Lecture Notes on Differential Privacy

15-316: Software Foundations of Security & Privacy

Frank Pfenning*

Lecture 22
December 6, 2024

1 Introduction

In Lecture 12 we looked at ways of relaxing noninterference so that we could reason about the information flow security of programs that leak some, but not "too much" information about their secrets. To address this problem in the special context of authorization checks, we discussed a type system due to Volpano and Smith [2000]. They introduced a **match** construct and a corresponding typing rule that we later generalized to an arbitrary declassification construct.

if declassify$(guess = pin)$
then $auth := 1$
else $auth := 0$

For this program, our security policy assigned pin high security level and $guess$ and $auth$ low security level.

We did not formally introduce the notion then, but we can define the *feasible set* as those initial states that lead to the observable outcome.

$$\Omega_\Sigma(\omega, \alpha) = \{\omega' \mid \Sigma \vdash \omega \approx_\mathsf{L} \omega' \text{ and } \Sigma \vdash \mathsf{eval}\ \omega\ \alpha \approx_\mathsf{L} \mathsf{eval}\ \omega'\ \alpha\}$$

The password checker above is acceptable because its feasible set is rather large, say, $2^{64} - 1$ if the outcome is $auth = 0$. Every time we run this program with a new guess we can reduce the size of the feasible set by 1 (even if this temporal evolution isn't part of the formal definition). By contrast, if we replaced equality by **declassify**$(guess \leq pin)$ then we can cut the size of the feasible set in half every

*Almost entirely based on notes by Matt Fredrikson including some edits by Giselle Reis and Ryan Riley time we run the program. As you saw in Lab 2 this can easily be used for an attack on a server to discover the pin.

You can think of the size of feasible set as a quantitative measure of the uncertainty that a user (possibly an attacker) has regarding the high security part of the initial state, once they know the outcome.

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* [Dwork, 2006] 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 [Dwork and Roth, 2014], from machine learning [Chaudhuri et al., 2011] to web browser data collection [Erlingsson et al., 2014]. 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

As discussed above, 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* [Warner, 1965], 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 $\mathbf{flip}()$ is a random function that flips an unbiased coin. In other words,

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

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 $\mathsf{RandResp}$, is as follows, where we assume that the variable out is publicly-observable and b is not (e.g., $\Gamma = x : \mathsf{H}, b : \mathsf{H}, out : \mathsf{L}$).

```
$b := \mathbf{flip}()$
$\mathbf{if}\ b = 1\ \mathbf{then}$
  $out := F(x)$
$\mathbf{else}$
  $out := \mathbf{flip}()$
```

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 out, and if $b = 0$ after being assigned, then out does not depend at all on x, so x could be anything as though the program satisfied non-interference.

But perhaps this does not seem quite right. Let us 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 $\mathsf{RandResp}$ as though it were a function of x that returns the value in out after executing. If $x = 0$, then,

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

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,

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

So we see that $\mathsf{RandResp}$ 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 $\mathsf{RandResp}$ outputs exactly the same value as $F(x)$, and so can go about inferring x by computing feasible sets as before.

This is not 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 $\mathsf{RandResp}$ 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 **flip**() by adding a parameter $0 \le p \le 1$ controlling the bias of the coin.

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

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

```
$b := \textbf{flip}(p)$
if $b = 1$ then
  $out := F(x)$
else
  $out := \textbf{flip}(p)$
```

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

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

When F is the identity function then we have,

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

So if we set $p \ge 1/2$, 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 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 memory m 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 m as perhaps being an 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 $\mathsf{eval}\ (\omega, m)\ \alpha$. The goal is to make sure that the results do not reveal too much information about any single individual's data stored in m.

Definition 1 (ϵ-Differential Privacy) *Let $\epsilon \geq 0$. A program α satisfies ϵ-differential privacy if for all possible memory configurations m_1 and m_2 that differ in* exactly one index*, and all states ω and ν, the following inequality holds:*

$$\Pr[\mathsf{eval}\ (\omega, m_1)\ \alpha = \nu] \leq e^{\epsilon} \cdot \Pr[\mathsf{eval}\ (\omega, m_2)\ \alpha = \nu]$$

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

We write $m_1 \sim_1 m_2$ if m_1 and m_2 differ in one index. 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 f that takes a memory configuration m as input and returns a single discrete value rather than a state. This leads to the following equivalent definition.

Definition 2 (ϵ-Differential Privacy (functional form)) *Let $\epsilon \geq 0$. A function f satisfies ϵ-differential privacy if for all possible inputs m_1 and m_2 with $m_1 \sim_1 m_2$ and all return values s the following inequality holds:*

$$\Pr[f(m_1) = s] \leq e^{\epsilon} \cdot \Pr[f(m_2) = s]$$

The probabilities in this expression are taken over the randomness of f'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 2 is a property of the function f, and *not* of the data being computed on or any particular output of f. 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 2 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 will get into some high-level intuitive interpretations of this definition in a little while, but first let us think about its various components and how they relate to f'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 us look at the behavior of an ϵ-differentially private f for extremal values of ϵ.

Suppose that we make $\epsilon = 0$. Then Definition 2 requires that for any $m_1 \sim_1 m_2$ and outputs s, the stated inequality holds. Notice that the definition is symmetric in the values that m_1 and m_2 take; there is nothing that distinguishes them from each other, so f must also satisfy:

$$\Pr[f(m_2) = s] \leq \Pr[f(m_1) = s]$$

Combining the two inequalities, it must be that $\Pr[f(m_1) = s] = \Pr[f(m_2) = s]$ for all $m_1 \sim_1 m_2$ What does this mean for the privacy of individuals in m_1 and m_2, and the utility of f?

- 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 2 needs to hold for *all* pairs $m_1 \sim_1 m_2$. So, if our actual input is m_1, then all inputs m_2 that we obtain by changing one index in m_1 must produce the same distribution of outputs in f.

- As for utility, you probably guessed that $\epsilon = 0$ is not great. In fact, because f's output distribution needs to remain the same for all adjacent inputs, we can observe that by transitivity f's output distribution needs to remain the same for *all* inputs. In other words, the results cannot contain any information about the input, which clearly means no utility is possible.

On the other hand, when ϵ tends to infinity, then the inequality is automatically satisfied because the right-hand side become arbitrarily large. Therefore, no constraint is imposed on the function and privacy drops off very quickly. This yields maximal utility, because we can have all available data without randomness.

3.1 Interpreting the Definition

Now that we have thought about the definition and some of its technical implications, let us 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 [Dwork, 2006], 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 f 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 2 requires that the likelihood of f 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 f, is in terms of how much individuals' data can influence, or cause changes to, f's response. We have talked about influence before in the context of noninterference, which required that the H-typed parts of the initial state have no influence on the L-typed parts of the final state:

For all ω_1, ω_2, $\Sigma \vdash \omega_1 \approx_{\mathsf{L}} \omega_2$ implies $\Sigma \vdash \mathsf{eval}\ \omega_1\ \alpha \approx_{\mathsf{L}} \mathsf{eval}\ \omega_2\ \alpha$

We might rewrite Definition 2 more concisely as follows.

For all m_1, m_2, $m_1 \sim_1 m_2$ implies $\Pr[f(m_1) = s] \leq e^{\epsilon} \cdot \Pr[f(m_2) = s]$

Notice the similarities between these definitions:

- 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 low-security variables, so that the final state is indistinguishable regardless of the initial high-security variables. For differential privacy, the neighbor relation works similarly by letting one 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 low-security 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 f'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 these definitions, 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 us go back to our example of randomized response. Does it satisfy differential privacy? Let us keep things simple and assume that F is the identity function that just returns the contents of $m[0]$, $p = 1/2$ and all variables and memory cells hold values in the set $\{0, 1\}$. This corresponds to the following program:

```
$f(m) =$
 $b :=$ **flip**$(p)$
 **if** $b = 1$ **then**
  $out := m[0]$
 **else**
  $out :=$ **flip**$(p)$
```

It turns out that this does indeed satisfy ϵ-differential privacy. Normally, we would do our calculation and then find a tight ϵ, but let's anticipate it.

Theorem 3 *f satisfies $\ln(3)$-differential privacy when $p = 1/2$ and $m[0] \in \{0, 1\}$.*

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

$$\Pr[f(m_1) = s] \leq e^\epsilon \cdot \Pr[f(m_2) = s]$$

Because this instantiation of randomized response only depends on the contents of a single memory cell, i.e. $m[0]$, There are two possible configurations of neighboring inputs: $m_1[0] = 1, m_2[0] = 0$ and $m_1[0] = 0, m_2[0] = 1$.

Case: $m_1[0] = 1, m_2[0] = 0$. The inequality has to be satisfied for all s, that is, for $s = 1$ and $s = 0$.

Subcase: $s = 1$. Then we calculate the left-hand side

$$\begin{aligned} \Pr[f(m_1) = 1] \quad &= \Pr[b = 1] + \Pr[b = 0 \wedge \mathbf{flip}(p) = 1] \\ &= p + (1 - p)p = 3/4 \end{aligned}$$

and the right-hand side

$$\begin{aligned} \Pr[f(m_2) = 1] \quad &= \Pr[b = 0 \wedge \mathbf{flip}(p) = 1] \\ &= (1 - p)p = 1/4 \end{aligned}$$

For the inequality to hold we have to have

$$\Pr[f(m_1) = 1] = 3/4 \leq e^\epsilon \cdot 1/4 = e^\epsilon \cdot \Pr[f(m_2) = 1]$$

We see $3 \leq e^\epsilon$ and therefore $\epsilon = \ln(3)$ is a tight bound.

Subcase: $s = 0$. Again, we calculate both sides:

$$\begin{aligned} \Pr[f(m_1) = 0] \quad &= \Pr[b = 0 \wedge \mathbf{flip}(p) = 0] \\ &= (1 - p)^2 = 1/4 \end{aligned}$$

And for the other side:

$$\begin{aligned}\Pr[f(m_2) = 0] &= \Pr[b = 1] + \Pr[b = 0 \wedge \mathbf{flip}(p) = 0] \\ &= p + (1-p)(1-p) = 3/4\end{aligned}$$

Summarizing

$$\Pr[f(m_1) = 0] = 1/4 \leq e^{\epsilon} \cdot 3/4 = e^{\epsilon} \cdot \Pr[f(m_2) = 0]$$

so $1/3 \leq e^{\epsilon}$, which is satisfied for any $\epsilon \geq 0$.

Case: $m_1[0] = 0, m_2[0] = 1$. In the case where $p = 1/2$ this turns out to be symmetric to the previous case.

Subcase: $s = 1$.

$$\begin{aligned}\Pr[f(m_1) = 1] &= \Pr[b = 0 \wedge \mathbf{flip}(p) = 1] = (1-p)p = 1/4 \\ \Pr[f(m_2) = 1] &= \Pr[b = 1] + \Pr[b = 0 \wedge \mathbf{flip}(p) = 1] \\ &= p + (1-p)p = 3/4\end{aligned}$$

Subcase: $s = 0$.

$$\begin{aligned}\Pr[f(m_1) = 0] &= \Pr[b = 1] + \Pr[b = 0 \wedge \mathbf{flip}(p) = 0] \\ &= p + (1-p)p = 3/4 \\ \Pr[f(m_2) = 1] &= \Pr[b = 0 \wedge \mathbf{flip}(p) = 0] \\ &= (1-p)(1-p) = 3/4\end{aligned}$$

So indeed the probabilities are simply inverted in this case

□

Now what would we expect for $p = 1/4$? We go into the branch where we reveal $m[0]$ less frequently, instead returning a random (if biased) answer instead. This would indicate that the privacy budget could be less than $\ln(3)$. Conversely, if $p = 3/4$ we go into the revealing branch with much higher probably, so we would need a higher privacy budget to be allowed to do that.

If we did our math right, there would be four inequalities that should be satisfied for general p:

$$\begin{aligned}&\frac{p + (1-p)p}{(1-p)p} \leq e^{\epsilon} \\ &\frac{(1-p)(1-p)}{p + (1-p)(1-p)} \leq e^{\epsilon} \\ &\frac{(1-p)p}{p + (1-p)p} \leq e^{\epsilon} \\ &\frac{p + (1-p)p}{(1-p)(1-p)} \leq e^{\epsilon}\end{aligned}$$

The middle two fractions are less than 1 and are therefore automatically satisfied.

Plugging in $p = 1/4$, we get $\epsilon \geq \ln(7/3)$, which is indeed less than $\ln(3)$. Plugging in $p = 3/4$, we get $\epsilon \geq \ln(15)$, which is indeed greater than $\ln(3)$. One could also imagine inverting the question: given a certain privacy budget, can we choose a suitable p to make the function comply?

4 Differentially Private Programming

In the preceding section we used an informal proof of adherence to ϵ-differential privacy 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., $\mathbf{flip}(p)$), 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.

Alternatively, we could go the route of information flow and devise a *type system* so that type-checking a program would guarantee that it satisfies ϵ-differential privacy. Perhaps surprisingly, this is also possible! There is a sequence of two supremely elegant papers [Reed and Pierce, 2010, Gaboardi et al., 2013] that lay out such type systems, which also provide useful techniques for composing differentially private computations in various ways.

References

Kamalika Chaudhuri, Claire Monteleoni, and Anand D. Sarawate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12: 1069–1109, 2011.

Cynthia Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages, and Programming, II (ICALP 2006)*, pages 1–12, Venice, Italy, July 2006. Springer LNCS 4052.

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

Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. RAPPOR: Randomized aggregatable privacy-preserving ordinal response. In *Conference on Computer and Communications Security (CCS 2014)*, pages 1054–1067, Scottsdale, Arizana, November 2014. ACM.

Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. Linear dependent types for differential privacy. In Roberto Giacobazzi and Radhia Cousot, editors, *40th Symposium on Principles of Programming Languages (POPL 2013)*, pages 357–370, Rome, Italy, January 2013. ACM.

Jason Reed and Benjamin C. Pierce. Distances makes the types grow stronger: A calculus for differential privacy. In P. Hudak and S. Weirich, editors, *15th International Conference on Functional Programming (ICFP 2010)*, pages 157–168, Baltimore, Maryland, September 2010. ACM.

Dennis M. Volpano and Geoffrey Smith. Verifying secrets and relative secrecy. In M. N. Wegman and T. W. Reps, editors, *Symposium on Principles of Programming Languages*, pages 268–276, Boston, Massachusetts, January 2000. ACM.

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