Fingerprinting web users through font metrics

David Fifield[1] and Serge Egelman[1,2]

[1] University of California, Berkeley
[2] International Computer Science Institute
{fifield,egelman}@cs.berkeley.edu

Abstract. We describe a web browser fingerprinting technique based on measuring the onscreen dimensions of font glyphs. Font rendering in web browsers is affected by many factors—browser version, what fonts are installed, and hinting and antialiasing settings, to name a few—that are sources of fingerprintable variation in end-user systems. We show that even the relatively crude tool of measuring glyph bounding boxes can yield a strong fingerprint, and is a threat to users' privacy. Through a user experiment involving over 1,000 web browsers and an exhaustive survey of the allocated space of Unicode, we find that font metrics are more diverse than User-Agent strings, uniquely identifying 34% of participants, and putting others into smaller anonymity sets. Fingerprinting is easy and takes only milliseconds. We show that of the over 125,000 code points examined, it suffices to test only 43 in order to account for all the variation seen in our experiment. Font metrics, being orthogonal to many other fingerprinting techniques, can augment and sharpen those other techniques.

We seek ways for privacy-oriented web browsers to reduce the effectiveness of font metric–based fingerprinting, without unduly harming usability. As part of the same user experiment of 1,000 web browsers, we find that whitelisting a set of standard font files has the potential to more than quadruple the size of anonymity sets on average, and reduce the fraction of users with a unique font fingerprint below 10%. We discuss other potential countermeasures.

1 Introduction

Web browser fingerprinting exploits measurable characteristics of browsers to build an identifier that can be used to track the same browser over time. Fingerprinting works even when cookies are disabled, and can be hard for users to defend themselves against. A fingerprint is composed of a variety of measurements of the browser environment, typically acquired through client-side JavaScript. Previous studies have identified many sources of fingerprintable variation, including the User-Agent string, the list of system fonts, and the list of installed browser plugins [8, 5].

In this work, we examine another facet of font-based device fingerprinting, the measurement of individual glyphs. Figure 1 shows how the same character in the same style may be rendered with different bounding boxes in different

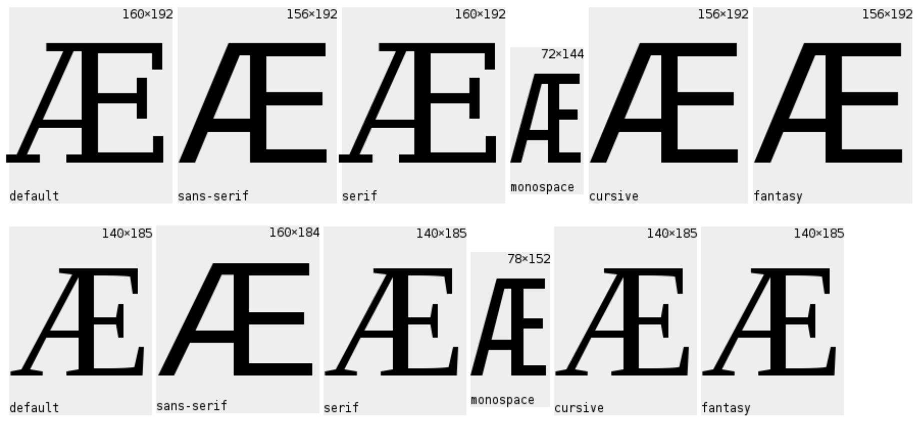

Fig. 1. The Unicode code point U+00C6 LATIN CAPITAL LETTER AE rendered at `font-size: 1000%` in various styles in Firefox 24 (top) and Chromium 35 (bottom). Even when JavaScript is forbidden from reading the pixel data, it can tell the difference between browsers by measuring the dimensions of rendered glyphs. Notice that Firefox has chosen a sans-serif and Chromium a serif font for the CSS cursive and fantasy families. The browsers chose different serif fonts from among those available, and even the same font in the same style appears at different sizes.

browsers. The same effect can serve to distinguish between instances of even the same browser on the same OS, when there are differences in configuration that affect font rendering—and we find that such differences are surprisingly common. By rendering glyphs at a large size, we magnify even small differences so they become detectable. The test is invisible—the glyphs are drawn on the background and never actually appear onscreen—and fast, taking less than a second when the code points tested come from a carefully chosen small set.

At the most basic level, font metrics can tell when there is no installed font with a glyph for a particular code point, by comparing its dimensions to those of a placeholder "glyph not found" glyph. But even further, font metrics can distinguish different fonts, different versions of the same font, different default font sizes, and different rendering settings such as those that govern hinting and antialiasing. Even the "glyph not found" glyph differs across configurations.

Font metric–based fingerprinting is weaker than some other known fingerprinting techniques. For example, it is probably strictly inferior to canvas fingerprinting [16], which gets not only bounding boxes but also pixel data. However, it is relevant because it is as yet effective against Tor Browser, a browser whose threat model includes tracking by fingerprinting [23, Section 4.6], and which already defends against easier, more powerful attacks. For instance, Tor Browser was highlighted in a recent study [4] as being the only browser to resist canvas fingerprinting.

We performed an experiment with more than 1,000 web users that tested the effectiveness of font fingerprinting across more than 125,000 code points of Unicode. 34% of users were uniquely identified; the others were in various anonymity sets of size up to 61. We found that the same fingerprinting power, with this user population, can be achieved by testing only 43 code points. We tested a proposed anti-fingerprinting defense of using standard fonts in the web browser, and found it to be effective, more than quadrupling the size of anonymity sets on average.

2 Related work

Eckersley [8] investigated the potential of fingerprinting in the absence of usual tracking technologies like cookies. The well-known Panopticlick experiment collected hundreds of thousands of submissions and is still ongoing. Fingerprints are derived from a variety of features: User-Agent string, HTTP request headers, whether cookies are enabled, time zone, screen size, browser plugins and their versions, whether certain long-term state storage ("evercookies") are blocked, and the list of system fonts. These limited features uniquely identified 84% of participants. Fingerprints that had changed slightly between visits were found to be nevertheless linkable to previous fingerprints.

Previous studies [8, 5] have considered fingerprinting using the list of installed fonts; that is, a list of names like "Courier" and "Lucida." An ordered list of font names is available from the Java and Flash plugins. Nikiforakis et al. [19] describe how to get an unordered list of font names from JavaScript when Java and Flash are not available. For each of a long list of known font names, render a reference string using that font, and—using the same APIs that we use in this work to measure individual glyphs—compare its rendered dimensions against a database of known fonts. The technique has been known since at least 2007 [20] and was found to be in use by a large fingerprinting company.

Mowery and Shacham [16] found the HTML canvas element [18, Section 4.11.4] to be a rich source of variation. They measured an entropy of 5.73 bits, with 116 unique fingerprints in a population of 294. Their technique is to ask the browser to draw shapes and text to a pixel buffer, and then read back the resulting bitmap. Variations in how browsers draw antialiased lines, for example, are fingerprintable characteristics. They tested font rendering using both system and web fonts, and found, as we do, that the appearance of nominally identical fonts differs across systems. Like us, they recruited users for their experiment from Mechanical Turk and had a similar sample size. Canvas fingerprinting is more powerful than what we describe in this work; however our technique works even when HTML canvas is absent or disabled.

Mowery et al. [15] fingerprinted JavaScript implementations using performance benchmarks. A web browser's JavaScript implementation is an integral part of the browser, and optimization techniques such as just-in-time compilation mean that timing characteristics of even the underlying physical processor may be exposed. They were able to correctly identify a browser family 98% of the time. They also show how the use of a privacy technology, in this case NoScript, can paradoxically make a user more identifiable, by leaking individualized blocking preferences. Mulazzani et al. [17] used the success and failure of standard JavaScript test suites to identify different JavaScript engines.

Acar et al. [5] in 2013 tested the prevalence of fingerprinting in the wild, scanning the top million Alexa sites with a focus on font probing. Their system, FPDetective, found 404 of the top million sites using JavaScript font probing, and discovered some previously unknown commercial fingerprinting providers. They also found fingerprinting scripts that disguised themselves, for example by removing themselves from the DOM after execution.

A further study by Acar et al. [4] in 2014 measured the prevalence of canvas fingerprinting, evercookies, and "cookie syncing" in the wild. They found canvas fingerprinting in use by 5% of the top 100,000 Alexa sites, mostly because of third-party advertisement code. They found instances of evercookies restoring ordinary HTTP cookies and vice versa, and discovered a new evercookie vector used by trackers. They quantified the effect of cookie syncing, the sharing of identifying tokens across domain in circumvention of the same-origin policy.

Previous work has used the measurement of bounding boxes as a means of detecting what fonts are installed, and in turn using the list of installed fonts as a fingerprint feature. The technique we describe in this work is different: its output is not a list of font names, but a list of individual glyph dimensions. It does not require a list of candidate font names known in advance. While it may be possible to infer some characteristics of the target system, such as the list of installed fonts, from a font-metric fingerprint, that is not the main goal. The goal is only to hash as much variation as possible into some kind of unique identifier. Glyph dimensions have the potential to be more sensitive than font names (as the "same" named font may in fact be different on different systems), but they also may miss obscure fonts that are never selected by the browser unless asked for by name. Of course, there is no reason for a tracker to limit itself to one kind of fingerprinting. In this study we consider font metric fingerprinting in isolation, with the understanding that it can be combined with other techniques for better performance.

3 Methodology

We collected measurements through a web page with a JavaScript program that inserts code points into the DOM and measures the dimensions of their corresponding glyphs. The program renders in turn 125,776 Unicode code points over the course of a few minutes. The list consists of every code point in every assigned block of Unicode 7.0.0 [25], with the exception that only the first 256 code points are included from the two Supplementary Private Use Area blocks (U+F0000–U+FFFFF and U+100000–U+10FFFF), which would otherwise contain 65,536 code points each. The code points cover every writing system known to Unicode.

Each code point is drawn six times, once with no font specified (default), then once in each of the five generic CSS families (sans-serif, serif, monospace, cursive, fantasy) [14, Section 15.3.1]. Generic font family names are usually used in a CSS rule to express a rough idea of how text should look, when no specific matching named font is found. These generic CSS family names are mapped to concrete fonts at the browser's discretion, depending in part on user preferences and what fonts are available. Fonts were rendered very large, with CSS style `font-size: 10000%`, in order to better distinguish small differences in dimensions. At this size, typical dimensions for the letter 'A' in the default style are 1155×1900 pixels. The size of each code point is measured indirectly, by placing it in a box and measuring the size of the box. The box is emptied before refreshing the browser UI, so the user does not actually see anything appear onscreen. Thus, fingerprinting can occur without the user's awareness.

We recruited users from Amazon Mechanical Turk and did not impose any restrictions on participation (e.g., geographic region, completion rate, etc.) in order to yield a diverse sample. For each submission, we recorded only the browser's User-Agent string, the elapsed time, and the height and width in pixels of every code point in every font style. Participants were paid \$0.25 each. In order to detect duplicate submissions by the same user, the web page set a cookie with a randomly generated token and a lifetime of 30 days.

Following Eckersley [8], we use entropy as the measure of variation. For a vector of categorical values S, the probability of observing a particular value v is $P_S(v) = \frac{|x \in S : x = v|}{|S|}$; that is, the number of observations of that value divided by the length of the vector. The entropy of S is the sum of the entropies of all the distinct values it comprises:

$$H(S) = -\sum_{v \in S} P_S(v) \log_2 P_S(v).$$

We will be considering the case where the entries of S are the observed dimensions of a certain code point across all experiment submissions. If we think of the data set as a matrix with one row for every user submission and one column for every code point, an individual S is one of the columns.

In order to compute conditional entropy given a set of code points already measured, we will partition the submissions (rows of the matrix) into equivalence sets according to equality in the already-measured code points, so that two submissions are in the same equivalence set if and only if all their corresponding measured code points have the same dimensions. We consider a column of the partitioned matrix not as a single vector, but a set $\mathcal{S}$ of vectors, one for each partition. The entropy of $\mathcal{S}$ is the sum of the entropies of each of its constituent vectors, each scaled by its length.

$$H(\mathcal{S}) = -\sum_{S \in \mathcal{S}} \frac{|S|}{\sum_{T \in \mathcal{S}} |T|} H(S).$$

Such partitions may be further subdivided along additional code points, until all partitions contain elements that are equal in every code point not already measured, at which point the remaining conditional entropy is zero and no further distinctions can be made.

4 Results

We received 1,022 submissions. After removing 6 that had a duplicate cookie, there remained 1,016. The maximum entropy possible, if every submission were distinct, is therefore $\log_2(1016) = 9.989$ bits. Table 1 shows how the submissions broke down with respect to operating system and web browser. Our user sample was drawn from Mechanical Turk users and its composition is not representative of that of the web as a whole.

Table 1. Operating systems and web browsers parsed from User-Agent strings.

523	51%	Windows 7	504	50%	Chrome 36
245	24%	Windows 8	241	24%	Firefox 31
80	8%	other Windows (XP or Vista)	155	15%	Chrome 37
72	7%	OS X 10.9	41	4%	other Chrome (6–35 or 38–39)
40	4%	other OS X (10.5–10.8 or 10.10)	31	3%	other Firefox (9–30 or 32–34)
39	4%	GNU/Linux other than Android	27	3%	Safari (4–8)
10	1%	Android	17	2%	Internet Explorer (9–11)
7	1%	iOS			

Considering 4-tuples (OS, OS version, browser, browser version), there were 94 distinct OS+browser combinations, having an entropy of 4.151 bits. There were 48 (5%) unique combinations, and 28 (3%) were in a set of size 2. The largest set of identical OS+browser combinations, Chrome 36 on Windows 7, contained 281 elements.

The User-Agent string is more variable than OS+browser, as it may contain additional information such as the browser's minor release number. The User-Agent is also useful as a trivial baseline of fingerprintability. Within the input set of 1,016, there were 175 distinct User-Agent strings, having an entropy of 5.148 bits. There were 116 submissions (11%) with a unique User-Agent, and another 42 (4%) that were in a set of size 2. The most common User-Agent appeared 220 times, and was "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.143 Safari/537.36"; i.e., Chrome 36 on 64-bit Windows 7.

Now, on to the fingerprinting attack. There were 444 distinct complete font metric measurements, having an entropy of 7.599 bits. There were 349 submissions (34%) that were identified uniquely by font metrics, and another 84 (8%) that were in a set of size 2. The largest anonymity set contained 61 submissions, 50 of which also shared the most common User-Agent (the others were slight variations: 32-bit Windows instead of 64-bit, or a different micro-release of Chrome). The most common User-Agent appeared 220 times; we observed 46 different font fingerprints for it, 29 of them unique.

Two or more fingerprinting techniques may be combined in order to extract more variation. The combination of font metrics and User-Agent, where two fingerprints are considered equal only if their User-Agents are equal and all their corresponding font metrics are equal, leads to 531 distinct submissions and an entropy of 8.058 bits. 440 of those (43%) are identified uniquely, and another 76 are in a set of size 2. The largest anonymity set contained 51 elements, which happened to be Chrome 36 on Windows 8.1.

Table 2. Code points with the most and least individual entropy.

rank	individual entropy (bits)	code point	name
#1	4.908178	U+20B9	INDIAN RUPEE SIGN
2	4.798824	U+20B8	TENGE SIGN
3	4.698577	U+FBEE	ARABIC LIGATURE YEH WITH HAMZA ABOVE WITH WAW ISOLATED FORM
4	4.698577	U+FBF0	ARABIC LIGATURE YEH WITH HAMZA ABOVE WITH U ISOLATED FORM
5	4.698577	U+FBF2	ARABIC LIGATURE YEH WITH HAMZA ABOVE WITH OE ISOLATED FORM
6	4.698577	U+FBF4	ARABIC LIGATURE YEH WITH HAMZA ABOVE WITH YU ISOLATED FORM
7	4.657576	U+F002	*Private Use Area*
8	4.652798	U+F001	*Private Use Area*
9	4.646632	U+FD3D	ARABIC LIGATURE ALEF WITH FATHATAN ISOLATED FORM
10	4.640043	U+FBF8	ARABIC LIGATURE YEH WITH HAMZA ABOVE WITH E INITIAL FORM
11	4.640043	U+FBFB	ARABIC LIGATURE UIGHUR KIRGHIZ YEH WITH HAMZA ABOVE WITH ALEF MAKSURA INITIAL FORM
⋮	⋮	⋮	
125,766	2.573742	U+202A	LEFT-TO-RIGHT EMBEDDING
125,767	2.573742	U+202B	RIGHT-TO-LEFT EMBEDDING
125,768	2.573742	U+202D	LEFT-TO-RIGHT OVERRIDE
125,769	2.573742	U+202E	RIGHT-TO-LEFT OVERRIDE
125,770	2.481283	U+202C	POP DIRECTIONAL FORMATTING
125,771	2.462760	U+000C	FORM FEED (FF)
125,772	2.462760	U+000D	CARRIAGE RETURN (CR)
125,773	0.156341	U+00AD	SOFT HYPHEN
125,774	0.000000	U+0009	CHARACTER TABULATION
125,775	0.000000	U+000A	LINE FEED (LF)
125,776	0.000000	U+0020	SPACE

Table 2 shows the code points with the greatest and least individual entropy across all submissions. The top of the list includes many code points from the Currency Symbols, Private Use Area, Arabic, and Georgian blocks of Unicode. The Private Use Area block is one in which font designers are free to do what they like; the meanings of the code points is left unspecified. The bottom of the list has mostly whitespace and control characters. Only three code points were identical in every submission, always having a size of 0×0: U+0009 CHARACTER TABULATION, U+000A LINE FEED, and U+0020 SPACE. All three are considered "inter-element whitespace" in HTML [18, Section 3.2.4] and do not count as text when they are the only thing appearing in an element. There are two other inter-element whitespace characters, U+000C FORM FEED and U+000D CARRIAGE RETURN; all submissions had them with zero width (except for one oddball Chrome 36 with a width of 1), and some browsers give them zero height while others give them the line height. The only code point with less entropy than a whitespace character was was U+00AD SOFT HYPHEN, with 0.156 bits, which apparently renders at 0×0 in all browsers but Internet Explorer, where it has nonzero height.

The full suite of 125,776 code points took a mean time to test of 570 seconds with a standard deviation of 394. The shortest test took 70 seconds and the longest 50 minutes. Figure 2 shows the distribution of elapsed times. Part of

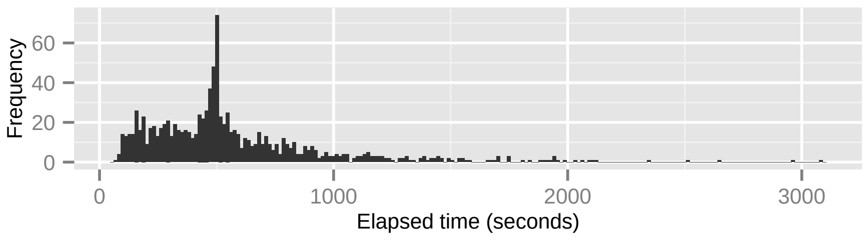

Fig. 2. Time taken to measure all code points in all styles.

the variance is definitely attributable to differing CPU speeds, and disk latency as seldom-used font files are dredged off of disk. The test likely took longer for users who moved the tab it was running in to the background. The spike at around 500 seconds is probably explained by the throttling that browsers apply to timers running in background tabs [24, 27]: the program tests 256 code points in a batch, and if the browser throttles to one batch per second, it takes about $125776/256 = 491.3125$ seconds to test all batches.

Though our data collection experiment took many minutes, fingerprinting requires only milliseconds. We found a subset of 43 code points that suffices to account for all the variation found in the complete set. The reduced subset is shown in Table 3 and a sample fingerprint using it is in Appendix A. We constructed the subset using a greedy algorithm that first selected the code point having the highest individual entropy, then the one with the highest conditional entropy given that the first code point had already been measured, and so on until only uniform, zero-entropy subsets remained.

It is important to remember that entropy measurement is limited by sample size. For example, we measured 5.148 bits of entropy for the User-Agent from our population of 1,016 browsers, while the Panopticlick experiment [8] measured 10.0 bits from a population of about 470,000. We have measured 7.599 bits of entropy in font metric measurements, out of a theoretical maximum 9.989. Before running this experiment, we had done a preliminary test of 496 browsers (under slightly different conditions: Unicode 6.3.0 and `font-size: 2000%`) and measured 7.080 bits of entropy out of a theoretical maximum of 8.954. We expect the entropy to continue to grow, though from the limited sample size it is not possible to say whether or where variability will hit a plateau. Figures 4 and 5 in Section 6 give a rough idea of how entropy may be expected to increase with sample size.

5 Sources of variation

We have seen that the dimensions of individual glyphs can vary widely, and that some code points are more variable than others. Figure 3 compares the variation

Table 3. Code points with the greatest conditional information gain. These 43 code points suffice to capture all the variation of the full set of 125,776. The conditional entropy on each line measures the variation remaining conditioned on the code points on preceding lines already having been measured. Note that the selected code points do not simply appear in order of increasing rank; at each step the algorithm chooses one, the measurement of which gives the most additional information. Slanted type indicates a Unicode block name when a code point is not individually named. There is nothing magic about the set shown here; many others would do just as well. A sample fingerprint using this code point set appears in Appendix A.

rank	individual entropy (bits)	conditional entropy (bits)	code point	name
#1	4.908178	4.908178	U+20B9	INDIAN RUPEE SIGN
190	4.223916	0.843608	U+2581	LOWER ONE EIGHTH BLOCK
18	4.607439	0.496079	U+20BA	TURKISH LIRA SIGN
933	4.008738	0.264101	U+A73D	LATIN SMALL LETTER AY
6,715	3.794592	0.217025	U+FFFD	REPLACEMENT CHARACTER
2	4.798824	0.173474	U+20B8	TENGE SIGN
194	4.215221	0.120687	U+05C6	HEBREW PUNCTUATION NUN HAFUKHA
676	4.063433	0.075592	U+1E9E	LATIN CAPITAL LETTER SHARP S
5,876	3.892304	0.067049	U+097F	DEVANAGARI LETTER BBA
367	4.137402	0.060762	U+F003	*Private Use Area*
100,605	3.440790	0.045069	U+1CDA	VEDIC TONE DOUBLE SVARITA
90,538	3.517391	0.035899	U+17DD	KHMER SIGN ATTHACAN
6,029	3.879878	0.028690	U+23AE	INTEGRAL EXTENSION
7,176	3.763447	0.028359	U+0D02	MALAYALAM SIGN ANUSVARA
62,371	3.549727	0.025836	U+0B82	TAMIL SIGN ANUSVARA
55,549	3.603737	0.022298	U+115A	HANGUL CHOSEONG KIYEOK-TIKEUT
101,598	3.429199	0.020307	U+2425	SYMBOL FOR DELETE FORM TWO
683	4.063107	0.015840	U+302E	HANGUL SINGLE DOT TONE MARK
55,755	3.598234	0.015405	U+A830	NORTH INDIC FRACTION ONE QUARTER
5,872	3.894021	0.014138	U+2B06	UPWARDS BLACK ARROW
122,695	3.894021	0.012554	U+21E4	LEFTWARDS ARROW TO BAR
297	4.163269	0.011433	U+20BD	RUBLE SIGN
806	4.028184	0.010647	U+2C7B	LATIN LETTER SMALL CAPITAL TURNED E
7,967	3.702500	0.010586	U+20B0	GERMAN PENNY SIGN
3	4.698577	0.010389	U+FBEE	ARABIC LIGATURE YEH WITH HAMZA ABOVE WITH WAW ISOLATED FORM
55,358	3.616671	0.007269	U+F810	*Private Use Area*
56,251	3.583220	0.006550	U+FFFF	*Specials*
102,938	3.382354	0.005807	U+007F	DELETE
33	4.593589	0.005638	U+10A0	GEORGIAN CAPITAL LETTER AN
73,091	3.523493	0.005521	U+1D790	MATHEMATICAL SANS-SERIF BOLD ITALIC CAPITAL ALPHA
96,023	3.486238	0.003839	U+0700	SYRIAC END OF PARAGRAPH
99,164	3.449583	0.003839	U+1950	TAI LE LETTER KA
55,116	3.618169	0.003553	U+3095	HIRAGANA LETTER SMALL KA
54,880	3.620506	0.003194	U+532D	*CJK Unified Ideographs*
125,759	2.831178	0.002712	U+061C	ARABIC LETTER MARK
869	4.020008	0.002712	U+20E3	COMBINING ENCLOSING KEYCAP
6,702	3.796600	0.002712	U+FFF9	INTERLINEAR ANNOTATION ANCHOR
7,849	3.708330	0.001969	U+0218	LATIN CAPITAL LETTER S WITH COMMA BELOW
872	4.018562	0.001969	U+058F	ARMENIAN DRAM SIGN
962	4.004011	0.001969	U+08E4	ARABIC CURLY FATHA
99,577	3.445643	0.001969	U+09B3	*Bengali*
55,774	3.596681	0.001969	U+1C50	OL CHIKI DIGIT ZERO
102,439	3.404409	0.001969	U+2619	REVERSED ROTATED FLORAL HEART BULLET
		7.599160	bits total entropy	

observed in two selected code points. There is variation even within the same browser on the same operating system. In this section we explore the causes of these phenomena.

Text rendering is a subtle and complex part of a web browser. Even in the Latin alphabet, layout is more than simply stacking boxes together: considerations such as ligatures, kerning, and combining characters come into play. Some other writing systems are even more complex, causing browsers to rely on OS-provided libraries for text layout. These libraries, including Pango on GNU/Linux, Graphics Device Interface (GDI) or DirectWrite on Windows, and Core Text on Mac OS X, are independent code bases and do not behave identically. Browsers additionally impose their own customizations atop the base text rendering.

The fonts that are installed by default are different on different operating systems. This fact, combined with the differences in layout engines, contribute to a strong per-OS fingerprinting effect. To disguise this effect completely would be difficult, and in Section 6 we assume that OS and browser are inherently fingerprintable, and only seek to reduce further fingerprintability.

Even systems having the "same" named fonts installed may be fingerprintable because they have different revisions of the same font. For example, both Debian 7.6 and Ubuntu 14.04 include the DejaVu fonts, but Debian has version 2.33 of the font and Ubuntu has version 2.34. We found that there are detectable differences in some code points rendered using DejaVu, including some which are listed in the DejaVu changelog [1] as having been added or modified in version 2.34.

Different font rendering settings can distinguish end-user systems. We tracked down one-pixel differences in the width of 134 code points, on two systems that were configured very similarly (Tor Browser 4.0-alpha-1 on Debian "jessie", with the same font files and libraries), to different font hinting settings.

Six of the 43 points selected by our distinguishing algorithm and shown in Table 3 are currency symbols. Here they are shown along with their unconditional entropies and ranks:

rank	individual entropy (bits)	code point	name
#1	4.908178	U+20B9	INDIAN RUPEE SIGN
2	4.798824	U+20B8	TENGE SIGN
18	4.607439	U+20BA	TURKISH LIRA SIGN
297	4.163269	U+20BD	RUBLE SIGN
872	4.018562	U+058F	ARMENIAN DRAM SIGN
7,967	3.702500	U+20B0	GERMAN PENNY SIGN

The code points corresponding to the rupee and tenge signs are the two most entropic overall, and other currency symbols form a hotspot of high entropy. Five of those listed (all but U+20B0) are in the top 1% overall. It may be that relative newness of the glyphs which these code points represent contributes to their variability. The sign for the Kazakhstani tenge was approved by the National Bank of Kazakhstan in 2007 and added to Unicode 5.2 in 2009 [26]. The Indian rupee sign was presented by the Government of India and added to Unicode 6.0

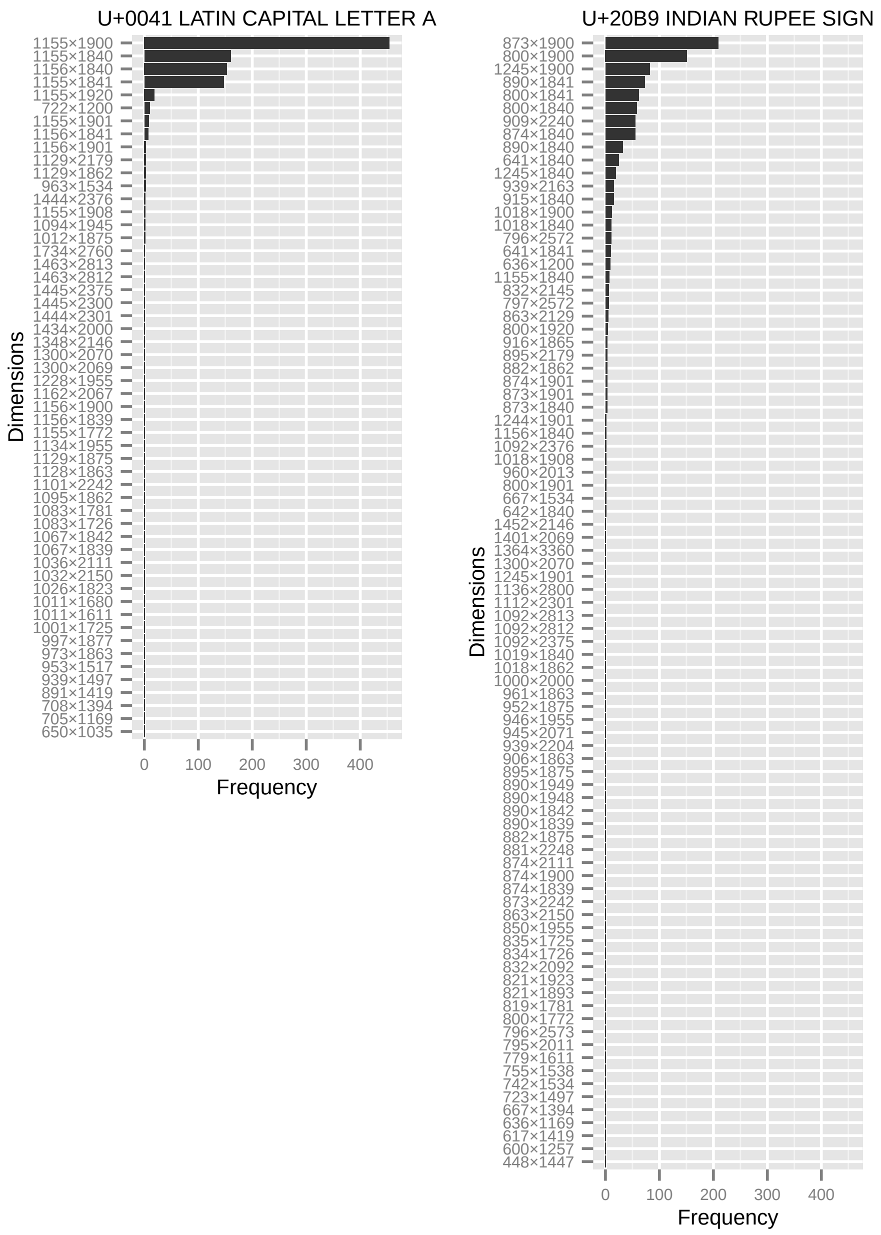

Fig. 3. Frequencies of particular measured dimensions for two selected code points in the default style. Measurements of U+0041 (the letter 'A') had 54 distinct values, 38 of them unique, with an entropy of 2.575 bits. U+20B9 (the currency symbol '₹') took on 87 distinct values, 50 of them unique, with an entropy of 4.288 bits. Note the several occurrences of dimensions that differ by one pixel, for example 1155×1840 and 1156×1840, and a "long tail" of infrequently seen dimensions.

in 2010. The Armenian dram sign was added to Unicode 6.1 in 2012; the Turkish lira sign to Unicode 6.2 in 2012; and the ruble sign to Unicode 7.0 in June, 2014. All these glyphs were newly created symbols, the results of various public design competitions. For comparison, a much older currency symbol, U+20AC EURO SIGN, introduced in Unicode 2.1.2 in 1998, is in the bottom 4% of variability, at rank #123,190 with 3.301 bits.

The Private Use Area block, U+E000–U+F8FF, has high variability. Font designers are free to give their own meaning to code points in this block, so what glyphs are shown depends heavily on what fonts are available.

6 Defenses against fingerprinting

Fingerprinting is made more difficult, in general, by reducing differences across systems; or by making those differences harder to measure.

A simple idea to eliminate variation due to font file availability is to ship a set of standard fonts with the web browser, and use only those (plus downloadable web fonts), at the exclusion of any other fonts that may be installed on the system. This approach has been suggested by Mowery and Schacham [16, Section 5] and on the bug trackers of Mozilla [22] and Tor [21].

We tested this idea: during our data-gathering experiment, in addition to the six generic styles previously mentioned, we tested a style that consisted only of standardized `@font-face` web fonts downloaded from our server. The style included Linux Libertine [13] as an example of an ordinary proportional font, as well as a version of GNU Unifont [6] specially modified to have a glyph for every code point tested, in order to prevent any fallback to system fonts.

The effect on fingerprintability is summarized in Table 4, and in Figures 4 and 5. In our experiment, the defense saved about 2.6 bits, reducing entropy to near the "baseline entropy" of operating system plus browser.

Table 4. Entropy of different variables across the 1,016 submissions. "Standard fonts" uses the simulated defense discussed in Section 6. "OS+browser" is the 4-tuple (OS, OS version, browser, browser version) extracted from the User-Agent string, without any other User-Agent information. Lower numbers are better in all columns except "largest set."

Variable	entropy	# distinct	# unique	largest set
System fonts and User-Agent	8.058 bits	531	440	51
System fonts	7.599 bits	444	349	61
Standard fonts and User-Agent	6.128 bits	270	197	181
User-Agent	5.148 bits	175	116	220
Standard fonts	4.957 bits	150	99	203
OS+browser	4.151 bits	94	48	281

Shipping standard fonts is a promising approach, but also a difficult one. It takes cultural and linguistic understanding to select a set of fonts that will adequately cover the most common writing systems. Font files are large—those cov-

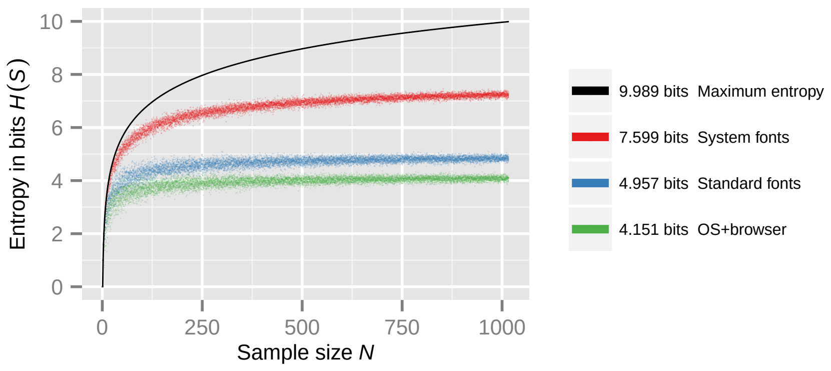

Fig. 4. How entropy in different variables changes for different sample sizes. The black line is $\log_2(N)$, the maximum entropy possible if all browsers were measurably different. For each N, we formed a vector S composed of N individual submissions sampled with replacement from the overall population of 1,016, and computed the entropy of S in different variables. We repeated the sampling ten times for each N. The red line "System fonts" shows the effectiveness of font metric fingerprinting with no countermeasures. The blue line "Standard fonts" shows a simulation of the standard-font defense described in Section 6. The difference between the red and blue lines, $7.599-4.957 = 2.642$ bits, is the reduction in entropy achieved through the simulation of standardized fonts. The green line "OS+browser" is the entropy of only the OS and browser components of the User-Agent, with other information stripped away; it represents a lower bound on entropy achievable if we assume that different OSes and browsers are intrinsically distinguishable by some means. The entropy of User-Agent is not shown but would be slightly above "Standard fonts," 5.148 bits at the right side.

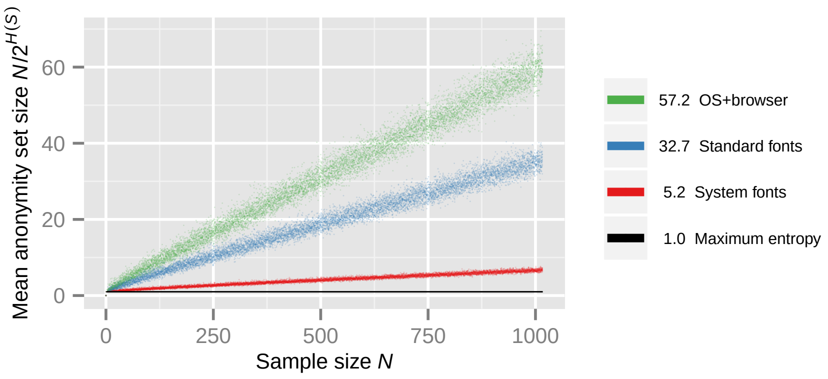

Fig. 5. Another view of the same data. This plot shows $N/2^{H(S)}$, where H is the entropy function. This quantity is the (geometric) mean size of the anonymity set that a random element finds itself a part of. (See Appendix B for a proof.) With maximum entropy, every element would be the sole member of its own anonymity set.

ering east Asian scripts, for example, can be several megabytes uncompressed—adding to the size of downloads. Including fonts with the browser requires the browser developer to assume ongoing maintenance and expertise costs previously held by the operating system developer.

The attack as we have designed it relies on client-side execution of JavaScript. Simply disabling JavaScript is unlikely to be an effective defense, however. Heiderich et al. [11] show how to use CSS to measure the size of DOM elements, for example by shrinking a container through animation and causing its contents to reflow.

Tor Browser already imposes a limit on the number of fonts that can be used to render a document, in order to defend against font enumeration attacks. Unfortunately this defense is ineffective against the attack we have described, because the attack uses only generic font names.

Randomizing the size of onscreen glyphs, or just randomizing the sizes reported to JavaScript, would frustrate fingerprinting. One would need to take care not to allow the randomization to be simply averaged away, and keep in mind that a browser's randomizing its dimensions is itself a detectable feature. FireGloves [2] was a proof-of-concept fingerprint-resistant browser extension for Firefox that randomized the reported size of DOM objects, among other things. FPBlock [12] proposes to track data that depends on HTML element elements, and prevent its exfiltration through means such as XMLHttpRequest.

Using a standardized operating system such as Tails [3] is an effective way to blend in with other users of the same operating system.

7 Future work

We hope to collaborate with the maintainers of Tor Browser to develop and deploy a patch along the lines of the standard-font defense described in Section 6. The Tor Browser maintainers have indicated a willingness to work with us and a ticket tracks development progress [9]. Tor Browser is a good target for deployment of a defense, because it already defends against other, more direct and powerful attacks that are still effective in other browsers, even in private browsing mode.

Canvas fingerprinting could be strengthened using the information gain–based selection technique we have used to refine the set of code points tested. Rather than testing only the 26 letters of the English alphabet, canvas fingerprinting could test carefully selected code points from Unicode.

Our technique could perhaps be strengthened by testing more than one code point at a time, using combinations designed to reveal differences in the handling of ligatures, kerning, combining characters, right-to-left text, and other font features. Font technologies such as OpenType [10] support a large number of features that are being made available to CSS [7].

8 Conclusion

We have presented and evaluated a new web fingerprinting attack based on measuring the onscreen size of font glyphs. We conducted a user experiment to test nearly the entire repertoire of Unicode in various CSS font styles, and then developed a narrow set of code points that can quickly and effectively fingerprint web users. We simulated a standard-font defense against fingerprinting. Font metric–based fingerprinting can supplement other techniques in order to increase their effectiveness.

9 Source code

Source code for the web experiment and analysis programs is available in a Git repository at https://repo.eecs.berkeley.edu/git-anon/users/fifield/fontfp.git.

10 Acknowledgments

We thank Mike Perry for suggesting the idea of testing what code points lack font coverage as a means of fingerprinting, and for guidance during development of the test code; Gunes Acar for extensive conversation on this technique and fingerprinting in general; Georg Koppen for comments on a draft of this paper and on the history of font measurement; Alex Kantchelian for advice regarding information gain measurements; Kamil Jozwiak, Benjamin Smedberg, and John Daggett for help regarding fonts in Firefox; and the tor-assistants mailing list for help testing Tor Browser.

References

1. DejaVu fonts full changelog (version 2.34), http://dejavu-fonts.org/wiki/Full_changelog
2. FireGloves, http://fingerprint.pet-portal.eu/?menu=6
3. Tails, https://tails.boum.org/
4. Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., Diaz, C.: The Web never forgets: Persistent tracking mechanisms in the wild. In: Proceedings of the 21st ACM conference on Computer and Communications Security (CCS 2014) (November 2014), https://securehomes.esat.kuleuven.be/~gacar/persistent/the_web_never_forgets.pdf
5. Acar, G., Juarez, M., Nikiforakis, N., Diaz, C., Gürses, S., Piessens, F., Preneel, B.: FPDetective: Dusting the web for fingerprinters. In: Proceedings of the 20th ACM conference on Computer and Communications Security (CCS 2013) (November 2013), https://www.cosic.esat.kuleuven.be/publications/article-2334.pdf
6. Czyborra, R.: GNU Unifont, http://unifoundry.com/unifont.html
7. Daggett, J.: CSS fonts module level 3. Candidate recommendation, W3C (Oct 2013), http://www.w3.org/TR/2013/CR-css-fonts-3-20131003/

8. Eckersley, P.: How unique is your web browser? In: Proceedings of the 10th Privacy Enhancing Technologies Symposium. pp. 1–18 (July 2010), https://panopticlick.eff.org/browser-uniqueness.pdf
9. Fifield, D.: #13313: Enable bundled fonts in Tor Browser (Oct 2014), https://trac.torproject.org/projects/tor/ticket/13313
10. FontShop International: OpenType user guide (Apr 2012), https://www.fontfont.com/staticcontent/downloads/FF_OT_User_Guide.pdf
11. Heiderich, M., Niemietz, M., Schuster, F., Holz, T., Schwenk, J.: Scriptless attacks: Stealing the pie without touching the sill. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. pp. 760–771. CCS 2012, ACM, New York, NY, USA (2012), http://www.nds.rub.de/media/emma/veroeffentlichungen/2012/08/16/scriptlessAttacks-ccs2012.pdf
12. Kim, D.: Detection and prevention of web-based device fingerprinting (May 2014), http://www.cs.utexas.edu/~dkim/papers/webfingerprint-poster_sp14.pdf, poster presented at the 2014 IEEE Symposium on Security and Privacy
13. Libertine Open Fonts Project: Linux Libertine, http://www.linuxlibertine.org/
14. Lie, H.W., Çelik, T., Bos, B., Hickson, I.: Cascading style sheets level 2 revision 1 (CSS 2.1) specification. W3C recommendation, W3C (Jun 2011), http://www.w3.org/TR/2011/REC-CSS2-20110607
15. Mowery, K., Bogenreif, D., Yilek, S., Shacham, H.: Fingerprinting information in JavaScript implementations. In: Wang, H. (ed.) Proceedings of W2SP 2011. IEEE Computer Society (May 2011), https://cseweb.ucsd.edu/~hovav/dist/jspriv.pdf
16. Mowery, K., Shacham, H.: Pixel perfect: Fingerprinting canvas in HTML5. In: Fredrikson, M. (ed.) Proceedings of W2SP 2012. IEEE Computer Society (May 2012), https://cseweb.ucsd.edu/~hovav/dist/canvas.pdf
17. Mulazzani, M., Reschl, P., Huber, M., Leithner, M., Schrittwieser, S., Weippl, E.: Fast and reliable browser identification with javascript engine fingerprinting. In: Web 2.0 Workshop on Security and Privacy (W2SP) (5 2013), http://www.sba-research.org/wp-content/uploads/publications/jsfingerprinting.pdf
18. Navara, E.D., Berjon, R., Leithead, T., O'Connor, E., Pfeiffer, S., Faulkner, S.: HTML5. Candidate recommendation, W3C (Feb 2014), http://www.w3.org/TR/2014/CR-html5-20140731/
19. Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F., Vigna, G.: Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In: Proceedings of the 2013 IEEE Symposium on Security and Privacy. pp. 541–555. SP '13, IEEE Computer Society, Washington, DC, USA (2013), https://seclab.cs.ucsb.edu/media/uploads/papers/sp2013_cookieless.pdf
20. Patel, L.: JavaScript/CSS font detector (Mar 2007), http://www.lalit.org/lab/javascript-css-font-detect/
21. Perry, M.: #2872: Limit the fonts available in TorBrowser (Apr 2011), https://trac.torproject.org/projects/tor/ticket/2872
22. Perry, M.: Bug 732096 - Add a preference to prevent local font enumeration, comment 18 (Mar 2012), https://bugzilla.mozilla.org/show_bug.cgi?id=732096#c18
23. Perry, M., Clark, E., Murdoch, S.: The design and implementation of the Tor Browser. Tech. rep. (Mar 2013), https://www.torproject.org/projects/torbrowser/design/
24. Russell, K.: Issue 66078: Background tabs with webgl slow down browser due to missing flow control (Dec 2010), https://code.google.com/p/chromium/issues/detail?id=66078
25. Unicode, Inc.: Blocks (Unicode character database) (Apr 2014), http://www.unicode.org/Public/7.0.0/ucd/Blocks.txt

26. Unicode, Inc.: DerivedAge (Unicode character database) (May 2014), http://www.unicode.org/Public/7.0.0/ucd/DerivedAge.txt
27. Zbarsky, B.: Bug 633421 - Clamp setTimeout/setInterval to something higher than 10ms in inactive tabs (Feb 2011), https://bugzilla.mozilla.org/show_bug.cgi?id=633421

A Sample fingerprint

This is a sample font metric fingerprint using the fast code point testing set of Table 3. The system represented is Tor Browser (Firefox 24.8.0) in Tails 1.1.1 [3]. The fingerprint can be hashed into a single short identifier rather than being stored in the long form shown here.

	default	sans-serif	serif	monospace	cursive	fantasy
U+20B9	636×1200	636×1200	636×1200	602×1200	636×1200	636×1200
U+2581	769×1200	769×1200	769×1200	602×1200	769×1200	769×1200
U+20BA	636×1200	636×1200	636×1200	602×1200	636×1200	636×1200
U+A73D	824×1200	818×1200	824×1200	818×1200	818×1200	818×1200
U+FFFD	1025×1200	1025×1200	1025×1200	602×1200	1025×1200	1025×1200
U+20B8	636×1200	636×1200	636×1200	602×1200	636×1200	636×1200
U+05C6	307×1200	441×1200	307×1200	441×1200	441×1200	441×1200
U+1E9E	829×1200	769×1200	829×1200	769×1200	769×1200	769×1200
U+097F	524×1598	524×1598	524×1598	524×1598	524×1598	524×1598
U+F003	1000×1226	977×1200	1000×1226	1000×1219	977×1200	977×1200
U+1CDA	636×1200	636×1200	636×1200	602×1200	636×1200	636×1200
U+17DD	0×2002	0×2002	0×2002	0×1856	0×2002	0×2002
U+23AE	521×1200	521×1200	521×1200	602×1200	521×1200	521×1200
U+0D02	886×1472	886×1472	886×1472	886×1472	886×1472	886×1472
U+0B82	763×2000	763×2000	763×2000	763×2000	763×2000	763×2000
U+115A	1000×1226	1000×1219	1000×1226	1000×1219	1000×1219	1000×1219
U+2425	500×1200	500×1200	500×1200	500×1200	500×1200	500×1200
U+302E	0×1226	0×1219	0×1226	0×1219	0×1219	0×1219
U+A830	636×1200	636×1200	636×1200	602×1200	636×1200	636×1200
U+2B06	838×1200	838×1200	838×1200	838×1200	838×1200	838×1200
U+21E4	838×1200	838×1200	838×1200	602×1200	838×1200	838×1200
U+20BD	636×1200	636×1200	636×1200	602×1200	636×1200	636×1200
U+2C7B	491×1200	491×1200	491×1200	491×1200	491×1200	491×1200
U+20B0	636×1200	636×1200	636×1200	602×1200	636×1200	636×1200
U+FBEE	500×1200	500×1200	500×1200	500×1200	500×1200	500×1200
U+F810	16×1200	16×1200	16×1200	1000×1230	16×1200	16×1200
U+FFFF	636×1200	636×1200	636×1200	602×1200	636×1200	636×1200
U+007F	600×1200	600×1200	600×1200	602×1200	600×1200	600×1200
U+10A0	723×1200	840×1200	723×1200	840×1200	840×1200	840×1200
U+1D790	774×1200	774×1200	774×1200	774×1200	774×1200	774×1200
U+0700	1000×1200	1000×1200	1000×1200	1000×1200	1000×1200	1000×1200
U+1950	500×1200	500×1200	500×1200	500×1200	500×1200	500×1200
U+3095	1000×1200	1000×1200	1000×1200	1000×1200	1000×1200	1000×1200
U+532D	16×1200	16×1200	16×1200	1000×1230	16×1200	16×1200
U+061C	636×1200	636×1200	636×1200	602×1200	636×1200	636×1200
U+20E3	0×1200	0×1200	0×1200	0×1200	0×1200	0×1200
U+FFF9	0×1200	0×1200	0×1200	602×1200	0×1200	0×1200
U+0218	685×1200	635×1200	685×1200	602×1200	635×1200	635×1200
U+058F	636×1200	636×1200	636×1200	602×1200	636×1200	636×1200
U+08E4	636×1200	636×1200	636×1200	602×1200	636×1200	636×1200
U+09B3	636×1200	636×1200	636×1200	602×1200	636×1200	636×1200
U+1C50	500×1200	500×1200	500×1200	500×1200	500×1200	500×1200
U+2619	896×1200	896×1200	896×1200	602×1200	896×1200	896×1200

B Mean anonymity set size from entropy

This appendix contains a proof of the claim in Figure 5, that an entropy measurement implies a mean anonymity set size. Refer to Section 3 for notation.

Claim. Let S be a vector of categorical values with N elements and k distinct values $v_1, \ldots, v_k$. For $i \in 1, \ldots, k$, let c_i signify the number of times v_i appears in S: $P_S(v_i) = c_i/N$. Then the quantity $N/2^{H(S)}$, shown in Figure 5, is $\left(\prod_{i=1}^k c_i^{c_i}\right)^{\frac{1}{N}}$; that is, the geometric mean of the vector that results from replacing each element of S with the number of times that element appears (a vector where each c_i appears c_i times).

Proof.

$$\begin{aligned}
&N/2^{H(S)} \\
=&N/2^{\left(-\sum_{i=1}^k P_S(v_i)\log_2 P_S(v_i)\right)} \\
=&N \cdot 2^{\left(\sum_{i=1}^k \frac{c_i}{N}\log_2 \frac{c_i}{N}\right)} \\
=&N \prod_{i=1}^k 2^{\left(\frac{c_i}{N}\log_2 \frac{c_i}{N}\right)} \\
=&N \prod_{i=1}^k \left(\frac{c_i}{N}\right)^{\frac{c_i}{N}} = N\left(\prod_{i=1}^k \frac{c_i^{c_i}}{N^{c_i}}\right)^{\frac{1}{N}}
\end{aligned}$$

Because $\sum_{i=1}^k c_i = N$, $\prod_{i=1}^k N^{c_i} = N^N$, and so

$$\begin{aligned}
&N\left(\prod_{i=1}^k \frac{c_i^{c_i}}{N^{c_i}}\right)^{\frac{1}{N}} = N\left(\frac{\prod_{i=1}^k c_i^{c_i}}{N^N}\right)^{\frac{1}{N}} \\
=&N\frac{\left(\prod_{i=1}^k c_i^{c_i}\right)^{\frac{1}{N}}}{N} = \left(\prod_{i=1}^k c_i^{c_i}\right)^{\frac{1}{N}}.
\end{aligned}$$

□