Group and Attack: Auditing Differential Privacy

Johan Lokna
jlokna@student.ethz.ch
ETH Zurich
Switzerland

Anouk Paradis
anouk.paradis@inf.ethz.ch
ETH Zurich
Switzerland

Dimitar I. Dimitrov
dimitar.iliev.dimitrov@inf.ethz.ch
ETH Zurich
Switzerland

Martin Vechev
martin.vechev@inf.ethz.ch
ETH Zurich
Switzerland

ABSTRACT

(ϵ, δ) differential privacy has seen increased adoption recently, especially in private machine learning applications. While this privacy definition allows provably limiting the amount of information leaked by an algorithm, practical implementations of differentially private algorithms often contain subtle vulnerabilities. This motivates the need for effective tools that can audit (ϵ, δ) differential privacy algorithms before deploying them in the real world. However, existing state-of-the-art-tools for auditing (ϵ, δ) differential privacy directly extend the tools for ϵ-differential privacy by fixing either ϵ or δ in the violation search, inherently restricting their ability to efficiently discover violations of (ϵ, δ) differential privacy.

We present a novel method to efficiently discover (ϵ, δ) differential privacy violations based on the key insight that many (ϵ, δ) pairs can be grouped as they result in the same algorithm. Crucially, our method is orthogonal to existing approaches and, when combined, results in a faster and more precise violation search.

We implemented our approach in a tool called Delta-Siege and demonstrated its effectiveness by discovering vulnerabilities in most of the evaluated frameworks, several of which were previously unknown. Further, in 84% of cases, Delta-Siege outperforms existing state-of-the-art auditing tools. Finally, we show how Delta-Siege outputs can be used to find the precise root cause of vulnerabilities, an option no other differential privacy testing tool currently offers.

CCS CONCEPTS

• **Security and privacy** → **Privacy-preserving protocols**; • **Mathematics of computing** → *Statistical software.*

KEYWORDS

Differential privacy, Auditing

ACM Reference Format:
Johan Lokna, Anouk Paradis, Dimitar I. Dimitrov, and Martin Vechev. 2023. Group and Attack: Auditing Differential Privacy. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 22 pages. https://doi.org/10.1145/3576915.3616607

CCS '23, November 26–30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0050-7/23/11.
https://doi.org/10.1145/3576915.3616607

Table 1: State-of-the-art algorithms where Delta-Siege found a privacy violation. Violations of MST, AIM, and Google's DP Gaussian were previously unknown, Meta's and Diffprivlib's were found recently by [26].

Library	DP Mechanism	Differential privacy def.
IBM's Diffprivlib [28]	Laplace	ϵ
	FloatGauss	(ϵ, δ)
	Analytic FloatGauss	(ϵ, δ)
Google's DP Library [25]	Gauss[a]	(ϵ, δ)
Meta's Opacus [51]	Gauss	(ϵ, δ)
MST [40]		$(\epsilon, \delta)^b$
AIM [41]		$(\epsilon, \delta)^b$

[a] For Google's DP library, we used the Python bindings from PyDP [45] as of January 10th 2023.

[b] These mechanisms rely on the privacy of intermediate results to ensure the privacy of the end-to-end mechanism. We showed that the intermediate results are not private, thus invalidating the privacy guarantees of the mechanisms.

1 INTRODUCTION

The proliferation of data and applications built on top of it, from healthcare to traffic prediction, has only increased interest in data-driven algorithms. However, the necessary data is often sensitive; both medical records and localization data contain personal information. When using data containing sensitive information, one must respect an individual's right to privacy, both from an ethical and legal perspective [13, 20, 24]. To design algorithms respecting privacy, the golden standard is *differential privacy*: a framework enabling the exact quantification of the amount of private information leaked by an algorithm. Since its introduction, differential privacy [21] has been widely adopted by major tech companies such as Google [25], Meta [51], and IBM [28], alongside governmental institutions such as the US Census Bureau[19].

Flawed Implementations. However, there is as often a gap between theory and practice. Multiple works have identified and exploited vulnerabilities in implementations, showing how slight deviations from the theoretical algorithms can be enough to compromise differential privacy guarantees. Those vulnerabilities can completely invalidate any guarantees. For instance, [31, 26, 42] showed that common implementations of the Laplacian and Gaussian mechanisms are vulnerable to attacks exploiting the last few bits of their floating point representation, allowing an attacker to completely recover the original data.

Need for Automated Auditing. As shown by the existence of these attacks, if differential privacy is ever going to live up to its promise, privacy-preserving algorithms' implementations must be thoroughly vetted before deployment. This has led to substantial research interest in tools [44, 9, 36] designed to automatically and efficiently audit state-of-the-art ϵ-differentially private algorithms.

(ϵ, δ) *differential privacy.* However, most recent differentially private algorithms use a more permissive definition of differential privacy: (ϵ, δ) differential privacy. This definition is harder to audit efficiently as the privacy parameter space is larger, and many privacy parameters now become incomparable; (ϵ, δ) is a stronger parameter than (ϵ', δ') only if *both* $\epsilon < \epsilon'$ and $\delta < \delta'$. Violations are hence harder to find and characterize, which in turn makes a direct extension of the auditing tools mentioned above slow and inefficient. For instance, a common extension strategy is to fix multiple values of δ and run the complete ϵ auditing process for each, which is not only slow but can easily miss some violations. This motivates the need for an effective method that can efficiently explore the space of (ϵ, δ) potential violations.

This Work. In this work, we present a novel method to efficiently explore potential violations of (ϵ, δ) differential privacy. The key technical insight of our work is that many (ϵ, δ) differentially private algorithms combine ϵ and δ into a single privacy parameter ρ. This means that multiple values of (ϵ, δ) may lead to the exact same algorithm, as the value of ρ entirely determines the algorithm. Therefore, such an algorithm must satisfy (ϵ, δ) differential privacy for an entire set of (ϵ, δ) values. By using ρ directly, we can hence audit multiple privacy claims *at the same time*. Importantly, our method is orthogonal to many of the existing auditing tools and can be used to amplify their auditing power; it can also be leveraged to audit ϵ differential privacy.

We implemented our method within the state-of-the-art auditing framework DP-Opt [44], resulting in our new tool Delta-Siege. As shown in Table 1, Delta-Siege detected violations in most audited algorithms, including ones thought to be correctly implemented. Further, we found that 84% of the time, Delta-Siege outperforms other state-of-the-art auditing tools, finding violations up to 16 times more severe. Finally, we demonstrate how Delta-Siege can be leveraged to provide practical counter-examples when violations are found, enabling users to understand the source of the vulnerabilities in audited mechanisms.

Main Contributions. Our main contributions are:

- A characterization of differential privacy algorithms that combines ϵ and δ into a single privacy parameter ρ that efficiently describes the space of (ϵ, δ) differential privacy violations.
- A method to efficiently search for (ϵ, δ) differential privacy violations for such algorithms.
- An end-to-end implementation of this method extending DP-Opt [44] in our tool Delta-Siege.
- A thorough evaluation against state-of-the-art differentially private algorithms showing that Delta-Siege drastically outperforms related works in almost all tested scenarios.
- A new method to identify the root causes of violations discovered using the DP-Opt framework.

2 BACKGROUND

We now provide the background needed to understand our contributions. Intuitively, a randomized algorithm $M : \mathbb{A} \rightarrow \mathbb{B}$ is differentially private if for any neighboring inputs a, a', the output of $M(a)$ and $M(a')$ are indistinguishable with high probability. Therefore, when observing the output of M, an adversary cannot confidently know whether the original input was a or a'.

To formalize this definition, we first define for some a in $\mathbb{A}$ its set of neighbors as $\mathcal{N}(a)$. This set of neighbors depends on the algorithm at hand; if a is a database, $\mathcal{N}(a)$ is often defined as the set of all databases where a row of a was either dropped or added. If a is a vector, $\mathcal{N}(a)$ can be the set of all vectors a' such that $||a - a'|| \leq 1$ for some norm $|| \cdot ||$.

We formalize the "indistinguishable with high probability" notion using the standard definition of (ϵ, δ) differential privacy:

Definition 1. *For any $\epsilon \geq 0$ and δ in [0, 1], we say a randomized algorithm $M : \mathbb{A} \rightarrow \mathbb{B}$ is (ϵ, δ)-differentially private if and only if*

$$\forall a \in \mathbb{A}, \forall a' \in \mathcal{N}(a), \forall S \subseteq \mathbb{B} \ : \ \frac{\mathbb{P}[M(a) \in S] - \delta}{\mathbb{P}[M(a') \in S]} \leq \exp(\epsilon) \quad (1)$$

We denote $\mathcal{P} = \mathbb{R}_{\geq 0} \times [0, 1]$ the set of valid (ϵ, δ) parameter pairs.

Auditing through attacks. In this work, we focus on auditing differentially private algorithms. Given an algorithm M claimed to be (ϵ, δ) differentially private, we look for an attack (a, a', S) that would invalidate this claim. Such an attack is a violation if:

$$\frac{\mathbb{P}[M(a) \in S] - \delta}{\mathbb{P}[M(a') \in S]} > \exp(\epsilon)$$

A violation here indicates a vulnerability in the algorithm, namely that the claimed differential privacy does not hold.

3 OVERVIEW

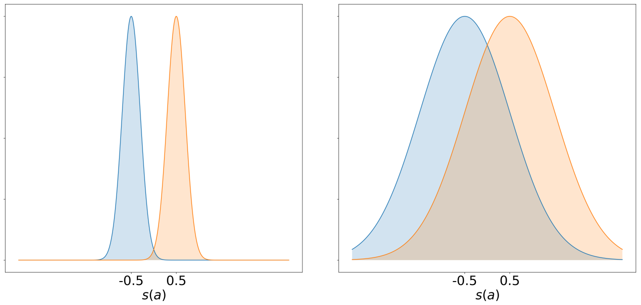
We illustrate in this section the key technical insight of our work on a running example, namely how to amplify the effectiveness of existing attacks on differential privacy by considering not only a single (ϵ, δ) pair but an equivalence class of these.

Our running example will be the Gaussian mechanism, introduced in Section 3.1. Then, in Section 3.2, we illustrate how we can group multiple privacy claims for this example.

3.1 Gaussian Mechanism

Algorithm 1 shows a differentially private mechanism M based on Gaussian noise.

Choosing σ. The function BuildsM builds for given parameters ϵ, δ a mechanism M that is (ϵ, δ) differentially private. To do so, it first computes in Line 2 which amount of noise should be used, quantified by the standard deviation $\sigma = \frac{1}{\epsilon}\sqrt{2 \log \frac{1.25}{\delta}}$. Using this value, it then builds the instantiation M_σ of the mechanism. This instantiation computes some function f from its input a. This result, stored in s in Line 5, may leak private information from a and should be protected. To do so, instead of returning the result directly, some



(a) Low σ: low privacy and high ϵ, δ.

(b) High σ: high privacy and low ϵ, δ.

Figure 1: Probability distribution of the output of $M_\sigma(a)$ (blue) and $M_\sigma(a')$ (orange) for different values of σ. We assume $f(a) = -0.5$ and $f(a') = 0.5$.

Algorithm 1 Building a differentially private M

```
1: func BuildsM($\epsilon, \delta$)
2:     $\sigma \leftarrow \frac{1}{\epsilon}\sqrt{2 \log \frac{1.25}{\delta}}$
3:     return $M_\sigma(\cdot)$
4: func $M_\sigma(a)$
5:     $s \leftarrow f(a)$
6:     Sample $Z \sim \mathcal{N}(0, \sigma^2)$
7:     return $s + Z$
```

Gaussian noise Z with standard deviation σ is added to it. It has been shown in [22] that M_σ is (ϵ, δ) differentially private [1].

Privacy Intuition and σ. From the formula defining σ in Line 2, it is clear that for lower values of ϵ or δ, meaning higher privacy, we will end up with higher values for the standard deviation. We show this visually in Fig. 1. Here, suppose $f(a) = -0.5$ and $f(a') = 0.5$. We then show in blue the probability distribution of $M_\sigma(a)$ and in orange that of $M_\sigma(a')$. For smaller values of σ, shown in Fig. 1a, we can deduce from the output of M its input with high probability; if it is close to -0.5, it is almost certain that the input was a; a consequence of the blue probability being very high and the orange very low. That is, a smaller standard deviation σ corresponds to having low privacy guarantees, implying high values of ϵ, δ. On the other hand, if σ is high as in Fig. 1b, we cannot deduce with confidence the input to M_σ from its output, as the distributions overlap substantially. Thus, we obtain stronger privacy guarantees, meaning smaller values for ϵ, δ.

3.2 Grouping Privacy Claims

We now show how we can use the standard deviation σ to characterize more potential violations of claimed (ϵ, δ) differential privacy and audit multiple privacy claims at once.

Auditing a differentially-private mechanism. In the auditing context, we are given an instance M_{σ_0} of the Gaussian mechanism described above, with $\sigma_0 = \sigma(\epsilon_0, \delta_0)$. Our goal is to find, if it exists, a counter-example (a, a', S) showing that M_{σ_0} is not (ϵ_0, δ_0) differentially private as claimed. Suppose we discovered some inputs a, a', output set S and values $\hat{\epsilon}_0, \hat{\delta}_0$ such that:

$$\frac{\mathbb{P}\left[M_{\sigma_0}(a) \in S\right] - \hat{\delta}_0}{\mathbb{P}\left[M_{\sigma_0}(a') \in S\right]} > \exp(\hat{\epsilon}_0) \tag{2}$$

We want to characterize whether this example is a violation of the (ϵ, δ) differential privacy claim about $M_{(\cdot)}$.

Clear (ϵ, δ) Violations. If both $\hat{\epsilon}_0 \geq \epsilon_0$ and $\hat{\delta}_0 \geq \delta_0$, Eq. (2) translates directly into a violation of (ϵ_0, δ_0) differential privacy for M_{σ_0}:

$$\frac{\mathbb{P}\left[M_{\sigma_0}(a) \in S\right] - \delta_0}{\mathbb{P}\left[M_{\sigma_0}(a') \in S\right]} \geq \frac{\mathbb{P}\left[M_{\sigma_0}(a) \in S\right] - \hat{\delta}_0}{\mathbb{P}\left[M_{\sigma_0}(a') \in S\right]} > \exp(\hat{\epsilon}_0) \geq \exp(\epsilon_0)$$

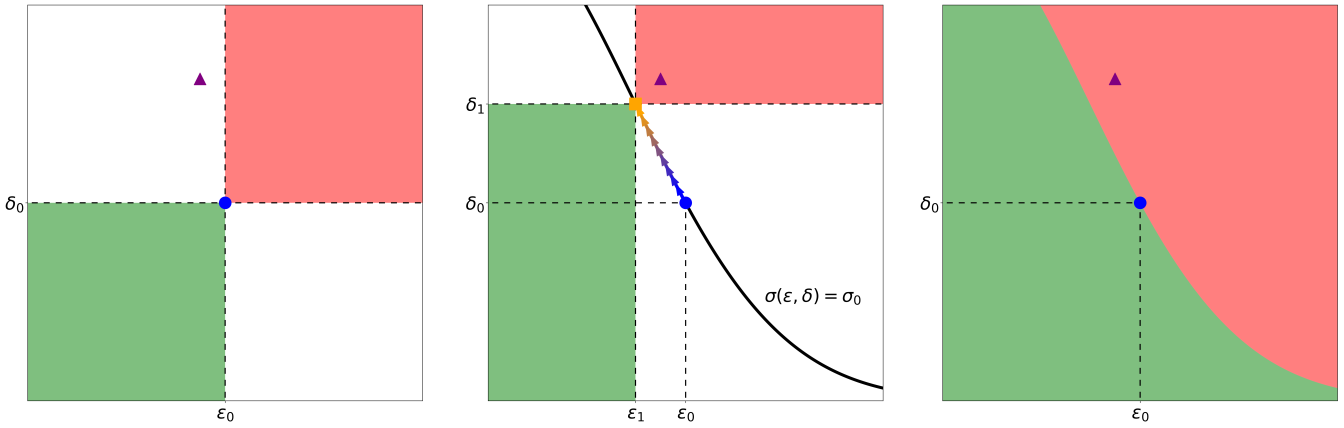
We visualize this set of clear violations in red in Fig. 2a. On the other hand, if both $\hat{\epsilon}_0 < \epsilon_0$ and $\hat{\delta}_0 < \delta_0$, Eq. (2) is not a violation of the claimed (ϵ_0, δ_0) differential privacy, as $(\hat{\epsilon}_0, \hat{\delta}_0)$ is a strictly tighter privacy condition than (ϵ_0, δ_0). Further, if M_{σ_0} is precisely (ϵ_0, δ_0) differentially private, that is, it is not differentially private for any $(\epsilon < \epsilon_0, \delta < \delta_0)$, we *expect* that M_{σ_0} is not $(\hat{\epsilon}_0, \hat{\delta}_0)$ differentially private and that there exist some (a, a', S) such that Eq. (2) holds. In Fig. 2a, we show in green this set of strictly tighter conditions, for which finding a violation as in Eq. (2) does not invalidate the privacy claim of M_{σ_0}.

However, this still leaves a significant portion of the space white; if $(\hat{\epsilon}_0, \hat{\delta}_0)$ belongs to that white space (shown as ▲ in Fig. 2a), we cannot immediately conclude anything from this fact. We will now show how by using σ *level sets*, we can figure out whether ▲ violates the differential privacy claims about $M_{(\cdot)}$.

Using σ to extend the violation set. The key insight in our work is that multiple values of (ϵ, δ) may yield the same value of σ. Using this, we would like to choose (ϵ_1, δ_1) such that $\sigma_1 = \sigma(\epsilon_1, \delta_1) = \sigma_0$ and ▲ is in the clear violation set (red quadrant) for (ϵ_1, δ_1). To do so, we plot in Fig. 2b the level set $\{(\epsilon, \delta) \mid \sigma(\epsilon, \delta) = \sigma_0\}$. We can now simply pick (ϵ_1, δ_1) in this set as shown by 🟧. Eq. (2) is now a clear (ϵ_1, δ_1) differential privacy violation for $M_{\sigma_1 = \sigma_0}$ as both $\hat{\epsilon}_0 \geq \epsilon_1$ and $\hat{\delta}_0 \geq \delta_1$. We have hence shown that the *general* differential privacy mechanism $M_{(.)}$ described by BuildsM in Algorithm 1 is not differentially private as claimed. If it were, $M_{\sigma_1 = \sigma_0}$ would have been (ϵ_1, δ_1) differentially private, and we have just shown this to be false using Eq. (2).

σ Violations. The procedure we used above to translate Eq. (2) into a clear differential privacy violation for $M_{\sigma_1 = \sigma_0}$ can be used for any (ϵ, δ) that is above the level set in Fig. 2b, that is, where $\sigma(\epsilon, \delta) < \sigma_0$. We can hence define those as the new violation set, shown in red in Fig. 2c. Symmetrically, any (ϵ, δ) such that $\sigma(\epsilon, \delta) > \sigma_0$ can be translated to a tighter condition which we expect M_{σ_0} to violate. Again, we show those in green.

[1] Here we assume that the function f has outputs in $\mathbb{R}$ and has L_2 sensitivity $\Delta \leq 1$, that is to say for any neighboring inputs a, a', $|f(a) - f(a')| \leq 1$.



(a) (ϵ_0, δ_0) differential privacy violation set.

(b) Moving the violation set.

(c) σ_0 differential privacy violation set.

Figure 2: Extending the set of identified differential privacy violations using σ. Here ● is (ϵ_0, δ_0), ▲ $(\hat{\epsilon}_0, \hat{\delta}_0)$ and ■ (ϵ_1, δ_1)

Auditing multiple privacy claims at once. We leveraged the insight that because $M_{(\cdot)}$ is defined entirely through σ (and does not use (ϵ, δ) directly), many values of (ϵ, δ) yield the same mechanism M_{σ_0}. Hence, any instantiation M_{σ_0} should be (ϵ, δ) differentially private for *all* (ϵ, δ) in the level set $\{(\epsilon, \delta) \mid \sigma(\epsilon, \delta) = \sigma_0\}$. Therefore, when given an instantiation M_{σ_0} of a privacy mechanism $M_{(\cdot)}$, we can audit all those privacy claims *simultaneously*, and any violation will be enough to show that $M_{(\cdot)}$ is not actually private.

3.3 Using Delta-Siege in practice

We implemented the grouping approach explained above in our tool Delta-Siege, which we present in more detail in Section 5. We now showcase how we can use this tool not only to audit but also to debug implementations of differential privacy mechanisms.

Continuous integration. Delta-Siege is fully automated and takes between 30s and 5 minutes to audit most differentially private mechanisms. It can therefore be run directly as part of a continuous integration pipeline. OpenDP [45] has already expressed interest.

Finding Usable Counter-Examples. While finding violations, and therefore avoiding incorrectly implemented differentially private mechanisms, is an important security goal from a user perspective, allowing programmers to find the *root cause* of such violations is equally important from a programmer perspective. Therefore, Delta-Siege provides for each found violation the precise (ϵ_1, δ_1) (■ in Fig. 2b) with which to instantiate the differentially private mechanism $M_{(\cdot)}$, and the found violation $(\hat{\epsilon}_0, \hat{\delta}_0)$ (▲ in Fig. 2b). Further, Delta-Siege provides the set S for which Eq. (2) was verified, allowing programmers to inspect it in detail. We will show in Section 6 how we can reconstruct attacks against the mechanism using this set S and therefore understand the source of the vulnerabilities.

4 CHARACTERIZING PRIVACY VIOLATIONS

We showed in Section 3 that using the standard deviation σ directly, instead of (ϵ, δ), made finding privacy violations for implementations of the Gaussian mechanism easier. In this section, we generalize this insight by defining *ρ-ordered differentially private mechanisms* and use it to construct a criterion for discovering privacy violations.

4.1 ρ-ordered Differentially Private Mechanisms

A necessary characteristic of the standard deviation σ in Section 3 was that it was non-increasing in both parameters. Formally, for a function $\rho : \mathcal{P} \longrightarrow \mathbb{R}$, we say ρ is non-increasing in both parameters if and only if for any ϵ_0, δ_0 both $\rho_{\epsilon_0} : \delta \mapsto \rho(\epsilon_0, \delta)$ and $\rho_{\delta_0} : \epsilon \mapsto \rho(\epsilon, \delta_0)$ are non-increasing.

Upwards-invertibility. Another crucial property of σ was that from a potential violation $(\hat{\epsilon}_0, \hat{\delta}_0)$ of (ϵ_0, δ_0) differential privacy such that:

$$\hat{\sigma}_0 = \sigma(\hat{\epsilon}_0, \hat{\delta}_0) < \sigma_0 = \sigma(\epsilon_0, \delta_0)$$

we could build a clear violation (ϵ_1, δ_1) such that $\epsilon_1 \leq \hat{\epsilon}_0, \delta_1 \leq \hat{\delta}_0$ and $\sigma(\epsilon_1, \delta_1) = \sigma_0$. We call this property *upwards-invertibility* and define it as follows:

Definition 2. *Let $\rho : \mathcal{P} \longrightarrow \mathbb{R}$ be non-increasing in both parameters. ρ is upwards-invertible if for any (ϵ_0, δ_0) and $(\hat{\epsilon}_0, \hat{\delta}_0)$ in $\mathcal{P}$ such that $\rho(\hat{\epsilon}_0, \hat{\delta}_0) \leq \rho(\epsilon_0, \delta_0)$ there exists some ϵ_1, δ_1 such that $\hat{\delta}_0 \geq \delta_1 \geq 0$, $\hat{\epsilon}_0 \geq \epsilon_1 \geq 0$ and $\rho(\epsilon_1, \delta_1) = \rho(\epsilon_0, \delta_0)$.*

All continuous non-increasing functions enjoy the following property:

Lemma 1. *If $\rho : \mathcal{P} \longrightarrow \mathbb{R}$ is continuous and non-increasing in both parameters, then it is upwards-invertible.*

Proof. For given $\epsilon_0, \hat{\epsilon}_0, \delta_0, \hat{\delta}_0$ as in Definition 2, we can define $f : [0, 1] \longrightarrow \mathbb{R}$, which maps λ to $\rho(\lambda\epsilon_0, \lambda\delta_0)$. We then have that $f(0) = \rho(0, 0) \geq \rho(\hat{\epsilon}_0, \hat{\delta}_0) \geq f(1)$. Using the continuity of f, we can define λ in $[0, 1]$ such that $f(\lambda) = \rho(\hat{\epsilon}_0, \hat{\delta}_0)$. Finally, $\epsilon_1 = \lambda\epsilon_0$ and $\delta_1 = \lambda\delta_0$ are as needed. □

Using the above definition, we can finally define a ρ-ordered differentially private mechanism:

Definition 3. *Let $\rho : \mathcal{P} \longrightarrow \mathbb{R}$ be upwards invertible and non-increasing in both parameters. A parameterized randomized mechanism $M_{(\cdot)} : \mathbb{A} \to \mathbb{B}$ is ρ-ordered differentially private if and only if for any $(\epsilon, \delta) \in \mathcal{P}$*

$$\forall r \geq \rho(\epsilon, \delta) \ : \ M_r \text{ is } (\epsilon, \delta) \text{ differentially private} \tag{3}$$

A widely applicable definition. This definition directly applies to the Gaussian mechanism; for instance by choosing $\rho(\epsilon, \delta) = \sigma(\epsilon, \delta) = \frac{1}{\epsilon}\sqrt{2 \log \frac{1.25}{\delta}}$. This function is non-increasing in both parameters and continuous; therefore, also upwards invertible, using Lemma 1. Note that many other definitions of ρ would also be possible; for instance, using the variance σ^2 would also work. We show in Section 7.4 that most investigated (ϵ, δ) differentially private mechanisms are ρ-ordered differentially private.

A more widely applicable method. While we build our approach with ρ-ordered differentially private mechanisms in mind, we will show in Section 4.3 that we can apply it more generally and that it performs well in those cases too. Further, our results are always sound, regardless of the audited mechanism properties: any reported violation was statistically checked. Our method can therefore be applied directly to any mechanism defined using a ρ privacy parameter, without any preliminary proof-work.

4.2 Violation Criterion

We now define a violation criterion to prove that some mechanism M is not (ϵ, δ) differentially private. To do so, we first define for any probabilities p, p' in $[0, 1]$ the *worst violated* ρ and denote it by $\mathcal{V}(\rho, p, p')$:

$$\mathcal{V}(\rho, p, p') = \min\left\{\rho(\epsilon, \delta) \middle| (\epsilon, \delta) \in \mathcal{P} \text{ and } \tfrac{p-\delta}{p'} \geq \exp(\epsilon)\right\} \tag{4}$$

To gain intuition, consider for a given attack (a, a', S) the value $\mathcal{V}(\rho, \mathbb{P}[M(a) \in S], \mathbb{P}[M(a') \in S])$. It is the minimum of ρ over all (ϵ, δ) for which a, a' and S are a counter-example showing that M is not (ϵ, δ) differentially private. Now, using Eq. (4), we can derive the following property:

Lemma 2. *Let $\rho : \mathcal{P} \longrightarrow \mathbb{R}$ be non-increasing in both parameters and upwards-invertible and $M_{(\cdot)} : \mathbb{A} \to \mathbb{B}$ be a ρ-ordered differentially private mechanism. Then we have that for any (ϵ_0, δ_0) in $\mathcal{P}$, a, a' in $\mathbb{A}$ and S subset of $\mathbb{B}$,*

$$\mathcal{V}(\rho, \mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a) \in S\right], \mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a') \in S\right]) \geq \rho(\epsilon_0, \delta_0)$$

This lemma states that for any (ϵ, δ) where (a, a', S) shows that $M_{\rho(\epsilon_0,\delta_0)}$ is not (ϵ, δ) differentially private, we have $\rho(\epsilon, \delta) \geq \rho(\epsilon_0, \delta_0)$. This fits our intuition. Suppose M is the Gaussian mechanism, and σ_0 is the variance needed to ensure M_{σ_0} is (ϵ_0, δ_0) differentially private. Then for any (ϵ, δ) such that M_{σ_0} is not (ϵ, δ) differentially private, a greater standard deviation $\sigma(\epsilon, \delta) \geq \sigma_0$ would be needed to ensure (ϵ, δ) differential privacy. We now provide a proof of this lemma.

Proof. Have $\rho, M, \epsilon_0, \delta_0, a, a'$ and S as in the lemma. We show this by contraposition. Suppose that

$$\mathcal{V}(\rho, \mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a) \in S\right], \mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a') \in S\right]) < \rho(\epsilon_0, \delta_0)$$

Then there exists some (ϵ, δ) such that both

$$\frac{\mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a) \in S\right] - \delta}{\mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a') \in S\right]} \geq \exp(\epsilon)$$

and $\rho(\epsilon, \delta) < \rho(\epsilon_0, \delta_0)$. Using the upwards invertibility of ρ, we can define $\delta \geq \delta' \geq 0$ and $\epsilon \geq \epsilon' \geq 0$ such that $\rho(\epsilon', \delta') = \rho(\epsilon_0, \delta_0)$. Putting it all together, we finally get that

$$\begin{aligned}
\frac{\mathbb{P}\left[M_{\rho(\epsilon',\delta')}(a) \in S\right] - \delta'}{\mathbb{P}\left[M_{\rho(\epsilon',\delta')}(a') \in S\right]} &\overset{(1)}{\geq} \frac{\mathbb{P}\left[M_{\rho(\epsilon',\delta')}(a) \in S\right] - \delta}{\mathbb{P}\left[M_{\rho(\epsilon',\delta')}(a') \in S\right]} \\
&= \frac{\mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a) \in S\right] - \delta}{\mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a') \in S\right]} \\
&\geq \exp(\epsilon) \\
&\overset{(2)}{\geq} \exp(\epsilon')
\end{aligned}$$

Further, as $\rho(\epsilon, \delta) < \rho(\epsilon', \delta')$ either $\delta' < \delta$ or $\epsilon' < \epsilon$, so either (1) or (2) is strict, resulting in

$$\frac{\mathbb{P}\left[M_{\rho(\epsilon',\delta')}(a) \in S\right] - \delta'}{\mathbb{P}\left[M_{\rho(\epsilon',\delta')}(a') \in S\right]} > \exp(\epsilon')$$

That is, $M_{\rho(\epsilon',\delta')}$ is not (ϵ', δ') differentially private. This is a contradiction as we assumed M to be ρ-ordered differentially private, and hence $M_{\rho(\epsilon',\delta')}$ to be (ϵ', δ') differentially private. □

Using Lemma 2, we can finally define our violation criterion, which is simply its negation:

Criterion 3. *Let $\rho : \mathcal{P} \longrightarrow \mathbb{R}$ be non-increasing in both parameters and upwards-invertible. To show that a ρ-ordered randomized mechanism $M_{(\cdot)} : \mathbb{A} \to \mathbb{B}$ is not ρ-parameter differentially private, it is enough to find $(\epsilon_0, \delta_0) \in \mathcal{P}$, a, a' in A and S subset of B, such that:*

$$\mathcal{V}(\rho, \mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a) \in S\right], \mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a') \in S\right]) < \rho(\epsilon_0, \delta_0) \tag{5}$$

Attack Strength. Besides the criterion above, it is often interesting to estimate the violation magnitude of a given attack: is the attack barely violating the claimed privacy or completely invalidating it? For an algorithm claiming ϵ_0 differential privacy, this would simply be the ratio $\frac{\epsilon_m}{\epsilon_0}$ where ϵ_m is the maximum value such that:

$$\frac{\mathbb{P}\left[M_0(a) \in S\right]}{\mathbb{P}\left[M_0(a') \in S\right]} \geq \exp(\epsilon_m)$$

However, for (ϵ, δ) differential privacy, this cannot be directly translated, as there is no clear partial order on $\mathcal{P}$. We first need to define the worst violated ρ_m as:

$$\rho_m = \mathcal{V}(\rho, \mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a) \in S\right], \mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a') \in S\right])$$

We can then define the magnitude μ of an attack on an algorithm claiming (ϵ_0, δ_0) differential privacy as $\mu = \frac{\rho(\epsilon_0,\delta_0)}{\rho_m}$: the greater the magnitude is, the greater the violation is. Note that the numerator and denominator are switched compared to the definition of attack magnitude for ϵ-differential privacy. This is because high ρ means high privacy, whereas high ϵ means low privacy.

4.3 Beyond ρ-ordered Mechanisms

As shown above, we developed our method for ρ-ordered differentially private mechanisms. We now show how it can be applied to more general mechanisms, while still ensuring sound results.

No upwards-invertibility. Let us first study the case of a differentially private mechanism $M_{\rho(\cdot,\cdot)}$ where ρ is non-increasing in both parameters but not upwards-invertible. Suppose we have now found a potential violation, that is a, a', S, ϵ_0 and δ_0 such that:

$$\mathcal{V}(\rho, \mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a) \in S\right], \mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a') \in S\right]) < \rho(\epsilon_0, \delta_0) \quad (6)$$

As ρ is not upwards-invertible, we cannot simply use Criterion 3 to prove that the mechanism is not differentially private. Instead, we must *explicitly* find this violation. To do so, we want to define $(\hat{\epsilon_0}, \hat{\delta_0})$ such that:

$$\rho(\hat{\epsilon_0}, \hat{\delta_0}) = \mathcal{V}(\rho, \mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a) \in S\right], \mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a') \in S\right])$$

In practice, we estimate the value of $\mathcal{V}(\rho, \cdot, \cdot)$ by evaluating on ρ on witnesses (see Section 4.4), and therefore always have access to $(\hat{\epsilon_0}, \hat{\delta_0})$ for the minimum value of ρ on those witnesses.

We are now exactly in the case shown in Fig. 2b: we have a candidate violation $(\hat{\epsilon_0}, \hat{\delta_0})$, and need to find (ϵ_1, δ_1) such that $\rho(\epsilon_1, \delta_1) = \rho(\epsilon_0, \delta_0)$ and $(\hat{\epsilon_0}, \hat{\delta_0})$ is a clear violation of (ϵ_1, δ_1) privacy, that is, $\epsilon_1 < \hat{\epsilon_0}$ and $\delta_1 < \hat{\delta_0}$. To find such (ϵ_1, δ_1), we can now simply use a binary search, as ρ is non-increasing in both parameters. This search may fail, but if it succeeds, the violation is confirmed.

Further generalizability. In the above, we used ρ monotonicity to search for values. If ρ is not non-decreasing, our search strategy may not be efficient. However, any found result will still be correct. Our method can hence be applied to any differentially private mechanism $M_{\rho(\cdot,\cdot)}$, and its results are always statistically verified, whether or not ρ satisfies the properties outlined above.

Sensitivity. Finally, our approach does not rely on any assumptions on the neighborhood definition or sensitivity of the audited mechanism M. We simply take as given a, a' neighbors, and work with the output values of M for those inputs. Therefore, our approach can be used for any norm and value of sensitivity. For instance, we will show in Section 7 that we can use our method for both L_2 norm (for Gaussian and Laplacian mechanisms) and symmetric distance (for MST and AIM).

4.4 Estimating $\mathcal{V}$

We now introduce the methods we need to apply the definitions above in practice.

Upper-bound on $\mathcal{V}$. Criterion 3 uses the value of

$$\mathcal{V}(\rho, \mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a) \in S\right], \mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a') \in S\right])$$

However, we often cannot compute this value directly as we do not know the values of either probabilities. Luckily, to demonstrate a violation of privacy claims using Eq. (5), an upper bound on this value of $\mathcal{V}$ is enough. Further, based on the definition of $\mathcal{V}$ in Eq. (4), it is clear that for any $0 \leq \underline{p} \leq \mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a) \in S\right]$ and $1 \geq \overline{p} \geq \mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a') \in S\right]$, it holds that:

$$\mathcal{V}(\rho, \mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a) \in S\right], \mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a') \in S\right]) \leq \mathcal{V}(\rho, \underline{p}, \overline{p})$$

Our method can hence be easily integrated into any tool that provides a lower bound $\underline{p}$ of $\mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a) \in S\right]$ and an upper bound $\overline{p}$ of $\mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a) \in S\right]$.

Computing $\mathcal{V}$. Even for given $\underline{p}, \overline{p}$, computing $\mathcal{V}(\rho, \underline{p}, \overline{p})$ may not be trivial as it requires minimizing ρ over the set

$$\left\{\rho(\epsilon, \delta) \middle| (\epsilon, \delta) \in \mathcal{P} \text{ and } \frac{\underline{p}-\delta}{\overline{p}} \geq \exp(\epsilon)\right\}$$

To do so in practice, we first use that ρ is non-increasing in both parameters to perform the following rewrite [2]:

$$\min_{(\epsilon,\delta)\in\mathcal{R}} \rho(\epsilon, \delta) = \min_{\delta \leq \underline{p}} \rho\left(\log\left(\frac{\underline{p}-\delta}{\overline{p}}\right), \delta\right)$$

This reduces the search to a one-dimensional search over the interval $[0, \underline{p}]$. We then use the following approximation strategy, applicable for any function ρ. We log-uniformly discretize the subset $[10^{-9} \cdot \underline{p}, \underline{p}]$ with 900 sampling points, compute ρ values at each point and use as an approximation the minimum over all test points. We have found in practice that this simple method performs well. More precise strategies could be developed for a given ρ.

5 DELTA-SIEGE: AMPLIFYING DP-OPT

So far we defined ρ-ordered differentially private algorithms and a criterion to discover more (ϵ, δ) violations. We now show we can easily integrate this criterion in DP-Opt [44] to augment its auditing power. We first recall the key ideas of DP-Opt.

High-level idea. To audit a given $M_{\rho(\epsilon_0,\delta_0)}$ with M a ρ-ordered differentially private algorithm, DP-Opt first picks a, a' in $\mathbb{A}$. It then builds a set $S \subseteq \mathbb{B}$ that maximizes $\mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a) \in S\right]$ and minimizes $\mathbb{P}\left[M_{\rho(\epsilon_0,\delta_0)}(a') \in S\right]$. Finally, it verifies the attack statistically. In the following, we use M_0 to denote $M_{\rho(\epsilon_0,\delta_0)}$.

Reducing the Search Space. Searching through every possible attack set $S \subseteq \mathbb{B}$ is in general intractable. DP-Opt therefore follows DP-Sniper [9] and uses the posterior probability $p(\cdot|\cdot)$ to build threshold attacks S^t. $p(a|b)$ is defined as $\mathbb{P}\left[A = a \mid B = b\right]$ where the random variables A and B are built using the following procedure: first set the random variable A to a or a' with equal probability 0.5, then compute $B = M_0(A)$. Intuitively, if $p(a|b)$ is high for some

[2] If ρ is not non-increasing, the rewrite holds with $\leq$. Hence, our estimate of $\mathcal{V}$ is greater than its actual values, and the results are still sound.

Algorithm 2 Delta-Siege

```
1:  func Delta-Siege($\rho, a, a', \epsilon_0, \delta_0, \alpha, N_{\text{train}}, N_{\text{attack}}, N_{\text{bound}}$)
2:      $M_0 \leftarrow M_{\rho(\epsilon_0,\delta_0)}$
3:      $\tilde{p} \leftarrow \text{LearnP}(N_{\text{train}}, M_0, a, a')$
4:      $t^{\text{best}}, \tilde{S}^{\text{best}} \leftarrow \text{PickAttack}(\tilde{p}, N_{\text{attack}}, \alpha, \rho, M_0, a, a')$
5:      $\epsilon^*, \delta^*, \rho^* \leftarrow \text{EvalAttack}(\tilde{S}^{\text{best}}, N_{\text{bound}}, \alpha, \rho, M_0, a, a')$
6:      return $\epsilon^*, \delta^*, \rho^*$
7:  func PickAttack($\tilde{p}, N, \alpha, \rho, M_0, a, a'$)
8:      for $t$ in $[0, 1]$ do
9:          $\tilde{S}^t \leftarrow \{b \in \mathbb{B} \mid \tilde{p}(a|b) \geq t\}$
10:         $\underline{p}(t) \leftarrow \text{LowerBound}\left(\mathbb{P}\left[M_0(a) \in \tilde{S}^t\right], N, \alpha/2\right)$
11:         $\overline{p}(t) \leftarrow \text{UpperBound}\left(\mathbb{P}\left[M_0(a') \in \tilde{S}^t\right], N, \alpha/2\right)$
12:         $\epsilon^*(t), \delta^*(t), \rho^*(t) \leftarrow \mathcal{V}(\rho, \underline{p}(t), \overline{p}(t))$
13:     $t^{\text{best}} \leftarrow \arg\min_{t\in[0,1]} \rho^*(t)$
14:     $\tilde{S}^{\text{best}} \leftarrow \{b \in \mathbb{B} \mid \tilde{p}(a|b) \geq t^{\text{best}}\}$
15:     return $t^{\text{best}}, \tilde{S}^{\text{best}}$
16: func EvalAttack($S, N, \alpha, \rho, M_0, a, a'$)
17:     $\underline{p} \leftarrow \text{LowerBound}\left(\mathbb{P}\left[M_0(a) \in S\right], N, \alpha/2\right)$
18:     $\overline{p} \leftarrow \text{UpperBound}\left(\mathbb{P}\left[M_0(a') \in S\right], N, \alpha/2\right)$
19:     return $\mathcal{V}(\rho, \underline{p}, \overline{p})$
20: func $\mathcal{V}(\rho, \underline{p}, \overline{p})$
21:     $\delta^* \leftarrow \arg\min_{\delta\in[0,\underline{p}]} \rho\left(\log\left(\frac{\underline{p}-\delta}{\overline{p}}\right), \delta\right)$
22:     $\epsilon^* \leftarrow \log\left(\frac{\underline{p}-\delta^*}{\overline{p}}\right)$
23:     $\rho^* \leftarrow \rho\left(\epsilon^*, \delta^*\right)$
24:     return $\epsilon^*, \delta^*, \rho^*$
```

b, then when the algorithm M_0 outputs b, it is highly likely that the input to M_0 was a and not a'. Finally, for any t in $[0, 1]$, DP-Opt defines $S^t = \{b \in \mathbb{B} \mid p(a|b) \geq t\}$.

5.1 Overview

We now describe Delta-Siege, that is, the DP-Opt algorithm after applying our method. The complete algorithm is shown in Algorithm 2 and we highlight in red the changes from the original DP-Opt approach. Note that only three such changes were enough to apply our method.

Picking a, a'. We found in practice that when using Delta-Siege, simple choices of neighboring inputs a, a' were enough to find violations in most audited algorithms. For instance, for the synthetic data generation algorithms, we use for a and a' databases made of a few entries, as detailed in Section 7.2. We also noted that sometimes swapping a and a' could yield better results. Therefore, we always run lines Lines 3 to 4 twice: once for the original a, a' and once with their values swapped. Finally, we pick the attack with the highest magnitude before proceeding to Line 5.

Building the Attack Set S. Lines 3 to 4 build the attack set S. As mentioned above, DP-Opt builds attack sets of the shape $S^t = \{b \in \mathbb{B} \mid p(a|b) \geq t\}$. As we do not have access to $p(a|b)$ directly, we train in Line 3 a classifier to learn an approximation $\tilde{p}$ using a fixed number of samples N_{train}. We describe our training procedure in more detail in Section 5.2. Delta-Siege then uses this approximation to pick the best threshold t^{best} using the function PickAttack (Line 4). To do so, for any t in $[0, 1]$ the attack set $\tilde{S}^t = \{b \in \mathbb{B} \mid \tilde{p}(a|b) \geq t\}$ is constructed. The optimal threshold t^{best} is then selected to minimize the quantity

$$\mathcal{V}(\rho, \mathbb{P}\left[M_0(a) \in \tilde{S}^t\right], \mathbb{P}\left[M_0(a') \in \tilde{S}^t\right]).$$

As this quantity cannot be exactly computed, DP-Opt computes for all t a lower bound $\underline{p}(t)$ on $\mathbb{P}\left[M_0(a) \in \tilde{S}^t\right]$ and an upper bound $\overline{p}(t)$ on $\mathbb{P}\left[M_0(a') \in \tilde{S}^t\right]$ (both with confidence $1 - \alpha/2$, in Lines 10 to 11), and optimizes $\mathcal{V}(\rho, \underline{p}, \overline{p})$ (Line 12) instead. Finally, the attack set $\tilde{S}^{\text{best}}$ is constructed from the optimal threshold t^{best} (Line 14).

Validating the Attack. After building the attack as above, DP-Opt validates it in Line 5. It simply computes its magnitude again, using fresh samples, and gets an upper bound ρ^* on

$$\mathcal{V}(\rho, \mathbb{P}\left[M_0(a) \in \tilde{S}^t\right], \mathbb{P}\left[M_0(a') \in \tilde{S}^t\right])$$

that holds with confidence $1 - \alpha$. If this upper bound is strictly less than $\rho(\epsilon_0, \delta_0)$, we can use Criterion 3 and conclude that the mechanism $M_{(\cdot)}$ violates its privacy claims, with confidence $1 - \alpha$.

Generating (ϵ, δ) Counter Examples. Finally, we describe how Delta-Siege builds (ϵ, δ) counter-examples for found vulnerabilities, allowing further investigation of the discovered issues. Using the notations from Fig. 2, such a counter example consists of a violated (ϵ_1, δ_1) and $(\hat{\epsilon}_0, \hat{\delta}_0)$ which is a clear violation. More formally, those values should be such that $\epsilon_1 \leq \hat{\epsilon}_0, \delta_1 \leq \hat{\delta}_0$ with one of the two inequalities strict and $\rho(\epsilon_0, \delta_0) = \rho(\epsilon_1, \delta_1)$. Finding $(\hat{\epsilon}_0, \hat{\delta}_0)$ is rather straightforward. As described in Section 4.4, when computing ρ^*, we transformed the problem into optimization over δ, which we solved by computing ρ on 900 log-uniform samples of δ. We can hence simply use for $\hat{\epsilon}_0, \hat{\delta}_0$ the values ϵ^*(Line 22) and δ^*(Line 21) corresponding to the optimal choice for ρ. Then, for (ϵ_1, δ_1), we define $\delta_1 = \hat{\delta}_0$, and use binary search to find $\epsilon_1 = \max\{\epsilon \mid \rho(\epsilon, \delta_1) = \rho(\epsilon_0, \delta_0)\}$. If $\epsilon_1 < \hat{\epsilon}_0$, we have found a clear violation. If this is not the case, Delta-Siege reports no violation, as the violation could not be confirmed.

5.2 LearnP

As described in Section 5.1, we do not know the true probability distribution $p(a|b)$, and rely on samples to train a machine learning model $\tilde{p}(a|b)$ that approximates this distribution. We collect N_{train} samples each from both $M_0(a)$ and $M_0(a')$, resulting in a dataset D. We use D to train a logistic regression model that takes as input the individual bits of the floating point representation of b and outputs the probability distribution $\tilde{p}(A = a|b)$. In practice, a simple logistic regression model outperformed more complex models and also has the side benefit of being easily interpretable. We train the model using Adam [33] with learning rate of 0.1 and ℓ_1 weight regularisation with strength 10^{-4}.

6 ROOT CAUSE ANALYSIS OF DP VIOLATIONS

As discussed in the previous section, Delta-Siege can provide explicit (ϵ, δ) counter-examples for all uncovered violations. In this section, we demonstrate that Delta-Siege outputs can further be used to *identify* the root cause of a violation. We exemplify this procedure for the inversion Laplace mechanism, the textbook implementation of Laplace sampling[3].

Root Cause Discovery. We audited the inversion Laplace mechanism M for $\epsilon = 1.0$, with inputs $a = 0.0$ and $a' = 1.0$. This resulted in a violated $\epsilon' = 9.74$, using the learned logistic regression $\tilde{p}(a|b)$ and a threshold t. This means that Delta-Siege found that (with confidence 0.95)

$$\mathbb{P}(M(a) \in S) > e^{\epsilon'}\mathbb{P}(M(a') \in S)$$

where S is defined by $\tilde{p}$ and t as $S = \{b \mid \tilde{p}(a|b) > t\}$.

To better understand this attack, we analyzed the weights W of the learned logistic regression classifier $\tilde{p}$. We took the three largest in absolute value weights W_i from W, corresponding to the three most important inputs. We recall that each input is an individual bit in the floating point representation of some mechanism output b. The three most important weights corresponded to the sign bit, the highest precision bit from the exponent, and the lowest precision mantissa bit. By looking at the signs of the weights W_i, we further determined that $\tilde{p}(a|b)$ reached high values when the sign and mantissa bits of the output are set to 1, while the exponent bit is set to 0. Visually, this means most of the set S consists of floating point numbers whose bit representation is

$$\underbrace{\texttt{1}}_{\text{sign}}\ \underbrace{\texttt{0xxxxxxxxxx}}_{\text{exponent}}\underbrace{\texttt{xxx...xxxxx1}}_{\text{mantissa}} \tag{7}$$

where x can be 0 or 1. To validate this hypothesis, we created p_b the perfect classifier for floating point numbers with the above representation and its set S_b (as defined by p_b and t). Then, we measured its overlap to the set S found by Delta-Siege. We defined the overlap as $\frac{|S \cap S_b|}{|S|}$ and approximated it by sampling N_{test} floating point numbers and counting how many of them were in $S \cap S_b$ and S. We obtained a score of $\frac{49707}{51135} \approx 0.972$, confirming our hypothesis.

Understanding The Root Cause. We now manually investigate the root cause identified above. We would like to show that for input $a' = 1.0$ and any noise z, we always have that $a' + z$ cannot be of the shape described in Eq. (7), while for input $a = 0.0$, there exists (many) noise values z such that $a + z$ is like Eq. (7). Let us first show this on an example by taking some noise z of the shape:

$$\underbrace{\texttt{1}}_{\text{sign}}\ \underbrace{\texttt{01111111111}}_{\text{exponent}}\underbrace{\texttt{11x...xxxxx1}}_{\text{mantissa}}$$

If we now compute $a + z = z$, we see that $a + z$ satisfies Eq. (7). In contrast, $a' + z$ gives:

$$\begin{aligned} &\underbrace{\texttt{0}}\underbrace{\texttt{01111111111}}\underbrace{\texttt{000...000000}} \\ +&\underbrace{\texttt{1}}\underbrace{\texttt{01111111111}}\underbrace{\texttt{11x...xxxxx1}} \\ =&\underbrace{\texttt{1}}\underbrace{\texttt{01111111110}}\underbrace{\texttt{1xx...xxxx10}} \end{aligned} \tag{8}$$

where the resulting mantissa in Eq. (8) was bit-shifted to ensure that its leading zero was removed, and the exponent was decremented accordingly. This readjustment introduces a 0 on the least bit of the mantissa. While we show this here only for one example, this is more generally the case: for input $a' = 1.0$, there is no noise value z such that $a' + z$ has its sign bit set to 1 and highest exponent bit to 0 and least bit of mantissa 1. We pose this as the following theorem:

THEOREM 4. *Any mechanism that is based on IEEE754 floating point addition, denoted $\oplus$, of some randomly generated noise Z with the result of some computation $f(a)$ on the private input a is not (ϵ, δ) differentially private if:*

- *f can attain the values 0.0 and 1.0 for some neighbouring input values a and $a' \in \mathcal{N}(a)$*
- *floating point values for Z with the property that they have negative sign bit, their mantissa has its highest precision bit set to 1, and their exponent has its lowest precision bit set to 0 can be generated by the noise-generation part of the mechanism with probability larger than δ.*

We point out that while the attack was derived automatically from the classifier $\tilde{p}(a|b)$, we have manually proven its validity (see Section B). Further, this attack is a variant of the floating point issue first described in Section 2 of [26] for the Laplace mechanism.

General Root Cause Discovery. While the example above was done on a single instantiation of the Laplace mechanism, we believe this procedure is widely applicable. To confirm this, we repeated it for the Opacus' Float Gauss, where we uncovered the same vulnerability, demonstrating that the procedure works for auditing (ϵ, δ) differentially private algorithms as well. Further, while in the example above, we chose to use the top three weights of our classifier to simplify the classifier decision, we can treat this as a hyperparameter of our simplification procedure. One way to set it is based on the desired level of overlap of the sets S and S_b. Finally, we point out that Delta-Siege can be used with any classifier type. In this paper, we focused on using logistic regression for $\tilde{p}(a|b)$ as it performs well on all experiments in our evaluation. However, other classifier types whose decisions are easier to interpret, such as decision trees, could be used instead, further simplifying the root cause analysis.

7 EVALUATION

In this section, we conduct an extensive evaluation of Delta-Siege on multiple state-of-the-art differentially private algorithms. We structure this evaluation along the following research questions:

Q1 *Can Delta-Siege accurately audit state-of-the-art differentially private algorithms?* We show that Delta-Siege can be used to audit recent and complex differentially private algorithms, such as synthetic data generation. Further, Delta-Siege identified violations in many of the audited algorithms, several of which were previously unknown.

Q2 *How much does grouping help to find violations?* We show through an ablation study that using ρ to group mechanisms allows finding more violations and finding them faster. Further, we show that all but two of our benchmarks are ρ-ordered.

Q3 *How does Delta-Siege compare to other automated differential privacy auditing methods?* We demonstrate that Delta-Siege almost always finds stronger violations than existing methods, both for ϵ and for (ϵ, δ) differential privacy.

[3] We here use the implementation from DP-Sniper[9].

Table 2: Benchmarks from [23, 28, 45, 51]

	Mechanisms		
Library	Laplace	Float Gauss	Discrete Gauss
OpenDP [23]	✓	✓	
IBM's Diffprivlib [28]	✓	✓✓ [a]	✓
PyDP [45]	✓	✓	
Opacus [51]		✓	

[a] Diffprivlib offers a second more precise method to compute the standard deviation required for privacy when using Gaussian noise, based on [3]. We refer to this more precise method as Analytic Float Gauss.

7.1 Benchmarks

To answer the research questions above, we evaluate Delta-Siege on differentially private algorithms gathered from three different sources: differential privacy libraries, NIST competitions, and related work evaluation sections.

Libraries. Differential privacy algorithms have been implemented in multiple libraries. We follow [16][4] and focus on four main ones: Google's differential privacy library [25], OpenDP [23], IBM's Diffprivlib [28] and Meta's Opacus [51]. As Google's differential privacy library does not offer a Python implementation, we use PyDP [45] the Python bindings for it implemented by OpenMinded. Note that those bindings are slightly outdated, so vulnerabilities found here could have since been fixed in the current version of the Google library.

Investigated algorithms. In each of those four libraries, we examine the Laplacian and Gaussian mechanisms. As those methods are the basis of multiple other algorithms, any vulnerability found in either is likely to propagate to many other algorithms in the same framework. Further, note that some libraries do not offer those mechanisms as part of their API, but instead use them as part of a more complex system. In those cases, we identify the relevant code and run it directly. We further ensure that doing so does not remove crucial privacy-preserving steps. For instance, in the Opacus library, we audit directly the function `_generate_noise`, which is used to generate Gaussian noise.

We show precisely which algorithms were available in each library in Table 2, and provide links to each algorithm implementation in the Appendix in Table 7. All four libraries offer a floating point Gaussian mechanism, and Diffprivlib offers two, the second with (in theory) tighter privacy guarantees. Diffprivlib further offers a discrete Gaussian mechanism. Finally, all libraries but Opacus offer a Laplacian mechanism.

Synthetic Data Generation Algorithms. We now add to our benchmarks more complex algorithms: the MST algorithm [40] (first place in the 2018 NIST Differential Privacy Synthetic Data Challenge[5]) and the AIM algorithm [41] (the latest synthetic data generation method from the same authors). Both of these mechanisms are graphical models for creating synthetic data based on marginal queries. Both first *privately* measure marginals on the original dataset, then build a synthetic dataset built on those noisy marginals. As the privacy proof of this process relies exclusively on the privacy of the noisy marginals, we directly use those measured marginals as input when auditing those mechanisms.

Table 3: Parameters used by Delta-Siege to audit the differentially private mechanisms in our benchmark.

Mechanism	ϵ_0	δ_0	Input
Gaussian Mechanism1	$\{0.1, 1.0\}$	$\{10^{-6}, 10^{-3}, 10^{-1}\}$	$\{0, 1\}$
Gaussian Mechanism2	$\{0.1, 1.0, 3.0, 10.0\}$	$\{10^{-6}, 10^{-3}, 10^{-1}\}$	$\{0, 1\}$
Laplacian Mechanism	$\{0.1, 1.0, 3.0, 10.0\}$	$\{0\}$	$\{0, 1\}$
MST	$\{3.0\}$	$\{10^{-6}, 10^{-1}\}$	Drop last row or not
AIM	$\{3.0\}$	$\{10^{-6}, 10^{-1}\}$	Drop last row or not

Algorithms from Related Works. To compare Delta-Siege's performance against existing auditing tools for ϵ differential privacy, we use benchmarks from the evaluation sections of these tools. In particular, we use the Laplacian mechanism implementation used by DP-Sniper, which we refer to as Laplacian Inversion. As this implementation is known to be vulnerable to floating point attacks [42], we also investigate the fixed implementation from OpenDP.

To compare Delta-Siege to the handcrafted attacks demonstrated in [27, 31], we further extend our benchmarks to include Gaussian mechanism implementations they have shown to be vulnerable. Those are the Box-Muller [10], Polar [38] and Ziguart [39] methods. Finally, we extend our benchmarks with the Opacus Gaussian mechanism implementation of Gaussian noise, which implements the fix recommended by [27].

Responsible Disclosure. After the submission of this paper, we notified all authors of the found vulnerabilities and offered to share the paper. We further informed them that we would wait three months before making the paper publicly available.

7.2 Experimental Set-up

Picking ϵ and δ. Delta-Siege audits a specific instantiation of a differentially private algorithm, for given (ϵ, δ) values. To analyze the quality of the auditing in different scenarios, we run Delta-Siege for several choices of (ϵ, δ) for each of the benchmarks described above, as shown in Table 3. The Laplace mechanism is ϵ-differentially private, so we only use $\delta = 0$ in this case, and four values of ϵ. For all the Gaussian mechanisms, some only accept limited values of ϵ. For those, we investigate two values for ϵ and three for δ, shown as Gaussian Mechanism 1 in Table 3. For others, we investigate all four values for ϵ and three for δ. Finally, as MST and AIM are computationally very intensive, we limit the number of investigated values to one for ϵ and two for δ.

Choosing inputs a, a'. As mentioned in Section 5, we pick naive input values a, a'. We will show in the remainder of this section that those are enough to find violations in almost all evaluated algorithms. For the Laplace and Gaussian mechanisms, we use $a = 0$ and $a' = 1$. Note that this results in a trivial sensitivity of $\Delta = 1$. Hence, any found violations are unlikely to be due to sensitivity miscalculations (a known flaw of multiple differentially

[4] Note that we do not evaluate on the Chorus library [32] as doing so would have required porting the library from Scala to Python.

[5] https://www.nist.gov/ctl/pscr/open-innovation-prize-challenges/past-prize-challenges/2018-differential-privacy-synthetic

Table 4: Results of our auditing with Delta-Siege of the libraries [45, 51, 28, 23]. We report for each mechanism if a vulnerability was found, and if the mechanism was known to be vulnerable (to the best of our knowledge). For all mechanisms where we found violations, we give an example of a clear violation, described by the expected (ϵ, δ) privacy (🟧 in Fig. 2) and a violated (ϵ, δ)(▲ in Fig. 2). For mechanisms where we did not find violations, we report the closest violated (ϵ, δ) we could find.

				Violations		
Library	Mechanism	Found?	Known?	Expected (ϵ, δ) 🟧	Violated (ϵ, δ) ▲	μ
OpenDP [23]	Laplace	✗		(1.000, 0.000)	(0.993, 0.000)	0.993
	Float Gauss	✗		(0.172, 0.056)	(0.069, 0.056)	0.680
Diffprivlib [28]	Laplace	✓	✓ [26]	(1.000, 0.000)	(5.784, 0.000)	5.784
	Float Gauss	✓	✓ [26]	(0.722, 0.001)	(5.877, 0.001)	8.013
	Analytic Float Gauss	✓	✓ [26]	(0.424, 0.004)	(7.153, 0.004)	8.967
	Discrete Gauss	✗		(0.138, 0.043)	(0.131, 0.043)	0.979
PyDP [45]	Laplace	✗		(1.000, 0.000)	(0.990, 0.000)	0.990
	Gauss	✓	✗	(0.501, 0.002)	(2.192, 0.002)	3.338
Opacus [51]	Gauss	✓	✗	(0.818, 0.0001)	(4.890, 0.0001)	5.976
MST [40]		✓[a]	✗	(1.875, 0.001)	(3.665, 0.001)	3.029
AIM [41]		✓[a]	✗	(2.036, 0.001)	(2.609, 0.001)	1.405

[a] As mentioned in Table 1, we found violation in intermediate results, invalidating the proof of privacy of those algorithms.

private algorithms [16]). For the MST and AIM algorithms, we use a simple database consisting of ten columns as follows:

$$a = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 1&1&1&1&1&1&1&1&1&1 \end{bmatrix} \quad \text{and} \quad a' = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \end{bmatrix}$$

We use here databases with really few entries, as the noise used for both MST and AIM does not depend on the number of entries and more entries significantly slow down the mechanisms (and therefore also slows down Delta-Siege as it runs those mechanisms).

Number of samples. Building an attack for a differentially private algorithm M requires sampling multiple times from $M(a)$ and $M(a')$ and more samples make building an accurate attack easier. As the Laplace and Gaussian mechanisms are fast to run, we use 10^6 unique samples each for training the classifier, finding the threshold parameter, selecting the optimal attack, and estimating the empirical privacy guarantees; so a total of $4 \cdot 10^6$ samples for the auditing framework. In contrast, for the more computationally intensive synthetic data generation mechanisms the number of samples is limited due to computational constraints. Creating one data point takes on average just over 1.9 seconds for MST and 1.7 seconds for AIM; in comparison, the Laplace mechanism uses only $2 \cdot 10^{-6}$ seconds to generate a sample. Therefore, we restrict the number of samples used for those to at most 25'000, unless mentioned otherwise.

Implementation and Hardware. Our implementation is publicly available[6]. All experiments were conducted using less than 1GB of RAM on a single core from AMD EPYC 7742 CPU, with clock speed 2250MHz and a total of 64 cores.

7.3 Q1: Finding new vulnerabilities

As mentioned in Section 1 and shown in more detail in Table 4, we detect several severe violations in a wide range of real-world implementations of privacy-preserving algorithms, each claiming to provide (ϵ, δ) differential privacy. We show the results of our auditing in Table 4.

Flawed Implementations. As teased in Table 1, we find violations in seven of the twelve audited mechanisms. Out of those, four were previously undocumented, to the best of our knowledge. We are the first to find vulnerabilities in the two synthetic data generation mechanisms MST and AIM. While we do not discover vulnerabilities in the final output of those mechanisms (the synthetic dataset), we find that the marginals they are built on are not differentially private as expected. This invalidates the privacy proof of those mechanisms, as they rely entirely on the privacy of those marginals.

Tighter Bounds. For implementations where we find no violations, Delta-Siege can give a sense of how tight the privacy bounds are. For instance, we find an attack strength μ greater than 0.99 for both OpenDP and PyDP Laplacian mechanisms. It is hence likely that the privacy bounds of both of those are tight. In contrast, for OpenDP Float Gauss, we can only achieve $\mu = 0.680$. There is hence a chance that its privacy bounds could be refined, allowing for more precision.

Confirming violations with library developers. We reported all found violations to the respective library owners. Diffprivlib acknowledged our bug reports. Further, as mentioned in Table 4, the vulnerabilities had already been identified in [26]. PyDP library owners did not reply, but after inspecting their code we believe the vulnerability is again due to floating point addition. For Opacus, library owners also acknowledged the bug, and we were also able to identify its root cause, see Section 6. Finally, the authors of MST and AIM acknowledged the bug. After inspecting their implementation, we believe the vulnerabilities' root cause is a similar floating point vulnerability to the one described in Section 6.

7.4 Q2: Using the ρ-ordered definition

We now evaluate in more detail the applicability and effect of our definition of ρ-ordered.

A Widely Applicable Definition. We show in Table 5 the ρ function that can be used for each of the investigated algorithms. Note that multiple such definitions are often possible, we here only showcase one per algorithm. For the Laplace and floating point Gaussian

[6] https://github.com/eth-sri/Delta-Siege

Table 5: Definition of ρ for all investigated algorithms. Δ denotes the sensitivity.

Method	$\rho(\epsilon, \delta)$	Decreasing and Upwards Invertible
Laplacian Mechanism	Δ/ϵ	✓
Gaussian Mechanism	$2\Delta^2 \log(1.25/\delta)/\epsilon^2$	✓
Analytic Gaussian Mechanism	see [3]	✗
Discrete Gaussian Mechanism	see [14]	✗
MST	see Eq. (9)	✓
AIM	see Eq. (9)	✓

mechanisms, we can define ρ to be the variance of each algorithm. For both, ρ is continuous and decreasing in both parameters, which is enough to satisfy our definition, using Lemma 1. The proof of privacy for both MST and AIM relies on a conversion between (ϵ, δ) differential privacy and γ-zero-concentrated differential privacy [12, 43]. We use $\rho(\epsilon, \delta) = 1/\gamma$ where $\gamma > 0$ is the smallest values such that γ-zero-concentrated differential privacy implies (ϵ, δ) differential privacy. The exact formula is given below and we show in Section A that ρ is upwards invertible and non-increasing.

$$\rho(\epsilon, \delta) = \max\left\{\frac{1}{\gamma} > 0 \mid \exists \alpha > 1 : \frac{\exp\left((\alpha-1)(\alpha\gamma-\epsilon)\right)}{\alpha-1}\left(1-\frac{1}{\alpha}\right)^{\alpha} \geq \delta\right\} \tag{9}$$

Therefore, for all those mechanisms, any ρ-violation can always be transformed into a clear (ϵ, δ) violation. In contrast, the discrete and analytic Gauss mechanisms of the DiffPrivlib library use standard deviation functions σ that combine both ϵ and δ into a single parameter, which we were not able to prove were upwards invertible. Note that as explained in Section 4.3, we can still use Delta-Siege to audit this mechanism, but there is no guarantee that found ρ-violation will translate to clear (ϵ, δ) violations.

Evaluating ρ. We use an ablation study to quantify the impact of the extended violation set permitted by ρ (visualized in Fig. 2c). To explore the smaller set of clear violations of (ϵ, δ) privacy (shown in Fig. 2a), we design two strategies based on related work:

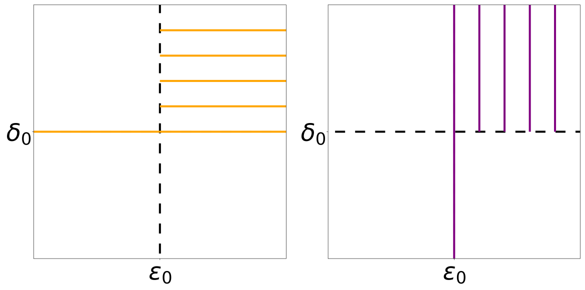
1. *ϵ-Maximization* (based on [15]): we set $\hat{\delta}_0$ to a predetermined set of values and optimize for $\hat{\epsilon}_0$. This corresponds to optimizing along the orange lines in Fig. 3a.
2. *δ-Maximization* (based on Eureka [37]): we set $\hat{\epsilon}_0$ to a predetermined set of values and optimize for $\hat{\delta}_0$. This corresponds to optimizing along the purple lines in Fig. 3b.

We implemented both strategies based on DP-Opt, as we did for Delta-Siege.

Finding More Violations. We first evaluated all three violation-finding strategies on the Diffprivlib implementation of the Analytic Gauss mechanism, for $\epsilon_0 = 10$ and $\delta_0 = 0.1$. We show the results in Fig. 4. Here we ran each strategy ten times and plotted for each run the (ϵ, δ) pair such that:

$$\frac{\mathbb{P}\left[M_0(a) \in S\right] - \delta}{\mathbb{P}\left[M_0(a') \in S\right]} \geq \exp(\epsilon)$$

and $\rho(\epsilon, \delta)$ is minimum. Both the ϵ maximization and the δ maximization strategies failed to find any violation, whereas half of the runs using ρ were able to find one. Further, none of the violations found with ρ were in the clear violation set (upper right quadrant in Fig. 2a): our extended definition was needed to characterize such violations.



(a) ϵ-Maximization

(b) δ-maximization

Figure 3: Two search strategies to find violations of (ϵ, δ) differential privacy

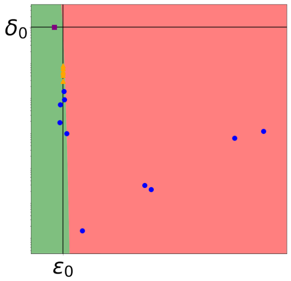


Figure 4: Worst violated (ϵ, δ) found for the Opacus implementation of the Gauss mechanism. (▲) are found by-ϵ maximization, (■) by δ-maximization, and (●) using ρ-ordering

Finding Violations Faster. We now evaluate the three strategies in a more computationally intensive setting; auditing MST algorithm, with $\epsilon_0 = 3.0$ and $\delta_0 = 10^{-6}$. To compare the violations found by each optimization strategy, we use the magnitude μ as defined in Section 4.2. We then show in Fig. 5 the maximum magnitude of the discovered violations as a function of the number of samples used.

From the plot it is clear that Delta-Siege always requires less samples to find the same violation magnitude as the other mechanisms, and hence less computational resources to discover violations. Running Delta-Siege with 12'500 samples is enough to conclude that the implementation is flawed. On the other hand, δ-Maximization must be run with 25'000 samples to reach the same conclusion and ϵ-Maximization never detects any violations at all. As the sample complexity completely dominates the runtime of the complete audit, employing ρ-ordering allows us to reduce the computational cost of auditing by a factor of 2.

7.5 Q3: Comparison to state of the art

We now compare Delta-Siege to other state-of-the-art differential privacy algorithms auditing tools.

Baselines. We consider here only two baselines: DP-Sniper [9] and DP-Opt [44]. We will discuss other related works in Section 8.

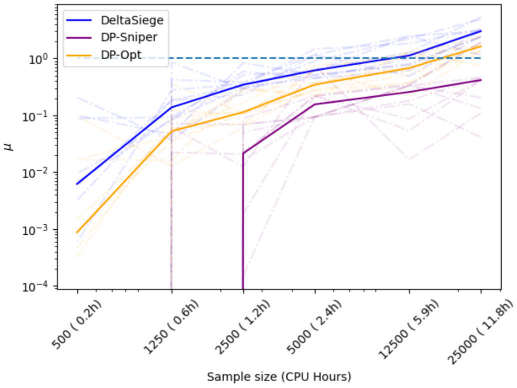


Figure 5: Comparison of ϵ-maximization, δ-minimization and our strategy. We show the magnitude of the violations found for the MST mechanism with $\delta_0 = 10^{-6}$ and $\epsilon_0 = 3.0$, for varying number of samples. We use ten batches of samples and report the median.

The two selected methods both search for a threshold attack S^t based on a classifier trained from samples as described in Section 5. As both works are focused on ϵ differential privacy only, we extend them to handle (ϵ, δ) privacy by fixing some values of δ and running each tool for these fixed values; this corresponds to the ϵ-maximization strategy as shown in Fig. 3a. We use the same samples and approximation of the posterior probability $\tilde{p}$ when evaluating the different baselines. Further, when working on the implementation of Delta-Siege, we found a bug in the DP-Sniper implementation[7], which lowered the accuracy of the learned classifier. This bug also affected DP-Opt, which builds on DP-Sniper. We run for both their fixed version, where we patch this bug.

We now briefly describe each work. DP-Sniper searches for violations of ϵ differential privacy. Importantly, Bichsel et al. assume that one cannot confidently estimate probabilities below a threshold c. Under this assumption, the authors show that it is optimal to choose t to be the c-quantile when sampling from $\tilde{p}(a \mid b)$, described in Section 5.

DP-Opt [44] is another auditing framework based on the ideas from DP-Sniper. The main difference to the previous baseline is that instead of setting the threshold t to a specific quantile, DP-Opt optimizes it in order to maximize $\hat{\epsilon}_0$.

More Violations Found. We show in Table 6 a summary of the comparison the algorithms from our benchmark. Complete numerical results can be found in Appendix Section D. We first focus on the reported number of violations. We audit multiple versions of each evaluated algorithm M, for the values of ϵ_0 and δ_0 shown in Table 3. We report the number of instantiations where we found violations of the claimed privacy guarantees. Delta-Siege consistently finds more violations than the other tools. For almost all algorithms in Table 6 that are found to be vulnerable in at least one instantiation, Delta-Siege finds violations in at least 75% of the instantiations, which is more consistent than both DP-Sniper and DP-Opt. We found that the missed violations are often for high values of ϵ_0 and δ_0 for all three tools. Further for Box Muller, Polar and Ziggart Gaussian mechanisms, both DP-Sniper and DP-Opt fail to find any violation when the mechanisms are instantiated with $\epsilon = 0.1$ and $\delta = 0.1$ (see Table 11 in Appendix). In contrast, Delta-Siege is able to consistently find ρ-violations and convert them to violations for a high ϵ and a low $\delta < 0.1$. This illustrates again the case shown in Fig. 4.

Bigger Violations. We also report in Table 6 the maximum magnitude of the violated $(\hat{\epsilon_0}, \hat{\delta_0})$, as defined in Section 7.4. For clarity, here we only report values for privacy parameters $\epsilon_0 = 1.0, \delta_0 = 10^{-6}$ for the Laplacian and Gaussian mechanisms as those are close to values used in practical applications of (ϵ, δ) differential privacy [18]. We find that Delta-Siege finds violations for every mechanism except the OpenDP Laplacian mechanism. When looking into the extended results, we see that Delta-Siege finds larger or equal violations than both other tools for *more than 80% of all evaluated instances.*

This shows that using ρ-ordering allows us to audit differentially private algorithms significantly more accurately and is often necessary to detect violations in state-of-the-art methods.

8 RELATED WORK

We now discuss related differential privacy tools. We first discuss the strategies other tools use to explore the space of (ϵ, δ) potential violations, then the methods they use to find attacks.

Exploring (ϵ, δ). Multiple auditing methods [44, 9] focus exclusively on ϵ differential privacy. Others [36] set δ to 0 (which gives an infinite variance for the Gaussian mechanism). We also mentioned in Section 7.4 the δ maximization strategy used by [37] and the ϵ maximization strategy used by [15, 30]. As demonstrated in Section 7.4, all the above strategies miss violations that our method can find using ρ-ordering. Finally, [34] directly audits Rényi differential privacy, but, as we will show below, inherently cannot derive confident bounds on the found violations.

Building and Evaluating Attacks. We now describe different auditing frameworks from prior work. Note that our method could easily be implemented for most of them, making their auditing

[7] We confirmed this with the DP-Sniper developers.

Table 6: Comparison to DP-Sniper and DP-Opt on all instantiations (with privacy parameters as defined in Table 3) of each benchmark. We report for each mechanism the number of violations across 10 runs of each of its instantiations. We also report the median of the magnitude μ of violations over ten runs for parameters $\epsilon = 1.0, \delta = 10^{-6}$.

	Delta-Siege		DP-Sniper		DP-Opt	
Method [Implementation]	# violations	$\mu_{\text{Delta-Siege}}$	# violations	$\frac{\mu_{\text{Delta-Siege}}}{\mu_{\text{DP-Sniper}}}$	# violations	$\frac{\mu_{\text{Delta-Siege}}}{\mu_{\text{DP-Opt}}}$
Laplacian Mechanism [Inversion]	30 / 40	9.009	27 / 40	1.115≥ **1**	30 / 40	1≥ **1**
Laplacian Mechanism [OpenDP]	0 / 40	0.993	5 / 40	1.054≥ **1**	0 / 40	1.001≥ **1**
Gaussian Mechanism [Box Muller]	59 / 60	10.615	37 / 60	1.508≥ **1**	48 / 60	1.398≥ **1**
Gaussian Mechanism [Polar]	58 / 60	9.140	36 / 60	1.935≥ **1**	46 / 60	1.322≥ **1**
Gaussian Mechanism [Zigguart]	60 / 60	9.664	34 / 60	1.551≥ **1**	47 / 60	1.336≥ **1**
Gaussian Mechanism [Opacus]	24 / 60	5.976	21 / 60	3.657≥ **1**	23 / 60	1.228≥ **1**

faster and more precise. [9, 44] train a classifier to distinguish between the outputs of $M(a)$ and $M(a')$ and create an attack based on this classifier. Further, both works provide statistical guarantees on the found attack but focus exclusively on ϵ differential privacy. [36] is also based on [9] and uses approximate bounds (the Katz-log confidence interval) to find more powerful attacks. However, as the same approximate bounds are used for evaluation, the found attacks cannot be statistically confirmed. The Eureka framework [37] again uses attacks based on classifiers to audit (ϵ, δ) differentially private algorithms. They derive guarantees on the performance of the attack by conditioning on the excess risk of the trained classifier compared to the Bayes optimal classifier. [35] is limited to finite output spaces and shows that its approach is optimal in a minimax setting. However, it does not present confidence intervals for the produced results.

Another line of work tries to estimate the output distributions $\mathbb{P}[M(a)]$ and $\mathbb{P}[M(a')]$ directly through kernel density methods. This approach has been implemented for both ϵ differential privacy [2] and Rényi differential privacy [34]. Neither work could derive confident estimates in this setting; therefore, both works only present asymptotic bounds on their estimates.

Membership Inference Attacks. A common use case for differential privacy is protecting against membership inference attacks [29]. In this scenario, an upper bound on the true positive and false positive rates of such attacks can be derived from the differential privacy claim. [15, 48, 30] explicitly implemented this, and could all be extended with our method as mentioned above. Other works [17, 46] also find such attacks but do not show how they could be converted to proven (ϵ, δ) violations.

Verifying Differential Privacy. Many works aim at *verifying* differential privacy claims. For instance, ChekDP [49] and DiPC [4] can not only detect violations of privacy claims but also be used for explicit verification. Some other works include [52, 7] based on a relational type system, [47, 8, 5, 6, 1] based on approximate couplings, or [50] based on shadow execution. However, the proofs in those works rely on the assumption that pre-implemented methods, such as the Laplacian or Gaussian mechanisms, are correct, which we have shown to be sometimes false.

To the best of our knowledge, only OpenDP has endeavored to verify *actual implementations* of differential privacy[8]. However, such an undertaking requires a lot of expert work, and may often be frustrated when the proofs identify implementation vulnerabilities. Fixing those vulnerabilities often implies heavy changes to the implementation and therefore restarting the proof process anew. For instance, [16] is a result of this proof effort: when trying to prove a mechanism correct, they found a vulnerability that stopped the proof work. We believe using Delta-Siege would be a great complement to this proof effort, as it may weed out many erroneous implementations before the proof process is even started, therefore allowing this costly effort to focus only on the most promising implementations.

9 CONCLUSION

We presented a new method for characterizing (ϵ, δ) differential privacy violations. Our method can be easily implemented for existing differential privacy auditing tools, as we demonstrated by implementing it for DP-Opt, resulting in Delta-Siege.

Delta-Siege found multiple severe violations of differential privacy in state-of-the-art algorithms, many of which were previously unknown. It also consistently outperforms other auditing tools for differential privacy.

ACKNOWLEDGMENTS

This work was supported by an ETH Zurich Research Grant.

REFERENCES

[1] Aws Albarghouthi and Justin Hsu. 2017. Synthesizing coupling proofs of differential privacy. *Proceedings of the ACM on Programming Languages*, 2, POPL, 1–30.

[2] Önder Askin, Tim Kutta, and Holger Dette. 2021. Statistical quantification of differential privacy: A local approach. *CoRR*, abs/2108.09528. https://arxiv.org/abs/2108.09528 arXiv: 2108.09528.

[3] Borja Balle and Yu-Xiang Wang. 2018. Improving the gaussian mechanism for differential privacy: analytical calibration and optimal denoising. In *International Conference on Machine Learning*. PMLR, 394–403.

[4] Gilles Barthe, Rohit Chadha, Vishal Jagannath, A. Prasad Sistla, and Mahesh Viswanathan. 2020. Deciding differential privacy for programs with finite inputs and outputs. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science* (LICS ’20). Association for Computing Machinery, Saarbrücken, Germany, 141–154. ISBN: 9781450371049. DOI: 10.1145/3373718.3394796.

[5] Gilles Barthe, George Danezis, Benjamin Grégoire, César Kunz, and Santiago Zanella-Beguelin. 2013. Verified computational differential privacy with applications to smart metering. In *2013 IEEE 26th Computer Security Foundations Symposium*. IEEE, 287–301.

[6] Gilles Barthe, Noémie Fong, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016. Advanced probabilistic couplings for differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 55–67.

[8] See https://opendp.org/blog/introducing-opendp-library-v06.

[7] Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, Aaron Roth, and Pierre-Yves Strub. 2015. Higher-order approximate relational refinement types for mechanism design and differential privacy. *ACM SIGPLAN Notices*, 50, 1, 55–68.

[8] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Beguelin. 2012. Probabilistic relational reasoning for differential privacy. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 97–110.

[9] Benjamin Bichsel, Samuel Steffen, Ilija Bogunovic, and Martin Vechev. 2021. Dpsniper: black-box discovery of differential privacy violations using classifiers. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, (May 2021), 391–409. doi: 10.1109/SP40001.2021.00081.

[10] G. E. P. Box and Mervin E. Muller. 1958. A Note on the Generation of Random Normal Deviates. *The Annals of Mathematical Statistics*, 29, 2, 610–611. doi: 10.1214/aoms/1177706645.

[11] 2016. *Chapter 2.4.5 floating-point operations. Computer Systems: A programmer's perspective.* (3rd ed.). Pearson, 160.

[12] Mark Bun and Thomas Steinke. 2016. Concentrated differential privacy: simplifications, extensions, and lower bounds. *CoRR*, abs/1605.02065. http://arxiv.org/abs/1605.02065 arXiv: 1605.02065.

[13] 2022. California consumer privacy act. State of California. (Jan. 17, 2022). Retrieved Jan. 17, 2022 from https://leginfo.legislature.ca.gov/faces/codes_displayText.xhtml?division=3.&part=4.&lawCode=CIV&title=1.81.5.

[14] Clément L. Canonne, Gautam Kamath, and Thomas Steinke. 2020. The discrete gaussian for differential privacy. *CoRR*, abs/2004.00010. https://arxiv.org/abs/2004.00010 arXiv: 2004.00010.

[15] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramèr. 2022. Membership inference attacks from first principles. In *2022 IEEE Symposium on Security and Privacy (SP)*, 1897–1914. doi: 10.1109/SP46214.2022.9833649.

[16] Sílvia Casacuberta, Michael Shoemate, Salil Vadhan, and Connor Wagaman. 2022. Widespread underestimation of sensitivity in differentially private libraries and how to fix it. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 471–484.

[17] Christopher A. Choquette-Choo, Florian Tramer, Nicholas Carlini, and Nicolas Papernot. 2021. Label-only membership inference attacks. In *Proceedings of the 38th International Conference on Machine Learning* (Proceedings of Machine Learning Research). Marina Meila and Tong Zhang, (Eds.) Vol. 139. PMLR, (July 2021), 1964–1974. https://proceedings.mlr.press/v139/choquette-choo21a.html.

[18] Damien Desfontaines. 2021. A list of real-world uses of differential privacy. https://desfontain.es/privacy/real-world-differential-privacy.html. Ted is writing things (personal blog). (Oct. 2021).

[19] 2021. Differential privacy for census data explained. National Conference of State Legislatures. (Nov. 10, 2021). Retrieved Jan. 12, 2022 from https://www.ncsl.org/health/differential-privacy-for-census-data-explained.

[20] 2022. Digital markets act. European Commission. (Sept. 14, 2022). Retrieved Jan. 12, 2022 from https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32022R1925/.

[21] Cynthia Dwork and Aaron Roth. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9, 3–4, 211–407. doi: 10.1561/0400000042.

[22] Cynthia Dwork and Aaron Roth. 2014. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9, 3–4, (Aug. 2014), 211–407. doi: 10.1561/0400000042.

[23] Marco Gaboardi, Michael Hay, and Salil Vadhan. 2020. A programming framework for opendp. *Manuscript, May.*

[24] 2018. General data protection regulation. European Commission. (May 25, 2018). Retrieved Jan. 12, 2022 from https://gdpr-info.eu/.

[25] Google. 2021. Differential privacy. https://github.com/google/differential-privacy. (2021).

[26] Samuel Haney, Damien Desfontaines, Luke Hartman, Ruchit Shrestha, and Michael Hay. 2022. Precision-based attacks and interval refining: how to break, then fix, differential privacy on finite computers. *arXiv preprint arXiv:2207.13793.* https://arxiv.org/abs/2207.13793.

[27] Naoise Holohan and Stefano Braghin. 2021. Secure random sampling in differential privacy. *CoRR*, abs/2107.10138. https://arxiv.org/abs/2107.10138 arXiv: 2107.10138.

[28] Naoise Holohan, Stefano Braghin, Pól Mac Aonghusa, and Killian Levacher. 2019. Diffprivlib: the IBM differential privacy library. *ArXiv e-prints*, 1907.02444 [cs.CR], (July 2019).

[29] Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Philip S. Yu, and Xuyun Zhang. 2022. Membership inference attacks on machine learning: a survey. *ACM Comput. Surv.*, 54, 11s, Article 235, (Sept. 2022), 37 pages. doi: 10.1145/3523273.

[30] Matthew Jagielski, Jonathan R. Ullman, and Alina Oprea. 2020. Auditing differentially private machine learning: how private is private sgd? *CoRR*, abs/2006.07709. https://arxiv.org/abs/2006.07709 arXiv: 2006.07709.

[31] Jiankai Jin, Eleanor McMurtry, Benjamin I. P. Rubinstein, and Olga Ohrimenko. 2021. Are we there yet? timing and floating-point attacks on differential privacy systems. *CoRR*, abs/2112.05307. https://arxiv.org/abs/2112.05307 arXiv: 2112.05307.

[32] Noah Johnson, Joseph P Near, Joseph M Hellerstein, and Dawn Song. 2020. Chorus: a programming framework for building scalable differential privacy mechanisms. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 535–551.

[33] Diederik P Kingma and Jimmy Ba. 2014. Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

[34] Tim Kutta, Önder Askin, and Martin Dunsche. 2022. Lower bounds for rényi differential privacy in a black-box setting. (2022). doi: 10.48550/ARXIV.2212.04739.

[35] Xiyang Liu and Sewoong Oh. 2019. Minimax optimal estimation of approximate differential privacy on neighboring databases. In *Advances in Neural Information Processing Systems*. H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, (Eds.) Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/7a674153c63cff1ad7f0e261c369ab2c-Paper.pdf.

[36] Fred Lu, Joseph Munoz, Maya Fuchs, Tyler LeBlond, Elliott Zaresky-Williams, Edward Raff, Francis Ferraro, and Brian Testa. 2022. A general framework for auditing differentially private machine learning. (2022). doi: 10.48550/ARXIV.2210.08643.

[37] Yun Lu, Yu Wei, Malik Magdon-Ismail, and Vassilis Zikas. 2022. Eureka: a general framework for black-box differential privacy estimators. Cryptology ePrint Archive, Paper 2022/1250. https://eprint.iacr.org/2022/1250. (2022). https://eprint.iacr.org/2022/1250.

[38] G. Marsaglia and T. A. Bray. 1964. A convenient method for generating normal variables. *SIAM Review*, 6, 3, 260–264. Retrieved Jan. 15, 2023 from http://www.jstor.org/stable/2027592.

[39] George Marsaglia and Wai Wan Tsang. 2000. The ziggurat method for generating random variables. *Journal of Statistical Software*, 5, 8, 1–7. doi: 10.18637/jss.v005.i08.

[40] Ryan McKenna, Gerome Miklau, and Daniel Sheldon. 2021. Winning the NIST contest: A scalable and general approach to differentially private synthetic data. *CoRR*, abs/2108.04978. https://arxiv.org/abs/2108.04978 arXiv: 2108.04978.

[41] Ryan McKenna, Brett Mullins, Daniel Sheldon, and Gerome Miklau. 2022. AIM: an adaptive and iterative mechanism for differentially private synthetic data. *CoRR*, abs/2201.12677. https://arxiv.org/abs/2201.12677 arXiv: 2201.12677.

[42] Ilya Mironov. 2012. On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (CCS '12). Association for Computing Machinery, Raleigh, North Carolina, USA, 650–661. isbn: 9781450316514. doi: 10.1145/2382196.2382264.

[43] Ilya Mironov. 2017. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE, (Aug. 2017). doi: 10.1109/csf.2017.11.

[44] Ben Niu, Zejun Zhou, Yahong Chen, Jin Cao, and Fenghua Li. 2022. Dp-opt: identify high differential privacy violation by optimization. In *Wireless Algorithms, Systems, and Applications*. Lei Wang, Michael Segal, Jenhui Chen, and Tie Qiu, (Eds.) Springer Nature Switzerland, Cham, 406–416. isbn: 978-3-031-19214-2.

[45] OpenMinded. 2021. Pydp. https://github.com/OpenMined/PyDP. (2021).

[46] Md.Atiqur Rahman, Tanzila Rahman, Robert Laganière, and Noman Mohammed. 2018. Membership inference attack against differentially private deep learning model. *Trans. Data Priv.*, 11, 61–79.

[47] Jason Reed and Benjamin C Pierce. 2010. Distance makes the types grow stronger: a calculus for differential privacy. In *Proceedings of the 15th ACM SIGPLAN international conference on Functional programming*, 157–168.

[48] Theresa Stadler, Bristena Oprisanu, and Carmela Troncoso. 2022. Synthetic data–anonymisation groundhog day. In *31st USENIX Security Symposium (USENIX Security 22)*, 1451–1468.

[49] Yuxin Wang, Zeyu Ding, Daniel Kifer, and Danfeng Zhang. 2020. Checkdp: an automated and integrated approach for proving differential privacy or finding precise counterexamples. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (CCS '20). Association for Computing Machinery, Virtual Event, USA, 919–938. isbn: 9781450370899. doi: 10.1145/3372297.3417282.

[50] Yuxin Wang, Zeyu Ding, Guanhong Wang, Daniel Kifer, and Danfeng Zhang. 2019. Proving differential privacy with shadow execution. *CoRR*, abs/1903.12254. http://arxiv.org/abs/1903.12254 arXiv: 1903.12254.

[51] Ashkan Yousefpour et al. 2021. Opacus: User-friendly differential privacy library in PyTorch. *arXiv preprint arXiv:2109.12298.*

[52] Danfeng Zhang and Daniel Kifer. 2017. Lightdp: towards automating differential privacy proofs. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, 888–901.

Appendices

A ρ-ORDERING FOR MST & AIM

We see in Section A of [40] and Section 4 of [41] that the proof of (ϵ, δ) differential privacy in MST and AIM relies on the conversion between (ϵ, δ) differential privacy and zero-Concentrated Differential Privacy (γ-zero-Concentrated Differential Privacy is equivalent to $(\alpha, \gamma\alpha)$ Rényi differential privacy holding for all $\alpha > 1$). As noted in Proposition 4 in Section 2.2 of [41], $(\alpha, \gamma\alpha)$ Rényi differential privacy implies $\left(\epsilon, \frac{\exp((\alpha-1)(\alpha\gamma-\epsilon))}{\alpha-1}\left(1-\frac{1}{\alpha}\right)^{\alpha}\right)$ differential privacy. The authors state that they use the minimal γ so that γ-zero-Concentrated Differential Privacy implies (ϵ, δ) differential privacy according to the aforementioned proposition. We, therefore, define the privacy parameter for MST as:

$$\rho(\epsilon, \delta) = \max\left\{\frac{1}{\gamma} > 0 \mid \exists\alpha > 1 : \frac{\exp\left((\alpha-1)(\alpha\gamma-\epsilon)\right)}{\alpha-1}\left(1-\frac{1}{\alpha}\right)^{\alpha} \geq \delta\right\} \tag{10}$$

From communication with the authors, we have learned that MST relies on the same conversion from γ-zero-concentrated differential privacy to (ϵ, δ) differential privacy; this conversion is slightly different than the one described in the original paper [40]. Furthermore, inspecting the code from both algorithms confirms that they use the exactly same script to perfom this conversion [9]. ρ as in Eq. (10) therefore exactly parameterizes the AIM and MST algorithms.

Non-increasing. We first notice that we can rewrite Eq. (10) as $\rho(\epsilon, \delta) = \max \Gamma_{\epsilon,\delta}$ with

$$\Gamma_{\epsilon,\delta} = \left\{\frac{1}{\gamma} > 0 \mid \exists\alpha > 1 : F(\gamma, \alpha, \epsilon, \delta) \geq 0\right\}$$

using $F(\gamma, \alpha, \epsilon, \delta) = \frac{\exp((\alpha-1)(\alpha\gamma-\epsilon))}{\alpha-1}\left(1-\frac{1}{\alpha}\right)^{\alpha} - \delta$.

Have $\epsilon' \leq \epsilon$ and $\delta' \leq \delta$. We have that for any $1/\gamma \in \Gamma_{\epsilon,\delta}$ we can pick $\alpha > 1$ such that $F(\gamma, \alpha, \epsilon, \delta) \geq 0$. As F is non indreasing, we then have that $F(\gamma, \alpha, \epsilon', \delta') \geq F(\gamma, \alpha, \epsilon, \delta) \geq 0$, meaning that $1/\gamma \in \Gamma_{\epsilon',\delta'}$. Consequently, $\Gamma_{\epsilon,\delta} \subseteq \Gamma_{\epsilon',\delta'}$ which implies that $\rho(\epsilon, \delta) \leq \rho(\epsilon', \delta')$ or equivalently that ρ is non-increasing in both parameters.

Upwards-invertibility. . As F is continuous, we have that ρ must also be continuous. Therefore, by Lemma 1 ρ is upwards invertible.

B PROOF OF OUR FLOATING POINT ATTACK

In this section, we provide proof of Theorem 4 first introduced in Section 6. We restate the theorem for convenience below:

Theorem 4. *Any mechanism that is based on IEEE754 floating point addition, denoted $\oplus$, of some randomly generated noise Z with the result of some computation $f(a)$ on the private input a is not (ϵ, δ) differentially private if:*

- *f can attain the values 0.0 and 1.0 for some neighbouring input values a and $a' \in \mathcal{N}(a)$*
- *floating point values for Z with the property that they have negative sign bit, their mantissa has its highest precision bit set to 1, and their exponent has its lowest precision bit set to 0 can be generated by the noise-generation part of the mechanism with probability at least δ.*

Proof. We call any IEEE754 floating point number *attackable* if satisfies the property that it is negative, its mantissa has its highest precision bit set to 1, and its exponent has its lowest precision bit set to 0 like in Eq. (7).

To prove the theorem, we first show that the number $Z \oplus 0.0$ is an attackable number when Z is attackable. We then show that the number $Z' \oplus 1.0$ is never an attackable number for any floating point noise Z'. We finish the proof by considering the output set S consisting of all attackable floating point numbers that can be generated for the mechanism output $Z \oplus 0.0$ of a. For this choice of S, $\mathbb{P}\left[f(a') \in S\right]$ is 0, as all numbers in S cannot be generated by the mechanism output $Z \oplus 1.0$ of a' for any noise, as these numbers cannot be attackable. Therefore for this choice of S, the definition of differential privacy is violated if the probability $\mathbb{P}\left[f(a) \in S\right]$ is bigger than δ.

In order to complete our proof, we first need to prove that $Z \oplus 0.0$ is attackable when Z is. This directly follows from the observation $Z \oplus 0.0 = Z$. Further, we need to prove that $Z' \oplus 1.0$ cannot be an attackable number for any floating point noise Z'. This constitutes the main part of the proof effort and is described next.

By way of contradiction, assume that O' is an attackable floating point number, such that $O' = Z' \oplus 1.0$ for some Z'. As O' is negative, and all floating point numbers in the range $(-\infty, -0.0]$ are monotone in their bit representation, we know that O' is bigger than the floating point number -2.0 as -2.0's exponent has its lowest precision bit set to 1. Therefore, $O' \in (-2.0, -0.0)$. By the monotonicity of floating point addition [11], we can then conclude $Z' \in (-3.0, -1.0)$. We split this interval into two cases which we consider separately below.

Let's first assume $Z' \in (-2.0, -1.0)$. In that interval, Z' can be represented with the binary fraction $-1.\overline{m} \times 2^0$ where $\overline{m}$ is the binary representation of the mantissa of Z'. Similarly, the floating point number 1.0 is represented with the binary fraction 1.0×2^0. Therefore, O', representing their sum, is a floating point number of the type $-0.\overline{m} \times 2^0$. In order to be represented as IEEE754 floating point number, $-0.\overline{m} \times 2^0$ needs to be rewritten as $-1.\overline{m_s} \times 2^e$ for some negative e and shifted mantissa $\overline{m_s}$. The shifted mantissa $\overline{m_s}$ is generated from $\overline{m}$ by removing all of its leading zeros as well as the first 1 and adding as many zeros at the end of $\overline{m_s}$ as bits were removed from the front of $\overline{m}$. As at least one bit was removed from the front of $\overline{m}$ (the one corresponding to the first 1 bit), $\overline{m_s}$ ends in at least one 0. As $\overline{m_s}$ represents the mantissa of O', O' cannot have its mantissa's highest precision bit set to 1. This contradicts the assumption that O' is an attackable number.

Let's now assume $Z' \in (-3.0, -2.0]$. In this interval, Z' can be represented with the binary fraction $-1.\overline{0m} \times 2^1$ where $\overline{m}$ is the binary representation of the mantissa of Z' after the leading 0. Similarly, the floating point number 1.0 is represented with the binary fraction 0.1×2^1. The resulting sum representing O' then becomes $-0.\overline{1m} \times 2^1$, which is then converted as before to $-1.\overline{m0} \times 2^0$.

[9] Both algorithms use an IBM script to perform the conversion in practice. It can be found at: https://github.com/IBM/discrete-gaussian-differential-privacy/blob/master/cdp2adp.py

This, as before, introduces a trailing 0 in the mantissa of O', which leads to a contradiction with the assumption that O' is attackable.

Therefore, we conclude that no floating point numbers Z can make O' attackable, which finishes the proof of our theorem. □

C CODE REFERENCES

In Table 7, we present links to the different differentially private algorithms that we have audited in Section 7. Clicking on the URLs in the table leads to the documentation of the exact implementations used for each mechanism.

D EXTENDED RESULTS

In Tables 8 to 23, we present the results of attacking the different methods from Table 7 for different values of ϵ_0 and δ_0. We compare Delta-Siege against DP-Opt and DP-Sniper in terms of the number of violations found and μ. Further, we show the values of (ϵ_1, δ_1) and $(\hat{\epsilon_0}, \hat{\delta_0})$ from Fig. 2 for all methods.

Table 7: Code for the mechanisms tested in Section 7

Library	Mechanism	URL
OpenDP [23]	Laplacian	https://docs.opendp.org/
OpenDP [23]	Gaussian	https://docs.opendp.org/
Diffprivlib [28]	Laplacian	https://github.com/IBM/differential-privacy-library/
Diffprivlib [28]	Float Gaussian	https://github.com/IBM/differential-privacy-library/
Diffprivlib [28]	Analytic Float Gaussian	https://github.com/IBM/differential-privacy-library/
Diffprivlib [28]	Discrete Gaussian	https://github.com/IBM/differential-privacy-library/
PyDP [45]	Laplacian	https://github.com/OpenMined/PyDP/
PyDP [45]	Float Gaussian	https://github.com/OpenMined/PyDP/
Opacus [51]	Float Gaussian	https://github.com/pytorch/opacus/
private-pgm	MST [40]	https://github.com/ryan112358/private-pgm/
private-pgm	AIM [41]	https://github.com/ryan112358/private-pgm/

Table 8: Results for $\epsilon_0 = 0.1$ and $\delta_0 = 0$

	Delta-Siege			DP-Sniper			DP-Opt		
Method [Implementation]	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$
Laplace Mechanism [IBM]	9 / 10	24.459	$(0.100, 0)$ $(2.446, 0)$	10 / 10	12.685	$(0.100, 0)$ $(1.269, 0)$	10 / 10	27.028	$(0.100, 0)$ $(2.703, 0)$
Laplace Mechanism [Inversion]	10 / 10	38.269	$(0.100, 0)$ $(3.827, 0)$	10 / 10	20.998	$(0.100, 0)$ $(2.100, 0)$	10 / 10	43.728	$(0.100, 0)$ $(4.373, 0)$
Laplace Mechanism [OpenDP]	0 / 10	0.954	$(0.100, 0)$ $(0.095, 0)$	2 / 10	0.599	$(0.100, 0)$ $(0.060, 0)$	0 / 10	0.922	$(0.100, 0)$ $(0.092, 0)$
Laplace Mechanism [PyDP]	0 / 10	0.933	$(0.100, 0)$ $(0.093, 0)$	1 / 10	0.675	$(0.100, 0)$ $(0.068, 0)$	0 / 10	0.929	$(0.100, 0)$ $(0.093, 0)$

Table 9: Results for $\epsilon_0 = 0.1$ and $\delta_0 = 1 \times 10^{-6}$

	Delta-Siege			DP-Sniper			DP-Opt		
Method [Implementation]	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$
Analytic Gaussian Mechanism [IBM]	10 / 10	9.117	$(0.044, 0.00074)$ $(0.626, 0.00074)$	9 / 10	4.422	$(0.100, 1 \times 10^{-6})$ $(0.490, 1 \times 10^{-6})$	10 / 10	9.489	$(0.100, 1 \times 10^{-6})$ $(1.119, 1 \times 10^{-6})$
Discrete Gaussian Mechanism [IBM]	0 / 10	0.726	$(0.018, 0.0043)$ $(0.009, 0.0043)$	1 / 10	0.093	$(0.100, 1 \times 10^{-6})$ $(0.002, 1 \times 10^{-6})$	0 / 10	0.244	$(0.100, 1 \times 10^{-6})$ $(0.022, 1 \times 10^{-6})$
Gauss Mechanism [OpenDP]	0 / 10	0.090	$(0.056, 0.00035)$ $(0.002, 0.00035)$	1 / 10	0.000	$(0.100, 1 \times 10^{-6})$ $(-0.024, 1 \times 10^{-6})$	0 / 10	0.000	$(0.100, 1 \times 10^{-6})$ $(-0.001, 1 \times 10^{-6})$
Gaussian Mechanism [Box Muller]	10 / 10	8.645	$(0.082, 0.00011)$ $(0.778, 0.00011)$	7 / 10	3.644	$(0.100, 1 \times 10^{-6})$ $(0.364, 1 \times 10^{-6})$	8 / 10	7.911	$(0.100, 1 \times 10^{-6})$ $(0.791, 1 \times 10^{-6})$
Gaussian Mechanism [IBM]	7 / 10	1.337	$(0.078, 0.00027)$ $(0.105, 0.00027)$	4 / 10	0.779	$(0.100, 1 \times 10^{-6})$ $(0.078, 1 \times 10^{-6})$	6 / 10	1.213	$(0.100, 1 \times 10^{-6})$ $(0.121, 1 \times 10^{-6})$
Gaussian Mechanism [Opacus]	0 / 10	0.120	$(0.087, 3.1 \times 10^{-5})$ $(0.010, 3.1 \times 10^{-5})$	1 / 10	0.000	$(0.100, 1 \times 10^{-6})$ $(-0.019, 1 \times 10^{-6})$	1 / 10	0.097	$(0.100, 1 \times 10^{-6})$ $(0.010, 1 \times 10^{-6})$
Gaussian Mechanism [Polar]	8 / 10	1.487	$(0.077, 0.00029)$ $(0.118, 0.00029)$	6 / 10	2.794	$(0.100, 1 \times 10^{-6})$ $(0.279, 1 \times 10^{-6})$	6 / 10	1.343	$(0.100, 1 \times 10^{-6})$ $(0.134, 1 \times 10^{-6})$
Gaussian Mechanism [PyDP]	3 / 10	0.834	$(0.040, 0.00093)$ $(0.030, 0.00093)$	1 / 10	0.108	$(0.100, 1 \times 10^{-6})$ $(0.009, 1 \times 10^{-6})$	1 / 10	0.399	$(0.100, 1 \times 10^{-6})$ $(0.037, 1 \times 10^{-6})$
Gaussian Mechanism [Zigguart]	10 / 10	4.880	$(0.081, 0.00012)$ $(0.377, 0.00012)$	7 / 10	4.254	$(0.100, 1 \times 10^{-6})$ $(0.425, 1 \times 10^{-6})$	7 / 10	4.553	$(0.100, 1 \times 10^{-6})$ $(0.455, 1 \times 10^{-6})$

Table 10: Results for $\epsilon_0 = 0.1$ and $\delta_0 = 0.001$

	Delta-Siege			DP-Sniper			DP-Opt		
Method [Implementation]	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$
Analytic Gaussian Mechanism [IBM]	10 / 10	11.844	$(0.170, 2.8 \times 10^{-5})$ $(2.799, 2.8 \times 10^{-5})$	9 / 10	7.915	$(0.100, 0.001)$ $(1.207, 0.001)$	10 / 10	9.560	$(0.100, 0.001)$ $(1.523, 0.001)$
Discrete Gaussian Mechanism [IBM]	0 / 10	0.900	$(0.034, 0.01)$ $(0.029, 0.01)$	3 / 10	0.235	$(0.100, 0.001)$ $(0.015, 0.001)$	0 / 10	0.858	$(0.100, 0.001)$ $(0.082, 0.001)$
Gauss Mechanism [OpenDP]	0 / 10	0.371	$(0.066, 0.0046)$ $(0.013, 0.0046)$	0 / 10	0.000	$(0.100, 0.001)$ $(-0.088, 0.001)$	0 / 10	0.306	$(0.100, 0.001)$ $(0.022, 0.001)$
Gaussian Mechanism [Box Muller]	10 / 10	11.640	$(0.124, 2.8 \times 10^{-5})$ $(1.597, 2.8 \times 10^{-5})$	7 / 10	6.632	$(0.100, 0.001)$ $(0.663, 0.001)$	10 / 10	8.498	$(0.100, 0.001)$ $(0.850, 0.001)$
Gaussian Mechanism [IBM]	10 / 10	6.571	$(0.114, 0.00013)$ $(0.754, 0.00013)$	7 / 10	4.196	$(0.100, 0.001)$ $(0.420, 0.001)$	10 / 10	4.875	$(0.100, 0.001)$ $(0.487, 0.001)$
Gaussian Mechanism [Opacus]	1 / 10	0.223	$(0.127, 1.3 \times 10^{-5})$ $(0.028, 1.3 \times 10^{-5})$	2 / 10	0.000	$(0.100, 0.001)$ $(-0.061, 0.001)$	0 / 10	0.155	$(0.100, 0.001)$ $(0.016, 0.001)$
Gaussian Mechanism [Polar]	10 / 10	2.192	$(0.113, 0.00015)$ $(0.248, 0.00015)$	7 / 10	2.284	$(0.100, 0.001)$ $(0.228, 0.001)$	10 / 10	1.817	$(0.100, 0.001)$ $(0.182, 0.001)$
Gaussian Mechanism [PyDP]	10 / 10	1.724	$(0.059, 0.0049)$ $(0.129, 0.0049)$	4 / 10	0.000	$(0.100, 0.001)$ $(-0.012, 0.001)$	10 / 10	1.574	$(0.100, 0.001)$ $(0.175, 0.001)$
Gaussian Mechanism [Zigguart]	10 / 10	2.596	$(0.113, 0.00014)$ $(0.293, 0.00014)$	7 / 10	1.889	$(0.100, 0.001)$ $(0.189, 0.001)$	10 / 10	2.108	$(0.100, 0.001)$ $(0.211, 0.001)$

Table 11: Results for $\epsilon_0 = 0.1$ and $\delta_0 = 0.1$

	Delta-Siege			DP-Sniper			DP-Opt		
Method [Implementation]	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$
Analytic Gaussian Mechanism [IBM]	10 / 10	6.491	$(0.611, 0.0078)$ $(7.669, 0.0078)$	4 / 10	0.388	$(0.100, 0.1)$ $(-0.006, 0.1)$	10 / 10	2.777	$(0.100, 0.1)$ $(1.110, 0.1)$
Discrete Gaussian Mechanism [IBM]	0 / 10	0.988	$(0.486, 0.017)$ $(0.477, 0.017)$	0 / 10	0.000	$(0.100, 0.1)$ $(-1.939, 0.1)$	0 / 10	0.990	$(0.100, 0.1)$ $(0.096, 0.1)$
Gauss Mechanism [OpenDP]	0 / 10	0.680	$(0.198, 0.061)$ $(0.080, 0.061)$	0 / 10	0.000	$(0.100, 0.1)$ $(-2.369, 0.1)$	0 / 10	0.000	$(0.100, 0.1)$ $(-\infty, 0.1)$
Gaussian Mechanism [Box Muller]	9 / 10	2.465	$(0.182, 0.0003)$ $(0.436, 0.0003)$	0 / 10	0.000	$(0.100, 0.1)$ $(-2.215, 0.1)$	0 / 10	0.000	$(0.100, 0.1)$ $(-\infty, 0.1)$
Gaussian Mechanism [IBM]	10 / 10	6.189	$(0.193, 0.00011)$ $(1.449, 0.00011)$	0 / 10	0.000	$(0.100, 0.1)$ $(-2.250, 0.1)$	0 / 10	0.000	$(0.100, 0.1)$ $(-\infty, 0.1)$
Gaussian Mechanism [Opacus]	3 / 10	0.784	$(0.196, 8 \times 10^{-5})$ $(0.147, 8 \times 10^{-5})$	0 / 10	0.000	$(0.100, 0.1)$ $(-2.664, 0.1)$	0 / 10	0.000	$(0.100, 0.1)$ $(-\infty, 0.1)$
Gaussian Mechanism [Polar]	10 / 10	8.796	$(0.199, 5.8 \times 10^{-5})$ $(1.961, 5.8 \times 10^{-5})$	0 / 10	0.000	$(0.100, 0.1)$ $(-2.286, 0.1)$	0 / 10	0.000	$(0.100, 0.1)$ $(-\infty, 0.1)$
Gaussian Mechanism [PyDP]	10 / 10	5.751	$(0.764, 0.0027)$ $(6.953, 0.0027)$	0 / 10	0.000	$(0.100, 0.1)$ $(-0.748, 0.1)$	10 / 10	2.092	$(0.100, 0.1)$ $(0.654, 0.1)$
Gaussian Mechanism [Zigguart]	10 / 10	10.883	$(0.192, 0.00012)$ $(2.394, 0.00012)$	0 / 10	0.000	$(0.100, 0.1)$ $(-2.101, 0.1)$	0 / 10	0.000	$(0.100, 0.1)$ $(-\infty, 0.1)$

Table 12: Results for $\epsilon_0 = 1.0$ and $\delta_0 = 0$

	Delta-Siege			DP-Sniper			DP-Opt		
Method [Implementation]	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$
Laplace Mechanism [IBM]	10 / 10	5.784	$(1.000, 0)$ $(5.784, 0)$	10 / 10	6.011	$(1.000, 0)$ $(6.011, 0)$	10 / 10	6.171	$(1.000, 0)$ $(6.171, 0)$
Laplace Mechanism [Inversion]	10 / 10	9.009	$(1.000, 0)$ $(9.009, 0)$	10 / 10	8.082	$(1.000, 0)$ $(8.082, 0)$	10 / 10	9.009	$(1.000, 0)$ $(9.009, 0)$
Laplace Mechanism [OpenDP]	0 / 10	0.993	$(1.000, 0)$ $(0.993, 0)$	1 / 10	0.942	$(1.000, 0)$ $(0.942, 0)$	0 / 10	0.992	$(1.000, 0)$ $(0.992, 0)$
Laplace Mechanism [PyDP]	0 / 10	0.990	$(1.000, 0)$ $(0.990, 0)$	1 / 10	0.955	$(1.000, 0)$ $(0.955, 0)$	0 / 10	0.987	$(1.000, 0)$ $(0.987, 0)$

Table 13: Results for $\epsilon_0 = 1.0$ and $\delta_0 = 1 \times 10^{-6}$

	Delta-Siege			DP-Sniper			DP-Opt		
Method [Implementation]	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$
Analytic Gaussian Mechanism [IBM]	10 / 10	8.967	$(0.424, 0.0042)$ $(7.153, 0.0042)$	10 / 10	6.498	$(1.000, 1 \times 10^{-6})$ $(8.040, 1 \times 10^{-6})$	10 / 10	6.386	$(1.000, 1 \times 10^{-6})$ $(7.877, 1 \times 10^{-6})$
Discrete Gaussian Mechanism [IBM]	0 / 10	0.979	$(0.138, 0.043)$ $(0.131, 0.043)$	4 / 10	0.646	$(1.000, 1 \times 10^{-6})$ $(0.624, 1 \times 10^{-6})$	0 / 10	0.688	$(1.000, 1 \times 10^{-6})$ $(0.667, 1 \times 10^{-6})$
Gauss Mechanism [OpenDP]	0 / 10	0.680	$(0.172, 0.056)$ $(0.069, 0.056)$	0 / 10	0.237	$(1.000, 1 \times 10^{-6})$ $(0.216, 1 \times 10^{-6})$	0 / 10	0.226	$(1.000, 1 \times 10^{-6})$ $(0.205, 1 \times 10^{-6})$
Gaussian Mechanism [Box Muller]	10 / 10	10.615	$(0.674, 0.0021)$ $(7.152, 0.0021)$	10 / 10	7.041	$(1.000, 1 \times 10^{-6})$ $(7.041, 1 \times 10^{-6})$	10 / 10	7.594	$(1.000, 1 \times 10^{-6})$ $(7.594, 1 \times 10^{-6})$
Gaussian Mechanism [IBM]	10 / 10	8.013	$(0.722, 0.00083)$ $(5.877, 0.00083)$	10 / 10	4.257	$(1.000, 1 \times 10^{-6})$ $(4.257, 1 \times 10^{-6})$	10 / 10	6.382	$(1.000, 1 \times 10^{-6})$ $(6.382, 1 \times 10^{-6})$
Gaussian Mechanism [Opacus]	10 / 10	5.976	$(0.818, 0.0001)$ $(4.890, 0.0001)$	8 / 10	1.634	$(1.000, 1 \times 10^{-6})$ $(1.634, 1 \times 10^{-6})$	10 / 10	4.867	$(1.000, 1 \times 10^{-6})$ $(4.867, 1 \times 10^{-6})$
Gaussian Mechanism [Polar]	10 / 10	9.140	$(0.708, 0.0011)$ $(6.537, 0.0011)$	10 / 10	4.724	$(1.000, 1 \times 10^{-6})$ $(4.724, 1 \times 10^{-6})$	10 / 10	6.914	$(1.000, 1 \times 10^{-6})$ $(6.914, 1 \times 10^{-6})$
Gaussian Mechanism [PyDP]	10 / 10	4.037	$(0.678, 0.00022)$ $(3.839, 0.00022)$	10 / 10	2.292	$(1.000, 1 \times 10^{-6})$ $(2.470, 1 \times 10^{-6})$	10 / 10	3.776	$(1.000, 1 \times 10^{-6})$ $(4.303, 1 \times 10^{-6})$
Gaussian Mechanism [Zigguart]	10 / 10	9.664	$(0.691, 0.0016)$ $(6.687, 0.0016)$	10 / 10	6.232	$(1.000, 1 \times 10^{-6})$ $(6.232, 1 \times 10^{-6})$	10 / 10	7.235	$(1.000, 1 \times 10^{-6})$ $(7.235, 1 \times 10^{-6})$

Table 14: Results for $\epsilon_0 = 1.0$ and $\delta_0 = 0.001$

	Delta-Siege			DP-Sniper			DP-Opt		
Method [Implementation]	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$
Analytic Gaussian Mechanism [IBM]	10 / 10	6.189	$(0.673, 0.0091)$ $(7.877, 0.0091)$	10 / 10	5.400	$(1.000, 0.001)$ $(8.072, 0.001)$	10 / 10	5.607	$(1.000, 0.001)$ $(8.483, 0.001)$
Discrete Gaussian Mechanism [IBM]	0 / 10	0.992	$(0.459, 0.03)$ $(0.453, 0.03)$	1 / 10	0.939	$(1.000, 0.001)$ $(0.945, 0.001)$	0 / 10	0.955	$(1.000, 0.001)$ $(0.962, 0.001)$
Gauss Mechanism [OpenDP]	0 / 10	0.693	$(0.288, 0.085)$ $(0.125, 0.085)$	0 / 10	0.489	$(1.000, 0.001)$ $(0.435, 0.001)$	0 / 10	0.538	$(1.000, 0.001)$ $(0.485, 0.001)$
Gaussian Mechanism [Box Muller]	10 / 10	8.200	$(0.888, 0.0045)$ $(7.331, 0.0045)$	10 / 10	8.062	$(1.000, 0.001)$ $(8.062, 0.001)$	10 / 10	7.608	$(1.000, 0.001)$ $(7.608, 0.001)$
Gaussian Mechanism [IBM]	10 / 10	8.530	$(0.884, 0.0047)$ $(7.481, 0.0047)$	10 / 10	8.041	$(1.000, 0.001)$ $(8.041, 0.001)$	10 / 10	8.023	$(1.000, 0.001)$ $(8.023, 0.001)$
Gaussian Mechanism [Opacus]	10 / 10	6.042	$(0.999, 0.001)$ $(6.031, 0.001)$	10 / 10	3.298	$(1.000, 0.001)$ $(3.298, 0.001)$	10 / 10	5.891	$(1.000, 0.001)$ $(5.891, 0.001)$
Gaussian Mechanism [Polar]	10 / 10	8.206	$(0.893, 0.0042)$ $(7.320, 0.0042)$	10 / 10	8.022	$(1.000, 0.001)$ $(8.022, 0.001)$	10 / 10	7.818	$(1.000, 0.001)$ $(7.818, 0.001)$
Gaussian Mechanism [PyDP]	10 / 10	5.183	$(0.868, 0.0026)$ $(7.146, 0.0026)$	10 / 10	4.445	$(1.000, 0.001)$ $(6.254, 0.001)$	10 / 10	5.063	$(1.000, 0.001)$ $(7.415, 0.001)$
Gaussian Mechanism [Zigguart]	10 / 10	8.427	$(0.891, 0.0044)$ $(7.488, 0.0044)$	10 / 10	8.065	$(1.000, 0.001)$ $(8.065, 0.001)$	10 / 10	7.926	$(1.000, 0.001)$ $(7.926, 0.001)$

Table 15: Results for $\epsilon_0 = 1.0$ and $\delta_0 = 0.1$

	Delta-Siege			DP-Sniper			DP-Opt		
Method [Implementation]	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$
Analytic Gaussian Mechanism [IBM]	10 / 10	4.011	$(0.749, 0.15)$ $(9.887, 0.15)$	10 / 10	3.395	$(1.000, 0.1)$ $(8.068, 0.1)$	10 / 10	3.952	$(1.000, 0.1)$ $(10.440, 0.1)$
Discrete Gaussian Mechanism [IBM]	0 / 10	0.996	$(1.209, 0.049)$ $(1.203, 0.049)$	0 / 10	0.000	$(1.000, 0.1)$ $(-\infty, 0.1)$	0 / 10	0.996	$(1.000, 0.1)$ $(0.992, 0.1)$
Gauss Mechanism [OpenDP]	0 / 10	0.799	$(1.313, 0.049)$ $(0.944, 0.049)$	0 / 10	0.000	$(1.000, 0.1)$ $(-2.076, 0.1)$	0 / 10	0.756	$(1.000, 0.1)$ $(0.619, 0.1)$
Gaussian Mechanism [Box Muller]	10 / 10	1.622	$(0.996, 0.1)$ $(1.616, 0.1)$	0 / 10	0.581	$(1.000, 0.1)$ $(0.581, 0.1)$	10 / 10	1.637	$(1.000, 0.1)$ $(1.637, 0.1)$
Gaussian Mechanism [IBM]	10 / 10	1.474	$(0.958, 0.12)$ $(1.412, 0.12)$	2 / 10	0.733	$(1.000, 0.1)$ $(0.733, 0.1)$	10 / 10	1.676	$(1.000, 0.1)$ $(1.676, 0.1)$
Gaussian Mechanism [Opacus]	0 / 10	0.907	$(0.996, 0.1)$ $(0.903, 0.1)$	0 / 10	0.000	$(1.000, 0.1)$ $(-0.233, 0.1)$	2 / 10	0.935	$(1.000, 0.1)$ $(0.935, 0.1)$
Gaussian Mechanism [Polar]	10 / 10	1.693	$(0.996, 0.1)$ $(1.686, 0.1)$	3 / 10	0.902	$(1.000, 0.1)$ $(0.902, 0.1)$	10 / 10	1.710	$(1.000, 0.1)$ $(1.710, 0.1)$
Gaussian Mechanism [PyDP]	10 / 10	2.956	$(1.510, 0.038)$ $(7.742, 0.038)$	10 / 10	1.918	$(1.001, 0.1)$ $(3.093, 0.1)$	10 / 10	2.026	$(1.001, 0.1)$ $(3.389, 0.1)$
Gaussian Mechanism [Zigguart]	10 / 10	1.610	$(0.996, 0.1)$ $(1.603, 0.1)$	0 / 10	0.652	$(1.000, 0.1)$ $(0.652, 0.1)$	10 / 10	1.624	$(1.000, 0.1)$ $(1.624, 0.1)$

Table 16: Results for $\epsilon_0 = 3.0$ and $\delta_0 = 0$

	Delta-Siege			DP-Sniper			DP-Opt		
Method [Implementation]	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$
Laplace Mechanism [IBM]	10 / 10	1.161	$(3.000, 0)$ $(3.484, 0)$	3 / 10	0.944	$(3.000, 0)$ $(2.831, 0)$	10 / 10	1.134	$(3.000, 0)$ $(3.403, 0)$
Laplace Mechanism [Inversion]	10 / 10	1.403	$(3.000, 0)$ $(4.209, 0)$	7 / 10	1.099	$(3.000, 0)$ $(3.297, 0)$	10 / 10	1.403	$(3.000, 0)$ $(4.209, 0)$
Laplace Mechanism [OpenDP]	0 / 10	0.990	$(3.000, 0)$ $(2.971, 0)$	2 / 10	0.918	$(3.000, 0)$ $(2.755, 0)$	0 / 10	0.990	$(3.000, 0)$ $(2.970, 0)$
Laplace Mechanism [PyDP]	0 / 10	0.993	$(3.000, 0)$ $(2.978, 0)$	2 / 10	0.921	$(3.000, 0)$ $(2.763, 0)$	0 / 10	0.995	$(3.000, 0)$ $(2.984, 0)$

Table 17: Results for $\epsilon_0 = 3.0$ and $\delta_0 = 1 \times 10^{-6}$

	Delta-Siege			DP-Sniper			DP-Opt		
Method [Implementation]	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$
AIM Internal	6 / 10	1.405	$(2.036, 0.00069)$ $(2.609, 0.00069)$	0 / 10	0.311	$(3.008, 1 \times 10^{-6})$ $(1.564, 1 \times 10^{-6})$	5 / 10	0.882	$(3.008, 1 \times 10^{-6})$ $(2.792, 1 \times 10^{-6})$
Analytic Gaussian Mechanism [IBM]	10 / 10	4.849	$(0.707, 0.062)$ $(9.205, 0.062)$	10 / 10	2.384	$(3.000, 1 \times 10^{-6})$ $(8.076, 1 \times 10^{-6})$	10 / 10	2.885	$(3.000, 1 \times 10^{-6})$ $(10.133, 1 \times 10^{-6})$
Discrete Gaussian Mechanism [IBM]	2 / 10	0.994	$(1.020, 0.021)$ $(1.011, 0.021)$	1 / 10	0.660	$(3.000, 1 \times 10^{-6})$ $(1.864, 1 \times 10^{-6})$	0 / 10	0.746	$(3.000, 1 \times 10^{-6})$ $(2.142, 1 \times 10^{-6})$
Gauss Mechanism [OpenDP]	0 / 10	0.777	$(0.925, 0.06)$ $(0.628, 0.06)$	0 / 10	0.460	$(3.000, 1 \times 10^{-6})$ $(1.292, 1 \times 10^{-6})$	0 / 10	0.477	$(3.000, 1 \times 10^{-6})$ $(1.341, 1 \times 10^{-6})$
Gaussian Mechanism [PyDP]	10 / 10	4.517	$(0.929, 0.034)$ $(8.948, 0.034)$	10 / 10	2.391	$(3.000, 1 \times 10^{-6})$ $(8.108, 1 \times 10^{-6})$	10 / 10	2.784	$(3.000, 1 \times 10^{-6})$ $(9.716, 1 \times 10^{-6})$
MST Internal	10 / 10	3.029	$(1.875, 0.0015)$ $(3.665, 0.0015)$	2 / 10	0.417	$(3.008, 1 \times 10^{-6})$ $(1.862, 1 \times 10^{-6})$	9 / 10	1.635	$(3.008, 1 \times 10^{-6})$ $(3.939, 1 \times 10^{-6})$

Table 18: Results for $\epsilon_0 = 3.0$ and $\delta_0 = 0.001$

	Delta-Siege			DP-Sniper			DP-Opt		
Method [Implementation]	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$
Analytic Gaussian Mechanism [IBM]	10 / 10	3.789	(0.793, 0.15) (9.488, 0.15)	10 / 10	2.182	(3.000, 0.001) (8.105, 0.001)	10 / 10	2.692	(3.000, 0.001) (10.733, 0.001)
Discrete Gaussian Mechanism [IBM]	0 / 10	0.996	(1.335, 0.045) (1.326, 0.045)	1 / 10	0.975	(3.000, 0.001) (2.877, 0.001)	0 / 10	0.975	(3.000, 0.001) (2.877, 0.001)
Gauss Mechanism [OpenDP]	0 / 10	0.819	(2.218, 0.013) (1.708, 0.013)	0 / 10	0.792	(3.000, 0.001) (2.270, 0.001)	0 / 10	0.784	(3.000, 0.001) (2.242, 0.001)
Gaussian Mechanism [PyDP]	10 / 10	2.677	(1.728, 0.03) (7.567, 0.03)	10 / 10	2.187	(2.998, 0.001) (8.123, 0.001)	10 / 10	2.299	(2.998, 0.001) (8.680, 0.001)

Table 19: Results for $\epsilon_0 = 3.0$ and $\delta_0 = 0.1$

	Delta-Siege			DP-Sniper			DP-Opt		
Method [Implementation]	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$
AIM Internal	0 / 10	0.323	(5.664, 0.001) (2.951, 0.001)	0 / 10	0.000	(3.047, 0.1) (−1.897, 0.1)	0 / 10	0.056	(3.047, 0.1) (0.244, 0.1)
Analytic Gaussian Mechanism [IBM]	10 / 10	1.141	(0.000, 0.49) (1.073, 0.49)	0 / 10	0.000	$(\infty, 0)$ (0.000, 0)	0 / 10	0.000	$(\infty, 0)$ (0.000, 0)
Discrete Gaussian Mechanism [IBM]	0 / 10	0.000	$(\infty, 0)$ (0.000, 0)	0 / 10	0.000	$(\infty, 0)$ (0.000, 0)	0 / 10	0.000	$(\infty, 0)$ (0.000, 0)
Gauss Mechanism [OpenDP]	0 / 10	0.857	(4.927, 0.0052) (4.047, 0.0052)	0 / 10	0.460	(3.000, 0.1) (0.866, 0.1)	0 / 10	0.819	(3.000, 0.1) (2.198, 0.1)
Gaussian Mechanism [PyDP]	0 / 10	0.000	$(\infty, 0)$ (0.000, 0)	0 / 10	0.000	$(\infty, 0)$ (0.000, 0)	0 / 10	0.000	$(\infty, 0)$ (0.000, 0)
MST Internal	1 / 10	0.651	(5.234, 0.0027) (3.959, 0.0027)	0 / 10	0.000	(3.047, 0.1) (−1.284, 0.1)	0 / 10	0.173	(3.047, 0.1) (0.730, 0.1)

Table 20: Results for $\epsilon_0 = 10.0$ and $\delta_0 = 0$

	Delta-Siege			DP-Sniper			DP-Opt		
Method [Implementation]	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$
Laplace Mechanism [IBM]	0 / 10	0.968	(10.000, 0) (9.681, 0)	0 / 10	0.432	(10.000, 0) (4.324, 0)	0 / 10	0.963	(10.000, 0) (9.631, 0)
Laplace Mechanism [Inversion]	0 / 10	0.962	(10.000, 0) (9.621, 0)	0 / 10	0.425	(10.000, 0) (4.250, 0)	0 / 10	0.962	(10.000, 0) (9.621, 0)
Laplace Mechanism [OpenDP]	0 / 10	0.958	(10.000, 0) (9.579, 0)	0 / 10	0.425	(10.000, 0) (4.248, 0)	0 / 10	0.954	(10.000, 0) (9.544, 0)
Laplace Mechanism [PyDP]	0 / 10	0.960	(10.000, 0) (9.600, 0)	1 / 10	0.000	(10.000, 0) $(-\infty, 0)$	0 / 10	0.961	(10.000, 0) (9.605, 0)

Table 21: Results for $\epsilon_0 = 10.0$ and $\delta_0 = 1 \times 10^{-6}$

	Delta-Siege			DP-Sniper			DP-Opt		
Method [Implementation]	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$
Analytic Gaussian Mechanism [IBM]	10 / 10	1.126	(0.000, 0.51) (1.148, 0.51)	0 / 8	0.000	$(\infty, 0)$ (0.000, 0)	0 / 8	0.000	$(\infty, 0)$ (0.000, 0)
Discrete Gaussian Mechanism [IBM]	0 / 10	0.000	$(\infty, 0)$ (0.000, 0)	0 / 10	0.000	$(\infty, 0)$ (0.000, 0)	0 / 10	0.000	$(\infty, 0)$ (0.000, 0)
Gauss Mechanism [OpenDP]	0 / 10	0.863	(5.727, 0.011) (4.668, 0.011)	0 / 10	0.609	$(10.000, 1 \times 10^{-6})$ $(5.633, 1 \times 10^{-6})$	0 / 10	0.616	$(10.000, 1 \times 10^{-6})$ $(5.707, 1 \times 10^{-6})$
Gaussian Mechanism [PyDP]	0 / 10	0.000	$(\infty, 0)$ (0.000, 0)	0 / 10	0.000	$(\infty, 0)$ (0.000, 0)	0 / 10	0.000	$(\infty, 0)$ (0.000, 0)

Table 22: Results for $\epsilon_0 = 10.0$ and $\delta_0 = 0.001$

	Delta-Siege			DP-Sniper			DP-Opt		
Method [Implementation]	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$
Analytic Gaussian Mechanism [IBM]	10 / 10	1.072	$(0.000, 0.62)$ $(1.691, 0.62)$	0 / 10	0.000	$(\infty, 0)$ $(0.000, 0)$	0 / 10	0.000	$(\infty, 0)$ $(0.000, 0)$
Discrete Gaussian Mechanism [IBM]	0 / 10	0.000	$(\infty, 0)$ $(0.000, 0)$	0 / 10	0.000	$(\infty, 0)$ $(0.000, 0)$	0 / 10	0.000	$(\infty, 0)$ $(0.000, 0)$
Gauss Mechanism [OpenDP]	0 / 10	0.854	$(6.995, 0.042)$ $(5.539, 0.042)$	0 / 10	0.694	$(10.000, 0.001)$ $(6.251, 0.001)$	0 / 10	0.722	$(10.000, 0.001)$ $(6.568, 0.001)$
Gaussian Mechanism [PyDP]	0 / 10	0.000	$(\infty, 0)$ $(0.000, 0)$	0 / 10	0.000	$(\infty, 0)$ $(0.000, 0)$	0 / 10	0.000	$(\infty, 0)$ $(0.000, 0)$

Table 23: Results for $\epsilon_0 = 10.0$ and $\delta_0 = 0.1$

	Delta-Siege			DP-Sniper			DP-Opt		
Method [Implementation]	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$	# violations	μ	(ϵ_1, δ_1) $(\hat{\epsilon}_0, \hat{\delta}_0)$
Analytic Gaussian Mechanism [IBM]	10 / 10	1.024	$(0.000, 0.68)$ $(3.906, 0.68)$	0 / 10	0.000	$(\infty, 0)$ $(0.000, 0)$	0 / 10	0.000	$(\infty, 0)$ $(0.000, 0)$
Discrete Gaussian Mechanism [IBM]	0 / 10	0.000	$(\infty, 0)$ $(0.000, 0)$	0 / 10	0.000	$(\infty, 0)$ $(0.000, 0)$	0 / 10	0.000	$(\infty, 0)$ $(0.000, 0)$
Gauss Mechanism [OpenDP]	0 / 10	0.791	$(7.430, 0.35)$ $(4.814, 0.35)$	1 / 10	0.600	$(10.000, 0.1)$ $(4.478, 0.1)$	0 / 10	0.711	$(10.000, 0.1)$ $(5.837, 0.1)$
Gaussian Mechanism [PyDP]	0 / 10	0.000	$(\infty, 0)$ $(0.000, 0)$	0 / 10	0.000	$(\infty, 0)$ $(0.000, 0)$	0 / 10	0.000	$(\infty, 0)$ $(0.000, 0)$