

RAPPOR in Reality

Akash Jain

Advised by Arvind Narayanan and Nick Feamster

April 29, 2016

Submitted to: Princeton University

Department of Computer Science

*This Thesis represents my own work in accordance with University
Regulations*

Abstract

RAPPOR-JS is a practical tool that democratizes access to differential privacy techniques for the bulk collection of data. It harnesses Google’s recent RAPPOR algorithm and is designed to be used by researchers with no background in privacy or programming. By making certain technical decisions on behalf of researchers, a major hurdle to the widespread adoption of differential privacy techniques in real-world settings has been dramatically lowered.

This paper justifies the design, parameter and privacy choices made in building RAPPOR-JS. It also discusses the circumstances under which this tool can be soundly and reliably used. The final project can be found at <http://rapporjs.com>.

Acknowledgements

I would like to thank my parents for providing me with the best education they possibly could. It was the greatest gift they could have bestowed.

The friends I made throughout Princeton made the journey all the more pleasant. In particular, I built many wonderful memories with the Terrace crew. And also with my girlfriend, Asavari Sinha.

My advisor Arvind Narayanan has been everything one could ask for in mentor: helpful, generous, and enthusiastic. He validated my foolhardy decision to voluntarily write a thesis.

Google and the open source community's work underpins everything in the pages that follow.

And last but by no means least, I deeply appreciate my loyal thesis fairies: Daniel Kim, Catherine Idylle, Michael Kosk (x2!), Daniel Taub, and Grace Lin.

Contents

1	Introduction	6
1.1	Problem Statement	6
1.2	Motivation and Goal	6
2	Background	7
2.1	Measures of Privacy	7
2.2	Randomized Response and Differential Privacy	8
2.3	RAPPOR	10
2.3.1	RAPPOR Client	11
2.3.2	RAPPOR Server	13
2.3.3	RAPPOR Privacy Guarantees	13
2.4	Comparison to My Project	14
3	Approach	15
3.1	Set Up	15
3.2	Threat Model	15
3.3	Web Library	16
3.4	Parameter Experimentation	17
3.4.1	Scenario 1	18
3.4.2	Scenario 2	19
3.4.3	Scenario 3	21
3.4.4	Optimal Parameters	23
3.5	Using ϵ_1 vs ϵ_∞	23
4	Implementation	29
4.1	Website	29

4.2	Client-side Code	29
4.3	Server Code	30
4.4	UI / UX	32
4.4.1	Generating Client Code	32
4.4.2	Analyzing Client Data	33
5	Evaluation	33
5.1	Web Application	33
5.2	Optimizing Parameter Choices	34
5.3	Choosing Privacy Bounds	35
5.4	Timing Results	36
6	Ethics	38
6.1	Using Differential Privacy	38
6.2	Privacy Paradigms	39
7	Discussion and Conclusion	41
7.1	Summary	41
7.2	Limitations	41
7.3	Future work	43
8	Appendix	44
8.1	CSV Files	44
8.2	RAPPOR-JS Screenshots	44

1 Introduction

1.1 Problem Statement

Nowadays, more and more researchers and organizations are collecting data from large sets of users in order to improve their products and services. A consulting firm might collect data from members of the public about their recent taxi journeys. A marketing firm for a new hospital might track users who live nearby across the web and report how long they spend on WebMD. Generally, annual Big Data spending is projected to reach nearly \$50 billion by 2019 by respected digital research firm IDC [1].

But mass collection of data poses its own problems. While such data is undeniably useful to researchers (and businesses), it also poses a privacy threat to consumers. Such data is often sensitive in nature, revealing a wealth of consumers' individual preferences and personal characteristics. Thus it is important to develop methods for analyzing a collection of sensitive data without sacrificing individuals' privacy [2].

Researchers also employ such tools when they want to gather data from a large volume of respondents, whether it's by issuing surveys or collecting data in the background.

In this project, I develop a tool to allow companies and researchers to collect data from large sets of users in a *privacy-preserving* manner. It is designed to be used by entities who have only a passing acquaintance with both data privacy and computer programming.

1.2 Motivation and Goal

A common technique used by organizations collecting data is *anonymization*. Here, user data is collected, and any personally-identifying / sensitive attributes of the data, e.g. name, social security number, etc., are removed. The remaining data attributes are analyzed in aggregate.

But this approach often falls short: there have been many successful demonstrations of *deanonymizing* datasets, e.g. Narayanan's widely cited deanonymization of a Netflix

database of users’ movie preferences [3]. Other researchers have questioned whether deanonymization can even provide meaningful protection at all if it is still to be useful to researchers afterwards [4]. Finally, as a general rule, any collection mechanism which relies on trusting a third party is inherently suboptimal.

These problems and questions have prompted researchers to focus on mechanisms that provide a stronger guarantee of privacy, one of which is *differential privacy*. A key property of differential privacy is to give users a certain level of privacy protection *without requiring them to trust the intentions of the data aggregator*. In particular, Google’s new RAPPOR (Randomized Aggregatable Privacy-Preserving Ordinal Response) mechanism meets this higher standard of privacy preservation. However, so far Google’s algorithm has seen widespread use only within Google’s own Chrome web browser [5].

This paper explains how differential privacy provides users with a strong guarantee of confidentiality, and **provides a simple, plug-and-play web application to allow technically unsophisticated researchers to employ RAPPOR** to collect data from users. It further explores certain properties of RAPPOR’s privacy bounds and analyzes optimal parameter choices in order to remove this burden from researchers.

I apply this algorithm to a scenario of a single researcher wishing to discern many users’ web browsing habits (e.g. which language they use). I also make the whole implementation publicly available for others to adapt to their needs.

2 Background

2.1 Measures of Privacy

Differential Privacy was first proposed by Microsoft Research’s Cynthia Dwork in 2006 [6], and was notable for providing a mathematically quantifiable measure of privacy. The technique describes a promise, made by a data holder to a data subject: “You will not be affected,

adversely or otherwise, by allowing your data to be used in any study or analysis, no matter what other studies, data sets, or information sources, are available.” At best, differentially private database mechanisms make confidential data widely available for accurate data analysis, without resorting to data clean rooms, data usage agreements, or restricted views [7].

For example, consider a health insurance company trying to determine the fraction of its clients that smoke. If the insurer sent out a survey, an individual smoker-client might decline to answer, reasonably worrying that revealing his habit to the insurer might result encourage them to raise his premium. Ideally, in such a situation the insurer could still infer this aggregated statistic without knowing (confidently) whether or not any given customer smokes. Such situations motivated the invention of differentially private collection techniques.

Note that differential privacy is a definition, not an algorithm. For a given computational task T and a given value of ϵ there will be many differentially private algorithms for achieving T in an ϵ -differentially private manner [7].

2.2 Randomized Response and Differential Privacy

Informally, a differentially-private mechanism asks individuals to report data to which they have added carefully-designed noise; this noise ensures that any individual’s information cannot be learned, but an aggregator can accurately infer population statistics [2].

Randomized response is a technique that satisfies this differential privacy guarantee, and is easiest explained with an example. Imagine a researcher wishes to determine the answer to a sensitive question, such as “how many communists live in the US?”. The researcher asks each participant to do the following randomized process [7]:

1. Flip a fair coin (i.e. heads with probability $t = 0.5$)
2. If **heads**, then respond truthfully
3. If **tails**, then flip a second fair coin and respond “Yes” if heads and “No” if tails

Privacy comes from the plausible deniability of any outcome - the researcher does not know with high confidence whether or not any given respondent is a communist. Even if a respondent answered “Yes”, this answer occurs with probability ≥ 0.25 whether or not the respondent truly is a communist.

Accuracy comes from statistically controlling for the noise added (i.e. the spurious “Yes” and “No” responses). In the communist example, if p is the fraction of respondents that truly are communists, and y the fraction of answers that are “Yes”, $\mathbb{E}(y) = \frac{1-p}{4} + \frac{3p}{4} = \frac{1}{4} + \frac{p}{2}$ and thus $\hat{p} = 2y - \frac{1}{2}$, where \hat{p} is an unbiased estimator of p .

To quantify the strength of the differential privacy guarantee, we note the formal definition of differential privacy. A randomized algorithm A is ϵ -differentially private if [8]

$$\mathbb{P}\{A(D) \in S\} \leq e^\epsilon \mathbb{P}\{A(D') \in S\}$$

for all pairs of datasets $D = (x_1, x_2, \dots, x_{n-1}, x_n)$ and $D' = (x_1, x_2, \dots, x_{n-1}, x'_n)$ containing n responses and differing in a single response, and all measurable sets S .

Intuitively, the definition says that no matter what answer a respondent provides, it does not change the researcher’s estimate of the respondent’s ground truth value very much. Consider the hypothesis test for a researcher trying to figure out whether or not the i th respondent is a communist. The likelihood ratio is, where x_i is the i th respondent’s true answer and y_i their reported answer:

$$\frac{\mathbb{P}\{y_i = 1 | x_i = 1\}}{\mathbb{P}\{y_i = 1 | x_i = 0\}} = \frac{(1-t) \times 0.5 + t}{(1-t) \times 0.5} = \frac{1+t}{1-t}$$

For a fair coin, where $t = 0.5$, the likelihood ratio is 3, i.e. if a respondent answers “Yes”, it is 3 times more likely s/he truly is a communist than not. Thus, if we set ϵ to $\ln(3)$, we guarantee differential privacy. We can therefore see ϵ is sensitive to the underlying probability t that respondents answer the communism question truthfully instead of reporting

noise. This gives a natural interpretation of ϵ , seen below graphically:

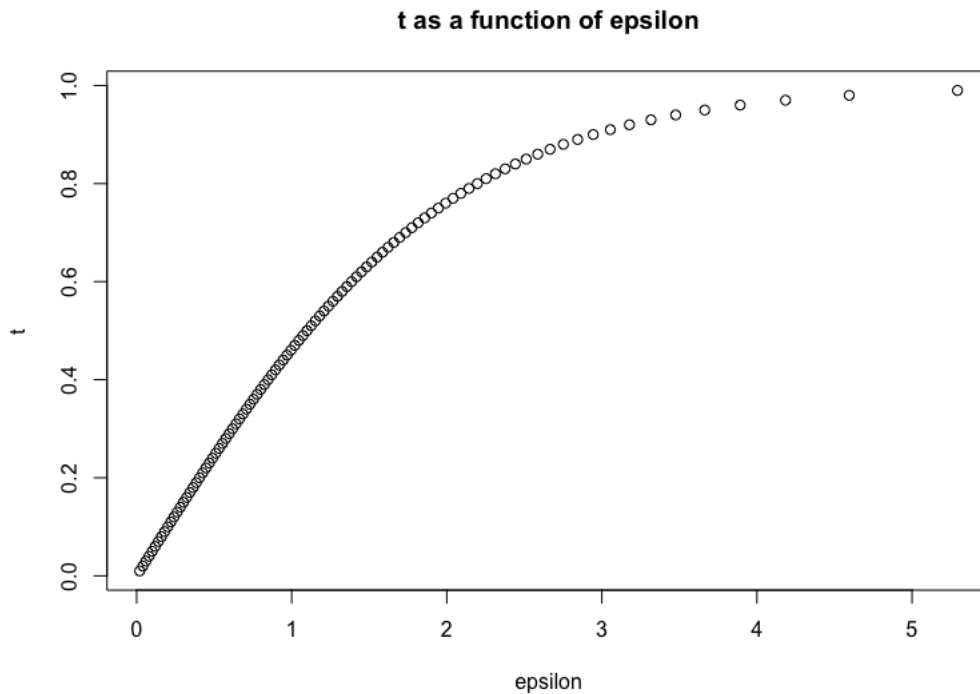


Figure 1: Lower ϵ values imply fewer responses are truthful and more responses are noise [9]

Note that as ϵ decreases, the strength of the privacy guarantee increases at the cost of decreased accuracy of the statistics inferred - more noise is present in the data. This trade-off is referred to in literature as the privacy-utility tradeoff [10].

2.3 RAPPOR

The recently-introduced RAPPOR algorithm is the first differentially private mechanism that has seen real-world deployment [5]. Developed by Google in November 2014, it provides users with a strong and measurable guarantee that their sensitive data will remain confidential. Crucially, it does *not* require users to trust the party they are sending their data to.

RAPPOR is motivated by the problem of multiple clients sending data (typically strings)

to a single server, analogous to several survey participants sending their noisy answers to a single researcher. RAPPOR performs two rounds of randomized response on the client side, to both mask the input and allow collection of data over time, before sending the response to the server. On the server side, RAPPOR aggregates the data and accepts a list of candidate words. It then determines which combination of candidate words best explains the data and returns the estimated marginal distribution of each of the candidate words.

One example of real-world deployment is by Google in their Chrome browser, to determine which pages their users set as their homepage. Each Chrome user sends Google diagnostic information that includes a noisy encoding of their home page, from which Google can infer which web pages make for popular homepages.

2.3.1 RAPPOR Client

Consider the example of Alice sending the string $X = \text{“dog”}$. The outline of RAPPOR on the client side is as follows [5]:

1. **Signal (B).** Hash X onto a Bloom filter of fixed-length k three times (or h times in general) to give a bit array B
2. **Permanent Randomized Response (PRR).** We employ the first layer of randomized response by doing the following to each bit B_i in B :

$$B'_i = \begin{cases} 1, & \text{with probability } f/2 \\ 0, & \text{with probability } f/2 \\ B_i & \text{with probability } 1 - f \end{cases}$$

where f is a user-supplied parameter that controls the level of privacy (intuitively f is the probability of flipping a coin vs responding with the truth). We refer to B' as the

PRR because it will also be memoized and reused for all subsequent values of B .

3. **Instantaneous Randomized Response (IRR).** We now create the report that will be sent to the server. To incorporate the second randomized response, we create a new bit array S , also of length k , and fill it as follows:

$$P(S_i = 1) = \begin{cases} q, & \text{if } B_i = 1 \\ p, & \text{if } B_i = 0 \end{cases}$$

Where p, q are again user-supplied parameters. The smaller the difference between p and q , the stronger the privacy guarantee.

4. **Report.** We send the IRR (and a list of the parameters used to generate it) to the server, which consists of a bitwise representation of the original data to be sent (“dog”) after undergoing two rounds of randomized response.

The following graphic from the Google paper illustrates the process of generating a report:

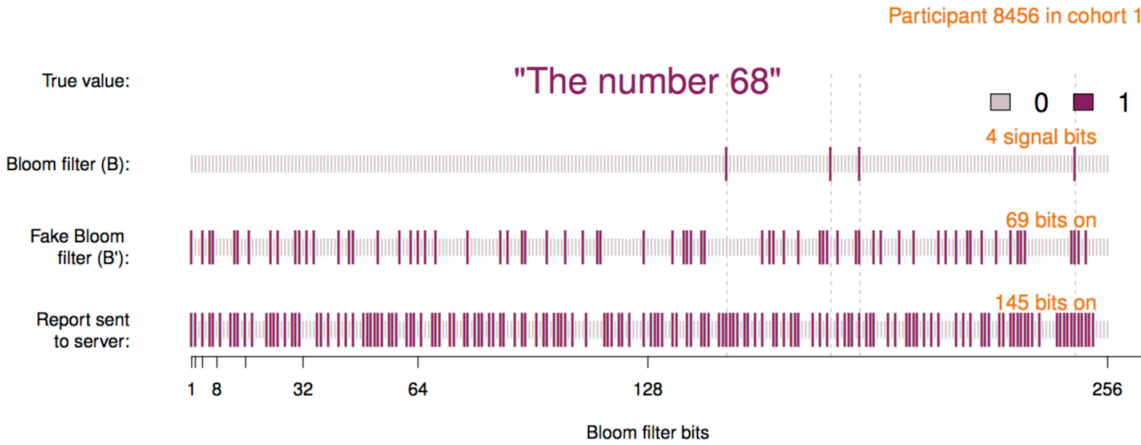


Figure 2: Performing RAPPOR on the string “the number 68” with $k = 128, h = 4$

As these steps show, the RAPPOR encoding scheme satisfies two different ϵ -differential

privacy guarantees: one against a weak adversary who sees only a single IRR, and one against a stronger adversary who sees infinitely many IRRs over time. The stronger adversary can reconstruct B' with arbitrary precision after seeing enough reports, but is unable to reliably infer B from a single copy of B' [2].

2.3.2 RAPPOR Server

On the server side, a sketch of the decoding process is sufficient for our purposes:

1. **Counts.** Construct a new vector Y of length k . For each index i , let Y_i be the sum of all the i th bits of the IRRs
2. **Map.** If there are n candidate strings, construct a matrix X of width k and height n . Each row of X should be the candidate string hashed h times onto a k -bit bloom filter
3. **Selection.** Use a LASSO regression to determine which subset of the candidate strings are relevant in explaining the IRRs
4. **Optimization.** Of the subset of strings chosen by LASSO, find the optimal linear combination of these strings that is closest to the aggregated IRRs.

For a more expansive explanation of the steps involved in reconstructing the marginal distributions, consult the original Google paper [5]. In particular, you can assign each IRR to a specific “cohort” to increase the accuracy of the selection and optimization processes without compromising the privacy guarantee.

2.3.3 RAPPOR Privacy Guarantees

RAPPOR offers two privacy guarantees: one for a report generated after performing the PRR, and one for a report generated after performing both the PRR and IRR [5]. The stronger privacy bound is the latter case, given by:

$$\epsilon_1 = h \log \left(\frac{q^*(1-p^*)}{p^*(1-q^*)} \right)$$

where $q^* = \frac{1}{2}f(p+q) + (1-f)q$ and $p^* = \frac{1}{2}f(p+q) + (1-f)p$. For the full derivation, see Google’s paper [5]. However, ϵ_1 only holds for a single IRR - if an adversary sees enough IRR reports, s/he can try and reconstruct the underlying PRR (e.g. by an averaging attack). Therefore, the privacy bound for the PRR is also important and holds no matter how many IRRs are analyzed:

$$\epsilon_\infty = 2h \ln \left(\frac{2-f}{f} \right)$$

2.4 Comparison to My Project

The value of Google’s contribution to the differential-privacy area of research is considerable. It is the first *practical* demonstration of a protocol that researchers in any field can use to collect data from users in a privacy-strong way.

But as of now, the discovery has had limited public impact. The code that Google has released is self-contained and designed primarily as a proof-of-concept. The client-side code is written in Python and run through a bash script on Linux - not something that an inexperienced researcher could easily use to collect survey data from users. The server-side code is written in R, requires multiple installation and setup steps, and features several bugs - it does not work out-of-the-box. The architecture is optimized for Google’s millions of users, rather than researchers’ thousands of users.

I sought to adapt RAPPOR to a truly plug-and-play solution, easily usable by researchers with very little coding experience or understanding of differential privacy. By **adding a straightforward interface, determining optimal choices of parameters, and rewriting/simplifying the algorithm**, I hope to further contribute to the discussion about the

practical usage of differential privacy.

The finished project RAPPOR-JS can be found at <http://rapporjs.com>. From there, a researcher can set up a study, generate JavaScript code to collect data from users, and analyze the results. More advanced documentation can be found at the project’s GitHub page: <https://github.com/a-jain/rappor-js>. This paper contains a description of my approach in creating RAPPOR-JS, and provides theoretical and empirical justification for the privacy bounds and parameter choices that RAPPOR-JS defaults to.

3 Approach

3.1 Set Up

RAPPOR operates under a model where multiple **users** each send data to a researcher’s single central **server**, which aggregates and analyzes the data.

For this project, I focus on the case where each user is a web user. Here, the researcher collecting data embeds a short snippet of JavaScript on a particular web page and as users land on that web page, it sends data to the server using RAPPOR’s encoding algorithm. This collection can either happen in the background (e.g. the JavaScript tests which browser the user is using and reports it silently), or explicitly with the user’s informed consent (e.g. the researcher sends the results of a web form encoded with RAPPOR). This decision is explored in more depth in Section 6 on Ethics.

3.2 Threat Model

RAPPOR-JS is built according to the following specification: it assumes the researcher using it is *well-intentioned but technically untrained*. Therefore we make few assumptions about the security and integrity of data transfer and storage.

We consider three kinds of adversary: a **basic** adversary that can intercept any single report, a **windowed** adversary that can intercept all reports sent during a specific window in time from a specific user, and a **total** adversary has access to all reports collected by the researcher. RAPPOR provides explicit trade-offs and guarantees against each of these adversaries, as explained further in Section 6 of Google’s paper [5]. In essence, RAPPOR promises users the strongest protection against a basic adversary (the ϵ_1 bound), medium protection against a windowed adversary (something between ϵ_1 and ϵ_∞), and a baseline level against a total adversary (at least ϵ_∞).

An important observation from Google is that the ϵ_∞ privacy bounds only hold when the underlying RAPPOR’d values are either all the same or not temporally correlated. The guarantees are only “slightly violated” if a user’s values change very slowly. But if the values change rapidly and are temporally correlated (e.g. a client reporting their age in days), the privacy guarantees become progressively weaker. One solution could be to increase the noise added to reports exponentially, or to spread a user’s privacy budget over several reports [5].

3.3 Web Library

The accompanying `RapporExamine` library is a proof of concept that shows how researchers can collect data on users’ browsing habits using RAPPOR. It can detect the following features:

1. Which browser is being used
2. Whether an ad blocker is being used
3. Which plugins are installed
4. What the browser’s primary language is
5. If the browser have touch support

I allow researchers to automatically generate the code to detect each feature by providing them with a simple form to fill, as shown in Section 4.4.1. Generating the client code programmatically avoids the problem of a researcher not being technically experienced enough to use the APIs themselves, and reduces the skills gap to a problem of copy and pasting. **Thus the level of technical competency required of a researcher to use RAPPOR-JS is comparable to knowing how to embed a YouTube video in a blog.**

3.4 Parameter Experimentation

Given the constraint of allowing a non-technical researcher to use this tool, it is vital to hide the complexity of choosing optimal parameters from the user. Therefore, some assumptions need to be made about the number of users being collected from, the distribution of data being collected, and the level of privacy protection ϵ desired. We choose 3 typical scenarios of data collection from 10,000 users and empirically compute parameters (number of cohorts m , length of bloom filter k , number of hashes h , PRR parameter f , IRR parameters p, q) for each, **constrained by the privacy bound and optimizing for utility**. For greater information on each of the parameters, refer to Section 2 of the Google paper [5].

I emphasize that this is not an authoritative, statistically robust test but rather an investigation to see which parameters RAPPOR is sensitive to. The approach suffers from, among other factors, overfitting to the tested strings and a lack of statistical robustness. Nevertheless, it is helpful to examine how the choice of parameters affects RAPPOR in the three scenarios below:

1. Researcher collecting whether or not users have ad blockers installed every day for 3 months \implies 2 strings collected, $\epsilon_\infty \leq \ln(3)$
2. Researcher collecting users' language as a one-off \implies 10 strings collected, uniformly distributed, $\epsilon_1 \leq \ln(3)$

3. Researcher collecting custom info monthly over 6 months \implies 5 strings collected, exponentially distributed, $\epsilon_1 \leq \ln(3)$

3.4.1 Scenario 1

We first derive the parameters f, h since they are solely a function of the ϵ_∞ privacy guarantee described in Section 2.3.3. The optimal values under this constraint are $f = 0.87, h = 2$ to give an ϵ_∞ value of $1.046 < \ln(3)$. The tighter ϵ_∞ bound is more appropriate than ϵ_1 , because each user is sending a report 90 times as opposed to just once.

We can trivially set $p = 0, q = 1$ since we are not concerned with privacy afforded by the IRR. We now empirically optimize k, m by sending 5000 `true` reports and 5000 `false` reports, and infer:

k	m	p	q	f	h	min freq	# true	# false	RMSE	key
16	8					600	1184	8815	3815	d33cd8be6e6a
	16					379	546	8121	3845	0c7561c3d4e1
	32	0	1	0.87	2	377	0	4117	3590	eb387343987a
	64					587	7149	0	3848	d80d1d5c05c3
	128					632	3832	1923	2327	62562f621cfb
32	8					600	6045	3594	1045	f4d4a5b8d530
	16					379	2991	7008	2008	b91f7ae8293e
	32	0	1	0.87	2	377	5176	4823	176	e7b064e80328
	64					587	6111	3888	1111	72a7bb49c87b
	128					632	4718	5281	281	c5b1147d7423

Table 1: **Bold** results reflect the best results in each column. We can see that $k = 32$ consistently performs better than $k = 16$. We can also observe that $m \geq 32$ tends to lead to more accurate results (lower RMSE). Note that m , the number of cohorts, is arbitrary.

`Min freq` refers to the minimum number of times a string needs to appear in the aggregated data for RAPPOR to have a $\geq 95\%$ chance of inferring it. `# true / false` refers to RAPPOR’s estimate of the number of times `true` and `false` respectively appear in the aggregated data. `RMSE` refers to the Root Mean Squared Error between the inferred data and the actual true data - it is a measure of accuracy of the inference. `Key` refers to the researcher key used to generate the data - you can go to <http://www.rapporjs.com/generate/<key>> to analyze the same data yourself.

We have modelled the collection for any one day out of the 3 months - the same inference process would be repeated for the other days.

3.4.2 Scenario 2

We send 10 strings, “2test1”, “2test2”, ..., “2test10”, each 1,000 times, and mode the researcher testing the aggregated data for exactly these strings as well. This scenario will test RAPPOR at detecting strings near its detection threshold.

We begin by optimizing the parameters constrained by the looser privacy bound $\epsilon_1 \leq \ln(3)$, since the data is being sent as a one-off. We know from the PRR step that minimizing f decreases the noise sent, and also from the IRR step that minimizing p and maximizing q also decreases the noise sent. Given this, we select a subset of pareto-optimal¹parameter combinations to try, fixing $k = 32$ for convenience given the strong results from *Scenario 1*.

The column labels are the same as under *Scenario 1*, with the addition of `# TPs`, which tests for the number of strings RAPPOR successfully detects (max 10), and two measures of RMSE. RMSE_{TPs} is a measure of accuracy - it characterizes the accuracy of the strings that RAPPOR did detect. RMSE_{all} is a broader measure of error that characterizes the overall performance of RAPPOR, including the strings it failed to detect. Note that there should naturally be an inverse correlation between `# TPs` and RMSE_{all} .

¹No parameter combinations being tested are strictly inferior to others being tested

k	m	p	q	f	h	min freq	# TPs	RMSE _{TPs}	RMSE _{all}	key
32	16	0.1	0.5	0.81	3	600	5	862	933	89b3b93c25ed
		0.1	0.8	0.81	2	379	4	1804	1379	52425d76c235
		0.1	0.9	0.83	2	378	4	1935	1448	8c1455f8c721
		0.1	0.9	0.89	3	587	6	928	957	de20079f1196
		0.4	0.9	0.84	3	632	5	818	913	6b1b325b000c
	32	0.1	0.5	0.81	3	600	6	410	710	46faecf077ec
		0.1	0.8	0.81	2	379	7	740	826	6a65d82c7759
		0.1	0.9	0.83	2	378	7	757	837	c96cecd7da53
		0.1	0.9	0.89	3	587	6	795	883	f2dc7b2d5f8d
		0.4	0.9	0.84	3	632	7	515	697	af5a135176d4
	64	0.1	0.5	0.81	3	600	5	331	744	fdd7acb42508
		0.1	0.8	0.81	2	379	7	532	705	dc7818a942e2
		0.1	0.9	0.83	2	378	7	791	859	348d83dfe06d
		0.1	0.9	0.89	3	587	6	654	810	cf1f3d1772fc
		0.4	0.9	0.84	3	632	8	337	539	fbffa0a3db30
	128	0.1	0.5	0.81	3	600	5	450	775	fdd7acb42508
		0.1	0.8	0.81	2	379	10	468	468	dc7818a942e2
		0.1	0.9	0.83	2	378	7	495	686	348d83dfe06d
		0.1	0.9	0.89	3	587	8	370	556	cf1f3d1772fc
		0.4	0.9	0.84	3	632	8	525	648	fbffa0a3db30

Table 2: $m = 16$ seems consistently worse in terms of RMSEs, while $m = 128$ seems consistently better. Within each ‘ m -block’, it is difficult to know whether or not variations in error should be attributed to chance

3.4.3 Scenario 3

We will send 5 strings, distributed with density $f(x) = 1467e^{\frac{x}{10}}$ for x in $[1, 5]$, giving a total of 10,001 strings labelled as “3test1”, “3test2”, \dots , “3test5”. We test for these 5 strings, as well as 5 other strings - “fake1”, “fake2”, \dots , “fake5”. This models the situation where the researcher is unsure of the universe of possible answers that users have submitted.

We use the ϵ_1 bound despite sending 5 temporally correlated reports from each user because 5 is under the threshold of ≈ 10 that we determine in Section 3.5. We therefore use the same parameters as in *Scenario 2*, and introduce an extra column # FPs that tracks how RAPPOR’s false positive rate - how many “fake” strings RAPPOR incorrectly detected in the aggregate data. The RMSE_{all} column is summed across the 5 “test” strings as well as the 5 “fake” strings.

k	m	p	q	f	h	min freq	# TPs	# FPs	RMSE _{all}	key
32	16	0.1	0.5	0.81	3	600	3	1	1131	c3cbe7f7c7e9
		0.1	0.8	0.81	2	379	5	0	536	39d2b644812a
		0.1	0.9	0.83	2	377	4	0	796	ecc61f50d70b
		0.1	0.9	0.89	3	587	2	2	1436	81d3dadd47a6
		0.4	0.9	0.84	3	632	2	2	1542	d4a71058287a
	32	0.1	0.5	0.81	3	600	3	0	1062	423940018967
		0.1	0.8	0.81	2	379	5	2	511	3c64790eab18
		0.1	0.9	0.83	2	377	4	1	710	0adb9e420a9c
		0.1	0.9	0.89	3	587	2	1	1540	f7b50c2523b7
		0.4	0.9	0.84	3	632	2	1	1556	e835a35ae306
	64	0.1	0.5	0.81	3	600	4	0	721	cb6fcb08a23b
		0.1	0.8	0.81	2	379	5	1	480	bd8666762493
		0.1	0.9	0.83	2	377	5	0	527	c2fcad32d1a4
		0.1	0.9	0.89	3	587	4	1	1030	8541b9409f16
		0.4	0.9	0.84	3	632	4	1	831	9038614ea819
	128	0.1	0.5	0.81	3	600	3	0	901	e045e3505817
		0.1	0.8	0.81	2	379	5	1	511	9dcca1143841
		0.1	0.9	0.83	2	377	5	1	369	1c9da10cfc3c
		0.1	0.9	0.89	3	587	3	0	1270	eba609020953
		0.4	0.9	0.84	3	632	5	1	876	ec67516db87f

Table 3: This table provides more insight. Parameter combinations where $h = 2$ consistently perform better, and in particular the second parameter combination (0.1, 0.8, 0.81, 2) performs consistently well regardless of m

3.4.4 Optimal Parameters

Again with the caveat that this is not intended to be an exhaustive modelling of different parameters, we can still reasonably surmise some patterns from our results. $k = 32$ seems to yield more robust results than $k = 16$, as does $m = 128$ versus $m \in \{16, 32\}$. Better accuracy in combinations where $h = 2$ is the most significant trend within the p, q, f, h variables, especially in Scenario 3 with the testing for “fake” strings. In particular, $h = 2$ as the optimal choice of h is corroborated by Google’s findings in their more general testing [5].

Pattern combination 2 in both Scenarios 2 and 3 consistently did well, leading to its position as the default parameter choice for the client-side RAPPOR implementation. Thus the defaults are $k = 32, m = 128, p = 0.1, q = 0.8, f = 0.81, h = 2$ which provide privacy bounds of $\epsilon_1 = 1.08$ and $\epsilon_\infty = 1.54$.

3.5 Using ϵ_1 vs ϵ_∞

RAPPOR provides two privacy guarantees: one from the first round of randomized response (ϵ_∞) and one from performing both rounds (ϵ_1). The Google paper notes that for just one report sent by a user the privacy guarantee is ϵ_1 , but if an infinite series of reports are sent (e.g. longitudinal collection), then the privacy bound is ϵ_∞ . But for the intermediate cases (e.g. 10 reports being sent), then the paper does not specify how to choose a privacy guarantee to handle this case. The most conservative approach would be to assume ϵ_∞ whenever a user is sending more than one report. But doing so could unnecessarily decrease the utility recoverable from the data. Thus we attempt to quantify: **after how many reports should RAPPOR-JS assume a guarantee of ϵ_∞ is more appropriate than ϵ_1 ?**

Say we are analyzing how N users’ use of ad blockers changes over T days, with daily reporting. Each user reports a string $\in [\text{true}, \text{false}]$, and the inference process leads to an estimate p_i of the proportion of users that are using an ad blocker on the i th day.

We impose the following constraint on the data, known to the researcher as well as any adversary: during the study if a user installs an ad blocker, s/he doesn't uninstall it afterwards. We thus remove the assumption that each report is independent and want to see if an adversary can determine a specific user's ad blocker installation date, or *turning point*. More precisely, if a user reports `[false, false, ..., false, true, true, ..., true]T` we want to determine when `false` \rightarrow `true` from RAPPOR's encoding of these reports.

We assume the user sends T reports using the parameters determined in Section 3.4.4 for a privacy guarantee of $\epsilon_1 = 1.0815 < \log(3)$ for a *single* report. **Given a vector d of T IRRs, we want to know if and when the user installed an ad blocker during the study with 95% certainty.** A straightforward $O(T)$ algorithm to find the maximum likelihood estimate (MLE) of the turning point of a sequence of one string followed by a sequence of another string follows:

1. Iterate through d and for each day's report d_i , determine the probability that d_i encodes either `true` or `false`, given knowledge of the mapping of the underlying string $s_i \in \{\text{true}, \text{false}\}$ to a k -bit Bloom filter (Step 1 in Section 2.3.1). For each d_i , if it is more likely to encode `false`, then label d_i as `false`. Do the same for `true`, but if both `false` and `true` are equally likely then label d_i as `pass`. The probabilities of inferring s_i given d_i , where $h = 2$ and no hash collisions, are:

$$\begin{aligned}\mathbb{P}\{s_i = \text{true} | d_i = \text{IRR}(\text{false})\} &= 2p^*(1-p^*)(1-q^*)^2 + 2(p^*)^2q^*(1-q^*) + (p^*)^2(1-q^*)^2 \\ &= 0.2184\end{aligned}$$

$$\begin{aligned}\mathbb{P}\{s_i = \text{pass} | d_i = \text{IRR}(\text{false})\} &= (1-p^*)^2(1-q^*)^2 + 4p^*(1-p^*)q^*(1-q^*) + (p^*)^2(q^*)^2 \\ &= 0.3643\end{aligned}$$

$$\begin{aligned}\mathbb{P}\{s_i = \text{false} | d_i = \text{IRR}(\text{false})\} &= 2(1-p^*)^2q^*(1-q^*) + 2p^*(1-p^*)(q^*)^2 + (1-p^*)^2(q^*)^2 \\ &= 0.4173\end{aligned}$$

where $p^* = \frac{1}{2}f(p + q) + (1 - f)p = 0.3835$ and $q^* = \frac{1}{2}f(p + q) + (1 - f)q = 0.5165$ are defined in LEMMA 1 of Google’s RAPPOR paper. The above probabilities derive from each of **true** and **false** mapping to $h = 2$ bits in a bloom filter and analyzing how each string’s mapped bits are probabilistically reflected in an IRR generated from it. Note that $\frac{\mathbb{P}\{s_i=\text{false}|d_i=\text{IRR}(\text{false})\}}{\mathbb{P}\{s_i=\text{true}|d_i=\text{IRR}(\text{false})\}} = 1.91 \leq e^{\epsilon_1} = 2.949$.

After this process, our labelled d vector will look something like Figure 3 below:

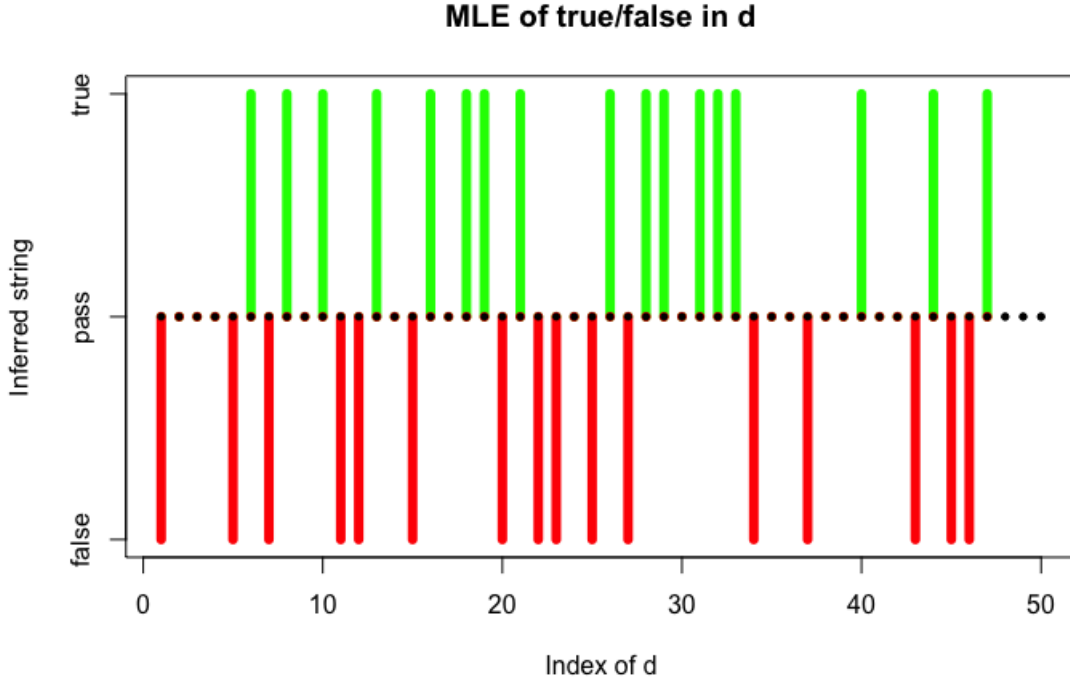


Figure 3: Example of MLE of underlying strings in d over 50 time periods and a true turning point of 25. Note how the first half of the graph shows relatively more **false**s and the second half shows relatively more **true**s

2. Create two new arrays A, A^* of length T , one for the **false** and one for the **true** strings respectively. Let each element of A be 1 if the same element in d is labelled **false**, and 0 otherwise. Let each element of A^* be 1 if the same element in d is labelled **true**, and 0 otherwise.

3. For A, A^* , convert each density array to a cumulative array, e.g. transform $[0, 1, 0, 1, 1, 1, 0]$ to $[0, 1, 1, 2, 3, 4, 4]$. Then, normalize each array by dividing each element in each array by the array's largest element (e.g. transform to $[0, 0.25, 0.25, 0.5, 0.75, 1, 1]$).
4. We now have two arrays A, A^* that represent our normalized estimate of the distribution of **false** and **true** strings in d . Graphically, we can depict them superimposed:

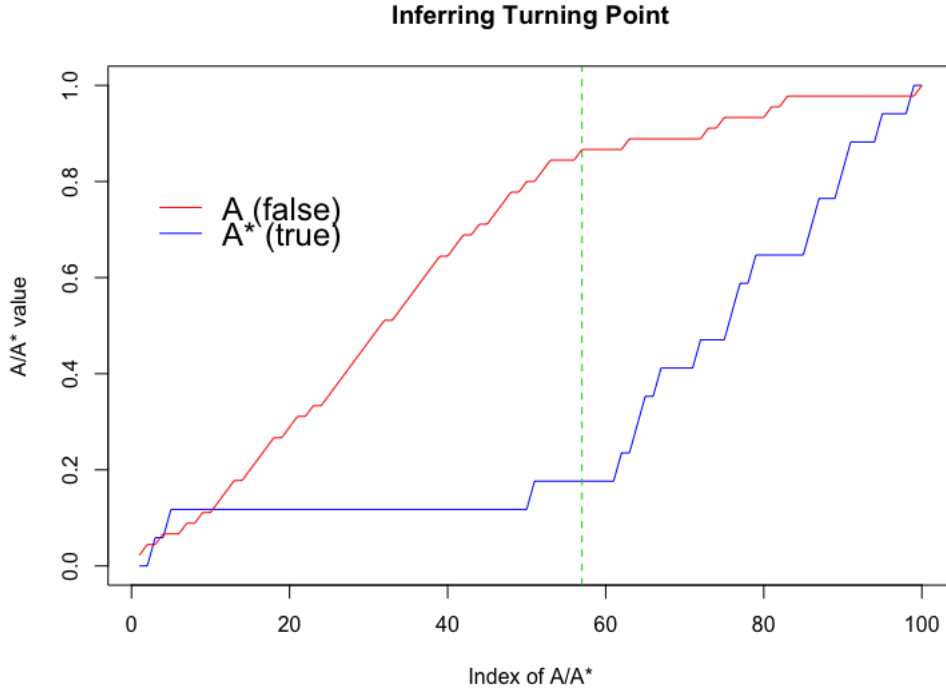


Figure 4: Estimated distribution of **false** and **true** in a sample d of length 100, where $d_k = 50$ is the true turning point and our algorithm has estimated $\hat{d}_k = 56$

5. Now we simply scan through the arrays and find the index for which the difference in A^* and A is the largest. This `max_difference` is our test statistic for determining if there exists a turning point.
6. If `max_difference` is above some threshold, then we return that there exists a turning point at the corresponding index. If not, we return that there is no turning point.

To determine the appropriate threshold, we run a Monte Carlo simulation to empirically determine the values of `max_difference` both where `false -> false` and `false -> true` (after $T/2$ periods) for different values of T , with 5,000 simulations per T value. Our goal is to perform the following test for each value of T :

H_0 : `max_difference` consistent with all underlying `false` strings

H_1 : `max_difference` implies underlying strings changed

For each T , we calculate the the mean and standard deviation of `max_difference` under two scenarios: 1) the underlying string does *not* change (μ_{ff}, σ_{ff}) and 2) the underlying string *does* change (μ_{ft}, σ_{ft}) . For a T value where the difference between the two means is statistically significant (as determined by the Z-Score), a researcher can confidently determine whether a turning point exists, and if it does, what its value is. The Z-Score for each T is:

$$Z = \frac{\mu_{ff} - \mu_{ft}}{\sqrt{\sigma_{ff}^2 + \sigma_{ft}^2}}$$

Thus for each T , if $Z > 1.65$ (we are only doing a one-tailed test), we reject the null hypothesis with confidence 95%. A graphical representation of the results are shown below (full results are in Section 8.1 of the Appendix):

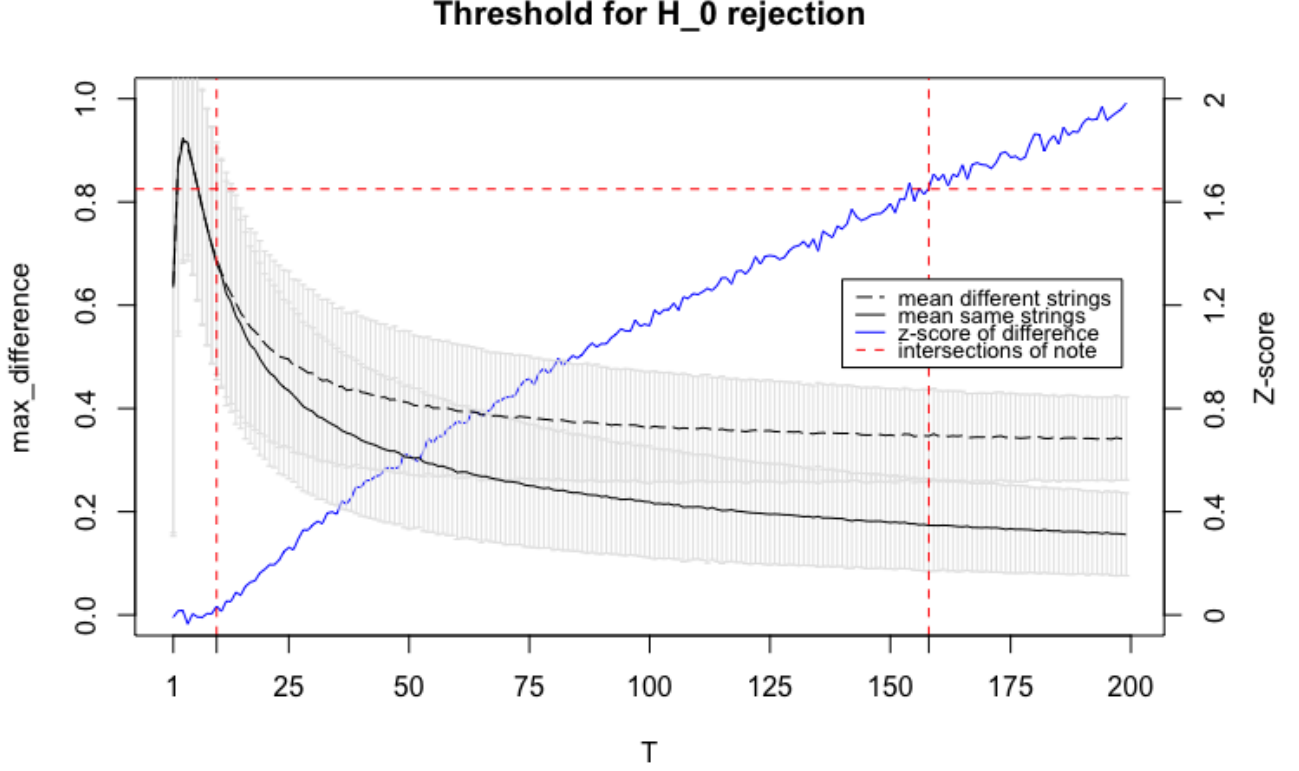


Figure 5: Where $T \geq 10$, in expectation we will infer the underlying strings have changed. Where $T \geq 157$, if we are given a vector with a turning point, we will infer its presence 95% of the time. The shaded regions represents each μ 's error bounds, the blue line is the Z-score of the difference between the μ s, and the red lines show the T values after which we can infer a turning point in expectation and with 95% confidence.

Therefore if a researcher is using these parameters and relying on an ϵ_1 privacy bound for a user sending over 10 reports, then the researcher will be leaking information about respondents that break the differential privacy guarantees. After 157 reports, an adversary can confidently learn that the underlying strings changed with high precision, and users have lost all privacy. Another Monte Carlo simulation reveals that at $T = 157$, if the true turning point is at $T = 78$, an adversary can estimate this with mean 77.47 and standard deviation 12.7.

4 Implementation

4.1 Website

All of a researcher's interaction with RAPPOR-JS happens via `http://rapporjs.com`. Here, the researcher can learn about the RAPPOR algorithm, generate the code to include on a website visited by his subjects ("CLIENT-SIDE CODE"), and analyze the results of their study. After generating the CLIENT-SIDE CODE, the researcher also receives two hyperlinks: one to access the results of the study (at any time) and one to simulate a user sending reports.

By default, researchers can conduct a survey using my own server (specifically by sending data over HTTP requests to `http://rapporjs.com/api/v1/<endpoint>`), but a researcher can specify a different server as well.

RAPPOR-JS itself runs on a Heroku server with a Node.js/Express back-end framework and AngularJS front-end framework and uses a MongoDB database hosted by MongoLab (`http://mlab.com`) to store data from studies. For certain operations more easily completed in Python, the Node environment forks a Python child process. The site uses Bootstrap's CSS system, so responsive design to different screen resolutions is fully incorporated.

4.2 Client-side Code

Here is an example of the code a researcher should include in a website visited by his users to report their zip code at most monthly:

```
% <script src="http://cdn.rapporjs.com/rappor.min.js">
% <script>
%   r = new RAPPOR({publicKey: "a93783fb29cd"});
%   r.send(<zip_code>, {freq: "monthly"});
% </script>
```

To make RAPPOR-JS easy for researchers to integrate into their existing projects, the CLIENT-SIDE CODE has a very simple API that hides most complexity from researchers. A default set of parameters has been already selected (explained in Section 3.4.4), and since the user is sending data to my own server, the code is prefilled with the ‘`publicKey`’ that allows RAPPOR-JS to assign a user’s report to the correct study. The researcher, within their results link, receives a ‘`privateKey`’ that is paired to the public key in the MongoDB database. Finally, the minified/gzipped JavaScript library is 1.6MB hosted on Amazon’s CloudFront CDN to spare the researcher downloading/uploading the library.

Behind the scenes, on `send()` the library does a series of chained asynchronous calls to store the client’s data on the database. It is written in Coffeescript with various CommonJS modules, and tested on Safari, Chrome and Firefox on OS X, and Safari and Chrome on iOS 9.2. The library is based on a Python script Google wrote in their original implementation [5], but adapted for JavaScript and made much more intuitive to use.

More experienced researchers can access other parts of the API more fully, as explained on the accompanying GitHub page. Namely, each function accepts an optional callback function, and the `send()` function accepts another parameter n that specifies how many times a given report should be sent.

4.3 Server Code

To analyze the results of a RAPPOR-JS study, the researcher submits a form on `http://www.rapporjs.com/generate` that contains the strings whose presence s/he wishes to test for in the aggregated data. On submission, s/he is redirected to the analysis app, which displays each candidate string the researcher tested, its estimated frequency in the data with confidence intervals, and many other summary statistics:

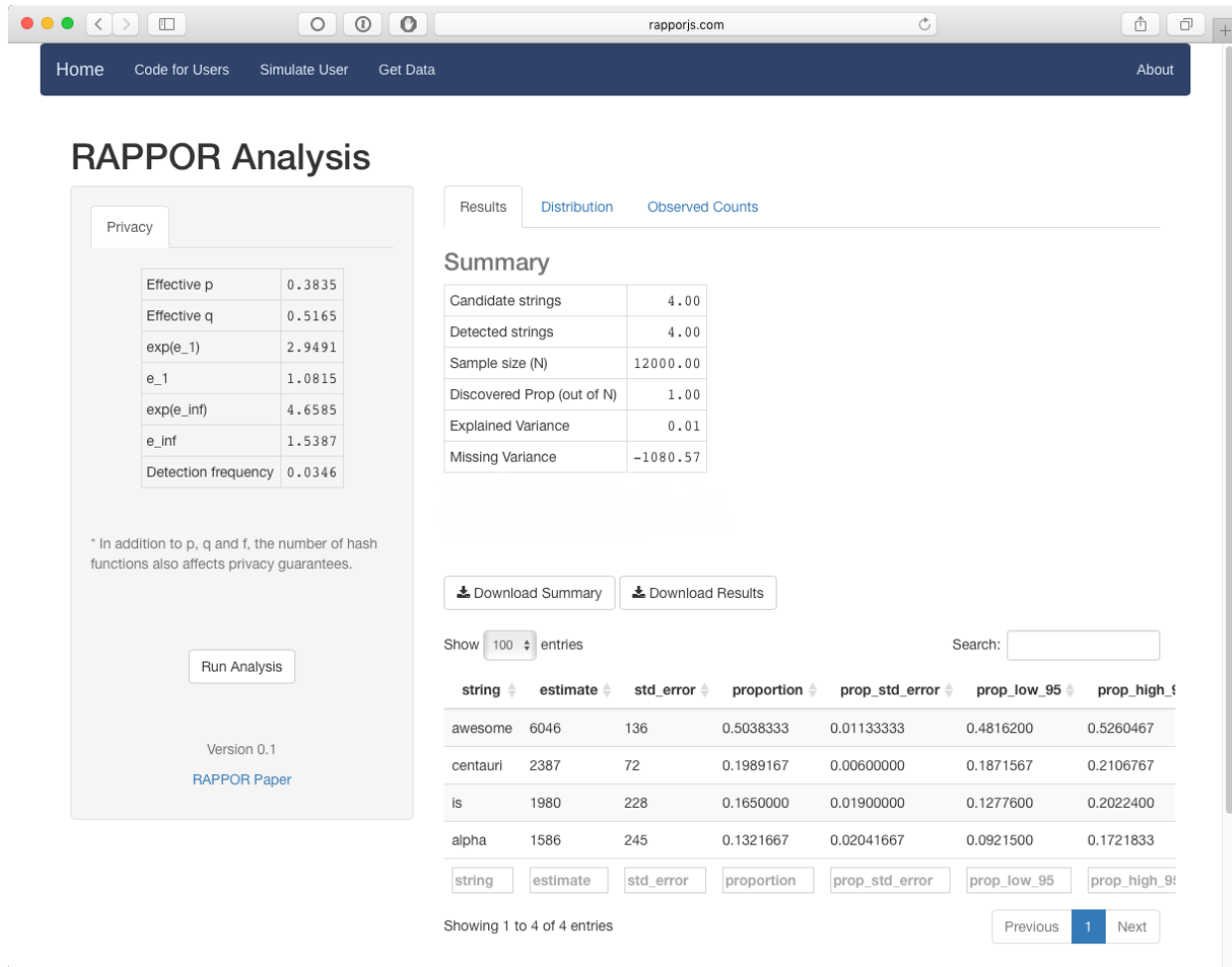


Figure 6: Sample of running analysis app with 2,000 reports of `alpha`, `centauri`, `is` and 6,000 reports of `awesome`

Behind the scenes, the Heroku server forks a Python process to retrieve the data from the MongoDB database and aggregates it into three `csv` files. Then, on the analysis app, R’s Shiny framework sends a GET request to the Heroku server to retrieve the `csv` files from Heroku (using the private key contained in URL) and decodes the RAPPOR’d data into useful aggregate statistics for the researcher in R.

4.4 UI / UX

Fundamental to the goal of making RAPPOR-JS able to be used by non-technically trained researchers is making the site easy to navigate. Too many friction points could frustrate users and discourage them from using the site. To that end, the site uses a simple UI, handles errors gracefully, and reduces unnecessary breaks in the user flow as much as possible. Below, I give sample researcher paths through the website: one for generating client code, and one for analyzing client data. Each step has an accompanying screenshot in the *Appendix*.

4.4.1 Generating Client Code

1. **Home.** The same fonts and palette of colors are used throughout the site for consistency [11]. The homepage details the steps required for using the site, and leaves a (simplified) explanation of the RAPPOR algorithm to the *About* page.
2. **Client-side Code.** This page features three columns to be completed sequentially by the researcher - s/he cannot skip any steps (enforced by buttons that are disabled until the previous step has been completed) - to generate embeddable JavaScript.

The first column gives researchers a choice to select different browser attributes they wish to collect, or alternatively lets them customize whatever they want to send (e.g. the results of a survey they're conducting). I give an example of what each option returns, and provide more details on the APIs for the web library on the GitHub page.

The second column asks the researcher to choose the time period s/he wants to collect over, from `always` to `monthly`.

The third column programmatically generates code that the researcher should embed in whichever page he wants his clients to visit. It also provides the researcher with the two hyperlinks mentioned earlier (to simulate a user and analyze the results). These links are unique to the particular study - no party can access the results without them.

3. **(Optional) Simulate user.** The simulation link generated on the previous page allows the researcher to simulate sending any number of client reports for testing purposes. After the researcher inputs the strings s/he wants to send, a green progress bar provides visual feedback on the progress of the reports being submitted.

4.4.2 Analyzing Client Data

To begin analyzing client data, researchers starts with the links they saved from the *Client-side Code* page. Following that link takes the researchers to the **Generate** page, where they enter the strings they wish to test their study for and the time range to search in. Upon submitting the form, they're taken to the **Analysis** page, where they can see the results of the study - an example was seen earlier in Figure 6.

5 Evaluation

Within the overall goal of creating a simple, plug-and-play application for researchers, there are several smaller subgoals: rewriting existing implementations to make them simpler, porting them to a web environment, determining appropriate privacy bounds, and optimizing choice of parameters. These subgoals have varying evaluation criteria, from a qualitative description of usability to algebraic and numerical proofs.

5.1 Web Application

CLAIM: I've been successful in allowing a nontechnical researcher, with neither technical knowledge nor knowledge of differential privacy, to use my web app.

JUSTIFICATION: Nowhere on the website is the phrase “differential privacy” mentioned, thus it is not necessary to either generate client code or analyze client data. The *Analysis* page does provide more technical information about the privacy bounds used, but an unsophisti-

cated researcher can simply ignore these and concentrate on the metrics s/he understands, e.g. inferred count of each string. The *About* page also provides a simplified explanation of randomized response geared to a technically inexperienced audience.

The basic technical requirements on a researcher are that s/he knows how to embed JavaScript into their website and understands to store their access link for when they want to analyze the data. If a researcher wishes to send a custom string for each client, the researcher must be sufficiently knowledgeable of JavaScript to implement that. Similarly, should a researcher wish to specify custom parameters or servers (or other parts of the full API on GitHub), and this again requires some JavaScript proficiency. I argue this is not a problem because if a researcher is in the position of wanting to change these default options, s/he is likely technical enough to be able to perform simple operations in JavaScript (e.g. creating a JavaScript object). The result for most untrained researchers is a technical knowledge burden at most equivalent to a blogger embedding a YouTube video in their blog - an upper bound I consider acceptable.

With regard to user flow, it takes 4 clicks from the homepage to generate the client-side code, including entering collection and frequency parameters (e.g. collecting browser info daily). It takes 1 click from the analysis link to analyze the aggregated data, plus entering the strings the researcher wants to test the data for. I also consider these bounds to be sufficient for most users not to be put off RAPPOR-JS.

5.2 Optimizing Parameter Choices

Picking the optimal set of parameters is tricky because it entail optimizing across many dimensions. Furthermore, different usage scenarios (e.g. recovering strings uniformly distributed vs exponentially distributed) likely favor different parameter combinations.

I consider the analysis in Section 3.4 to very much be a first step in a more thorough analysis of different parameter choices in RAPPOR. With more time, I would have liked to

set up a more elaborate testing framework to explore more combinations and find statistically significant results.

The effort was somewhat complicated by the fact that much of RAPPOR-JS is not straightforwardly automated - for example, the analysis app relies on manually reading of results from a table, bandwidth limits, etc.. These and other problems are certainly solvable, but required more time than I had available. In the interim, I am reasonably confident that the parameter combinations I selected will perform well enough for most research applications.

5.3 Choosing Privacy Bounds

In Google’s original paper, they leave undetermined the choice of when to use ϵ_1 vs ϵ_∞ , providing guidance only for special cases.

In this project, given the choice of ϵ guarantee should be invisible to researchers, I must decide this choice by proxy while balancing privacy with utility. To determine the appropriate bound, I mimic an adversary launching an attack on the data, and determine the thresholds after which an adversary either uncovers information about an individual either *on average* (10 reports with optimal parameters), or *with 95% significance* (157 reports with optimal parameters). I do not claim that this algorithm is optimal at recovering data, so these values should be taken as a lower bound - they are an attempt to provide a guideline on which bounds are appropriate for different scenarios.

Strictly speaking, RAPPOR’s differential privacy guarantees do not fully hold when reports are temporally correlated (they are weakened “only slightly” as the underlying values change very slowly [5]). However, given that the above scenario is plausible for a researcher scenario, and that many researchers would likely use RAPPOR-JS under these conditions anyway, it is valuable to investigate how robust the privacy guarantees are against temporally correlated reports.

We also note that situations where clients oscillate between only *two* values are the most

vulnerable to attack. For example, if an adversary doesn’t know if any given report is either `false`, `true`, or `dog`, then the probabilities derived in Step 1 of this algorithm would skew away from accurately reconstructing the underlying s_i data. More generally, as latent structure in the reported data decreases, it gets progressively harder for an adversary to reconstruct a user’s ground truth values.

5.4 Timing Results

We briefly note the client-side reporting running time is linear in the number of reports being sent, and in practice is bounded by network latency as opposed to computation time. Database space is also linear in the number of reports; each report takes around around 580 bytes of space (this number can be optimized if space is a major constraint).

For the inference process, we consider the total time required for the operation, from pressing submit on the *Generate* page to having the *Analysis* page finish loading. We also test how long it takes to refresh the *Analysis* page. Both measurements test how long it takes R to retrieve and process those CSV files. The first measurement better reflects real-world usage, and includes time for Python to retrieve data from the MongoDB and to generate aggregated `csv` files from them, (as well as Heroku’s and MongoLab’s own internal network latency). Refer to Section 4.3 for a more detailed explanation of the flow of events during the inference process.

We consider numbers of reports N with values $\{1K, 10K, 20K, 30K, 40K, 80K\}$, where the N reports are spread equally among 10 strings $\{ttest1, ttest2, \dots, ttest10\}$. The timings for ‘Refreshes’ are given from Chrome’s browser development tools, and timings for ‘Total Time’ are done with a physical stopwatch² as changing URLs complicate measuring it in JavaScript. Each measurement was taken 5 times, and the mean and standard deviation are reported. The results follow in Table 5.4.

²Consider all of these results around 0.3s slower than reality to account for my reaction time

N	DB Memory (MB)	Total Time (s)	Refreshes (s)	key
1,000	0.6	7.9 ± 0.3	3.46 ± 0.60	361c130e58c9
10,000	5.8	15.1 ± 0.4	3.29 ± 0.30	817316904504
20,000	11.6	20.9 ± 0.9	3.96 ± 0.66	1dcd3f9562d3
30,000	17.4	31.3 ± 1.6^3	4.43 ± 0.82	6443d8c5c158
40,000	23.2	NA	4.02 ± 0.80	b12d88e2e5ec
80,000	46.4	NA	NA	00e01eb8ac81

Table 4: Format is mean \pm std err. Consider $N = 1,000$ a baseline.

The NAs come from Heroku automatically returning an error when an HTTP request takes over 30 seconds. This bottleneck first occurs when the Python process on Heroku is trying to download 30,000 reports (17.4MB of data) from MongoLab and becomes a serious issue at 40,000 reports. In reality, this places a realistic upper bound for RAPPOR-JS analyzing at most 30,000 at once.

The most immediate conclusion is that it takes Python far longer to download, aggregate and process the data than for R to analyze them - for all N , $(\text{Total Time} - \text{Refreshes}) \gg \text{Refreshes}$. There is also a clear upwards trend (adjusted $R^2 = 0.98$) in Total Time as N grows. However, the relationship is not straightforwardly linear, which is likely a reflection of how nonlinear network latency is dominating the $O(N)$ aggregation algorithm. But even at worst, 30 seconds is an acceptable time limit.

The upwards trend in Refreshes is not statistically significant at the 95% level. Furthermore, the difference in timing is likely dominated by the constant factor and subject to considerable variability in each measurement.

We finally note that these results allow browser caching and come from warm starts,

³One of these was NA

which is crucial for both the Heroku server as well as the Shiny server. The MongoDB process is also shared with other users, so few performance guarantees can be made about these timings more generally.

6 Ethics

6.1 Using Differential Privacy

As explained more fully in Section 7.2, differential privacy is often described as a panacea for data collectors looking to preserve their users’ privacy. But it also relies on many assumptions about the nature of data being collected to provide full privacy protection, some of which may not strictly hold during common research situations (like Scenario 3 in Section 3.4.3). Conveying this information to a potential researcher is tricky - on one hand, flooding them with information about the use-cases of RAPPOR risks overwhelming a nontechnical researcher. On the other hand, providing a conveniently succinct summary leaves out important details and thus may unavoidably result in a serious breach of privacy for a subset of studies. This problem is highlighted as the *transparency paradox* by NYU Professor Helen Nissenbaum, who argues that the whole paradigm of conveying sophisticated probabilistic information to users (in plain-language or otherwise) for consent purposes is broken [12]. Furthermore, our initial goal was to create a solution for researchers that didn’t even have a conception of what differential privacy even is - a very high bar.

In RAPPOR-JS, we skirt this issue by presenting the app as a way for “researchers to collect data anonymously” - a glib placeholder that would need to be more thoughtfully replaced if the tool is to be publicized to a wider audience. One possible solution could be to optionally allow researchers to take a short Yes/No quiz to let them know how appropriate RAPPOR-JS is for them. Questions could be plainly worded, like “Do you have at least 400 respondents for each word you want to test for?”, and the results could allow RAPPOR-JS to customize

privacy bounds for the researcher from their answers.

6.2 Privacy Paradigms

Consider a researcher, Alice, who is correctly⁴ using RAPPOR-JS for a social study. She is a researcher at Princeton investigating child poverty in Trenton and comes up with the following study: she creates a website that allows users to estimate their tax obligation by inputting their income, ZIP code, child benefit payments and rent payments. She advertises this website to residents of Trenton (perhaps by using Facebook’s targeted ad platform), and after users submit the information to her, she secretly sends the information to her RAPPOR-JS study, as well as returning a tax calculation to the user. Alice thinks the study is ethical because she is only collecting RAPPOR’d data without a personally identifiable marker, but is concerned that forcing respondents to consent to this collection will put them off or encourage them to lie on their report (e.g. deterring respondents who don’t pay child support when they ought to). What should Alice do?

There are two big questions here: **1)** is the study ethical in the first place? **2)** If so, should she ask for consent from, or provide notice to, potential respondents?⁵

1) Many frameworks attempt to answer such questions, and in particular three come to mind: the Menlo Report [13], IRBs’ Common Rule [14], and Nissenbaum’s theory of Contextual Integrity [12].

The Menlo Report roughly orients around three tent-poles: a full stakeholder analysis, considerations of the study’s beneficence/harm, and whether the study shows respect for persons, law and public interest. The Common Rule variously provides guidance on each of these tent-poles and when analyzed together they form a reasonably comprehensive review

⁴Such that RAPPOR’s privacy guarantees are not violated

⁵In the interest of brevity, I provide sketches of answers and provide resources for interested readers to learn more.

of the ethical issues at stake. For a good example of how these and other frameworks come together, see Narayanan and Zevenbergen’s case study on Encore [15].

Nissenbaum advocates for considering the broader context in which the data collection is taking place, and for defaulting to the social norms of that broader context. Here, it is reasonable to say that users of an online calculator do not ordinarily expect their sensitive tax data to be sent to a third party without their explicit consent. However, social and legal norms on sharing tax data are so restrictive that informed consent may still be insufficient to allow for ethical sharing of data. Thus Nissenbaum’s framework would likely recommend against conducting such a study regardless of the consent question.

2) In considering whether or not Alice should ask for consent (assuming the study is deemed ethical), it is worth first asking what such consent would look like. Differential privacy is complex topic for those not versed in computer science or statistics, so a full explanation of RAPPOR can be ruled out. Even mentioning that the data will contain ‘no personally identifiable information’ is a half-truth: if an adversary intercepted a respondent’s report, there is still a small positive probability that the respondent’s true answers could be uncovered. But a more complex notice risks amounting to no more than a “charade” to a nonplussed respondent [16].

Even still, there is still some benefit to asking users for consent - if nothing else, it will force researchers to at least consider matters of privacy and make the results of a study more likely to be approved by an IRB [17]. Other scholars also argue that the baseline for *any* study on human subject *should* be to ask for consent. But on other other hand, according to the Common Rule anonymous observations of public activities and textual research are forms of research considered exempt from consent, even if they may reveal sensitive data about persons [15].

There are many, many other consideration that would go into answering the above questions that go beyond the scope of this paper. In general, the ethical norms surrounding bulk collection of data by computer scientists are still undetermined and further discussion is needed [15].

7 Discussion and Conclusion

7.1 Summary

RAPPOR-JS is a useful tool for researchers seeking to collect data in bulk from users in a privacy-preserving manner. It allows researchers with no little expertise in technical and privacy matters to flexibly create a system to collect and analyze data from a web browser.

This project also contributes to the discussion about using differential privacy in real life, and makes empirical estimates of the optimal parameters and privacy guarantees used in RAPPOR to that end. It also provides an attack algorithm researchers can use to test alternative parameter choices against.

7.2 Limitations

Many of the limitations of RAPPOR-JS are inherent to differential privacy itself. To succinctly sum up these problems, before justifiably using RAPPOR-JS a researcher must ask him- or herself the following questions:

1. If an adversary is 80% sure a client sent a particular value, is that level of privacy sufficient for users?
2. Are there enough participating users to analyze with RAPPOR (from Section 3.4.3 there need to be *at least* 379 for each string you desire to recover)?

3. Is it ok if the results have potentially significant and unpredictable noise added to them?
4. Is it ok if not all the strings users sent are recovered, and even some false positives are erroneously detected in the data?
5. Will all strings reported by users have frequency > 379 ?
6. Will users be sending discrete/categorical data?
7. Will users be sending many reports that are not significantly temporally correlated?
8. Do I know the strings I want to test for in advance?

If the answer to *any* of these questions is ‘no’, then RAPPOR-JS is likely unsuited to the researcher’s purposes. The expansiveness of this list of questions is somewhat at odds with prevailing academic literature, which often lauds differential privacy as “the gold standard” for data privacy [18]. Indeed, some scholars have sharply criticized the narrative around differential privacy techniques for exactly this reason: it promises far more than it delivers [19]. They use the academic hubris around techniques like differential privacy as evidence of researchers doing “crypto-for-crypto”, instead of for a practical, real-world application [20]. Even the esteemed Ed Felten is accused of grossly misrepresenting the potential of differential privacy in a blog post for the FTC [21] [19].

Evaluating these competing claims is beyond the scope of this paper, but it suffices to say that the true scope of research applications of differential privacy *in its current form* is limited. But given that RAPPOR is the first such algorithm that has actually seen widespread deployment, there is clearly significant scope for improvement. For example, a recent paper by Giulia Fanti from Google Research has already proposed an algorithm for handling the last question in the list above, about discovering “unknown-unknown” strings in the data [2]. The purpose in mentioning this debate is to highlight that while differential

privacy is no panacea to modern privacy problems, it has the potential to become more widely applicable after more practically-oriented research. In particular, a conclusion from a recent failed attempt to introduce differential privacy into a residential utility reporting system suggests research around the softer, non-technical issues with using differential privacy (e.g. from a legal and policy standpoint) is also needed [22].

With regard to specifically my implementation of RAPPOR - RAPPOR-JS - the chief limitation is that at current, the maximum number of reports that can be analyzed at once is around 30,000 (see Section 5.4). This problem can likely be mitigated by modifying how the API returns data to third parties, but at that point, it is likely there will be other problems related to using free/low tiers of service on many platforms (Heroku, MongoLab, ShinyApps). Procuring more funding should solve these problems!

7.3 Future work

There are various features that would make for compelling additions to RAPPOR-JS: statistically significant choices of parameters, client-side code dynamically adjusting parameters to reports being sent, optimizations for reporting binary strings, etc.. Improving how RAPPOR-JS handles temporally correlated data would be helpful as well, as described in Section 3.2. One solution could be to add noise exponentially to subsequent reports [5]. Finally, recent developments in Google’s RAPPOR algorithm could also be integrated into RAPPOR-JS, e.g. Fanti et al.’s work on determining joint distributions and “unknown-unknown” strings from aggregated data [2].

8 Appendix

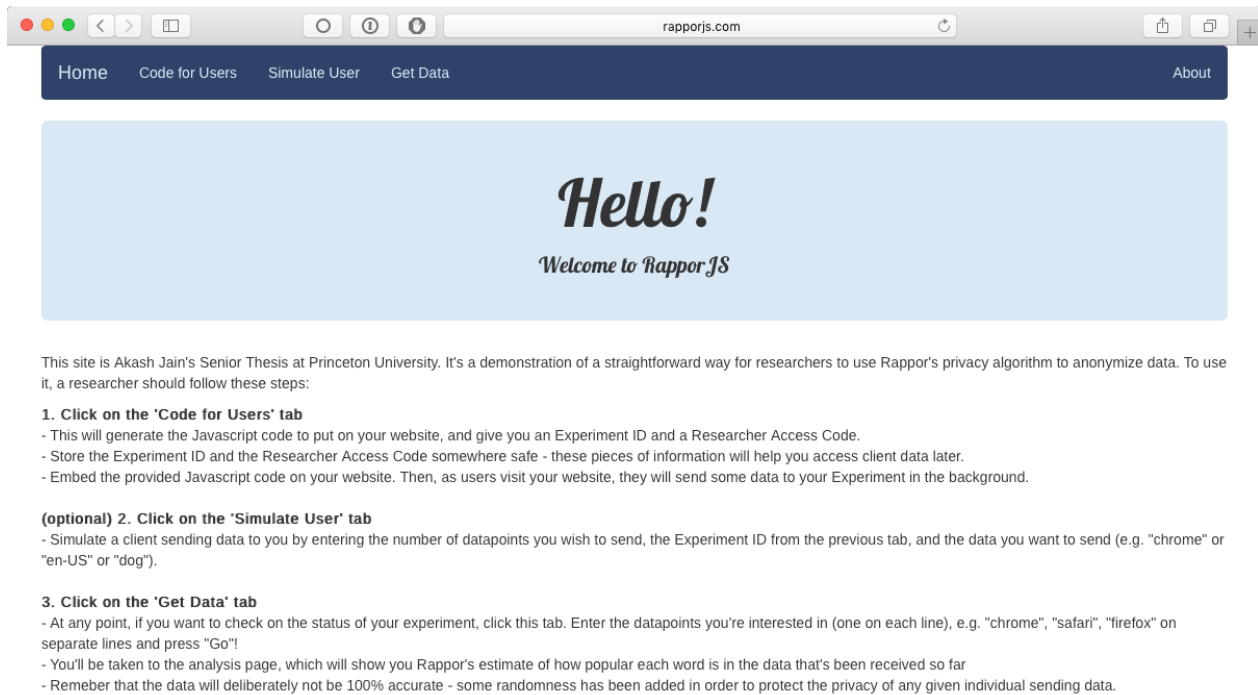
8.1 CSV Files

	diff_mean	diff_dev	same_mean	same_dev	z_score
10	0.691	0.224	0.68	0.226	0.034
20	0.526	0.179	0.482	0.174	0.177
30	0.468	0.162	0.394	0.137	0.349
40	0.432	0.147	0.339	0.116	0.494
50	0.411	0.139	0.305	0.1	0.617
60	0.395	0.129	0.276	0.091	0.75
70	0.383	0.123	0.258	0.083	0.843
80	0.377	0.117	0.242	0.078	0.96
90	0.369	0.111	0.229	0.074	1.05
100	0.363	0.108	0.219	0.07	1.12
110	0.361	0.101	0.21	0.068	1.242
120	0.355	0.1	0.199	0.063	1.321
130	0.354	0.096	0.193	0.061	1.426
140	0.352	0.093	0.187	0.06	1.495
150	0.349	0.09	0.179	0.057	1.594
160	0.347	0.088	0.173	0.054	1.684
170	0.345	0.085	0.169	0.054	1.742
180	0.345	0.082	0.164	0.051	1.864
190	0.342	0.081	0.161	0.051	1.904

8.2 RAPPOR-JS Screenshots

As accessed on April 27 2016.

1. Home page

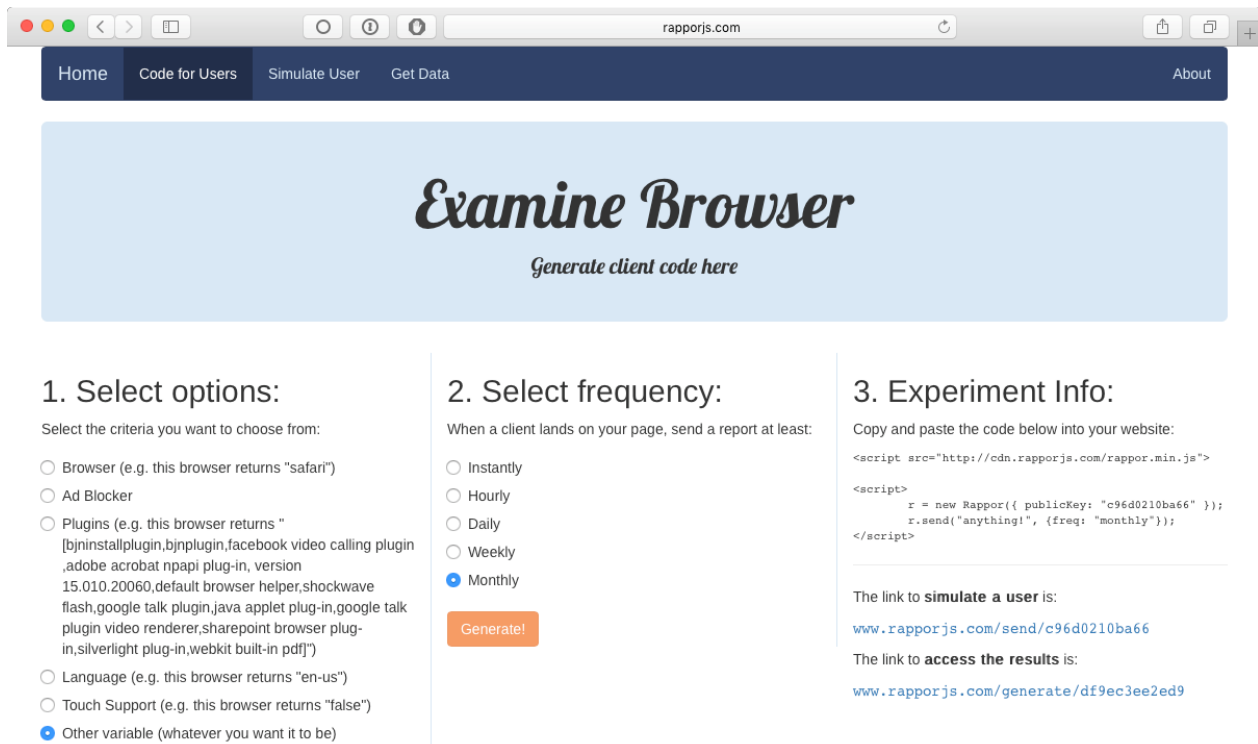


The screenshot shows the RapporJS website's home page. At the top is a dark blue navigation bar with links: Home, Code for Users, Simulate User, Get Data, and About. Below the navigation bar is a large light blue banner with the text "Hello!" in a large, bold, black serif font, and "Welcome to RapporJS" in a smaller, italicized black serif font below it. Underneath the banner, there is a paragraph of text explaining the site's purpose as a demonstration of a privacy algorithm. This is followed by three numbered steps: 1. Click on the 'Code for Users' tab, 2. Click on the 'Simulate User' tab (marked as optional), and 3. Click on the 'Get Data' tab. Each step includes a brief description of what the user will do and what information they will receive or provide.

This site is Akash Jain's Senior Thesis at Princeton University. It's a demonstration of a straightforward way for researchers to use Rappor's privacy algorithm to anonymize data. To use it, a researcher should follow these steps:

- 1. Click on the 'Code for Users' tab**
 - This will generate the Javascript code to put on your website, and give you an Experiment ID and a Researcher Access Code.
 - Store the Experiment ID and the Researcher Access Code somewhere safe - these pieces of information will help you access client data later.
 - Embed the provided Javascript code on your website. Then, as users visit your website, they will send some data to your Experiment in the background.
- (optional) 2. Click on the 'Simulate User' tab**
 - Simulate a client sending data to you by entering the number of datapoints you wish to send, the Experiment ID from the previous tab, and the data you want to send (e.g. "chrome" or "en-US" or "dog").
- 3. Click on the 'Get Data' tab**
 - At any point, if you want to check on the status of your experiment, click this tab. Enter the datapoints you're interested in (one on each line), e.g. "chrome", "safari", "firefox" on separate lines and press "Go"!
 - You'll be taken to the analysis page, which will show you Rappor's estimate of how popular each word is in the data that's been received so far
 - Remember that the data will deliberately not be 100% accurate - some randomness has been added in order to protect the privacy of any given individual sending data.

2. Client-side Code page



The screenshot shows the 'Client-side Code' page of the RapporJS website. The navigation bar is the same as the home page. The main content area has a light blue banner with the text "Examine Browser" in a large, bold, black serif font, and "Generate client code here" in a smaller, italicized black serif font below it. Below the banner, the page is divided into three columns. The first column, titled "1. Select options:", contains a list of criteria to choose from, each with a radio button. The second column, titled "2. Select frequency:", contains a list of frequencies with radio buttons and a "Generate!" button. The third column, titled "3. Experiment Info:", contains instructions on how to copy and paste the generated code, and links to simulate a user and access results.

1. Select options:
Select the criteria you want to choose from:

- ☐ Browser (e.g. this browser returns "safari")
- ☐ Ad Blocker
- ☐ Plugins (e.g. this browser returns "[b]installplugin,bjnpplugin,facebook video calling plugin,adobe acrobat npapi plug-in, version 15.010.20060,default browser helper,shockwave flash,google talk plugin,java applet plug-in,google talk plugin video renderer,sharepoint browser plug-in,silverlight plug-in,webkit built-in pdf[")
- ☐ Language (e.g. this browser returns "en-us")
- ☐ Touch Support (e.g. this browser returns "false")
- ☒ Other variable (whatever you want it to be)

2. Select frequency:
When a client lands on your page, send a report at least:

- ☐ Instantly
- ☐ Hourly
- ☐ Daily
- ☐ Weekly
- ☒ Monthly

[Generate!](#)

3. Experiment Info:
Copy and paste the code below into your website:

```
<script src="http://cdn.rapporjs.com/rappor.min.js">
</script>
<script>
  r = new Rappor({ publicKey: "c96d0210ba66" });
  r.send("anything!", {freq: "monthly"});
</script>
```

The link to **simulate a user** is:
www.rapporjs.com/send/c96d0210ba66

The link to **access the results** is:
www.rapporjs.com/generate/df9ec3ee2ed9

3. Simulate User page

Home Code for Users Simulate User Get Data About

Send a Report

Simulate a client

Simulate Rappor N times, with p probability of sending true (or an optional other string)

N

String

4. Generate CSVs page

Home Code for Users Simulate User Get Data About

Generate CSV files

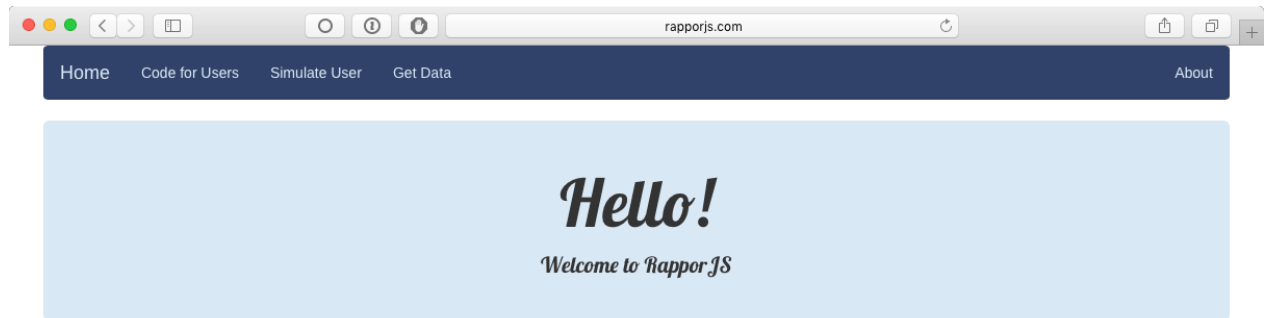
Analyze client data here

Create the files necessary for analyzing client data using Rappor

Candidates

Optional Date Range ☒ Get all reports? **From** **To**

5. Analysis page



This site is Akash Jain's Senior Thesis at Princeton University. It's a demonstration of a straightforward way for researchers to use Rappor's privacy algorithm to anonymize data. To use it, a researcher should follow these steps:

1. Click on the 'Code for Users' tab

- This will generate the Javascript code to put on your website, and give you an Experiment ID and a Researcher Access Code.
- Store the Experiment ID and the Researcher Access Code somewhere safe - these pieces of information will help you access client data later.
- Embed the provided Javascript code on your website. Then, as users visit your website, they will send some data to your Experiment in the background.

(optional) 2. Click on the 'Simulate User' tab

- Simulate a client sending data to you by entering the number of datapoints you wish to send, the Experiment ID from the previous tab, and the data you want to send (e.g. "chrome" or "en-US" or "dog").

3. Click on the 'Get Data' tab

- At any point, if you want to check on the status of your experiment, click this tab. Enter the datapoints you're interested in (one on each line), e.g. "chrome", "safari", "firefox" on separate lines and press "Go"!
- You'll be taken to the analysis page, which will show you Rappor's estimate of how popular each word is in the data that's been received so far
- Remember that the data will deliberately not be 100% accurate - some randomness has been added in order to protect the privacy of any given individual sending data.

References

- [1] Thor Olavsrud. *IDC says big data spending to hit \$48.6 billion in 2019* / CIO. Nov. 2015. URL: <http://www.cio.com/article/3004512/big-data/idc-predicts-big-data-spending-to-reach-48-6-billion-in-2019.html> (visited on 04/29/2016).
- [2] Giulia C. Fanti, Vasyl Pihur, and Úlfar Erlingsson. “Building a RAPPOR with the Unknown: Privacy-Preserving Learning of Associations and Data Dictionaries”. In: *CoRR* abs/1503.01214 (2015). URL: <http://arxiv.org/abs/1503.01214>.
- [3] Arvind Narayanan and Vitaly Shmatikov. “Robust De-anonymization of Large Sparse Datasets”. In: *Proceedings of the 2008 IEEE Symposium on Security and Privacy*. SP ’08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 111–125. ISBN: 978-0-7695-3168-7. DOI: 10.1109/SP.2008.33. URL: <http://dx.doi.org/10.1109/SP.2008.33>.
- [4] Paul Ohm. “Broken promises of privacy: Responding to the surprising failure of anonymization”. In: *UCLA law review* 57 (2010), p. 1701.
- [5] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. “RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response”. In: *Proceedings of the 21st ACM Conference on Computer and Communications Security*. Scottsdale, Arizona, 2014.
- [6] Cynthia Dwork. “Differential Privacy”. In: *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*. Vol. 4052. Lecture Notes in Computer Science. Venice, Italy: Springer Verlag, 2006, pp. 1–12. ISBN: 3-540-35907-9. URL: <http://research.microsoft.com/apps/pubs/default.aspx?id=64346>.
- [7] Cynthia Dwork and Aaron Roth. “The Algorithmic Foundations of Differential Privacy”. In: *Foundations and Trends in Theoretical Computer Science* 9.34 (2014), pp. 211–

407. ISSN: 1551-305X. DOI: 10.1561/04000000042. URL: <http://dx.doi.org/10.1561/04000000042>.
- [8] *Differential privacy and the AUC - An Ergodic Walk*. URL: <https://ergodicity.net/2014/04/21/differential-privacy-and-the-auc/> (visited on 04/28/2016).
 - [9] *differential privacy - An Ergodic Walk*. URL: <https://ergodicity.net/tag/differential-privacy/> (visited on 04/28/2016).
 - [10] Tiancheng Li and Ninghui Li. “On the tradeoff between privacy and utility in data publishing”. In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2009, pp. 517–526.
 - [11] *Corporate Orange & Blue color theme by julievonderropp - Adobe Color CC*. URL: <https://color.adobe.com/Corporate-Orange-Blue-color-theme-121815/> (visited on 04/28/2016).
 - [12] Helen Nissenbaum. “A contextual approach to privacy online”. In: *Daedalus* 140.4 (2011), pp. 32–48.
 - [13] D. Dittrich and E. Kenneally. *The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research*. Tech. rep. U.S. Department of Homeland Security, 2012.
 - [14] *Federal Policy for the Protection of Human Subjects ('Common Rule')*. URL: <http://www.hhs.gov/ohrp/regulations-and-policy/regulations/common-rule/index.html> (visited on 04/29/2016).
 - [15] Arvind Narayanan and Bendert Zevenbergen. “No Encore for Encore? Ethical Questions for Web-Based Censorship Measurement”. In: *Ethical Questions for Web-Based Censorship Measurement (September 24, 2015)* (2015).

- [16] William W Lowrance. *Privacy, confidentiality, and health research*. Vol. 20. Cambridge University Press, 2012.
- [17] Matthew Salganik. *Ethics (DRAFT)*. 2016.
- [18] Justin Hsu et al. “Differential privacy: An economic method for choosing epsilon”. In: *Computer Security Foundations Symposium (CSF), 2014 IEEE 27th*. IEEE. 2014, pp. 398–410.
- [19] Jane Bambauer, Krishnamurty Muralidhar, and Rathindra Sarathy. “Fool’s gold: an illustrated critique of differential privacy”. In: *Vand. J. Ent. & Tech. L.* 16 (2013), p. 701.
- [20] Phillip Rogaway. *The Moral Character of Cryptographic Work*. Tech. rep. IACR-Cryptology ePrint Archive, 2015.
- [21] Ed Felten. *What does it mean to preserve privacy? | Federal Trade Commission*. May 2012. URL: <https://www.ftc.gov/news-events/blogs/techftc/2012/05/what-does-it-mean-preserve-privacy> (visited on 04/29/2016).
- [22] Moritz Hardt. *Towards practicing differential privacy*. Mar. 2015. URL: <http://blog.mrtz.org/2015/03/13/practicing-differential-privacy.html> (visited on 04/29/2016).