# 9

# Differential Privacy and Computational Complexity

Our discussion of differential privacy has so far ignored issues of computational complexity, permitting both the curator and the adversary to be computationally unbounded. In reality, both curator and adversary may be computationally bounded.

Confining ourselves to a computationally bounded curator restricts what the curator can do, making it harder to achieve differential privacy. And indeed, we will show an example of a class of counting queries that, under standard complexity theoretic assumptions, does not permit *efficient* generation of a synthetic database, even though inefficient algorithms, such as SmallDB and Private Multiplicative Weights, are known. Very roughly, the database rows are digital signatures, signed with keys to which the curator does not have access. The intuition will be that any row in a synthetic database must either be copied from the original — violating privacy — or must be a signature on a *new* message, i.e., a forgery — violating the unforgeability property of a digital signature scheme. Unfortuately, this state of affairs is not limited to (potentially contrived) examples based on digital signatures: it is even difficult to create a synthetic database that maintains relatively

accurate two-way marginals.[1] On the positive side, given a set $\mathcal{Q}$ of queries and an $n$-row database with rows drawn from a universe $\mathcal{X}$, a synthetic database can be generated in time polynomial in $n$, $|\mathcal{X}|$, and $|\mathcal{Q}|$.

If we abandon the goal of a synthetic database and content ourselves with a data structure from which we can obtain a relatively accurate approximation to the answer to each query, the situation is much more interesting. It turns out that the problem is intimately related to the *tracing traitors* problem, in which the goal is to discourage piracy while distributing digital content to paying customers.

If the adversary is restricted to polynomial time, then it becomes easier to achieve differential privacy. In fact, the immensely powerful concept of *secure function evaluation* yields a natural way avoid the trusted curator (while giving better accuracy than randomized response), as well as a natural way to allow multiple trusted curators, who for legal reasons cannot share their data sets, to respond to queries on what is effectively a merged data set. Briefly put, secure function evaluation is a cryptographic primitive that permits a collection of $n$ parties $p_1, p_2, \ldots, p_n$, of which fewer than some fixed fraction are faulty (the fraction varies according to the type of faults; for "honest-but-curious" faults the fraction is 1), to cooperatively compute any function $f(x_1, \ldots, x_n)$, where $x_i$ is the input, or *value*, of party $p_i$, in such a way that no coalition of faulty parties can either disrupt the computation or learn more about the values of the non-faulty parties than can be deduced from the function output and the values of the members of the coalition. These two properties are traditionally called *correctness* and *privacy*. This privacy notion, let us call it *SFE privacy*, is very different from differential privacy. Let $V$ be the set of values held by the faulty parties, and let $p_i$ be a non-faulty party.[2] SFE privacy permits the faulty parties to learn $x_i$ if $x_i$ can be deduced from $V \cup \{f(x_1, \ldots, x_n)\}$; differential privacy would therefore not permit exact release of $f(x_1, \ldots, x_n)$. However, secure function evaluation

[1] Recall that the two-way marginals are the counts, for every pair of attribute values, of the number of rows in the database having this pair of values.

[2] In the honest but curious case we can let $V = \{x_j\}$ for any party $P_j$.

protocols for computing a function $f$ can easily be modified to obtain differentially private protocols for $f$, simply by defining a new function, $g$, to be the result of adding Laplace noise $\mathrm{Lap}(\Delta f/\varepsilon)$ to the value of $f$. In principle, secure function evaluation permits evaluation of $g$. Since $g$ is differentially private and the SFE privacy property, applied to $g$, says that nothing can be learned about the inputs that is not learnable from the value of $g(x_1, \ldots, x_n)$ together with $V$, differential privacy is ensured, provided the faulty players are restricted to polynomial time. Thus, secure function evaluation allows a computational notion of differential privacy to be achieved, even without a trusted curator, at no loss in accuracy when compared to what can be achieved with a trusted curator. In particular, counting queries can be answered with constant expected error while ensuring computational differential privacy, with no trusted curator. We will see that, without cryptography, the error must be $\Omega(n^{1/2})$, proving that computational assumptions provably buy accuracy, in the multiparty case.

## 9.1 Polynomial time curators

In this section we show that, under standard cryptographic assumptions, it is computationally difficult to create a synthetic database that will yield accurate answers to an appropriately chosen class of counting queries, while ensuring even a minimal notion of privacy.

This result has several extensions; for example, to the case in which the set of queries is small (but the data universe remains large), and the case in which the data universe is small (but the set of queries is large). In addition, similar negative results have been obtained for certain natural families of queries, such as those corresponding to conjunctions.

We will use the term *syntheticize* to denote the process of generating a synthetic database in a privacy-preserving fashion[3]. Thus, the results in this section concern the computational hardness of syntheticizing. Our notion of privacy will be far weaker than differential privacy, so hardness of syntheticizing will imply hardness of generating a synthetic

[3]In Section 6 a syntheticizer took as input a synopsis; here we are starting with a database, which is a trivial synopsis.

database in a differentially private fashion. Specifically, we will say that syntheticizing is hard if it is hard even to avoid leaking input items in their entirety. That is, some item is always completely exposed.

Note that if, in contrast, leaking a few input items is not considered a privacy breach, then syntheticizing is easily achieved by releasing a randomly chosen subset of the input items. Utility for this "synthetic database" comes from sampling bounds: with high probability this subset will preserve utility even with respect to a large set of counting queries.

When introducing complexity assumptions, we require a *security parameter* in order to express sizes; for example, sizes of sets, lengths of messages, number of bits in a decryption key, and so on, as well as to express computational difficulty. The security parameter, denoted $\kappa$, represents "reasonable" sizes and effort. For example, it is assumed that it is feasible to exhaustively search a set whose size is (any fixed) polynomial in the security parameter.

Computational complexity is an asymptotic notion — we are concerned with how the difficulty of a task increases as the sizes of the objects (data universe, database, query family) grow. Thus, for example, we therefore need to think not just of a distribution on databases of a single size (what we have been calling $n$ in the rest of this monograph), but of an ensemble of distributions, indexed by the security parameter. In a related vein, when we introduce complexity we tend to "soften" claims: forging a signature is not impossible — one might be lucky! Rather, we assume that no efficient algorithm succeeds with non-negligible probability, where "efficient" and "non-negligible" are defined in terms of the security parameter. We will ignore these fine points in our intuitive discussion, but will keep them in the formal theorem statements.

Speaking informally, a distribution of databases is *hard to synthesticize* (with respect to some family $\mathcal{Q}$ of queries) if for any efficient (alleged) syntheticizer, with high probability over a database drawn from the distribution, at least one of the database items can be extracted from the alleged syntheticizer's output. Of course, to avoid triviality, we will also require that when this leaked item is excluded from the input database (and, say, replaced by a random different item),

the probability that it can be extracted from the output is very small. This means that any efficient (alleged) syntheticizer indeed compromises the privacy of input items in a strong sense.

Definition 9.1 below will formalize our utility requirements for a syntheticizer. There are three parameters: $\alpha$ describes the accuracy requirement (being within $\alpha$ is considered accurate); $\gamma$ describes the fraction of the queries on which a successful synthesis is allowed to be inaccurate, and $\beta$ will be the probability of failure.

For an algorithm $A$ producing synthetic databases, we say that an output $A(x)$ is $(\alpha, \gamma)$-*accurate for a query set* $\mathcal{Q}$ if $|q(A(x)) - q(x)| \leq \alpha$ for a $1 - \gamma$ fraction of the queries $q \in \mathcal{Q}$.

**Definition 9.1** ($(\alpha, \beta, \gamma)$-Utility)**.** Let $\mathcal{Q}$ be a set of queries and $\mathcal{X}$ a data universe. A syntheticizer $A$ has $(\alpha, \beta, \gamma)$-utility for $n$-item databases with respect to $\mathcal{Q}$ and $\mathcal{X}$ if for any $n$-item database $x$:

$$\Pr\left[A(x) \text{ is } (\alpha, \gamma)\text{-accurate for } \mathcal{Q}\right] \geq 1 - \beta$$

where the probability is over the coins of $A$.

Let $\mathcal{Q} = \{\mathcal{Q}_n\}_{n=1,2,\ldots}$ be a query family ensemble, $\mathcal{X} = \{\mathcal{X}_n\}_{n=1,2,\ldots}$ be a data universe ensemble. An algorithm is said to be *efficient* if its running time is $\text{poly}(n, \log(|\mathcal{Q}_n|), \log(|\mathcal{X}_n|))$.

In the next definition we describe what it means for a family of distributions to be hard to syntheticize. A little more specifically we will say what it means to be hard to generate synthetic databases that provide $(\alpha, \gamma)$-accuracy. As usual, we have to make this an asymptotic statement.

**Definition 9.2** ($(\mu, \alpha, \beta, \gamma, \mathcal{Q})$-Hard-to-Syntheticize Database Distribution)**.** Let $\mathcal{Q} = \{\mathcal{Q}_n\}_{n=1,2,\ldots}$ be a query family ensemble, $\mathcal{X} = \{\mathcal{X}_n\}_{n=1,2,\ldots}$ be a data universe ensemble, and let $\mu, \alpha, \beta, \gamma \in [0, 1]$. Let $n$ be a database size and $\mathcal{D}$ an ensemble of distributions, where $\mathcal{D}_n$ is over collections of $n + 1$ items from $X_n$.

We denote by $(x, i, x_i') \sim \mathcal{D}_n$ the experiment of choosing an $n$-element database, an index $i$ chosen uniformly from $[n]$, and an additional element $x_i'$ from $\mathcal{X}_n$. A sample from $\mathcal{D}_n$ gives us a pair of databases: $x$ and the result of replacing the $i$th element of $x$ (under

a canonical ordering) with $x'_i$. Thus, we think of $\mathcal{D}_n$ as specifying a distribution on $n$-item databases (and their neighbors).

We say that $\mathcal{D}$ is $(\mu, \alpha, \beta, \gamma, \mathcal{Q})$-*hard-to Syntheticize* if there exists an efficient algorithm $T$ such that for any alleged efficient syntheticizer $A$ the following two conditions hold:

1. With probability $1-\mu$ over the choice of database $x \sim \mathcal{D}$ and the coins of $A$ and $T$, if $A(x)$ maintains $\alpha$-utility for a $1-\gamma$ fraction of queries, then $T$ can recover one of the rows of $x$ from $A(x)$:

$$\Pr_{\substack{(x,i,x'_i)\sim D_n \\ \text{coin flips of } A,T}} [(A(x) \text{ maintains } (\alpha,\beta,\gamma)\text{-utility}) \text{ and } (x \cap T(A(x)) = \emptyset)] \leq \mu$$

2. For *every* efficient algorithm $A$, and for every $i \in [n]$, if we draw $(x, i, x'_i)$ from $D$, and replace $x_i$ with $x'_i$ to form $x'$, $T$ cannot extract $x_i$ from $A(x')$ except with small probability:

$$\Pr_{\substack{(x,i,x'_i)\sim D_n \\ \text{coin flips of } A,\ T}} [x_i \in T(A(x'))] \leq \mu.$$

Later, we will be interested in offline mechanisms that produce arbitrary synopses, not necessarily synthetic databases. In this case we will be interested in the related notion of *hard to sanitize* (rather than hard to Syntheticize), for which we simply drop the requirement that $A$ produce a synthetic database.

## 9.2 Some hard-to-Syntheticize distributions

We now construct three distributions that are hard to syntheticize.

A *signature scheme* is given by a triple of (possibly randomized) algorithms $(\text{Gen}, \text{Sign}, \text{Verify})$:

- $\text{Gen} : 1^{\mathbb{N}} \to \{(\text{SK}, \text{VK})_n\}_{n=1,2,\ldots}$ is used to generate a pair consisting of a (secret) *signing* key and a (public) *verification* key. It takes only the security parameter $\kappa \in \mathbb{N}$, written in unary, as input, and produces a pair drawn from $(\text{SK}, \text{VK})_\kappa$, the distribution on (signature,verification) key pairs indexed by $\kappa$; we let

$p_s(\kappa), p_v(\kappa), \ell s(\kappa)$ denote the lengths of the signing key, verification key, and signature, respectively.

- Sign : $\mathrm{SK}_\kappa \times \{0,1\}^{\ell(\kappa)} \to \{0,1\}^{\ell s(\kappa)}$ takes as input a signing key from a pair drawn from $(\mathrm{SK}, \mathrm{VK})_\kappa$ and a message $m$ of length $\ell(\kappa)$, and produces a signature on $m$;
- Verify : $\mathrm{VK}_\kappa \times \{0,1\}^* \times \{0,1\}^{\ell(\kappa)} \to \{0,1\}$ takes as input a verification key, a string $\sigma$, and a message $m$ of length $\ell(\kappa)$, and checks that $\sigma$ is indeed a valid signature of $m$ under the given verification key.

Keys, message lengths, and signature lengths are all polynomial in $\kappa$.

The notion of security required is that, given any polynomial (in $\kappa$) number of valid (message, signature) pairs, it is hard to forge *any* new signature, even a new signature of a previously signed message (recall that the signing algorithm may be randomized, so there may exist multiple valid signatures of the same message under the same signing key). Such a signature scheme can be constructed from any one-way function. Speaking informally, these are functions that are easy to compute — $f(x)$ can be computed in time polynomial in the length (number of bits) of $x$, but hard to invert: for every probabilistic polynomial time algorithm, running in time polyomial in the security parameter $\kappa$, the probability, over a randomly chosen $x$ in the domain of $f$, of finding *any* valid pre-image of $f(x)$, grows more slowly than the inverse of any polynomial in $\kappa$.

**Hard to Syntheticize Distribution I:** Fix an arbitrary signature scheme. The set $\mathcal{Q}_\kappa$ of counting queries contains one counting query $q_{vk}$ for each verification key $vk \in \mathrm{VK}_\kappa$. The data universe $\mathcal{X}_\kappa$ consists of the set of all possible (message, signature) pairs of the form for messages of length $\ell(\kappa)$ signed with keys in $\mathrm{VK}_\kappa$.

The distribution $\mathcal{D}_\kappa$ on databases is defined by the following sampling procedure. Run the signature scheme generator $\mathrm{Gen}(1^\kappa)$ to obtain $(sk, vk)$. Randomly choose $n = \kappa$ messages in $\{0,1\}^{\ell(\kappa)}$ and run the signing procedure for each one, obtaining a set of $n$ (message, signature) pairs all signed with key $sk$. This is the database $x$. Note that all the messages in the database are signed with the *same* signing key.

A data universe item $(m, \sigma)$ satisfies the predicate $q_{vk}$ if and only if $\text{Verify}(vk, m, \sigma) = 1$, i.e., $\sigma$ is a valid signature for $m$ according to verification key $vk$.

Let $x \in_R \mathcal{D}_\kappa$ be a database, and let $sk$ be the signing key used, with corresponding verification key $vk$. Assuming that the syntheticizer has produced $y$, it must be the case that almost all rows of $y$ are valid signatures under $vk$ (because the fractional count of $x$ for the query $vk$ is 1). By the unforgeability properties of the signature scheme, all of these must come from the input database $x$ — the polynomial time bounded curator, running in time $\text{poly}(\kappa)$, cannot generate generate a *new* valid (message, signature) pair. (Only slightly) more formally, the probability that an efficient algorithm could produce a (message, signature) pair that is verifiable with key $vk$, but is not in $x$, is negligible, so with overwhelming probability any $y$ that is produced by an efficient syntheticizer will only contain rows of $x$.[4] This contradicts (any reasonable notion of) privacy.

In this construction, both $\mathcal{Q}_\kappa$ (the set of verification keys) and $\mathcal{X}_\kappa$ (the set of (message, signature) pairs) are large (superpolynomial in $\kappa$). When both sets are small, efficient differentially private generation of synthetic datasets is possible. That is, there is a differentially private syntheticizer whose running time is polynomial in $n = \kappa$, $|\mathcal{Q}_\kappa|$ and $|\mathcal{X}_\kappa|$: compute noisy counts using the Laplace mechanism to obtain a synopsis and then run the syntheticizer from Section 6. Thus, when both of these have size polynomial in $\kappa$ the running time of the syntheticizer is polynomial in $\kappa$.

We now briefly discuss generalizations of the first hardness result to the cases in which one of these sets is small (but the other remains large).

**Hard to Syntheticize Distribution II:** In the database distribution above, we chose a single $(sk, vk)$ key pair and generated a database of

[4]The quantification order is important, as otherwise the syntheticizer could have the signing key hardwired in. We first fix the syntheticizer, then run the generator and build the database. The probability is over all the randomness in the experiment: choice of key pair, construction of the database, and randomness used by the syntheticizer.

messages, all signed using $sk$; hardness was obtained by requiring the syntheticizer to generate a new signature under $sk$, in order for the syntheticized database to provide an accurate answer to the query $q_{vk}$. To obtain hardness for syntheticizing when the size of the set of queries is only polynomial in the security parameter, we again use digital signatures, signed with a unique key, but we cannot afford to have a query for each possible verification key $vk$, as these are too numerous.

To address this, we make two changes:

1. Database rows now have the form (verification key, message, signature). more precisely, the data universe consists of (key,message,signature) triples $\mathcal{X} = \{(vk, m, s) : vk \in \mathrm{VK}_\kappa, m \in \{0,1\}^{\ell(\kappa)}, s \in \{0,1\}^{\ell s(\kappa)}\}$.
2. We add to the query class exactly $2p_v(\kappa)$ queries, where $p_v(\kappa)$ is the length of the verification keys produced by running the generation algorithm $\mathrm{Gen}(1^\kappa)$. The queries have the form $(i, b)$ where $1 \leq i \leq p_v(\kappa)$ and $b \in \{0,1\}$. The meaning of the query "$(i, b)$" is, "What fraction of the database rows are of the form $(vk, m, s)$ where $\mathrm{Verify}(vk, m, s) = 1$ and the $i$th bit of $vk$ is $b$?" By populating a database with messages signed according to a single key $vk$, we ensure that the responses to these queries should be close to one for all $1 \leq i \leq p(\kappa)$ when $vk_i = b$, and close to zero when $vk_i = 1 - b$.

With this in mind, the hard to syntheticize distribution on databases is constructed by the following sampling procedure: Generate a signature-verification key pair $(sk, vk) \leftarrow \mathrm{Gen}(1^\kappa)$, and choose $n = \kappa$ messages $m_1, \ldots, m_n$ uniformly from $\{0,1\}^{\ell(\kappa)}$. The database $x$ will have $n$, rows; for $j \in [n]$ the $j$th row is the verification key, the $j$th message and its valid signature, i.e., the tuple $(vk, m_j, \mathrm{Sign}(m_j, sk))$. Next, choose $i$ uniformly from $[n]$. To generate the $(n+1)$st item $x'_i$, just generate a new message-signature pair (using the same key $sk$).

**Hard to Syntheticize Distribution III:** To prove hardness for the case of a polynomial (in $\kappa$) sized message space (but superpolynomial sized query set) we use a *pseudorandom function*. Roughly speaking, these are polynomial time computable functions with small descriptions that

cannot efficiently be distinguished, based only on their input-output behavior, from truly random functions (whose descriptions are long). This result only gives hardness of syntheticizing if we insist on maintaining utility for *all* queries. Indeed, if we are interested only in ensuring on-average utility, then the base generator for counting queries described in Section 6 yields an efficient algorithm for syntheticizing when the universe $\mathcal{X}$ is of polynomial size, even when $\mathcal{Q}$ is exponentially large.

Let $\{f_s\}_{s\in\{0,1\}^\kappa}$ be a family of pseudo-random functions from $[\ell]$ to $[\ell]$, where $\ell \in \text{poly}(\kappa)$. More specifically, we need that the set of all pairs of elements in $[\ell]$ is "small," but larger than $\kappa$; this way the $\kappa$-bit string describing a function in the family is shorter than the $\ell \log_2 \ell$ bits needed to describe a random function mapping $[\ell]$ to $[\ell]$. Such a family of pseudorandom functions can be constructed from any one-way function.

Our data universe will be the set of all pairs of elements in $[\ell]$: $\mathcal{X} = \{(a, b) : a, b \in [\ell]\}$. $\mathcal{Q}_\kappa$ will contain two types of queries:

1. There will be one query for each function $\{f_s\}_{s\in\{0,1\}^\kappa}$ in the family. A universe element $(a, b) \in \mathcal{X}$ satisfies the query $s$ if and only if $f_s(a) = b$.
2. There will be a relatively small number, say $\kappa$, truly random queries. Such a query can be constructed by randomly choosing, for each $(a, b) \in \mathcal{X}$, whether or not $(a, b)$ will satisfy the query.

The hard to syntheticize distribution is generated as follows. First, we select a random string $s \in \{0, 1\}^\kappa$, specifying a function in our family. Next, we generate, for $n = \kappa$ distinct values $a_1, \ldots, a_n$ chosen at random from $[\ell]$ without replacement, the universe element $(a, f_s(a))$.

The intuition is simple, relies only on the first type of query, and does not make use of the distinctness of the $a_i$. Given a database $x$ generated according to our distribution, where the pseudo-random function is given by $s$, the syntheticizer must create a synthetic database (almost) all of whose rows must satisfy the query $s$. The intuition is that it can't *reliably* find input-output pairs that do not appear in $x$. A little more precisely, for an arbitrary element $a \in [\ell]$ such that no

row in $x$ is of the form $(a, f_s(a))$, the pseudo-randomness of $f_s$ says that an efficient syntheticizer should have probability at most negligibly more than $1/\ell$ of finding $f_s(a)$. In this sense the pseudo-randomness gives us properties similar to, although somewhat weaker than, what we obtained from digital signatures.

Of course, for any given $a \in [\ell]$, the syntheticizer can indeed guess with probability $1/\ell$ the value $f_s(a)$, so without the second type of query, nothing obvious would stop it from ignoring $x$, choosing an arbitrary $a$, and outputting a database of $n$ copies of $(a, b)$, where $b$ is chosen uniformly at random from $[\ell]$. The intuition is now that such a synthetic database would give the wrong fraction — either zero or one, when the right answer should be about $1/2$ — on the truly random queries.

Formally, we have:

**Theorem 9.1.** Let $f : \{0,1\}^\kappa \to \{0,1\}^\kappa$ be a one-way function. For every $a > 0$, and for every integer $n = \text{poly}(\kappa)$, there exists a query family $\mathcal{Q}$ of size $\exp(\text{poly}(\kappa))$, a data universe $\mathcal{X}$ of size $O(n^{2+2a})$, and a distribution on databases of size $n$ that is $(\mu, \alpha, \beta, 0, \mathcal{Q})$-*hard-to-syntheticize* (i.e., hard to syntheticize for worst-case queries) for $\alpha \leq 1/3$, $\beta \leq 1/10$ and $\mu = 1/40n^{1+a}$.

The above theorem shows hardness of sanitizing with synthetic data. Note, however, that when the query set is small one can always simply release noisy counts for every query. We conclude that sanitizing for small query classes (with large data universes) is a task that separates efficient syntheticizing from efficient synopsis generation (sanitization with arbitrary outputs).

### 9.2.1 Hardness results for general synopses

The hardness results of the previous section apply only to syntheticizers — offline mechanisms that create synthetic databases. There is a tight connection between hardness for more general forms of privacy-preserving offline mechanisms, which we have been calling *offline query release mechanisms* or synopsis generators, and the existence of *traitor tracing* schemes, a method of content distribution in which (short) key

strings are distributed to subscribers in such a way that a sender can broadcast encrypted messages that can be decrypted by any subscriber, and any useful "pirate" decoder constructed by a coalition of malicious subscribers can be traced to at least one colluder.

A (private-key, stateless) traitor-tracing scheme consists of algorithms Setup, Encrypt, Decrypt and Trace. The Setup algorithm generates a key $bk$ for the broadcaster and $N$ subscriber keys $k_1, \ldots, k_N$. The Encrypt algorithm encrypts a given bit using the broadcaster's key $bk$. The Decrypt algorithm decrypts a given ciphertext using any of the subscriber keys. The Trace algorithm gets the key $bk$ and oracle access to a (pirate, stateless) decryption box, and outputs the index $i \in \{1, \ldots, N\}$ of a key $k_i$ that was used to create the pirate box.

An important parameter of a traitor-tracing scheme is its *collusion-resistance*: a scheme is $t$-resilient if tracing is guaranteed to work as long as no more than $t$ keys are used to create the pirate decoder. When $t = N$, tracing works even if all the subscribers join forces to try and create a pirate decoder. A more complete definition follows.

**Definition 9.3.** A scheme (Setup, Encrypt, Decrypt, Trace) as above is a *$t$-resilient traitor-tracing scheme* if (i) the ciphertexts it generates are semantically secure (roughly speaking, polynomial time algorithms cannot distinguish encryptions of 0 from encryptions of 1), and (ii) no polynomial time adversary $A$ can "win" in the following game with non-negligible probability (over the coins of Setup, $A$, and Trace):

$A$ receives the number of users $N$ and a security parameter $\kappa$ and (adaptively) requests the keys of up to $t$ users $\{i_1, \ldots, i_t\}$. The adversary then outputs a pirate decoder Dec. The Trace algorithm is run with the key $bk$ and black-box access[5] to Dec; it outputs the name $i \in [N]$ of a user or the error symbol $\perp$. We say that an adversary $A$ "wins" if it is both the case that Dec has a non-negligible advantage in decrypting ciphertexts (even a weaker condition than creating a usable pirate decryption device), *and* the output of Trace is not in $\{i_1, \ldots, i_t\}$, meaning that the adversary avoided detection.

[5]Black-box access to an algorithm means that one has no access to the algorithm's internals; one can only feed inputs to the algorithm and observe its outputs.

The intuition for why traitor-tracing schemes imply hardness results for counting query release is as follows. Fix a traitor tracing scheme. We must describe databases and counting queries for which query release is computationally hard.

For any given $n = \kappa$, the database $x \in \{\{0,1\}^d\}^n$ will contain user keys from the traitor tracing scheme of a colluding set of $n$ users; here $d$ is the length of the decryption keys obtained when the Setup algorithm is run on input $1^\kappa$. The query family $\mathcal{Q}_\kappa$ will have a query $q_c$ for each possible ciphertext $c$ asking "For what fraction of the rows $i \in [n]$ does $c$ decrypt to 1 under the key in row $i$?" Note that, since every user can decrypt, if the sender distributes an encryption $c$ of the bit 1, the answer will be 1: all the rows decrypt $c$ to 1, so the fraction of such rows is 1. If instead the sender distributes an encryption $c'$ of the bit 0, the answer will be 0: since no row decrypts $c'$ to 1, the fraction of rows decrypting $c'$ to 1 is 0. Thus, the exact answer to a query $q_c$, where $c$ is an encryption of a 1-bit messages $b$, is $b$ itself.

Now, suppose there were an efficient offline differentially private query release mechanism for queries in $\mathcal{Q}$. The colluders could use this algorithm to efficiently produce a synopsis of the database enabling a data analyst to efficiently compute approximate answers to the queries $q_c$. If these approximations are at all non-trivial, then the analyst can use these to correctly decrypt. That is, the colluders could use this to form a pirate decoder box. But traitor tracing ensures that, for any such box, the Trace algorithm can recover the key of at least one user, i.e., a row of the database. This violates differential privacy, contradicting the assumption that there is an efficient differentially private algorithm for releasing $\mathcal{Q}$.

This direction has been used to rule out the existence of efficient offline sanitizers for a particular class of $2^{\tilde{O}(\sqrt{n})}$ counting queries; this can be extended to rule out the existence of efficient *on-line* sanitizers answering $\tilde{\Theta}(n^2)$ counting queries drawn adaptively from a second (large) class.

The intuition for why hardness of offline query release for counting queries implies traitor tracing is that failure to protect privacy immediately yields some form of traceability; that is, the *difficulty* of providing an object that yields (approximate) functional equivalence for a set of

rows (decryption keys) while preserving privacy of each individual row (decryption key) — that is, the difficulty of producing an untraceable decoder — is precisely what we are looking for in a traitor tracing scheme.

In a little more detail, given a hard-to-sanitize database distribution and family of counting queries, a randomly drawn $n$-item database can act like a "master key," where the secret used to decrypt messages is the *counts* of random queries on this database. For a randomly chosen subset $S$ of polylog$(n)$ queries, a random set of polylog$(n)$ rows drawn from the database (very likely) yields good approximation to all queries in $S$. Thus, individual user keys can be obtained by randomly partitioning the database into $n/\text{polylog}(n)$ sets of polylog$(n)$ rows and assigning each set to a different user. These sets are large enough that with overwhelming probability their counts on a random collection of say polylog$(n)$ queries are *all* close to the counts of the original database.

To complete the argument, one designs an encryption scheme in which decryption is equivalent to computing approximate counts on small sets of random queries. Since by definition a pirate decryption box can decrypt, the a pirate box can be used to compute approximate counts. If we view this box as a sanitization of the database we conclude (because sanitizing is hard) that the decryption box can be "traced" to the keys (database items) that were used to create it.

## 9.3 Polynomial time adversaries

**Definition 9.4** (Computational Differential Privacy)**.** A randomized algorithm $C_\kappa : \mathcal{X}^n \to Y$ is *$\varepsilon$-computationally differentially private* if and only if for all databases $x, y$ differing in a single row, and for all nonuniform polynomial (in $\kappa$) algorithms $T$,

$$\Pr[T(C_\kappa(x)) = 1] \leq e^\varepsilon \Pr[T(C_\kappa(y)) = 1] + \nu(\kappa),$$

where $\nu(\cdot)$ is any function that grows more slowly than the inverse of any polynomial and the agorithm $C_\kappa$ runs in time polynomial in $n$, $\log|\mathcal{X}|$, and $\kappa$.

Intuitively, this says that if the adversary is restricted to polynomial time then computationally differentially private mechanisms provide the same degree of privacy as do $(\varepsilon, \nu(\kappa))$-differentially private algorithms. In general there is no hope of getting rid of the $\nu(\kappa)$ term; for example, when encryption is involved there is always some (neglibly small) chance of guessing the decryption key.

Once we assume the adversary is restricted to polynomial time, we can use the powerful techniques of *secure multiparty computation* to provide *distributed* online query release algorithms, replacing the trusted server with a distributed protocoal that simulates a trusted curator. Thus, for example, a set of hospitals, each holding the data of many patients, can collaboratively carry out statistical analyses of the union of their patients, while ensuring differential privacy for each patient. A more radical implication is that individuals can maintain their own data, opting in or out of each specific statistical query or study, all the while ensuring differential privacy of their own data.

We have already seen one distributed solution, at least for the problem of computing a sum of $n$ bits: randomized response. This solution requires no computational assumptions, and has an expected error of $\Theta(\sqrt{n})$. In contrast, the use of cryptographic assumptions permits much more accurate and extensive analyses, since by simulating the curator it can run a distributed implementation of the Laplace mechanism, which has constant expected error.

This leads to the natural question of whether there is some other approach, not relying on cryptographic assumptions, that yields better accuracy in the distributed setting than does randomized response. Or more generally, is there a separation between what can be accomplished with computational differential privacy and what can be achieved with "traditional" differential privacy? That is, does cryptography provably buy us something?

In the multiparty setting the answer is yes. Still confining our attention to summing $n$ bits, we have:

**Theorem 9.2.** For $\varepsilon < 1$, every $n$-party $(\varepsilon, 0)$-differentially private protocol for computing the sum of $n$ bits (one per party) incurs error $\Omega(n^{1/2})$ with high probability.

A similar theorem holds for $(\varepsilon, \delta)$-differential privacy provided $\delta \in o(1/n)$.

*Proof.* (sketch) Let $X_1, \ldots, X_n$ be uniform independent bits. The transcript $T$ of the protocol is a random variable $T = T(P_1(X_1), \ldots, P_n(X_n)$, where for $i \in [n]$ the protocol of player $i$ is denoted $P_i$. Conditioned on $T = t$, the bits $X_1, \ldots, X_n$ are still independent bits, each with bias $O(\varepsilon)$. Further, by differential privacy, the uniformity of the $X_i$, and Bayes' Law we have:

$$\frac{\Pr[X_i = 1|T = t]}{\Pr[X_i = 0|T = t]} = \frac{\Pr[T = t|X_i = 1]}{\Pr[T = t|X_i = 0]} \leq e^{\varepsilon} < 1 + 2\varepsilon.$$

To finish the proof we note that the sum of $n$ independent bits, each with constant bias, falls outside any interval of size $o(\sqrt{n})$ with high probability. Thus, with high probability, the sum $\sum_i X_i$ is not in the interval $[\text{output(T)} - o(n^{1/2}), \text{output(T)} + o(n^{1/2})]$. □

A more involved proof shows a separation between computational differential privacy and ordinary differential privacy even for the two-party case. It is a fascinating open question whether computational assumptions buy us anything in the case of the trusted curator. Initial results are negative: for *small* numbers of *real-valued* queries, i.e., for a number of queries that does not grow with the security parameter, there is a natural class of utility measures, including $L_p$ distances and mean-squared errors, for which any computationally private mechanism can be converted to a statistically private mechanism that is roughly as efficient and achieves almost the same utility.

## 9.4 Bibliographic notes

The negative results for polynomial time bounded curators and the connection to traitor tracing are due to Dwork et al. [28]. The connection to traitor tracing was further investigated by Ullman [82], who showed that, assuming the existence of 1-way functions, it is computationally hard to answer $n^{2+o(1)}$ arbitrary linear queries with differential privacy (even if without privacy the answers are easy to compute). In "Our Data, Ourselves," Dwork, Kenthapadi, McSherry, Mironov, and

Naor considered a distributed version of the precursor of differential privacy, using techniques from secure function evaluation in place of the trusted curator [21]. A formal study of *computational* differential privacy was initiated in [64], and the separation between the accuracy that can be achieved with $(\varepsilon, 0)$-differential privacy in the multiparty and single curator cases in Theorem 9.2 is due to McGregor et al. [58]. The initial results regarding whether computational assumptions on the adversary buys anything in the case of a trusted curator are due to Groce et al. [37].

Construction of pseudorandom functions from any one-way function is due to Håstad et al. [40].