrm{IC}[i][j]$. Finally, the DO obtains the grouped results $\{\mathrm{SC}[i], \mathrm{IC}[i]\}_{i=1}^{n_c}$ for the n_c columns.
- DO: commitment for significant bits. For each group $\mathrm{SC}[i][j]$ $(1 \le i \le n_c, 1 \le j \le n_g)$ of significant bits, the DO randomly selects $\gamma_{i,j} \in_R \mathbb{Z}_q$ and executes the $\mathrm{PC.Com}(\mathrm{H}_1(\mathrm{SC}[i][j]), \gamma_{i,j}, pp)$ algorithm to obtain the commitment $\Psi[i][j]$, where $\mathrm{H}_1 : \{0,1\}^* \to \mathbb{Z}_p$. After processing all groups, the DO sends the commitment set Ψ and the verification parameter set OP ($\mathrm{OP}[i][j] = \gamma_{i,j}$) to the recipient.
- DO and recipient: locking insignificant bits. For each group $\mathrm{IC}[i][j]$ $(1 \le i \le n_c, 1 \le j \le n_g)$ of insignificant bits, the DO interacts with the recipient to execute the $\mathrm{FPV.Lock}(pk, sk, \mathrm{IC}[i][j])$ algorithm. The DO and the recipient then obtain the locked bit-string $\Phi[i][j]$ and the verification parameter $\mathrm{PV}[i][j]$, respectively.

Fingerprint embedding. The fingerprint embedding algorithm is formally described as Figure 10 in Appendix B.

- Recipient: fingerprint generation. The recipient randomly selects a string $\zeta \in_R \{0,1\}^*$ and generates the signature $sig_{ID \| \zeta}$ for $ID \,\|\, \zeta$ using the $\mathrm{Sig.Sign}(sk_{sig}, ID \,\|\, \zeta)$ algorithm, where ID represents the recipient's identity. The recipient then sends ζ, ID, and $sign_{ID \| \zeta}$ to the DO.
- DO: fingerprint verification. The DO verifies the signature by executing the $\mathrm{Sig.Verify}(pk_{sig}, sign_{ID \| \zeta}, ID \,\|\, \zeta)$ algorithm. Only if the signature is valid does the DO compute the fingerprint $\mathrm{f} = \mathrm{H}_2(ID \,\|\, \zeta)$. The DO then initializes an $n_c \times n_g$ matrix M, setting all elements to t, to track the number of bits that can currently be modified in each group.
- DO: fingerprinting. The fingerprint embedding process follows the approach in [18], with UtiliClear additionally limiting the number of modifiable bits in each group of insignificant bits. For each insignificant bit $\mathrm{r}[i][j]$ ($\mathrm{r} \in \mathbf{R}$, $1 \le i \le n_c$, $s_i \le j \le e_i$), the DO determines whether a fingerprint bit should be embedded using its secret key sk, the primary key of the record $\mathrm{r}.key$, the attribute value $\mathrm{r}[i]$, and the position index j. Specifically, the DO computes a seed $\theta = sk \,\|\, \mathrm{r}.key \,\|\, \mathrm{r}[i] \,\|\, j$ and inputs it into the pseudorandom sequence generator S. A fingerprint bit is embedded only if $\mathrm{S}_1(\theta) \bmod \lfloor \frac{N \cdot d_i}{n_g \cdot t} \rfloor = 0$ and the number of modifiable bits within the group is greater than zero (i.e., $\mathrm{M}[i][\alpha] > 0$). Here, $d_i = e_i - s_i + 1$, $len = |\mathrm{r}[i]|$, and $\alpha = \mathrm{S}_1(\mathrm{r}[i][1, s_i - 1] \,\|\, \mathrm{r}[i][e_i + 1, len]) \bmod n_g + 1$. When embedding a fingerprint bit into the j-th bit of $\mathrm{r}[i]$, the DO computes the mark bit $\sigma = \beta \oplus \mathrm{f}[b]$ and modifies the $\mathrm{r}[i][j]$ by computing $\mathrm{r}[i][j] = \mathrm{r}[i][j] \oplus \sigma$, where $b = \mathrm{S}_2(\theta) \bmod l_f + 1$, l_f is the bit length of f, and the value β is determined by checking whether $\mathrm{S}_3(\theta)$ is even or odd. After embedding, the DO decreases the number of modifiable bits within the α-th group of the i-th column by 1 (i.e., $\mathrm{M}[i][\alpha] = \mathrm{M}[i][\alpha] - 1$).

After processing all insignificant bits, the DO obtains a fingerprinted database $\overline{\mathbf{R}}$ and sends it to the recipient.

Verification. Upon receiving the fingerprinted database $\overline{\mathbf{R}}$, the recipient can verify the modifications using the UtiliClear.Verify algorithm, as shown in Figure 6.

- Recipient: grouping. The recipient groups each column in the database, following the operations performed by

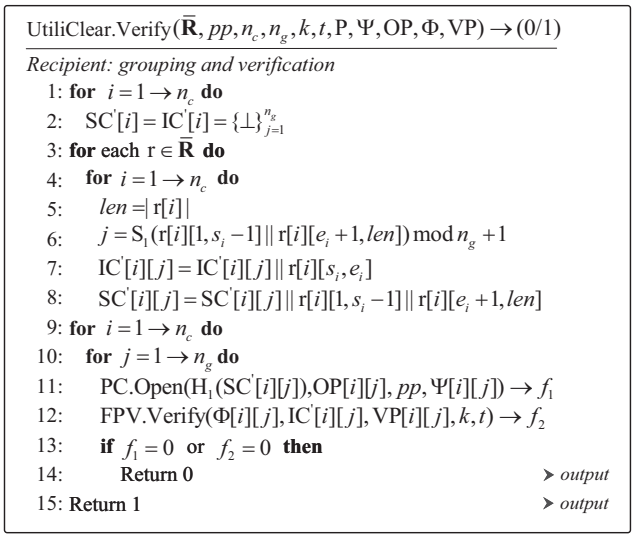


Figure 6: The verification algorithm (UtiliClear.Verify) of UtiliClear.

the DO during the preprocessing phase, and obtains the grouped results $\{\mathrm{SC}'[i], \mathrm{IC}'[i]\}_{i=1}^{n_c}$.

- Recipient: verification. For each group $\mathrm{SC}'[i][j]$ $(1 \le i \le n_c, 1 \le j \le n_g)$ of significant bits, the recipient executes the PC.Open algorithm using the parameters $(\mathrm{OP}[i][j], pp)$ to verify whether the committed significant bits in $\Psi[i][j]$ match $\mathrm{SC}'[i][j]$. For each group $\mathrm{IC}'[i][j]$ $(1 \le i \le n_c, 1 \le j \le n_g)$ of insignificant bits, the recipient interacts with the DO to execute the FPV.Verify algorithm using the verification parameters $\mathrm{VP}[i][j]$, verifying whether the number of differing positions between insignificant bits locked in $\Phi[i][j]$ and $\mathrm{IC}'[i][j]$ is no more than t. The recipient accepts the database only if all groups pass the verification.

Fingerprint extraction. Once discovering a pirated database, the DO can trace the dishonest recipient by extracting the recipient's fingerprint from the database. The fingerprint extraction process follows the approach in [18]. The formal description of the fingerprint extraction algorithm is provided as Figure 11 in Appendix A.

Specifically, the DO initializes two arrays, f_0 and f_1, to record the positions of '0' and '1' in the extracted fingerprint. For each insignificant bit, such as the j-bit of $\mathrm{r}[i]$, the DO determines whether a fingerprint bit was inserted using its secret key sk, the primary key of the record r.key, the attribute value $\mathrm{r}[i]$, and the position index j. Specifically, the DO computes a seed $\theta = sk \| \mathrm{r}.key \| \mathrm{r}[i] \| j$ and uses it as input of the pseudorandom sequence generator S. If $\mathrm{S}_1(\theta) \bmod \lfloor \frac{N \cdot d_i}{n_g \cdot t} \rfloor = 0$ and $b = \mathrm{S}_2(\theta) \bmod l_f + 1$, it indicates that the b-th fingerprint bit was inserted at this position. The DO then extracts the fingerprint bit α as $\alpha = \beta \oplus \mathrm{r}[i][j] \oplus \bar{\mathrm{r}}[i][j]$, where $\bar{\mathrm{r}} \in \bar{\mathbf{R}}$ is the corresponding record with r.key, and β is determined by whether $\mathrm{S}_2(\theta)$ is even or odd. If $\alpha = 1$, $\mathrm{f}_1[b]$ is incremented by 1; otherwise, $\mathrm{f}_0[b]$ is increased by 1.

After processing all insignificant bits, the DO can reconstruct the fingerprint from the arrays f_0 and f_1. For each position i within the fingerprint f', if $\mathrm{f}_1[i] > \mathrm{f}_0[i]$ then $\mathrm{f}'[i] = 1$; otherwise $\mathrm{f}'[i] = 0$. Once the fingerprint f' is reconstructed, the DO can accurately identify the dishonest recipient.

5.2 Theoretical Analysis

Modification extent verification. We present the theorem for modification extent verification as follows.

Theorem 2. *UtiliClear achieves modification extent verification if the* FPV *protocol is secure and the employed Pedersen commitment scheme satisfies the binding property.*

We provide the detailed proofs for Theorem 2 in Appendix B.2. Given that our proposed FPV protocol is secure as proved in Section 4.4, and the employed Pedersen commitment possesses the binding property [25], the conditions of Theorem 2 are satisfied. This confirms that UtiliClear achieves modification extent verification for the recipient.

Fingerprint robustness preservation. To protect the interests of the DO, the primary goal of UlitiClear is to ensure fingerprint integrity for the detection of illegal redistribution. We first prove the confidentiality of fingerprint positions from recipients.

Theorem 3. *UtiliClear ensures the confidentiality of fingerprint positions from recipients if the* FPV *is secure and the employed* GM *secret key is secure under the quadratic residuosity assumption.*

We provide the detailed proofs for Theorem 3 in Appendix B.3. Given that our proposed FPV is secure as proved in Section 4.4, and the GM secret key is secure under the quadratic residuosity assumption [34], the conditions of Theorem 3 are satisfied and UtiliClear ensures the confidentiality of fingerprint positions.

UtiliClear employs the advanced fingerprinting technique from [18] for embedding and extracting fingerprints. This method divides data into significant and insignificant bits, embedding the fingerprint by modifying the insignificant bits. By ensuring the confidentiality of fingerprint positions from recipients, UtiliClear inherits the fingerprint robustness comparable to the technique in [18]. This is further validated by the experiments in Section 6.3.

Database utility preservation. Since modification extent verification does not change the database and operates independently from the fingerprint embedding and extraction processes, it maintains the database utility in line with the technique described in [18]. This is further validated by the experiments in Section 6.3.

Data privacy preservation. During modification extent verification, the DO only provides the recipient with the ciphertext of database, encrypted using the GM cryptosystem [30].

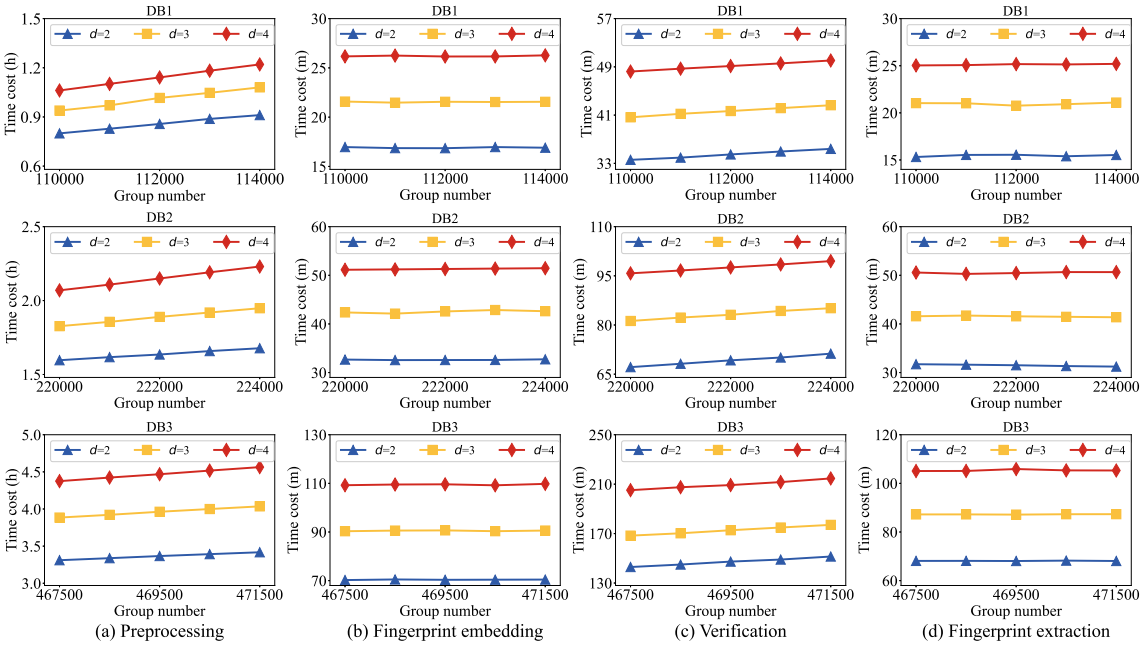


Figure 7: The DO's computation time across distinct phases, with varying group numbers and insignificant bit lengths d.

This ensures that the modification extent verification does not reveal any information about the actual database data to the recipient. Additionally, UtiliClear employs the fingerprint embedding technique [18], which provides ε-entry-level DP for the fingerprinted database contents, as demonstrated in Appendix B.4. This allows recipients to use the data for analysis while preventing access to sensitive details, which ensures data privacy for the database providers against the recipient. However, UtiliClear does not protect the privacy of data providers against the DO, which may cause a potential negative impact on the privacy of data providers.

6 Experiments

6.1 Experiment Setup

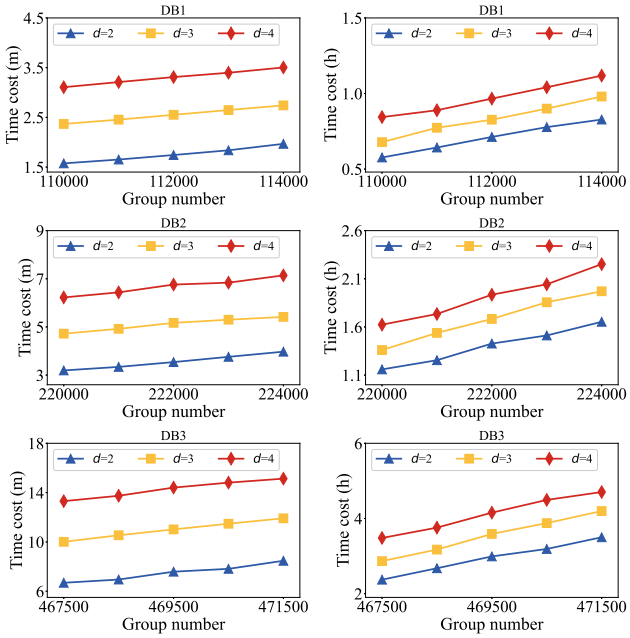
Implementation. Our scheme is fully implemented in C++ using standard cryptographic libraries, including OpenSSL [35] and GNU Multiple Precision Arithmetic [36]. The programs of the DO and recipient are deployed on two desktops, each equipped with a 24-core Intel Core i9-13900 processor and 16GB of memory.

Datasets. We evaluate UtiliClear using the Amazon Review dataset (34GB) [37], which consists of 230,139,802 records, each containing 4 attributes and associated with one of five rating labels. We extract two subsets, DB1 (8GB) containing 54,846,036 records and DB2 (16GB) containing 109,693,107 records, and evaluate our UtiliClear using the subsets with the original Amazon Review dataset (DB3). The datasets contains numeric data, categorical data, and text sentences. Numeric attributes are directly represented as bit-strings, while categorical attributes are encoded into 20-dimensional word vectors [38]. Text sentences are encoded into one-hot vectors [39].

Parameter setting. Our implementation employs the Reed-Solomon ECC [29], configured with an input size of 223 bytes, 16 bytes of error tolerance, and 255 bytes of output. Specifically, the FPV employed in UtiliClear is configured with an input string length of 2040 bits ($n = 2040$), a maximum of 128 modified bits ($t = 128$), and a challenge message length of 1784 bits ($k = 1784$). This ECC does not support decoding result verification. The division of significant and insignificant bits is usually negotiated between the DO and the recipient, based on the data type and database price. For simplicity, we set the last d bits (d = 2, 3, or 4) as insignificant in our experiments. For the number of groups, we adjust this parameter to ensure that the number of insignificant bits in each group does not exceed 255 bytes. Each insignificant bit group is then padded with '0' to ensure adaptability with the employed ECC. Thus, we set the minimum number of groups for DB1, DB2, and DB3 to 110000, 220000, and 467500, respectively. Additionally, we set the key's bit length κ of all cryptographic primitives to 256, which is a commonly used key length in practice. We also set the fingerprint bit length l_f to 128, which is large enough to represent a large number of recipients.

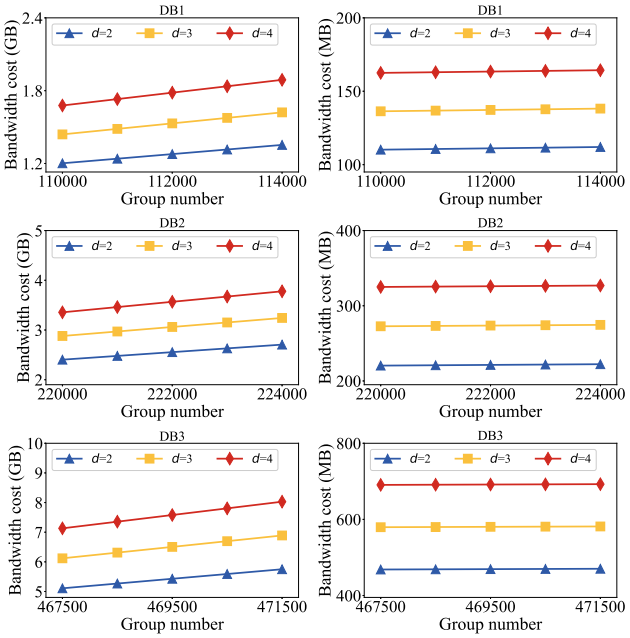
6.2 Evaluation on Efficiency

UtiliClear is the first scheme to support the verification of modification extent during the fingerprinting process, which inevitably incurs additional computation and communication



(a) Preprocessing (b) Verification

Figure 8: The recipient's computation time, with varying group numbers and insignificant bit lengths d.



(a) Preprocessing (b) Verification

Figure 9: Communication cost across different phases, with varying group numbers and insignificant bit lengths d.

overhead for both the recipient and the DO compared to previous fingerprinting schemes that lack this feature. Thus, directly comparing the efficiency of UtiliClear with previous schemes is less meaningful. Our experiments primarily focus on evaluating the effectiveness of UtiliClear.

Computation efficiency. Figure 7 and Figure 8 present the time costs for the DO and the recipient, respectively. During the preprocessing phase, the DO and the recipient collaborate to execute the FPV.Lock algorithm for each insignificant bit group. The DO also executes the PC.Com algorithm for each significant bit group. Dividing the database into more groups raises the time cost for both the recipient and the DO. However, fewer groups cause larger data sizes per group, requiring more available memory. Additionally, for databases with a larger number of insignificant bits (i.e., a larger parameter d), UtiliClear consumes more time to preprocess database.

During the fingerprint embedding phase, the DO's computation time increases with the number of insignificant bits but remains unaffected by the group number, as illustrated in Figure 7 (b). Fingerprint extraction, as the complementary process to fingerprint embedding, incurs similar time costs, as shown in Figure 7 (d).

For the verification phase, verifying each insignificant bit group requires one execution of the FPV.Verify algorithm, while validating each significant bit group requires one execution of the PC.Open algorithm. As illustrated in Figures 7 (c) and 8 (b), more groups increase the time costs for both the DO and the recipient.

Since each group can be processed independently using separate threads, our scheme can also leverage parallel execution for boosting the performance. Moreover, the DO and recipient are typically institutions with much more powerful computing resources in real-world scenarios.

Communication efficiency. Figure 9 (a) shows the communication overhead for the preprocessing phase. The DO transmits the commitment and verification parameters for each significant bit group to the recipient. Each execution of the commitment scheme incurs a fixed communication overhead, and the overall overhead grows linearly with the number of group. For insignificant bits, the ciphertext and its perturbed version are exchanged between the DO and the recipient, with the communication overhead primarily determined by the total number of insignificant bits.

For the verification phase, communication overhead arises solely from the FPV.Verify algorithm. As shown in Figure 9 (b), UtiliClear incurs higher communication overhead for processing databases with more insignificant bits, while the overhead remains unaffected by the number of groups.

Memory cost. The memory cost results are presented in Table 2. UtiliClear incurs an average memory cost of 580 MB for the DO and 620 MB for the recipient to process the largest 34GB dataset (DB3), which is practically reasonable. More groups cause smaller data sizes per group, reducing memory requirements for both the DO and the recipient. Additionally, since each group can be processed independently using separate threads, our scheme supports parallel execution, as long as sufficient memory is available.

Table 2: Memory cost (MB)

Database			DB1		DB2		DB3	
Group number (10^5)			1.10	1.14	2.20	2.24	4.675	4.715
DO	$d=2$	Average	426.54	396.68	434.55	405.81	432.34	401.51
		Maximum	570.21	542.69	596.17	562.94	570.65	539.08
	$d=3$	Average	495.07	458.91	507.36	474.52	504.81	469.25
		Maximum	643.97	615.98	672.83	631.17	654.02	612.70
	$d=4$	Average	568.87	531.51	579.02	542.57	571.89	539.66
		Maximum	732.26	697.45	740.97	693.04	735.96	696.12
Recipient	$d=2$	Average	452.23	425.19	472.53	446.10	461.34	423.62
		Maximum	568.42	541.97	598.58	563.79	572.62	537.47
	$d=3$	Average	537.42	500.02	552.91	514.63	536.67	503.25
		Maximum	646.29	613.16	671.78	628.05	652.34	614.72
	$d=4$	Average	613.63	579.68	623.14	588.01	616.79	583.97
		Maximum	730.42	694.74	737.58	695.73	731.47	692.15

6.3 Evaluation on Fingerprint Robustness and Database Utility

UtiliClear is the first solution to achieve modification extent verification in database fingerprinting by incorporating the FPV protocol. It is compatible with existing bit-level fingerprinting schemes. UtiliClear employs the advanced fingerprinting method from [18], with an additional constraint on the number of modifiable bits in each group. We present the fingerprint robustness experiment in Appendix C, where the results show that UtiliClear achieves similar fingerprint robustness to the scheme in [18], when applying modifications of similar extent to the fingerprinted databases. We also provide the database utility experiments in Appendix D. The results show that UtiliClear achieves comparable database utility with the scheme in [18] under similar modification extents. Additionally, as the number of modified bits increases, database utility decreases.

The above experiments demonstrate that UtiliClear does not compromise the fingerprint robustness or data utility of the employed fingerprinting scheme. The experiment results show UtiliClear's effectiveness on numeric-type data. It also supports real-valued data, with the process remaining the same as long as significant bits (e.g., sign bits) and insignificant bits (e.g., trailing fractional bits) are properly defined.

7 Related Work

The concept of database fingerprinting was first proposed in [33], where fingerprints are embedded by modifying one attribute value per record. Inspired by this work, several database fingerprinting schemes [40–42] have been developed to improve the efficiency of fingerprint embedding and enhance the robustness against various fingerprint removal attacks, such as random bit flipping attack [43], subset attack [40], and superset attack [44]. Recently, Ji *et al*. [16] introduced the correlation attack, which targets databases with strong inter-attribute correlations. By leveraging these correlations, recipients can remove most of fingerprint information with a high probability. To resist this attack, the scheme in [45] employs optimal transportation techniques. This solution can be applied as a post-processing step to any existing database fingerprinting scheme, effectively mitigating the threat of correlation attack.

Some prior works [15, 46, 47] employ database sanitization techniques, such as the (α,β)-privacy model [48] and DP [49], followed by fingerprinting to simultaneously ensure the identification of unauthorized redistribution and the protection of database privacy. These works achieve their objectives through a two-stage process, where data sanitization and fingerprinting are conducted separately. However, since both sanitization and fingerprinting involve noise addition, a significant number of records in the database are modified, greatly compromising its utility. To address this issue, the schemes in [17, 18] integrate data sanitization and fingerprinting into a unified mechanism. The scheme in [17] adds Gaussian noise with various pre-determined variances to the database and uses the combination of these noises as the fingerprint. In contrast, the scheme in [18] leverages the inherent randomness of fingerprinting to achieve provable entry-level DP for the distributed database, offering a stronger privacy model compared to previous works.

Summary. Existing database fingerprinting schemes focus on enabling the DO to detect unauthorized redistribution but conceal the actual extent of database modifications from the recipient. Additionally, to enhance the fingerprint robustness, the DO may be inclined to extensively modify the distributed database, which may ultimately reduce the database utility. In this paper, we address this challenge by developing a database fingerprinting scheme that allows recipients to specify and verify the extent of modifications made during the fingerprinting process, while preserving fingerprint robustness, database utility, and data privacy.

8 Conclusion

In this paper, we take a significant step towards achieving modification extent verification in database fingerprinting. We develop a novel FPV that enables a verifier to actually assess the extent of modifications made to a bit-string by a prover, while keeping the modification positions confidential from the verifier. Building on the FPV protocol, we further propose a new database fingerprinting scheme, UtiliClear, which allows the recipient to specify and verify the extent of modifications made to the fingerprinted database. Through theoretical proofs and comprehensive evaluation, we demonstrate that UtiliClear can achieve the novel property of modification extent verification, while maintaining fingerprint robustness, database utility and privacy comparable to the employed fingerprinting scheme.

Ethics Considerations

The authors strictly adhere to ethical standards throughout the research process, carefully reviewing the ethical considerations outlined in the ethical guidelines. We fully support the research ethics principles advocated by USENIX Security and critically evaluated ethical issues. Ethical standards are rigorously upheld, ensuring ethical conduct throughout the research.

The security goal of our protocol is to introduce a verification mechanism that enables recipients to verify the extent of modifications within the fingerprinted database. It is fully compatible with existing bit-level fingerprinting schemes while preserving fingerprint robustness, database utility, and data privacy. Developed from an impartial and unbiased perspective, the protocol promotes fair and transparent database transactions between database owners and recipients, although inevitably introducing additional computation and communication overhead.

Our protocol carefully adheres to fundamental ethical principles, ensuring fairness and reasonableness. In the event of a violation, either party can terminate the protocol without incurring economic loss. Our goal is to build digital trust for social good. We are unaware of any dual-use or harmful applications and remain open to any further guidance.

Compliance with the Open Science Policy

The source code and execution scripts of UtiliClear are now available at https://zenodo.org/records/14731665.

Acknowledgments

We sincerely thank the shepherd and the anonymous reviewers for their constructive and invaluable feedback. Guoai Xu was supported by the National Key R&D Program of China under Grant 2022YFB3104400 and by the Shenzhen Higher Education Stability Support Program under Grant GXWD20231130114233001. Zhongyun Hua was supported by the Guangdong Basic and Applied Basic Research Foundation under Grant 2024A1515012299.

References

[1] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[2] Michael Armbrust, Tathagata Das, Liwen Sun, Burak Yavuz, Shixiong Zhu, Mukul Murthy, Joseph Torres, Herman van Hovell, Adrian Ionescu, Alicja Łuszczak, et al. Delta lake: high-performance acid table storage over cloud object stores. *Proceedings of the VLDB Endowment*, 13(12):3411–3424, 2020.

[3] Maribel Fernández, Alex Franch Tapia, Jenjira Jaimunk, Manuel Martinez Chamorro, and Bhavani Thuraisingham. A data access model for privacy-preserving cloud-iot architectures. In *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies*, pages 191–202, 2020.

[4] Javen Kennedy, Pranav Subramaniam, Sainyam Galhotra, and Raul Castro Fernandez. Revisiting online data markets in 2022: A seller and buyer perspective. *ACM SIGMOD Record*, 51(3):30–37, 2022.

[5] Devriş İşler, Elisa Cabana, Alvaro Garcia-Recuero, Georgia Koutrika, and Nikolaos Laoutaris. Freqywm: Frequency watermarking for the new data economy. In *IEEE 40th International Conference on Data Engineering*, pages 4993–5007, 2024.

[6] Wenling Li, Ning Li, Jianen Yan, Zhaoxin Zhang, Ping Yu, and Gang Long. Secure and high-quality watermarking algorithms for relational database based on semantic. *IEEE Transactions on Knowledge and Data Engineering*, 35(7):7440–7456, 2023.

[7] Xinjia Li, Li Zhao, Yahong Li, and Lei Fang. A study on the application of digital watermarking techniques on relational databases. In *9th International Conference on Computer and Communications*, pages 2312–2316, 2023.

[8] Yuran Bi, Jinfei Liu, Chen Zhao, Junyi Zhao, Kui Ren, and Li Xiong. Share: Stackelberg-nash based data markets. In *IEEE 40th International Conference on Data Engineering*, pages 3573–3586, 2024.

[9] Emre Yilmaz, Tianxi Ji, Erman Ayday, and Pan Li. Genomic data sharing under dependent local differential privacy. In *Proceedings of the 20th ACM Conference on Data and Application Security and Privacy*, pages 77–88, 2022.

[10] Flatiron Health. [Online]. Available:https://flatiron.com/.

[11] Faten Chaabane, Maha Charfeddine, and Chokri Ben Amar. A survey on digital tracing traitors schemes. In *9th International Conference on Information Assurance and Security*, pages 85–90, 2013.

[12] Santiago Andrés Azcoitia and Nikolaos Laoutaris. A survey of data marketplaces and their business models. *ACM SIGMOD Record*, 51(3):18–29, 2022.

[13] Zhiwen Ren, Han Fang, Jie Zhang, Zehua Ma, Ronghao Lin, Weiming Zhang, and Nenghai Yu. A robust database watermarking scheme that preserves statistical characteristics. *IEEE Transactions on Knowledge and Data Engineering*, 36(6):2329–2342, 2024.

[14] Mashal Ahmad, Arsalan Shahid, Muhammad Yasir Qadri, Khalid Hussain, and Nadia N Qadri. Fingerprinting non-numeric datasets using row association and pattern generation. In *2017 International Conference on Communication Technologies*, pages 149–155, 2017.

[15] Sébastien Gambs, Julien Lolive, and Jean-Marc Robert. Entwining sanitization and personalization on databases. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 207–219, 2018.

[16] Tianxi Ji, Emre Yilmaz, Erman Ayday, and Pan Li. The curse of correlations for robust fingerprinting of relational databases. In *Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 412–427, 2021.

[17] Yun Hu, Aiqun Hu, Chunguo Li, Peng Li, and Chunyu Zhang. Towards a privacy protection-capable noise fingerprinting for numerically aggregated data. *Computers & Security*, 119:102755, 2022.

[18] Tianxi Ji, Erman Ayday, Emre Yilmaz, Ming Li, and Pan Li. Privacy-preserving database fingerprinting. In *30th Annual Network and Distributed System Security Symposium*, 2023.

[19] Ali Ataeemh Allami, Mukesh K Mohania, and Wei Jiang. Privacy-preserving watermarking transformation technique in a distributed environment. In *2018 IEEE Conference on Communications and Network Security*, pages 1–9, 2018.

[20] Muhammad Kamran and Muddassar Farooq. A formal usability constraints model for watermarking of outsourced datasets. *IEEE Transactions on Information Forensics and Security*, 8(6):1061–1072, 2013.

[21] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy*, pages 1074–1091. IEEE, 2021.

[22] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy*, pages 859–876. IEEE, 2020.

[23] David Heath and Vladimir Kolesnikov. Stacked garbling for disjunctive zero-knowledge proofs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 569–598. Springer, 2020.

[24] Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Annual International Cryptology Conference*, pages 704–737. Springer, 2020.

[25] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*, pages 129–140. Springer, 1991.

[26] Dario Catalano and Dario Fiore. Vector commitments and their applications. In *Public-Key Cryptography–PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography*, pages 55–72, 2013.

[27] Ari Juels and Martin Wattenberg. A fuzzy commitment scheme. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pages 28–36, 1999.

[28] Sonam Chauhan and Ajay Sharma. Improved fuzzy commitment scheme. *International Journal of Information Technology*, 14(3):1321–1331, 2022.

[29] Stephen B Wicker and Vijay K Bhargava. *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.

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

[31] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

[32] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 369–378, 1987.

[33] Yingjiu Li, Vipin Swarup, and Sushil Jajodia. Fingerprinting relational databases: Schemes and specialties. *IEEE Transactions on Dependable and Secure Computing*, 2(1):34–45, 2005.

[34] Marc Joye. Identity-based cryptosystems and quadratic residuosity. In *Public-Key Cryptography–PKC 2016: 19th IACR International Conference on Practice and Theory in Public-Key Cryptography*, pages 225–254, 2016.

[35] OpenSSL Library. (2015). Open Source Socket Layer [Online]. Available:https://www.openssl.org/.

[36] T. Granlund. (2010). GNU Multiple Precision Arithmetic Library [Online]. Available:http://gmplib.org/.

[37] Jianmo Ni. (2018). Amazon Review Data [Online]. Available:https://nijianmo.github.io/amazon/index.html.

[38] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[39] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[40] Fei Guo, Jianmin Wang, and Deyi Li. Fingerprinting relational databases. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, pages 487–492, 2006.

[41] Thach V Bui, Binh Q Nguyen, Thuc D Nguyen, Noboru Sonehara, and Isao Echizen. Robust fingerprinting codes for database. In *13th International Conference on Algorithms and Architectures for Parallel Processing*, pages 167–176, 2013.

[42] Donghui Hu, Qing Wang, Song Yan, Xiaojun Liu, Meng Li, and Shuli Zheng. Reversible database watermarking based on order-preserving encryption for data sharing. *ACM Transactions on Database Systems*, 48(2):1–25, 2023.

[43] Emre Yilmaz and Erman Ayday. Collusion-resilient probabilistic fingerprinting scheme for correlated data. *arXiv preprint arXiv:2001.09555*, 2020.

[44] Tianxi Ji, Erman Ayday, Emre Yilmaz, and Pan Li. Towards robust fingerprinting of relational databases by mitigating correlation attacks. *IEEE Transactions on Dependable and Secure Computing*, 20(4):2939–2953, 2022.

[45] T Ji, E Ayday, E Yilmaz, and P Li. Robust fingerprinting of genomic databases. In *30th International Conference of Intelligent Systems for Molecular Biology*, 2021.

[46] Sebastian Schrittwieser, Peter Kieseberg, Isao Echizen, Sven Wohlgemuth, Noboru Sonehara, and Edgar Weippl. An algorithm for k-anonymity-based fingerprinting. In *International Workshop on Digital Watermarking*, pages 439–452, 2012.

[47] Peter Kieseberg, Sebastian Schrittwieser, Martin Mulazzani, Isao Echizen, and Edgar Weippl. An algorithm for collusion-resistant anonymization and fingerprinting of sensitive microdata. *Electronic Markets*, 24:113–124, 2014.

[48] Vibhor Rastogi, Dan Suciu, and Sungho Hong. The boundary between privacy and utility in data publishing. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 531–542, 2007.

[49] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

A The Fingerprint Embedding and Extraction Algorithms

The UtiliClear.Fingembed algorithm in Figure 10 presents the steps of fingerprint embedding, while the UtiliClear.Fingextract algorithm in Figure 11 summarizes the detailed procedures of fingerprint extraction.

B Security Proof

B.1 Proof of Perturbation Security

In the FPV protocol, the verifier introduces perturbation to the original bit-sting x (or modified bit-string x') by XORing it with a random bit-string u and then permuting the result using a permutation function determined by a seed δ. The perturbation to x is denoted as $\text{Perm}(x \oplus u, \delta)$. We prove the security of perturbation, including the XOR and permutation processes, against the prover.

Theorem 4. *Given that the parameters* $u \in_R \{0,1\}^n$ *and* $\delta \in_R \Delta$ *are uniformly selected at random, the perturbation of* FPV *is secure against the prover, where n is the bit length of the* FPV *protocol's input and* Δ *represents the space of possible mappings for the permutation function.*

Proof. During the execution of the FPV protocol, the prover can obtain the following information: the original bit-string x, modified bit-string x', perturbed bit-string $y = \text{Perm}(x' \oplus u, \delta)$, and perturbed bit-string $z = \text{Perm}(x' \oplus u, \delta) \oplus cw$, where cw is the codeword chosen by the verifier. The perturbation security can be modeled as a game between a challenger $\mathcal{C}$ (i.e., the verifier) and an adversary $\mathcal{A}$ (i.e., the prover).

UtiliClear.Fingembed $(\mathbf{R}, kp_r, n_c, n_g, \mathrm{P}, t, sk, ID) \to (\mathrm{f}, \bar{\mathbf{R}})$

Recipient: fingerprint generation
$\zeta \in_R \{0,1\}^*$
$sign_{ID\|\zeta} = \text{Sig.Sign}(sk_{sig}, ID \| \zeta)$
Send ζ, ID, and $sign_{ID\|\zeta}$ to the DO
DO: fingerprinting
if $\text{Sig.Verify}(pk_{sig}, sign_{ID\|\zeta}, ID \| \zeta) = 1$ **then**
  $\mathrm{f} = \mathrm{H}_2(ID \| \zeta)$ ➤ *output*
otherwise: Terminate
$N = |\mathbf{R}|; l_f = |\mathrm{f}|$
for $i = 1 \to n_c$ **do**
  for $j = 1 \to n_g$ **do**
    $\mathrm{M}[i][j] = t$
for each $\mathrm{r} \in \mathbf{R}$ **do**
  for $i = 1 \to n_c$ **do**
    $d_i = e_i - s_i + 1$
    $len = |\mathrm{r}[i]|$
    $\alpha = \mathrm{S}_1(\mathrm{r}[i][1, s_i - 1] \| \mathrm{r}[i][e_i + 1, len]) \bmod n_g + 1$
    for $j = s_i \to e_i$ **do**
      $\theta = sk \| \mathrm{r}.key \| \mathrm{r}[i] \| j$
      if $\mathrm{S}_1(\theta) \bmod \lfloor N \cdot d_i / n_g \cdot t \rfloor = 0$ and $\mathrm{M}[i][\alpha] > 0$ **then**
        $b = \mathrm{S}_2(\theta) \bmod l_f + 1$
        if $\mathrm{S}_3(\theta) \bmod 2 = 1 : \beta = 1$
        otherwise : $\beta = 0$
        $\sigma = \beta \oplus \mathrm{f}[b]$
        $\mathrm{r}[i][j] = \mathrm{r}[i][j] \oplus \sigma$
        $\mathrm{M}[i][\alpha] = \mathrm{M}[i][\alpha] - 1$
$\bar{\mathbf{R}} = \mathbf{R}$ ➤ *output*

Figure 10: The fingerprint embedding algorithm (UtiliClear.Fingembed) of UtiliClear.

UtiliClear.Fingextract $(\mathbf{R}, \bar{\mathbf{R}}, n_c, n_g, \mathrm{P}, l_f, t, sk) \to (\mathrm{f}')$

$N = |\mathbf{R}|$
for $i = 1 \to l_f$ **do**
  $\mathrm{f}_0[i] = \mathrm{f}_1[i] = 0$
for each $\mathrm{r} \in \mathbf{R}$ **do**
  for $i = 1 \to n_c$ **do**
    $d_i = e_i - s_i + 1$
    for $j = s_i \to e_i$ **do**
      $\theta = sk \| \mathrm{r}.key \| \mathrm{r}[i] \| j$
      if $\mathrm{S}_1(\theta) \bmod \lfloor N \cdot d_i / n_g \cdot t \rfloor = 0$ **then**
        $b = \mathrm{S}_2(\theta) \bmod l_f + 1$
        if $\mathrm{S}_3(\theta) \bmod 2 = 1 : \beta = 1$
        otherwise : $\beta = 0$
        Retrieve $\bar{\mathrm{r}} \in \bar{\mathbf{R}}$ using $\mathrm{r}.key$
        $\alpha = \beta \oplus \mathrm{r}[i][j] \oplus \bar{\mathrm{r}}[i][j]$
        if $\alpha = 1 : \mathrm{f}_1[b]{+}{+}$
        otherwise : $\mathrm{f}_0[b]{+}{+}$
for $i = 1 \to l_f$ **do**
  if $\mathrm{f}_1[i] > \mathrm{f}_0[i] : \mathrm{f}'[i] = 1$ ➤ *output*
  otherwise : $\mathrm{f}'[i] = 0$ ➤ *output*

Figure 11: The fingerprint extraction algorithm (UtiliClear.Fingextract) of UtiliClear.

- Setup: The challenger $\mathcal{C}$ publishes the employed ECC parameters to the adversary $\mathcal{A}$.
- Challenge: The adversary $\mathcal{A}$ selects original bit-strings x_0, $x_1 \in_R \{0,1\}^n$ and modified bit-strings x_0', $x_1' \in_R \{0,1\}^n$ such that $\mathrm{Dis}(x_0, x_0') > t$ and $\mathrm{Dis}(x_1, x_1') > t$, where t is the error tolerance of ECC. It then sends them to the challenger $\mathcal{C}$. The challenger $\mathcal{C}$ randomly picks $i \in_R \{0,1\}$, $u \in_R \{0,1\}^n$, $\delta \in_R \Delta$, and computes $y = \mathrm{Perm}(u \oplus x_i, \delta)$, $z = \mathrm{Perm}(u \oplus x_i', \delta) \oplus cw$. The challenger $\mathcal{C}$ sends y and z to the adversary $\mathcal{A}$.
- Guess: The adversary $\mathcal{A}$ outputs a guess i' for i based on (y,z) to identify which data pair, (x_0, x_0') or (x_1, x_1'), was selected by the challenger $\mathcal{C}$.

In this game, the adversary $\mathcal{A}$ guesses whether the data pair is (x_0, x_0') or (x_1, x_1'), and we use X to denote its chosen data pair. Based on this guess, the adversary generates a perturbed data pair C through enumerating the used parameters. We calculate the conditional probability that C matches the received perturbed data pair (y,z) as follows:

$$\begin{aligned}
&\Pr\left[C = (y,z) \mid X = (x_0, x_0')\right] \\
=&\Pr\left[\mathrm{Perm}(u' \oplus x_0, \delta') = y \ \&\ \mathrm{Perm}(u' \oplus x_0', \delta') \oplus cw' = z\right] \\
=&\Pr\left[\mathrm{Perm}(u' \oplus x_0, \delta') = y\right] \cdot \Pr\left[\mathrm{Perm}(u' \oplus x_0', \delta') \oplus cw' = z\right] \\
\overset{(a)}{=}&\Pr\left[\mathrm{Perm}(u' \oplus x_0, \delta') = y\right] \\
\overset{(b)}{=}&\frac{C_n^{n_1} \cdot A_{n_1}^{n_1} \cdot A_{n_0}^{n_0}}{2^n \cdot n!} = \frac{1}{2^n},
\end{aligned}$$

$$\begin{aligned}
&\Pr\left[C = (y,z) \mid X = (x_1, x_1')\right] \\
=&\Pr\left[\mathrm{Perm}(u' \oplus x_1, \delta') = y \ \&\ \mathrm{Perm}(u' \oplus x_1', \delta') \oplus cw' = z\right] \\
=&\Pr\left[\mathrm{Perm}(u' \oplus x_1, \delta') = y\right] \cdot \Pr\left[\mathrm{Perm}(u' \oplus x_1', \delta') \oplus cw' = z\right] \\
\overset{(a)}{=}&\Pr\left[\mathrm{Perm}(u' \oplus x_1, \delta') = y\right] \\
\overset{(b)}{=}&\frac{C_n^{n_1} \cdot A_{n_1}^{n_1} \cdot A_{n_0}^{n_0}}{2^n \cdot n!} = \frac{1}{2^n}.
\end{aligned}$$

Here, n_1 and n_0 are the numbers of '1's and '0's in y, respectively, where $n_1 + n_0 = n$. The parameters u' and δ' are the adversary's guesses for u and δ, respectively. (a) $\Pr\left[\mathrm{Perm}(u' \oplus x_i', \delta') \oplus cw' = z\right] = 1$, because the adversary can always find a bit-string cw' such that $\mathrm{Perm}(u' \oplus x_i', \delta') \oplus cw' = z$ for any x_i', where $x = 1$ or 2. (b) The total number of possible perturbations is $2^n \cdot n!$, and for any x_i ($x = 1$ or 2), there are $C_n^{n_1} \cdot A_{n_1}^{n_1} \cdot A_{n_0}^{n_0}$ perturbations such that $\mathrm{Perm}(u' \oplus x_i, \delta') = y$.

Then the two conditional probabilities are equal, i.e.,

$$\Pr\left[C = (y,z) \mid X = (x_0, x_0')\right] = \Pr\left[C = (y,z) \mid X = (x_1, x_1')\right]$$

Thus, the adversary cannot determine which data pair was selected, and we can conclude that the perturbation of FVP satisfies the indistinguishability under chosen plaintext attack (IND-CPA), ensuring that the perturbation is secure.

As a result, the prover cannot infer the XOR value u or the seed δ of the permutation function. Therefore, both the XOR and permutation processes remain secure against the prover. □

B.2 Proof of Theorem 2

Proof. An adversary's target is to convince the recipient to accept a fingerprinted database $\overline{\mathbf{R}}$ that has bitwise difference from the original database $\mathbf{R}$ exceeding the specified threshold (i.e., $t \cdot n_c \cdot n_g$). The adversary and the recipient first execute the UtiliClear.Preproc algorithm before modification. Then the adversary attempts to pass the verification of the UtiliClear.Verify algorithm.

According to Definition 4, given two databases ($\mathbf{R}$ $\overline{\mathbf{R}}$) satisfying $\mathrm{Dis}(\mathbf{R}, \overline{\mathbf{R}}) > (t \cdot n_c \cdot n_g)$, the adversary needs to forgery locked string Φ' of insignificant bits such that the UtiliClear.Verify$(\overline{\mathbf{R}}, pp, n_c, n_g, k, t, \mathrm{P}, \Psi, \mathrm{OP}, \Phi', \mathrm{VP})$ algorithm outputs 1, where (pp, kp_o, kp_r) are generated using the UtiliClear.Setup(κ) algorithm, $(\Psi, \mathrm{OP}, \mathrm{VP})$ are the recipient's outputs of the UtiliClear.Preproc$(\mathbf{R}, pp, kp_o, k, \nu, \mathrm{P})$ algorithm, n_c is the number of columns, n_g is the number of groups, P contains the positions of insignificant bits, and t is the number of modifiable bits in each group.

$\mathrm{Dis}(\mathbf{R}, \overline{\mathbf{R}}) > (k \cdot t \cdot \nu)$ means that the adversary should make modifications to at least one group of significant bits or modified more than t bits on one group of insignificant bits. Thus, there exists at least one group pair (e.g., the j-th group of the i-th attribute) $\mathrm{IC}[i][j]$ and $\mathrm{IC}'[i][j]$ (or $\mathrm{SC}[i][j]$ and $\mathrm{SC}'[i][j]$) between $\mathbf{R}$ and $\overline{\mathbf{R}}$ satisfying $\mathrm{Dis}(\mathrm{IC}[i][j], \mathrm{IC}'[i][j]) > t$ (or $\mathrm{SC}[i][j] \neq \mathrm{SC}'[i][j]$).

If the adversary modified significant bits (i.e., $\mathrm{SC}[i][j] \neq \mathrm{SC}'[i][j]$), the commitment $\Psi[i][j]$ of $\mathrm{SC}[i][j]$ cannot pass the verification PC.Open$(\mathrm{H}_1(\mathrm{SC}'[i][j]), \mathrm{OP}[i][j], pp, \Psi[i][j])$ according to the binding property of Pedersen commitment. On the other hand, if the adversary modified more than t insignificant bits (i.e., $\mathrm{Dis}(\mathrm{IC}[i][j], \mathrm{IC}'[i][j]) > t$), the employed FPV protocol ensures that the adversary cannot forgery $\Phi'[i][j]$ to pass the verification. According to Theorem 1, the FPV protocol satisfies the fuzzy-binding property, indicating that any adversary has a negligible probability of convincing the recipient to accept $\mathrm{IC}'[i][j]$ (i.e., FPV.Verify$(\Phi'[i][j], \mathrm{IC}'[i][j], \mathrm{VP}[i][j]) \to 1$) for given $\mathrm{Dis}(\mathrm{IC}[i][j], \mathrm{IC}'[i][j]) > t$, FPV.Setup$(\kappa) \to (pk, sk)$, and FPV.Lock$(sk, pk, \mathrm{IC}[i][j]) \to (\Phi[i][j], \mathrm{VP}[i][j])$.

As a result, if $\mathrm{Dis}(\mathbf{R}, \overline{\mathbf{R}}) > (t \cdot n_c \cdot n_g)$, at least one group of significant bits cannot pass the verification of Pedersen commitment, or one group of insignificant bits fails to pass the verification of FPV, causing the UtiliClear.Verify algorithm to return 0. Therefore, UtiliClear can achieve modification extent verification for the recipient. □

B.3 Proof of Theorem 3

Proof. Given the DO's public key pk (i.e., the public key of GM encryption), a fingerprinted database $\overline{\mathbf{R}}$ of $\mathbf{R}$, the verification parameter sets $(\Psi, \mathrm{VP}, \mathrm{OP})$, the recipient attempts to obtain the differing positions between $\mathbf{R}$ and $\overline{\mathbf{R}}$ from these data $(\overline{\mathbf{R}}, pk, \Psi, \mathrm{VP}, \mathrm{OP})$.

We first demonstrate that the recipient cannot obtain the differing positions between $\mathbf{R}$ and $\overline{\mathbf{R}}$ by attempting all insignificant bits. Whether the position of an insignificant bit, e.g., the j-th bit of the i-th attribute of record r, was modified is determined by the seed $s = sk \parallel \mathrm{r}.key \parallel \mathrm{r}[i] \parallel j$. The DO's secret key sk is an essential part for the adversary to construct the seed. However, the secret key sk is secure under the quadratic residuosity assumption [34], preventing the recipient to obtain the secret key sk from pk. Thus, the recipient cannot attempt all insignificant bits to obtain the differing positions using the fingerprinted database $\overline{\mathbf{R}}$ and pk.

We then prove that the verification parameter sets $(\Psi, \mathrm{VP}, \mathrm{OP})$ cannot leak any information about the differing positions between $\mathbf{R}$ and $\overline{\mathbf{R}}$. Since the commitments Ψ and verification parameters OP are associated with significant bits, they cannot reveal any information of modified positions as the fingerprint is embedded into insignificant bits. For the verification parameter VP of insignificant bits, such as $\mathrm{VP}[i][j]$ for insignificant bits contained in the j-th group of i-the column (i.e., $\mathrm{IC}[i][j]$), it is generated from the locking phase of the FPV. According to Theorem 1, the proposed FPV satisfies the full-hiding property, indicating that the recipient cannot obtain the differing positions between the original data $\mathrm{IC}[i][j]$ and the modified data $\mathrm{IC}'[i][j]$. Therefore, the recipient cannot obtain any information of fingerprint positions from the verification parameter sets $(\Psi, \mathrm{VP}, \mathrm{OP})$.

As a result, UtiliClear can ensure the confidentiality of fingerprint positions from the recipient.

□

B.4 Proof of Database Privacy

In order to support database operations, such as intersection, union and uploading, the primary keys are required to be immutable [18, 33], when a database is fingerprinted. As a result, the conventional ε-DP [49], which obfuscates the presence or absence of a record in the database, is not appropriate in the setting of database distribution, and the common definition of neighboring databases that differ by one record in the literature of ε-DP does not apply in the context of fingerprinting database. The following alternative definitions of neighboring databases and ε-entry-level DP are formally presented in [18] for fingerprinting database.

Definition 5. *(Neighboring relational databases [18]) Given that* $\mathbf{R}$ *and* $\mathbf{R}'$ *differ by at most one item (i.e., an attribute of a single record), they are neighboring databases.*

Definition 6. *(ε-entry-level differential privacy [18]). A randomized algorithm* $\mathcal{F} : \mathbb{R} \to \mathbb{D}$ *satisfies* ε*-entry-level DP if, for any neighboring databases* $\mathbf{R}$ *and* $\mathbf{R}'$ *and all* $D \in \mathbb{D}$*, it holds that* $\Pr[\mathcal{F}(\mathbf{R}) = D] \leq e^{\varepsilon} \cdot \Pr[\mathcal{F}(\mathbf{R}') = D]$*, where* $\varepsilon > 0$.

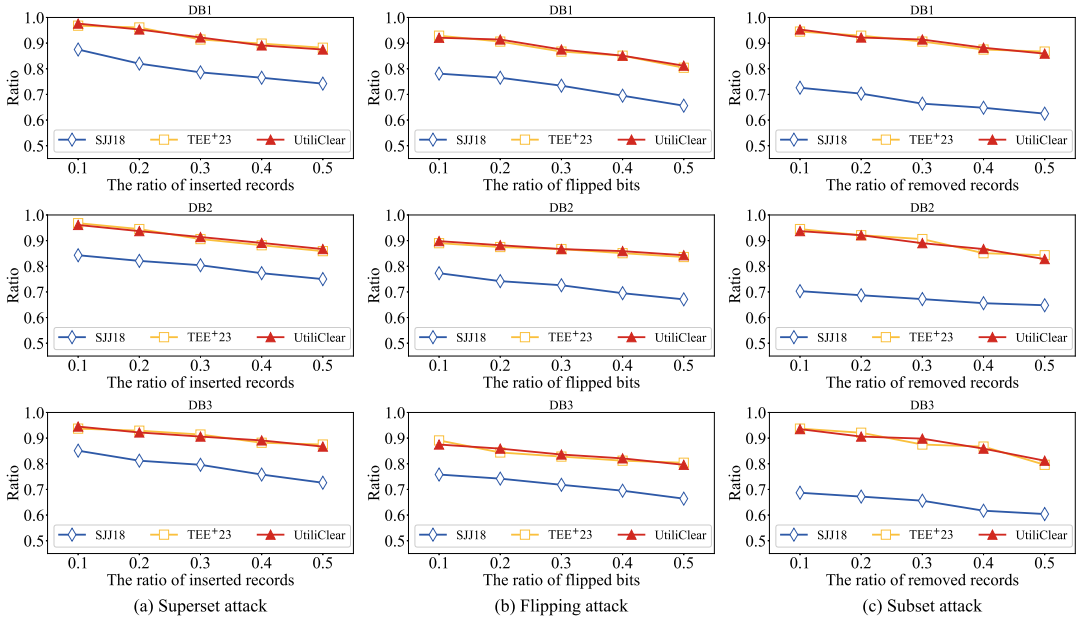
For any query or analysis performed on the fingerprinted database, ε-entry-level DP ensures that no specific attribute value of an individual record can be inferred from the result. UtiliClear employs the database fingerprinting method in [18] and introduces tolerable perturbation to the database by modifying insignificant bits. Assume $\mathrm{r}[i][j]$ (i.e., the j-th bit of i-th attribute of data record r) is selected to embed fingerprint bit, our scheme modifies its value to $\mathrm{r}[i][j] \oplus \sigma_{i,j}$, where $\sigma_{i,j}$ is a Bernoulli random variable with parameter τ.

Theorem 5. *Given the position of insignificant bits* $\{s_i, e_i\}_{i=1}^{n_c}$*, the number of groups* n_g*, and the number of modifiable bits in each group* t*, the fingerprinted database of UtiliClear satisfies* ε*-entry-level DP, if* $\frac{n_g \cdot t}{2N \cdot d_i} \geq \frac{1}{e^{\varepsilon/d_i}+1}$ *for* $(1 \leq i \leq n_c)$*, where* $d_i = e_i - s_i + 1$*, and* N *is the total number of records in the database.*

Proof. Given a pair of neighboring relational database $\mathbf{R}$ and $\mathbf{R}'$ that differ by one entry (i.e., the i-th attribute values between $\mathrm{r}[i]$ and $\mathrm{r}'[i]$ differ by at most K), our fingerprinting method $\mathcal{F}$ requires d_i bits to encode the difference and each bit is modified with a probability of τ. Given that the outputs of pseudorandom sequence generator S and hash function H_2 are uniformly distributed, a bit position j of $\mathrm{r}[i]$ is fingerprinted with the probability of the $\frac{n_g \cdot t}{N \cdot d_i}$, and the mark bit $\sigma_{i,j}$ takes value 1 with a probability of $\frac{1}{2}$. As a result, the probability of the bit being changed (i.e., $\mathrm{r}[i][j]$ is XORed by 1) is $\tau = \frac{n_g \cdot t}{2N \cdot d_i}$. Subsequently, by applying Definition 6, we have

$$
\begin{aligned}
&\frac{\Pr\left(\mathcal{F}(\mathbf{R}) = \overline{\mathbf{R}}\right)}{\Pr\left(\mathcal{F}(\mathbf{R}') = \overline{\mathbf{R}}\right)} \\
&= \prod_{j=s_i}^{e_i} \frac{\Pr\left(\mathrm{r}[i][j] \oplus \sigma_{i,j} = \bar{\mathrm{r}}[i][j]\right)}{\Pr(\mathrm{r}'[i][j] \oplus \sigma'_{i,j} = \bar{\mathrm{r}}[i][j])} \\
&= \prod_{j=s_i}^{e_i} \frac{\Pr(\sigma_{i,j} = \mathrm{r}[i][j] \oplus \bar{\mathrm{r}}[i][j])}{\Pr(\sigma'_{i,j} = \mathrm{r}'[i][j] \oplus \bar{\mathrm{r}}[i][j])} \\
&\overset{(a)}{=} \prod_{j=s_i}^{e_i} \frac{\tau^{(\mathrm{r}[i][j] \oplus \bar{\mathrm{r}}[i][j])} \cdot (1-\tau)^{(1-\mathrm{r}[i][j] \oplus \bar{\mathrm{r}}[i][j])}}{\tau^{(\mathrm{r}'[i][j] \oplus \bar{\mathrm{r}}[i][j])} \cdot (1-\tau)^{(1-\mathrm{r}'[i][j] \oplus \bar{\mathrm{r}}[i][j])}} \\
&\overset{(b)}{=} \prod_{j=s_i}^{e_i} \left(\frac{1-\tau}{\tau}\right)^{(\mathrm{r}[i][j] - \mathrm{r}'[i][j])(2\bar{\mathrm{r}}[i][j]-1)} \\
&\leq \prod_{j=s_i}^{e_i} \left(\frac{1-\tau}{\tau}\right)^{(|\mathrm{r}[i][j] - \mathrm{r}'[i][j]|(2\bar{\mathrm{r}}[i][j]-1))} \\
&\leq \prod_{j=s_i}^{e_i} \frac{1-\tau}{\tau},
\end{aligned}
$$

where (a) can be obtained as each bit $\mathrm{r}[i][j]$ ($\mathrm{r}'[i][j]$) of entry



(a) Superset attack (b) Flipping attack (c) Subset attack

Figure 12: Evaluation for the ratio of matched fingerprint bits against different attacks.

r[i] (or r′[i]) is modified independently with probability τ, and (b) follows from the formula $a \oplus b = (1-a)b + a(1-b)$ for two binary variables a and b.

Following this, by limiting the result $\prod_{j=s_i}^{e_i} \frac{1-\tau}{\tau} \leq e^{\varepsilon}$, we can obtain

$$\frac{1-\tau}{\tau} \leq e^{\varepsilon/d_i}$$
$$\frac{1}{e^{\varepsilon/d_i}+1} \leq \tau$$
$$\frac{1}{e^{\varepsilon/d_i}+1} \leq \frac{n_g \cdot t}{2N \cdot d_i},$$

which completes the proof. □

C Evaluation Results of Fingerprint Robustness

In the following fingerprint robustness comparisons, we compare UtiliClear with two relevant schemes (i.e., SJJ18 [15] and TEE$^+$23 [18]). We set the parameters t to 128, and configure n_g for DB1, DB2, and DB3 to 110000, 220000, and 4675000, respectively. Concurrently, we adjust the parameter p of TEE$^+$23 to 0.02, ensuring that the number of modified bits is roughly equivalent to that of UtiliClear.

The random bit flipping attack [43], subset attack [40], and superset attack [44] are common attacks against database fingerprint robustness. In the random bit flipping attack, a dishonest recipient randomly flips a proportion of bits to remove the fingerprint, distorting the fingerprint extraction of the DO. The subset attack erases the fingerprint by randomly removing a proportion of records, while the superset attack distorts the extracted fingerprint by inserting partial forged records into the fingerprinted database. We evaluate the fingerprint robustness against these attacks.

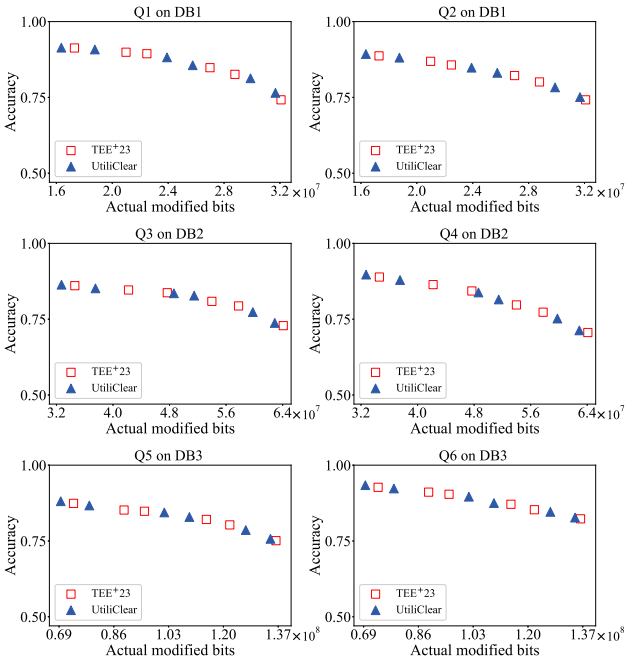
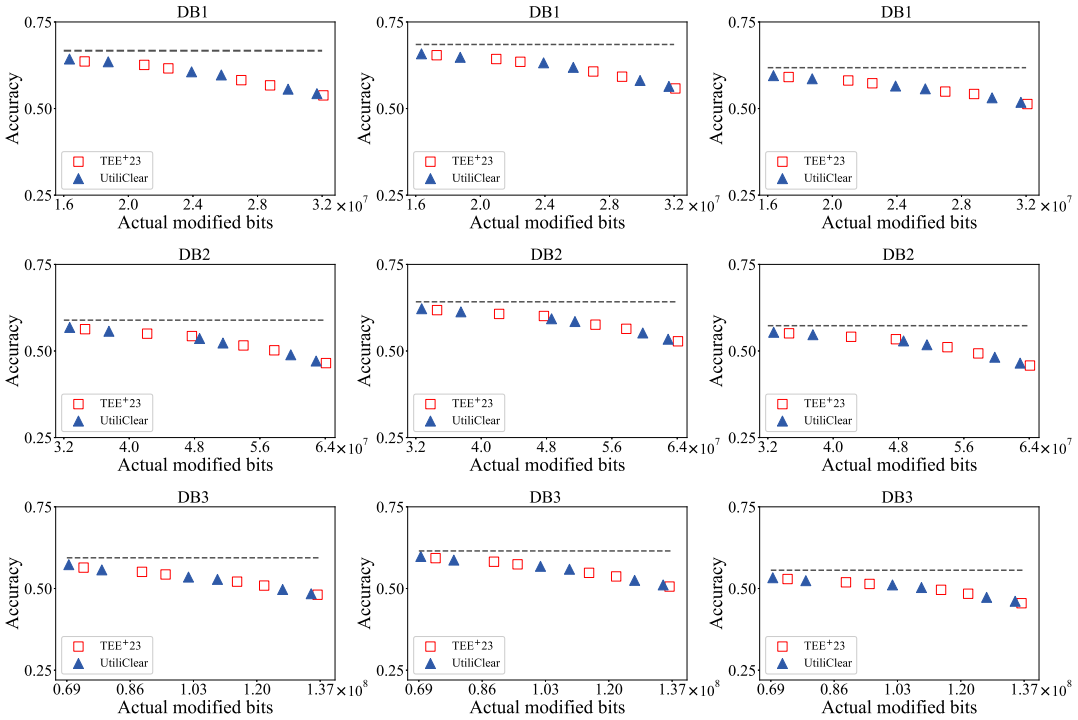


Figure 13: Query accuracy across different queries on fingerprinted databases, varying the extents of modifications.

Figure 12 presents the ratios of matched bits between the extracted fingerprints and original fingerprints across various ratios of inserted records, flipped bits, and removed records.



(a) KNN (b) SVM (c) LR

Figure 14: Classification accuracy across different tasks on fingerprinted databases, varying the extents of modifications.

UtiliClear achieves comparable fingerprint robustness with the employed scheme (i.e., TEE$^+$23) in three datasets, outperforming SJJ18. This is because UtiliClear and TEE$^+$23 insert fingerprint to all the attribute values of each record, while SJJ18 only inserts several forged records to embed fingerprint. The experiment results demonstrate that UtiliClear does not decrease the fingerprint robustness of the employed fingerprinting scheme.

D Evaluation Results of Database Utility

We employ the following eight customized SQL queries and four classifiers (i.e., k-nearest neighbors (KNN), support vector machine (SVM), and logistic regression (LR)) using 60% of the records for training to evaluate the utility of fingerprinted databases. Additionally, since SJJ18 [15] is a record-level database fingerprinting scheme, we cannot use the number of modified bits to measure the extent of its modifications to the database. Therefore, we only compare the database utilities of UtiliClear and TEE$^+$23, as they are bit-level fingerprinting schemes.

- Q1: SELECT key FROM DB1 WHERE overall $=$ 3 AND category $=$ Cell_phones_and_Accessories_5
- Q2: SELECT key FROM DB1 WHERE overall $=$ 1 AND category $=$ Gift_Cards_5
- Q3: SELECT key FROM DB2 WHERE overall $=$ 2 AND category $=$ Electronics_5
- Q4: SELECT key FROM DB2 WHERE overall $<$ 3 AND category $=$ Software_5
- Q5: SELECT key FROM DB2 WHERE overall $\leq$ 3 AND category $=$ Video_Games_5
- Q6: SELECT key FROM DB2 WHERE overall $=$ 5 AND category $=$ Prime_Pantry_5

We present the evaluation results for the utility of fingerprinted databases under different extents of modifications in Figures 13 and 14. Since theoretical number of modified bits is controlled by distinct parameters in UtiliClear and TEE$^+$22, we can only ensure that they make similar extents of modifications by adjusting parameters related to fingerprint density. When UtiliClear and TEE$^+$22 apply similar extents of modifications to fingerprinted databases, they exhibit comparable query accuracy and classification accuracy. This demonstrates that UtiliClear does not compromise the database utility of the employed fingerprinting scheme. Additionally, as shown in Figure 14, UtiliClear maintains classification accuracy comparable to that of the original database (indicated by black lines), when modifications are minimal. As modifications increase, both classification accuracy and query accuracy decline. This trend confirms that UtiliClear enables the recipient to assess the database's actual utility through modification extent verification.