15-316: Software Foundations of Security and Privacy

Lecture Notes on Provable Privacy

Matt Fredrikson

Carnegie Mellon University
Lecture 14

1 Introduction

In the previous lecture we looked at ways of relaxing noninterference so that we could use similar properties to reason about the information flow security of programs that don't leak "too much" about their secret inputs. To address this problem in the special context of such authorization checks, we discussed a type system due to Dennis Volpano and Geoff Smith[6]. They introduced a corresponding typing rule that allows the result of a `match` expression to flow to the class corresponding to the least of its given operands.

$$\text{(Match)}\quad \frac{\Gamma \vdash e : \ell_1 \quad \Gamma \vdash \tilde{e} : \ell_2}{\Gamma \vdash \texttt{match}(e, \tilde{e}) : \ell_1 \sqcap \ell_2}$$

They formalized the meaning of leaking "too much" by appealing to the complexity of an attacker who wishes to learn the secret state in the resulting type system, concluding that to do so would require exponential resources.

Theorem 1. *Let $\Gamma = (s : \texttt{H}, o : \texttt{L})$ and α be a deterministic program such that:*

1. *$\Gamma \vdash \alpha$ in the type system that includes* `match`*.*
2. *$\langle \omega, \alpha \rangle \Downarrow \nu$ for $\omega(s) = v, \omega(o) \neq v, \nu(o) = v$.*

So α always succeeds at transferring the value stored in s at the initial state into o at the final state. Then there exists some state $\hat{\omega}$ where:

1. *$\hat{\omega}(s)$ requires at least k bits to represent*
2. *Evaluating $\langle \hat{\omega}, \alpha \rangle$ results in greater than $poly(k)$ evaluations of* `match`*.*

In other words, the type system prevents polynomial-time attacks on some `H` *state.*

We then introduced a more general technique for understanding the degree to which programs leak sensitive information, based on the idea of a *feasible set* $\Omega_\Gamma(\alpha, \omega)$. This set corresponds to the initial states that *agree with the attacker's knowledge of the* `L` *values of the initial and final states*:

$$\Omega_\Gamma(\alpha, \omega) = \{\omega' \ : \ \omega' \approx_{\mathtt{L}} \omega \text{ and } \mathsf{Ev}(\omega', \alpha) \approx_{\mathtt{L}} \mathsf{Ev}(\omega, \alpha)\}$$

The feasible set characterizes all of the attacker's uncertainty about the initial state, and in particular the `H` part of the initial state (because the rest is known). Intuitively, any state in the feasible set will lead to *exactly* the same observations in the initial and final states, and so the attacker has no way of determining which of these states the program actually started in. From the attacker's perspective, they are *indistinguishable* from each other.

In today's lecture, we will look more carefully at a different set of techniques for revealing some useful information about secret state while controlling the attacker's level of uncertainty about it. These techniques all use randomness to produce approximate results for computations, while providing some form of cover for the true secret. We will look at a property called *differential privacy* [2] that formalizes the protections one might gain from this approach, and study some properties that make it useful for building computations that protect secret data. Differential privacy has been applied to a wide range of important computations to protect the privacy of source data [3], from machine learning [1] to web browser data collection [4]. We will not have time to cover these applications in any detail, but will instead focus on the core ideas behind the approach.

2 Quantifying uncertainty

In the previous lecture we mentioned that when the feasible set is large, it is an indication that the associated program did not reveal "too much" about its secret initial state. The reason for this is that when a large number of initial states remain consistent with an attacker's observations, then the attacker's uncertainty about which one was actually used is great. So perhaps we can reason about information flow security in terms of keeping the attacker's uncertainty about the secret high.

But what can we do if the program that we want to write is inherently "leaky" in that it results in small feasible sets? One way that we can make the attacker more uncertain about the secret initial state is to use randomness in our program. Consider for example a technique called *randomized response* [7], which is a privacy technique dating back to the 1960s with roots in the social sciences. Randomized response was motivated by survey collection, in situations where questions asked of respondents relate to sensitive issues. Randomized response gives these subjects *plausible deniability*, by providing a structured way of adding random "noise" to their answer.

In the following, assume that $\mathsf{flip}(p)$ is a random function that flips a biased coin with

parameter p. In other words,

$$\mathsf{flip}() = \begin{cases} 1 & \text{with probability } 1/2 \\ 0 & \text{with probability } 1/2 \end{cases} \tag{1}$$

Then suppose that F is a function that returns a value in $\{0, 1\}$, and that we wish to release $F(x)$ publicly while hiding the secret value x as much as possible. Then the randomized response program RandResp, is as follows, where we assume that the variable o is publicly-observable and b is not (e.g., $\Gamma = x : \mathtt{H}, b : \mathtt{H}, o : \mathtt{L}$).

$$\begin{array}{l} b := \mathsf{flip}() \\ \textbf{if } b = 1 \textbf{ then} \\ \quad o := F(x) \\ \textbf{else} \\ \quad o := \mathsf{flip}() \end{array} \tag{2}$$

In short, randomized response returns the true value of $F(x)$ with probability $1/2$, and a completely random answer with probability $1/2$. In terms of feasible sets, this appears to be an absolutely brilliant approach because now the attacker must be completely uncertain about the initial value of x. Why is this so? The adversary can only see o, and if $b = 0$ after being assigned, then o does not depend at all on x, so x could be anything as though the program satisfied non-interference.

But perhaps this doesn't seem quite right. Let's assume for a moment that $x \in \{0, 1\}$ and F is simply the identity function, and walk through the various possibilities. In the following, we will treat RandResp as though it were a function of x that returns the value in o after executing. If $x = 0$, then,

$$\Pr[\mathsf{RandResp}(0) = 0] = \Pr[b = 1] + \Pr[b = 0 \wedge \mathsf{flip}() = 0] = 1/2 + 1/4 = 3/4 \tag{3}$$

We could use the exact same reasoning to conclude that $\Pr[\mathsf{RandResp}(1) = 1] = 3/4$. Likewise we could reason about the probability that randomized response outputs an incorrect answer,

$$\Pr[\mathsf{RandResp}(0) = 1] = 1 - \Pr[\mathsf{RandResp}(0) = 0] = \Pr[b = 0 \wedge \mathsf{flip}() = 1] = 1/4 \tag{4}$$

So we see that RandResp outputs the *correct* value of $F(x)$ with fairly high probability of $3/4$, and an incorrect "random" value with probability $1/4$. In other words, most of the time the attacker is safe in assuming that RandResp outputs exactly the same value as $F(x)$, and so can go about inferring x by computing feasible sets as before.

This isn't to say that randomized response does nothing to protect x, and indeed it may offer ample protection for many applications because the attacker still has more uncertainty than they would otherwise. But by reasoning about the probabilities of various outcomes and what the attacker is able to infer from them, we arrived at a much more nuanced view of the degree of security than was suggested by looking at the feasible set of RandResp alone.

2.1 Quantifying a tradeoff

There are some arbitrary choices that have been made in this conception of randomized response, and they influence the degree of adversarial uncertainty of the secret input x. In particular, we could generalize $\mathsf{flip}()$ by adding a parameter $0 \leq p \leq 1$ controlling the bias of the coin.

$$\mathsf{flip}(p) = \begin{cases} 1 & \text{with probability } p \\ 0 & \text{with probability } 1-p \end{cases} \tag{5}$$

We could use this in RandResp as follows, assuming p is chosen to be some constant in advance.

$$\begin{array}{l} b := \mathsf{flip}(p) \\ \textbf{if } b = 1 \textbf{ then} \\ \quad o := F(x) \\ \textbf{else} \\ \quad o := \mathsf{flip}(p) \end{array} \tag{6}$$

Then updating the analysis we did before with this more general solution, we see that:

$$\begin{aligned} \Pr[\mathsf{RandResp}(x) = F(x)] &= \Pr[b = 1] + \Pr[b = 0 \wedge \mathsf{flip}(p) = F(x)] \\ &= p + (1-p)\Pr[F(x) = \mathsf{flip}(p)] \end{aligned} \tag{7}$$

When F is the identity function then we have,

$$\begin{aligned} \Pr[\mathsf{RandResp}(0) = 0] &= \Pr[b = 1] + \Pr[b = 0 \wedge \mathsf{flip}(p) = 0] = p + (1-p)^2 \\ \Pr[\mathsf{RandResp}(1) = 1] &= \Pr[b = 1] + \Pr[b = 0 \wedge \mathsf{flip}(p) = 1] = p + (1-p)p = 2p - p^2 \end{aligned}$$

So if we set $p \geq 1/2$, then we would be sure to have a more accurate answer in the sense that RandResp returns $F(x)$ with greater likelihood. But this comes at a tradeoff in information flow security, as the attacker can also be more confident (less uncertain) about the feasible set. Likewise, smaller values of p lead to a less accurate solution, but increase the attacker's uncertainty and so afford greater security.

3 Differential Privacy

Now we will turn to a property that is useful in many cases for characterizing the adversarial uncertainty one obtains through the use of randomized computation. In this setting, we will assume that the program α makes use of memory operations, and wants to prevent too much information about the contents of any cell in `Mem` from leaking through its output result. It is called *differential privacy*, and is an active area of study and application.

In the following, we will assume that all of the indices in `Mem` are secret and so typed `H`, and that all of the variables used by the program are typed `L`. So intuitively, think of the memory `Mem` as perhaps being a input where each cell holds the data of one individual that is to be used by α. The developer of α wishes to compute some useful aggregate fact about the individuals' data, and will store the result in the variables of

the final state $\mathsf{Ev}((\omega, \texttt{Mem}), \alpha)$. The goal is to make sure that the results do not reveal too much information about any single individual's data stored in `Mem`.

Definition 2. *ϵ-Differential Privacy.* Let $\epsilon > 0$. A program α satisfies ϵ-differential privacy if for all possible memory configurations $\texttt{Mem}_1, \texttt{Mem}_2$ that differ in *exactly one index*, and all states ω, ν, the following inequality holds:

$$\Pr[\mathsf{Ev}((\omega, \texttt{Mem}_1), \alpha) = \nu] \leq e^{\epsilon} \times \Pr[\mathsf{Ev}((\omega, \texttt{Mem}_2), \alpha) = \nu] \tag{8}$$

The probabilities in this expression are taken over the randomness of α's computation.

The fact that α is a program that does not explicitly "output" a single value is indeed irrelevant to the essence of this definition. It may be clearer for some to just think of α as a function that takes a memory configuration X as input and returns a single discrete value rather than a state. This leads to the following equivalent definition.

Definition 3. *ϵ-Differential Privacy (functional form).* Let $\epsilon > 0$. A function α satisfies ϵ-differential privacy if for all possible inputs X_1 and X_2 that differ in *exactly one index*, and all return values $s \in \mathsf{Range}(\alpha)$, the following inequality holds:

$$\Pr[\alpha(X_1) = s] \leq e^{\epsilon} \times \Pr[\alpha(X_2) = s] \tag{9}$$

The probabilities in this expression are taken over the randomness of α's outputs.

To keep notation as simple as possible, we will stick with the latter form of the definition for the remainder of the lecture.

First, notice that Definition 3 is a property of the function α, and *not* of the data being computed on or any particular output of α. In other words, when we speak of something as being differentially private, we are always referring to a process used to compute outputs from secret inputs. You may at times hear people refer to a piece of data as "differentially private", but do not get confused; when used correctly, this language means that the data was computed by a function that satisfies ϵ-differential privacy.

Second, the ϵ in Definition 3 is called the *privacy budget*, and controls the tradeoff between privacy and accuracy in much the same way that p did in our randomized response example before. We'll get into some high-level intuitive interpretations of this definition in a little while, but first let's think about its various components and how they relate to α's behavior directly.

Privacy budget ϵ. We hinted earlier that the privacy budget has an influence on both the degree of privacy established by the function, as well as the degree of approximation in the results. ϵ is our privacy budget, and it is a numeric real-valued quantity. To understand what it means, let's look at the behavior of an ϵ-differentially private α for extremal values of ϵ.

Suppose that we make $\epsilon = 0$. Then Definition 3 requires that for any X_1, X_2 that differ in one index, Equation 9 holds. However, notice that the definition is symmetric

in the values that X_1, X_2 take; there is nothing that distinguishes them from eachother, so α must also satisfy:

$$\Pr[\alpha(X_2) = s] \leq \Pr[\alpha(X_1) = s] \tag{10}$$

Combining equations 9 and 10, it must be that $\Pr[\alpha(X_1) = s] = \Pr[\alpha(X_2) = s]$ for all X_1, X_2 that differ in one index. What does this mean for the privacy of individuals in X_1 and X_2, and the utility of α?

- When it comes to privacy, we can conclude that $\epsilon = 0$ implies **no leakage** of information about the contents of *any* individual. Why does this hold for any index? Recall that Definition 3 needs to hold for *all* pairs X_1, X_2. So, if our actual input is X_1, then all input X_2 that we obtain by changing a index in X_1 must produce the same distribution of outputs in α.
- As for utility, you probably guessed that $\epsilon = 0$ isn't great. In fact, because α's output distribution needs to remain the same for all adjacent inputs, we can observe that by transitivity α's output distribution needs to remain the same for *all* inputs. In other words, the results can't contain any information about the input, which clearly means no utility is possible.

Clearly, $\epsilon = 0$ is good in terms of privacy but terrible for utility. We might expect that when ϵ is large, say 10, then the opposite is true. Note that $e^{10} \approx 22,000$. Probabilities range in $[0, 1]$, so in order for Equation 9 to place any meaningful limits on α's behavior, e.g. a limit on $\Pr[\alpha(X) = s]$ for some s, the probability of returning s on the neighboring input would need to be quite small ($\sim 4.5 \times 10^{-5}$). Conversely, because this large ϵ gives α quite a bit of freedom in its behavior, utility is not a problem.

So as ϵ grows, the privacy offered by ϵ-differential privacy drops off very quickly, and the utility begins to approach what we could achieve from return the exact answer without randomness.

Neighboring inputs. Definition 3 quantifies universally over pairs of inputs X_1, X_2 that differ in one index. Such pairs are called **neighbors**. What exactly do we mean by "differ in one index"? First of all, it's important to mention that we aren't concerned with the order of elements in X_1, X_2, so that if they were permutations of each other, we would consider them to be the same[1]. There are two reasonable interpretations.

1. X_1 and X_2 are identical, except that X_1 has an additional index that X_2 doesn't. So if we view inputs as sets (assuming all indices are unique within each input), then $|X_2| = |X_1| - 1$ and $X_2 \subseteq X_1$, $X_2 = X_1 \cup \{x_m\}$ for some x_m.
2. X_1 and X_2 have the same number of indices, but the value of one index is different. In other words, we could find a permutation of X_1 such that $X_1[1 \ldots N-1] = X_2[1 \ldots N-1]$, and $X_1[N] \neq X_2[N]$.

We will use by convention the latter definition of neighboring inputs.

[1] Usually the queries performed in this model are associative with respect to indices, so from the user's perspective, permuted inputs are the same

Randomized α. Is it essential that α be a random function? First of all, the Definition 3 is an inequality over probabilities, and the only source of randomness comes from α. So in a technical sense, we are required to assign probabilities to α's responses. However, we can interpret deterministic functions as a special case of randomized functions, so let's think about which deterministic functions might satisfy the definition.

Suppose that $X_1 = [0, 0, 0]$ and $X_2 = [0, 0, 1]$, and $\alpha(X_1) = s$. Any deterministic α whose value depends on the last element, so that $\alpha(X_2) = s'$ where $s \neq s'$ will give us:

$$\Pr[\alpha(X_1) = s] = 1, \Pr[\alpha(X_2) = s] = 0$$

so that for any ϵ Equation 9 fails to hold. Because the order of elements in X_1, X_2 doesn't matter, this means that α's response can't depend on *any* index in X. Thus, the only deterministic functions that satisfy Definition 3 are constant.

3.1 Interpreting the Definition

Now that we've thought about the definition and some of its technical implications, let's think about what it means for privacy.

Inference and protection from harm. One view of privacy is that it is about protecting individuals from harm that may arise from the release of their data. By learning things about individuals, a party with corrupt intent might use that information to limit their opportunities (e.g., deny them a job or a loan), offer differentiated services (e.g., higher prices for customers from affluent areas), or otherwise discriminate against them in numerous ways that play against their advantage.

One question that we might ask is, why not strive for a definition that prevents such parties from learning *anything* new about an individual from a result involving their data? If nothing new about the individual can be learned from the release, then no harm can follow. Researchers have contemplated this possibility before [2], and not suprisingly it turns out that doing so is at fundamental odds with a simultaneous goal of extracting useful insights from personal information.

Differential privacy aims to protect individuals from such harm to the greatest extent possible. The key to this is the *relative* nature of the definition. Rather than trying to prevent users from learning *anything* about an individual, we can think of the definition as trying to prevent users from learning new things about an individual relative to what they *could have* learned had the individual not shared their data. This is where the idea of neighboring inputs comes from: a neighboring input is one in which a particular individual's data takes a different value, which we can view as being a input where everyone *except* that individual shared (i.e., some other individual took their place). Differential privacy requires that any output of α be approximately as likely in both cases: one where the individual shared their data, and one where they did not.

For example, suppose that you are given the opportunity to share your medical records with a researcher who will use them in a study intended to improve treatments. You may rightly be concerned that if the researcher publishes results based

on your data, a data-savvy insurance provider might be able to infer something about your health status from these results in the future, and decide to raise your premiums or deny coverage. However, if the researcher applied differential privacy with an appropriately-chosen ϵ, then you might be reassured that no results that could come of the study would be that much more or less likely because of your decision to share. It follows that if an insurer were to base their decision on those differentially-private results, then they are similarly not much more or less likely to deny you coverage.

Plausible deniability. Another way of looking at the protection given by differential privacy is in terms of **plausible deniability**, or one's ability to make a believable claim that their data takes some value of their choosing, i.e., to "deny" a claim that their data took the value it did. Because Definition 3 requires that the likelihood of α responding with any value s is nearly identical regardless of what value the individual's data took, it would indeed be reasonable for the individual to claim that their data took another value; the probability of producing s would be about the same no matter what value they chose.

Indistinguishability and influence. Another way of viewing the definition, which brings us closer to the semantics of the computation done by α, is in terms of how much individuals' data can influence, or cause changes to, α's response. We've talked about influence before in the context of noninterference, which required that the `H`-typed initial state have no influence on the `L`-typed final state:

$$\forall \omega_1, \omega_2. \omega_1 \approx_{\mathtt{L}} \omega_2 \rightarrow \mathsf{Ev}(\omega_1, c) \approx_{\mathtt{L}} \mathsf{Ev}(\omega_2, c) \tag{11}$$

We might rewrite Definition 3 more concisely as follows.

$$\forall X_1, X_2. \mathsf{Neighbor}(X_1, X_2) \rightarrow \forall s. \Pr[\alpha(X_1) = s] \leq e^{\epsilon} \times \Pr[\alpha(X_2) = s] \tag{12}$$

Notice the similarities between Equations 11 and 12.

- In both cases, the definitions quantify over all pairs of inputs (i.e., initial states) that are related in a way that reflects what we are trying to protect. For noninterference, the relation does this by only constraining the `L` variables, so that the final state is indistinguishable regardless of the initial `H` variables. For differential privacy, the neighbor relation works similarly by letting each individual's data take an arbitrary value, and fixing the rest of the input.
- The right-hand side of the implication in each case describes the sort of changes that inputs, and more precisely inputs described by the left-hand side, are allowed to cause. Noninterference rules out any changes to `L` variables, whereas differential privacy places limits on the probability of variation in the response.

Viewed this way, differential privacy is a property which states that the influence of individual indices on α's response should remain low, so that responses computed under

neighboring inputs are "almost" indistinguishable. This is the essential property that allows for plausible deniability and protection from harm, and the core of differential privacy's strong guarantees.

Recall also that we were able to prove that programs satisfy noninterference, even to the point of designing type systems that simplify the task of writing noninterferent programs, and can be checked efficiently. Given the similarity between Equations 11 and 12, it should not be too surprising that we can also prove program's adherence to differential privacy. This is part of the appeal of using the definition in practice: it provides a crisp mathematical formulation of what it means to be private, that can be proved on real computations.

3.2 Proving differential privacy: randomized response

Now let's go back to our example of randomized response. Does it satisfy differential privacy? Let's keep things simple and assume that F is the identity function that just returns the contents of $\texttt{Mem}(0)$, $p = 1/2$ and all variables and memory cells hold values in the set $\{0, 1\}$. This corresponds to the following program.

$$\begin{array}{l} b := \mathsf{flip}(p) \\ \textbf{if } b = 1 \textbf{ then} \\ \quad o := \texttt{Mem}(0) \\ \textbf{else} \\ \quad o := \mathsf{flip}(p) \end{array} \tag{13}$$

It turns out that this does indeed satisfy ϵ-differential privacy.

Theorem 4. *The procedure* RandResp *satisfies* $\ln(3)$*-differential privacy when* $p = 1/2$, $F(\texttt{Mem}) = \texttt{Mem}(0)$, *and* $\texttt{Mem}(0) \in \{0, 1\}$.

Proof. Recall that we need to show that the following inequality holds over all pairs of neighboring inputs and all outputs s:

$$\Pr[\mathsf{RandResp}(X_1) = s] \leq e^{\epsilon} \times \Pr[\mathsf{RandResp}(X_2) = s]$$

Because this instantiation of randomized response only depends on the contents of a single memory cell, i.e. $\texttt{Mem}(0)$, There are two possible configurations of neighboring inputs: $X_1 = \{0 \mapsto 1\}, X_2 = \{0 \mapsto 0\}$ and $X_1 = \{0 \mapsto 0\}, X_2 = \{0 \mapsto 1\}$.

Let's consider the first configuration, for the case where the output is 1. We see that:

$$\begin{aligned} \Pr[\mathsf{RandResp}(\{0 \mapsto 1\}) = 1] &= \Pr[b_1 = 1] + \Pr[b_1 = 0 \wedge \mathsf{flip}(p) = 1] \\ &= p + (1 - p)p \\ &= 2p - p^2 = 3/4 \end{aligned}$$

Now for the right-hand side of the inequality with input X_2, the only way for the program to have output 1 given that $\texttt{Mem}(0) = 0$ would be for the first call to $\mathsf{flip}()$ assigned

to b_1 to have returned 0. Then

$$\begin{aligned}\Pr[\mathsf{RandResp}(\{0 \mapsto 0\}) = 1] &= \Pr[b_1 = 0 \wedge b_2 = 1] \\ &= (1-p)p \\ &= p - p^2 = 1/4\end{aligned}$$

So we have:

$$\frac{\Pr[\mathsf{RandResp}(\{0 \mapsto 1\}) = 1]}{\Pr[\mathsf{RandResp}(\{0 \mapsto 0\}) = 1]} = \frac{3/4}{1/4} = 3$$

Now let's consider the case where the output is 0.

$$\begin{aligned}\Pr[\mathsf{RandResp}(\{0 \mapsto 1\}) = 0] &= \Pr[b_1 = 0 \wedge \mathsf{flip}(p) = 0] \\ &= (1-p)^2 \\ &= 1 - (2p - p^2) = 1/4\end{aligned}$$

And for the other side:

$$\begin{aligned}\Pr[\mathsf{RandResp}(\{0 \mapsto 0\}) = 0] &= \Pr[b_1 = 1] + \Pr[b_1 = 0 \wedge \mathsf{flip}(p) = 0] \\ &= p + (1-p)(1-p) \\ &= 1 - (p - p^2) = 3/4\end{aligned}$$

So then,

$$\frac{\Pr[\mathsf{RandResp}(\{0 \mapsto 1\}) = 0]}{\Pr[\mathsf{RandResp}(\{0 \mapsto 0\}) = 0]} = \frac{1/4}{3/4} = 1/3$$

So for the first configuration,

$$\forall X_1, X_2, s.\mathsf{Neighbor}(X_1, X_2) \rightarrow \Pr[\mathsf{RandResp}(X_1) = s] \leq 3 \times \Pr[\mathsf{RandResp}(X_2) = s]$$

What is the corresponding privacy budget? We have only to solve for ϵ after equating e^ϵ with 3 from the equation immediately above. So $\epsilon = \ln(3)$. Then the theorem holds in this configuration.

Now moving on to the other possible configuration of neighboring inputs, $X_1 = \{0 \mapsto 0\}, X_2 = \{0 \mapsto 1\}$, we see that:

$$\Pr[\mathsf{RandResp}(\{0 \mapsto 0\}) = 1] = \Pr[b_1 = 0 \wedge \mathsf{flip}(p) = 1] = (1-p)p = 1/4$$
$$\Pr[\mathsf{RandResp}(\{0 \mapsto 1\}) = 1] = \Pr[b_1 = 1] + \Pr[b_1 = 0 \wedge \mathsf{flip}(p) = 1] = p + (1-p)p = 3/4$$

So indeed the probabilities are simply inverted in this case, giving us,

$$\frac{\Pr[\mathsf{RandResp}(\{0 \mapsto 0\}) = 1]}{\Pr[\mathsf{RandResp}(\{0 \mapsto 1\}) = 1]} = 1/3$$

It is not difficult to see that for the other outcome,

$$\frac{\Pr[\mathsf{RandResp}(\{0 \mapsto 0\}) = 0]}{\Pr[\mathsf{RandResp}(\{0 \mapsto 1\}) = 0]} = 3$$

Again for this configuration, $\epsilon = \ln(3)$, so the theorem holds in all cases and we conclude that RandResp satisfies $\ln(3)$-differential privacy. □

A good exercise is to generalize Theorem 4 so that p takes an arbitrary value between 0 and 1, letting users tune the privacy budget by setting p. Notice that we used an informal proof even though the primary object of analysis in this theorem was a program. It is possible to prove this theorem more formally, but to do so we would need a formal semantics for the programming language with random elements (e.g., $\mathsf{flip}(p)$), and logic for expressing properties of this language like dynamic logic, and sound proof rules for that logic. Such things exist, and also remain an active area of research, but are beyond the scope of this class.

4 Composing Differentially-Private Computations

There are many algorithms beyond randomized response that have been rigorously shown to satisfy differential privacy. Indeed, we could fill at least one semester-long course covering only a subset of them, and our goals in this class are more broad. If you are interested in this topic, then please consult the papers in the references section of these notes [1, 4, 5], and in particular the text by Cynthia Dwork and Aaron Roth [3], for more on these algorithms.

For the rest of the lecture, we will assume that we are in possession of a algorithm $\alpha_1, \ldots, \alpha_n$ that have already been shown to be differentially private. Our goal is to use them through some composition to implement a larger program that we can by extension show satisfies differential privacy. To support this, we will develop a set of *composition theorems* that allow us to draw such conclusions from the assumption that sub-components satisfy ϵ_i-DP.

Post-processing. The first important property we'll discuss covers post-processing, or computations that are performed that take the result of a differentially-private function as input. Differential privacy enjoys a post-processing guarantee when the post-processor is deterministic, as shown in Theorem 5.

Theorem 5. *Post-processing. Let $\alpha : \mathbf{X} \mapsto \mathbf{O}$ be a randomized ϵ-differentially private function, and $f : \mathbf{O} \mapsto \mathbf{Y}$ be any deterministic function. Then $f \circ \alpha$ is ϵ-differentially private.*

Proof. Let X_1, X_2 be neighboring inputs, and $Y \in \mathbf{Y}$ be any output of f. Let $I \in \mathbf{O}$ be such that $f(I) = Y$. Then,

$$\begin{aligned}\Pr[f(\alpha(X_1)) = Y] &= \Pr[\alpha(X_1) = I] \\ &\leq e^{\epsilon}\Pr[\alpha(X_2) = I] \\ &= e^{\epsilon}\Pr[f(\alpha(X_2)) = Y]\end{aligned}$$

□

The usefulness of the post-processing theorem is apparent: we can always perform deterministic computations over data produced by ϵ-DP computations, and still arrive at ϵ-DP results. Intuitively, the information content of a signal cannot be increased by local deterministic processing. If the input to f contains no information about an individual, then f cannot add any.

Sequential composition. The next type of composition that we'll consider applies a sequence of functions α_i, each of which provide ϵ_i-differential privacy, and releases the union of their results. We can still obtain a privacy guarantee, but the budgets increase additively.

Theorem 6. *(Sequential Composition [5]) Let $\alpha_i : \mathbf{X} \mapsto \mathbf{O}$, $1 \leq i \leq n$ be a sequence of n randomized ϵ_i-differentially private functions, and let $\alpha(X) = (\alpha_1(X), \ldots, \alpha_n(X))$. Then α is $(\sum_{1 \leq i \leq n} \epsilon_i)$-differentially private.*

Proof. Let $O \in \mathbf{O}^n$ be some value in the range of α, and X_1, X_2 be neighboring inputs. Then we can simply calculate:

$$\begin{aligned}
\frac{\Pr[\alpha(X_1) = O]}{\Pr[\alpha(X_2) = O]} &= \frac{\prod_{1 \leq i \leq n} \Pr[\alpha_i(X_1) = O]}{\prod_{1 \leq i \leq n} \Pr[\alpha_i(X_2) = O]} \\
&= \left(\frac{\Pr[\alpha_1(X_1) = O]}{\Pr[\alpha_1(X_2) = O]}\right) \cdots \left(\frac{\Pr[\alpha_n(X_1) = O]}{\Pr[\alpha_n(X_2) = O]}\right) \\
&\leq e^{\epsilon_1} \cdots e^{\epsilon_n} \\
&= e^{\epsilon_1 + \cdots + \epsilon_n}
\end{aligned}$$

□

The sequential composition theorem is crucial for any practical system that hopes to achieve differential privacy. Notably, it implies that for a fixed privacy budget ϵ, it isn't safe to apply a differentially-private computation an arbitrary number of times to the same input X. If the total sum of the computations' budgets exceeds ϵ, then the composed computation is no longer ϵ-differentially private. If we want to ensure a certain level of privacy in a computation composed of multiple queries, then we need to carefully account for the amount of privacy budget that is "consumed" by each query. If the amount ever exceeds our budget, then we can never answer another query from that input.

Parallel composition. The last form of composition that we'll look at is targeted towards the use of multiple differentially-private queries over **disjoint** partitions of X.

Theorem 7. *(Parallel Composition [5]) Let $\alpha_i : \mathbf{X} \mapsto \mathbf{O}$, $1 \leq i \leq n$ be a sequence of n randomized ϵ_i-differentially private functions, $P_1, \ldots, P_n$ be disjoint subsets of X, and let $\alpha(X) = (\alpha_1(P_1), \ldots, \alpha_n(P_n))$. Then α is $(\max_{1 \leq i \leq n} \epsilon_i)$-differentially private.*

Proof. Let $O \in \mathbf{O}^n$ be some value in the range of α, and X_1, X_2 be neighboring inputs. Let $P_{1,1}, \ldots, P_{1,n}$ be the disjoint subsets of X_1, and $P_{2,1}, \ldots, P_{2,n}$ similarly for X_2. Observe that there is only one partition index, i, at which $P_{1,i}$ and $P_{2,i}$ differ. Assume without loss of generality that the index is 1. Then:

$$\begin{aligned}\frac{\Pr[\alpha(X_1) = O]}{\Pr[\alpha(X_2) = O]} &= \frac{\prod_{1 \leq i \leq n} \Pr[\alpha_i(P_{1,i}) = O]}{\prod_{1 \leq i \leq n} \Pr[\alpha_i(P_{2,i}) = O]} \\ &= \frac{\Pr[\alpha_1(X_1) = O]}{\Pr[\alpha_1(X_2) = O]} \\ &\leq \max_{1 \leq i \leq n} \epsilon_i\end{aligned}$$

The second line follows because we assumed that the differing index occured in the first partition, so:

$$\prod_{2 \leq i \leq n} \Pr[\alpha_i(P_{1,i}) = O] = \prod_{2 \leq i \leq n} \Pr[\alpha_i(P_{2,i}) = O]$$

The last line follows under the pessimistic assumption that the largest ϵ_i applies to the first partition. □

Parallel composition is probably not as widely applicable as the sequential form, but can be very useful in certain cases because it does not expend privacy budget additively. Whenever computations can be broken into smaller sub-computations over disjoint data, applying the parallel composition theorem followed by post-processing can lead to strong utility for a fixed privacy budget.

References

[1] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 2011.

[2] C. Dwork. Differential privacy. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2006.

[3] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, Aug. 2014.

[4] Ú. Erlingsson, A. Korolova, and V. Pihur. RAPPOR: randomized aggregatable privacy-preserving ordinal response. *CoRR*, abs/1407.6981, 2014.

[5] F. D. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, 2009.

[6] D. Volpano and G. Smith. Verifying secrets and relative secrecy. In *Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages (POPL)*, 2000.

[7] S. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.