SCIENTIFIC REPORTS

OPEN

True Randomness from Big Data

Periklis A. Papakonstantinou[1,*], David P. Woodruff[2,*] & Guang Yang[3,*]

Received: 16 February 2016
Accepted: 18 August 2016
Published: 26 September 2016

Generating random bits is a difficult task, which is important for physical systems simulation, cryptography, and many applications that rely on high-quality random bits. Our contribution is to show how to generate provably random bits from uncertain events whose outcomes are routinely recorded in the form of *massive data sets*. These include scientific data sets, such as in astronomics, genomics, as well as data produced by individuals, such as internet search logs, sensor networks, and social network feeds. We view the generation of such data as the sampling process from a *big source*, which is a random variable of size at least a few gigabytes. Our view initiates the study of big sources in the randomness extraction literature. Previous approaches for big sources rely on statistical assumptions about the samples. We introduce a general method that provably extracts almost-uniform random bits from big sources and extensively validate it empirically on real data sets. The experimental findings indicate that our method is efficient enough to handle large enough sources, while previous extractor constructions are not efficient enough to be practical. Quality-wise, our method at least matches quantum randomness expanders and classical world empirical extractors as measured by standardized tests.

Randomness extraction is a fundamental primitive, and there is a large body of work on this; we refer the reader to the recent surveys[1,2] and references therein. The extracted true random bits are critical for numerical simulations of non-linear and dynamical systems, and also find a wide array of applications in engineering and science[3–7]. There are tangible benefits to linking randomness extraction to big sources. First, big sources are now commonplace[8,9]. Second, since they are in common use, adversaries cannot significantly reduce statistical entropy without making them unusable[10]. In addition, the ability to extract from big samples leverages the study of classical-world extractors[11] and quantum randomness expanders[12], since they allow us to post-process while ignoring local statistical dependencies. The contribution of our work is to give an extractor which has theoretical guarantees and which works efficiently, in theory and in practice, on massive data sets. In particular the model of processing massive data we study is the data stream model discussed in more detail below.

The main obstruction in big source extraction is the lack of available computational resources. Previously, the study of general extractors was largely theoretical. Note that no known theoretical construction could be applied even on samples of modest size, e.g., 10 megabytes (MB). Even if it had been possible to gracefully scale the performance of previous extractors, processing a 20 gigabyte (GB) sample would have taken more than 100,000 years of computation time and exabytes of memory. In contrast, our proposed method processes the same sample using 11 hours and 22 MB of memory. The proposed method is the first feasibility result for big source extraction, but is also the first method that works in practice.

Extractors from Big Sources

Let us now state things more precisely. Randomness can be extracted from an (n, k)-source X, where X is a random variable over $\{0, 1\}^n$ whose *min-entropy* $\mathrm{H}_\infty[X] = \min_{x\in\{0,1\}^n} \log_2(\Pr[X = x]^{-1})$ is at least k. The *min-entropy rate* of X is $\kappa = \mathrm{H}_\infty[X]/n$. Min-entropy is a worst-case statistic and in general cannot be replaced by the average-case *(Shannon) entropy* $\mathrm{H}[X] = \sum_{x\in\{0,1\}^n} \Pr[X = x]\log_2(\Pr[X = x]^{-1})$, an issue that we will elaborate more on later. To extract randomness from a source X, we need: (i) a sample from X, (ii) a small uniform random seed Y, (iii) a lower bound k for $\mathrm{H}_\infty[X]$, and (iv) a fixed error tolerance $\varepsilon > 0$. Formally, a (k, ε)-*extractor* Ext: $\{0, 1\}^n \times \{0, 1\}^d \to \{0, 1\}^m$ outputs Ext (X, Y) that is ε-*close to the uniform distribution*, i.e., $\frac{1}{2}\sum_{z\in\{0,1\}^m}\left|\Pr[\mathrm{Ext}(X, Y) = z] - \frac{1}{2^m}\right| \le \varepsilon$, when taking input from any (n, k)-source X together with a random seed Y. In typical settings $d = \mathrm{polylog}(n) = (\log_2 n)^c$ for a constant $c > 0$, and $m > d$. The seed is necessary since otherwise it is impossible to extract a single random bit from one source[2]. We note that other notions of the output being random, other than closeness to the uniform distribution, are possible and have been studied in a number of general science journal articles[13–16]. These are based on measures of randomness such as approximate entropy. Since our measure is *total variation distance* to the uniform distribution, our generated output provably appears random to *every* other specific measure, including e.g., approximate entropy.

[1]Rutgers University, MSIS, Piscataway, NJ 08853, USA. [2]IBM Research Almaden, San Jose, CA 95120, USA. [3]Institute for Computing Technology, CAS, Beijing, 100190, China. [*]These authors contributed equally to this work. Correspondence and requests for materials should be addressed to D.P.W. (email: dpwoodru@us.ibm.com)

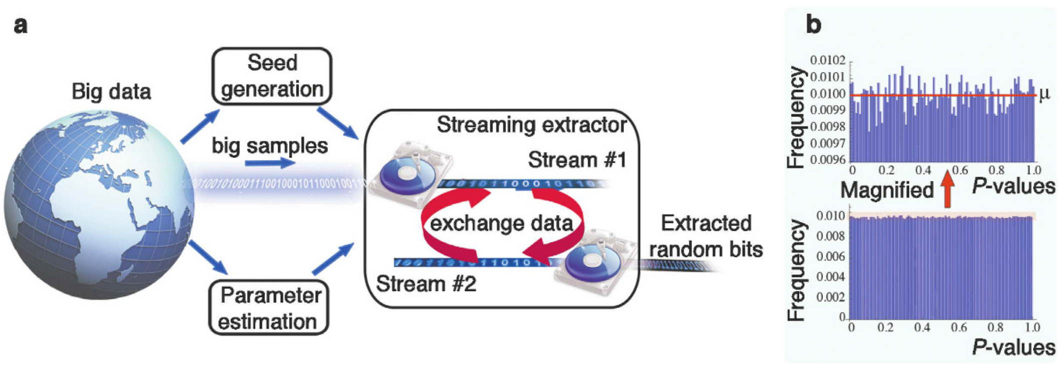

Figure 1. High-level overview of the extraction method. (**a**) The core of the proposed method is the streaming extractor. The description corresponds to one big source, with parameter estimation and initial seed generation done only once. Repeating the seed generation is optional, and is not needed for practical purposes. (**b**) The output of the proposed method is validated by standard statistical tests (NIST). On the ideal (theoretical) uniform distribution the P-values are uniformly distributed. We plot the histogram for 1,512,000 P-values proving a high-level indication about the uniformly extracted bits, i.e., well-concentrated frequencies around the expected frequency $\mu = 0.01$. This graph provides a visual overview (averaging), whereas detailed statistics are in the Supplementary Information.

What does it mean to extract randomness from big sources? Computation over big data is commonly formalized through the *multi-stream model of computation*, where, in practice, each stream corresponds to a hard disk[17]. Algorithms in this model can be turned into executable programs that can process large inputs. Formally, a *streaming extractor* uses a local memory and a constant number (e.g., two) of streams, which are *tapes* the algorithm reads and writes sequentially from left to right. Initially, the sample from X is written on the first stream. The seed of length $d = \text{polylog}(n)$ resides permanently in local memory. In each computation step, the extractor operates on one stream by writing over the current bit, then moving on to the next bit, and finally updating its local memory, while staying put on all other streams. The sum p of all passes over all streams is constant or slightly above constant and the local memory size is $s = \text{polylog}(n)$.

The limitations of streaming processing (tiny p and s) pose challenges for randomness extraction. For example, a big source X could be controlled by an adversary who outputs n-bit samples $x = x_1x_2\ldots x_n$ where $x_{t_1}\ldots x_{t_1+\Delta} = x_{t_2}\ldots x_{t_2+\Delta}$, for some $t_1 \neq t_2$ and large integer $\Delta > 0$. Besides such simple dependencies, an extractor must eliminate all possible determinacies without knowing any of the specifics of X. To do that, it should spread the input information over the output, a task fundamentally limited in streaming algorithms. This idea was previously[18] formalized, where it was shown that an extractor with only one stream needs either polynomial in n, denoted poly(n), many passes, or poly(n)-size local memory; i.e., *no single-stream extractor exists*. Even if we add a constant number of streams to the model, the so-far known extractors[19,20] cannot be realized with $o(\log_2 n)$ many passes (a corollary[17]), nor do they have a known implementation with tractable stream size.

An effective study on the limitations of every possible streaming extractor goes hand in hand with a concrete construction we provide. The main purpose of this article is to explain why such a construction is at all possible and our focus here is on the empirical findings. The following theorem relies on mathematical techniques that could be of independent interest (Supplementary Information pp. 26–30) and states that $\Omega(\log_2 \log_2 n)$ many passes are necessary for all multi-stream extractors. This constitutes our main impossibility result. This unusual, slightly-above-constant number of passes, is also sufficient, as witnessed by the two-stream extractor presented below.

Theorem. *Fix an arbitrary multi-stream extractor* Ext: $\{0, 1\}^n \times \{0, 1\}^d \to \{0, 1\}^m$ *with error tolerance* $\varepsilon = 1/\text{poly}(n)$, *such that for every input source X where* $H_\infty[X] \geq \kappa n$, *for any constant* $\kappa > 0$, *and uniform random seed Y, the output* Ext (X, Y) *is* ε*-close to uniform. If Ext uses sub-polynomial* $n^{o(1)}$ *local memory then it must make* $p = \Omega(\log_2 \log_2 n)$ *passes. Furthermore, the same holds for every constant* $\lambda \in \mathbb{Z}^+$ *number of input sources.*

Our RRB Extractor

We propose and validate a new empirical method for true randomness extraction from big sources. This method consists of a novel extractor and empirical methods to both estimate the min-entropy and generate the initial random seed. Figure 1 depicts a high-level view of the complete extaction method. This is the first complete general extraction method, not only for big sources but for every statistical source.

We propose what we call the *Random Re-Bucketing* (RRB) extractor. For our RRB extractor we prove (Supplementary Information pp. 11–25) that it outputs almost-uniform random bits when given as input a single sample from an arbitrary weak source – as long as the source has enough min-entropy. Mathematical guarantees are indispensable for extractors, since testing uniformity and estimating entropy of an unknown distribution, even approximately, is computationally intractable[21].

A key-feature of the RRB extractor in Fig. 2 is its simplicity, with the technical difficulty being in proving its correctness, which requires a novel, non-trivial analysis. RRB is the first extractor capable of handling big

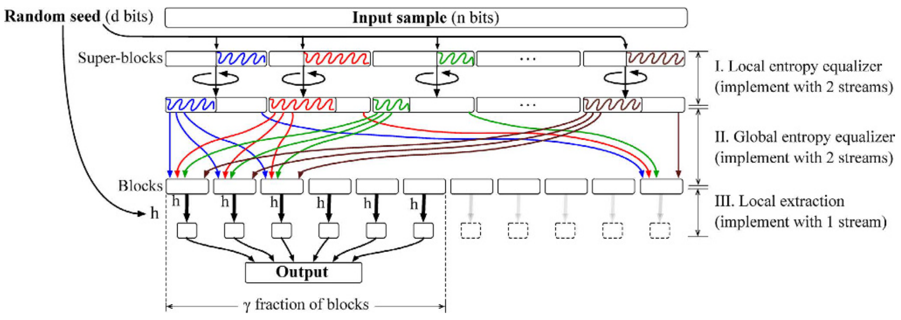
Random seed (d bits)
Input sample (n bits)
Super-blocks
I. Local entropy equalizer (implement with 2 streams)
II. Global entropy equalizer (implement with 2 streams)
Blocks
h
III. Local extraction (implement with 1 stream)
Output
γ fraction of blocks

Figure 2. The Random Re-Bucketing (RRB) extractor. The random seed of size polylog(n) is used only in Stages I & III. In Stage III the same local extractor h is used for the first γ fraction of blocks. The number of super-blocks b also depends on an error tolerance ε and the empirically estimated min-entropy rate κ. In the main body, we explain how to realize this description as an algorithm that uses *two streams*.

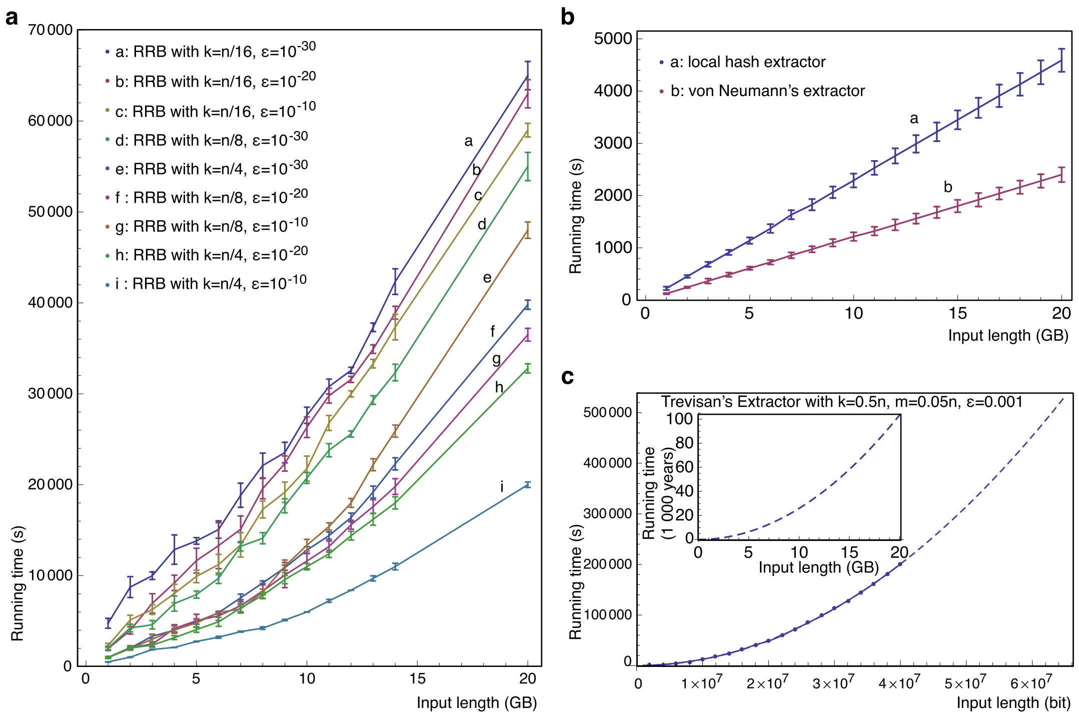

Figure 3. Running time of RRB compared with other extractors. Running time is measured on input samples of size 1 GB–20 GB for von Neumann extractor, local hash extractor (with block-size 1024 bits), and for RRB (with $k = n/4$, $n/8$, $n/16$ and $\varepsilon = 10^{-10}$, 10^{-20}, 10^{-30}). Trevisan's extractor is only measured for $\varepsilon = 0.001$ on samples of size up to 5 MB = 4×10^7 bits, since the available implementations of finite fields cannot handle larger samples or smaller ε. The running time of Trevisan's extractor on larger input size (in particular, 103,407 years for 20 GB input) is estimated by polynomial fitting assuming all data in the main memory, which is an unrealistic advantage. The exact form of the fittest polynomial is determined through cross-validation and standard analysis of polynomial norms.

sources without additional assumptions. Previous works require either (i) unrealistic running times or (ii) ad hoc assumptions about the source. In particular, the local extractors such as von Neumann[22] and Local Hash fail significantly in terms of output quality, whereas the throughput of Trevisan's extractor[19] and its followups degrade significantly (see Fig. 3) with the size of the sample[12] even with practical optimization considered; e.g., 103,407 years of computing time for a 20 GB input sample and $\varepsilon = 10^{-3}$, $\kappa = 1/2$. We note that we choose to compare to the Local Hash and von Neumann extractors since these are the only extractors experimented upon in previous work (see ref. 23 for empirical work using von Neumann's extractor, and see refs 12, 24 and 25 for empirical work using Local Hash), and importantly, both extractors happen to be streaming extractors. Thus, due to their special attention in previous work they are two ideal candidates for comparison. We refer the reader to Table 2, Fig. 3, and the Supplementary Information for details.

Data category	NIST Test Suite		DIEHARD Test Suite	
	# of tests	Observed (Ideal) freq.	# of tests	Observed (Ideal) freq.
Compressed audio	564	0 (2)	144	2 (3)
Comp. video	752	4 (2)	144	4 (3)
Comp. images	752	1 (2)	144	2 (3)
Comp. social network data	3008	9 (8)	576	10 (12)
Comp. DNA sequenced data	752	1 (2)	144	3 (3)
Comp. text	3008	11 (8)	576	14 (12)
Audio	752	2 (2)	144	1 (3)
Video	752	1 (2)	144	1 (3)
Images	564	0 (2)	62	2 (1)
Social network data	2256	6 (6)	432	5 (9)
DNA sequenced data	752	2 (2)	144	2 (3)
Text	1504	1 (4)	288	7 (6)
Total	15416	38 (45)	2942	53 (61)

Table 1. Empirically extracted bits versus ideal (theoretical) uniform random bits. Overall statistics of the empirically extracted bits using the proposed method. The second column is the total number of NIST tests per data category. The third column compares the number of NIST tests that the empirically extracted bits do not pass with the expected number of NIST tests (listed in parenthesis) that the ideal uniform random bits will not pass. Any number significantly above or below the one in parenthesis indicates non-uniformity (discussed in Empirical statistical tests on p. 5). Similarly, the fourth and fifth columns report the results for DIEHARD tests. Further statistics related to this table are reported in Supplementary Information Table S1.

Extractor and data setting	NIST Test Suite		
	Number of tests	Observed (Ideal) freq.	P-val $<$ 0.0001
Raw data	2256	1931 (6.09)	1561 (0.22)
Von Neumann	2256	966 (6.09)	785 (0.22)
Von Neumann on adversary	2256	1507 (6.09)	1435 (0.22)
Local hash	2256	16 (6.09)	1 (0.22)
Local hash on adversary	2256	781 (6.09)	170 (0.22)
RRB	2256	4 (6.09)	0 (0.22)
RRB on adversary	2256	5 (6.09)	1 (0.22)

Table 2. Comparative extraction quality performance. The raw data consists of 12 files each of size 1000 MB from the 12 data categories and the adversarial data are generated by simply replacing 10 MB in each file with fixed values. NIST tests are applied on the raw data and extraction output of von Neumann, local hash, and RRB extractors on raw data and adversarial data. The second column is the total number of NIST tests per setting. The third column is the number of NIST tests that fail because of proportion, and the fourth column is the number of NIST tests that fail because of the second-order P-value. All are compared with the expected number of the ideal uniform random bits. Except from RRB and "RRB on adversary", all other test results indicate non-uniform output (i.e. noticeably different from ideal uniform).

The RRB extractor consists of the following three stages.

I. Partition the n-bit long input into $b = O\left(\log_2 \frac{n}{\varepsilon}\right)$ many *super-blocks*, each of length n/b. Inside each super-block, choose uniformly and independently a random point to cyclically shift the super-block.
II. Re-bucket the b *super-blocks* into n/b many *blocks* each of size b, where the i-th block consists of the i-th bit from every (shifted) super-block, for $i = 1, 2, \ldots, n/b$.
III. Specify a *local extractor* h: $\{0, 1\}^b \to \{0, 1\}^{\kappa b/2}$ using the uniform random seed; for example, h can be a random Toeplitz matrix. Then, locally apply h on the first $b_O = \gamma n/b$ blocks, concatenate, and output the result. Here the *effectiveness factor* $\gamma = \Omega(1)$ denotes the fraction of blocks used for local extraction.

This extractor can be realized with two streams and local memory size polylog(n) (more details of streaming realization on p. 5). (I) Cyclically shift every super-block using in total 4 passes (every pass operates on all super-blocks). (II) Re-bucket with $O\left(\log_2 \log_2 \frac{n}{\varepsilon}\right)$ many passes and iterations where, in each iteration, the first and the second half of the first stream are shuffled with the help of the second stream. (III) Locally extract with 2 passes.

The implementation is scalable since it uses $\mathrm{O}(\log_2 b) = \mathrm{O}\left(\log_2 \log_2 \frac{n}{\varepsilon}\right)$ passes and a $\mathrm{O}(b \log_2 n) = \mathrm{O}\left(\log_2 \frac{n}{\varepsilon} \cdot \log_2 n\right)$ bit random seed. For example, 44 passes and 57 KB of a random seed suffice to extract 1 GB of randomness from a 20 GB input. (For min-entropy $k \geq 0.2n = 4$ GB, and error rate $\varepsilon = 10^{-27} < 1/n^2$. With a total of 50 passes, the error rate can be as small as $\varepsilon = 10^{-100}$. Most of the seed is used to sample a random Toeplitz hash h.).

Stage III with $\gamma = 1$ has been used before[26–28] in randomness extraction from sources of guaranteed *next-block-min-entropy*. This guarantee means that *every* block, as a random variable, has enough min-entropy left even after revealing all the blocks preceding it, i.e., it presumes strong inter-block independence. Such a precondition restricts the applicability of Stage III since it appears too strong for common big sources, especially when there is an adversary. However, by introducing Stages I & II we can provably fulfill the precondition for a theoretically lower bounded constant γ. In practice, a larger (i.e., better than in theory) γ can be empirically found and validated.

Stage I equalizes entropy within each super-block and, subsequently, Stage II distributes entropy globally. After Stage II, the following property holds. Let $(Z_1, \ldots, Z_{n/b}) \in (\{0, 1\}^b)^{\frac{n}{b}}$ denote the blocks of bits of the intermediate result at the end of Stage II. *Next-block-min-entropy* $\mathrm{H}_\infty[Z_i|Z_{i-1}, \ldots, Z_1] = \min_{z_1,\ldots,z_{i-1}} \mathrm{H}_\infty[Z_i|Z_{i-1} = z_{i-1}, \ldots, Z_1 = z_1]$ is the min-entropy of the i-th block Z_i conditioned on the worst choice of all the blocks preceding Z_i. We show (Supplementary Information pp. 15–25) that *for every* $i \in \{1, \ldots, b_O\}$,

$$\mathrm{H}_\infty[Z_i|Z_1, \ldots, Z_{i-1}] \geq \Omega(b) \tag{1}$$

with probability (over the choice of the random seed) greater than $1 - \varepsilon/n$, for $b = \mathrm{O}\left(\log_2 \frac{n}{\varepsilon}\right)$, $\gamma = \Omega(1)$ and $b_O = \Omega(n/b)$. Therefore, Stage III extracts ε-close to uniform random bits.

To invoke the extractor, it is necessary to find an initial random seed and estimate the min-entropy rate κ of the source. The proposed method includes an empirical realization of a multi-source extractor to obtain 4 MB initial randomness from 144 audio samples each of 4 MB. We also propose and validate an empirical protocol that estimates both κ and γ simultaneously by combining RRB itself with standardized uniformity tests.

Finally, we note that the RRB extractor bears some superficial similarities to the Advanced Encryption Standard (AES) block cipher, which is an encryption scheme widely used in practice. That is, at a high level both schemes efficiently mix information, though they do so in very different ways, e.g., in AES this is done on a much more local scale, whereas we mix information globally. Moreover, unlike the RRB extractor that we propose, the AES block cipher cannot have provable guarantees without proving that $P \neq NP$.

Methods

The proposed method is validated in terms of efficiency and quality, measured by standard quality test suites, NIST[29] and DIEHARD[30]. The results strongly support our new extractor construction on real-world samples. The empirical study compares multiple extraction methods on many real world data sets, and demonstrates that our extractor is the only one that works in practice on sufficiently large sources.

Our experiments are explained in more detail below and we summarize them here. Our samples range in size from 1.5 GB–20 GB and they are from 12 data categories: compressed/uncompressed text, video, images, audio, DNA sequenced data, and social network data. The empirical extraction is for $\varepsilon = 10^{-20}$ and estimated min-entropy rate ranging from 1/64 to 1/2, with extraction time from 0.85 hours to 11.06 hours on a desktop PC (Fig. 3). The extracted outputs of our method pass all quality tests, whereas the before-extraction-datasets fail almost everywhere (Tables 1 and 2). The output quality of RRB is statistically identical to the uniform distribution. Such test results provide further evidence supporting that the extraction quality is close to the ideal uniform distribution, besides the necessary[31] rigorous mathematical treatment.

Extraction method. The complete empirical method consists of: (i) initial randomness generation, (ii) parameter estimation, and (iii) streaming extraction. Components (ii) and (iii) rely on initial randomness.

We first extract randomness from multiple independent sources without using any seed. Then, we use RRB to expand this initial randomness further.

Parameter estimation determines a suitable pair (κ, γ) of min-entropy rate $\kappa = k/n$ and effectiveness factor $\gamma = b_O/\left(\frac{n}{b}\right)$.

Experimental set-up. We empirically evaluate the quality and the efficiency of our RRB extractor.

Quality evaluation is performed on big samples from twelve semantic data categories: compressed/uncompressed audio, video, images, text, DNA sequenced data, and social network data (for audio, video, and images the compression is lossy). The initial randomness used in our experiments consists of 9.375×10^8 bits $\approx$117 MB generated from 144 pieces of 4 MB compressed audio and one piece of 15 GB compressed video. The produced randomness is used for parameter estimation on samples ranging in size from 1 GB to 16 GB from each of the 12 categories. The estimated κ and γ vary within [1/64, 1/2] and [1/32, 1/2] respectively, cross-validated (i.e., excluding previously used samples) on samples of size 1.5 GB–20 GB with error tolerance $\varepsilon = 10^{-20}$. Final extraction quality is measured on all 12 categories by the standard NIST and DIEHARD batteries of statistical tests.

Operating system kernel-level measurements are taken for the running time and memory usage of RRB. These measurements are taken from RRB on input sizes 1 GB–20 GB, min-entropy rate $\kappa \in \{1/4, 1/8\}$, and error tolerance $\varepsilon \in \{10^{-10}, 10^{-20}\}$.

For comparison, we measure quality and efficiency for three of the most popular representatives of extractors. The quality of Local Hash and von Neumann extractors is evaluated on 12 GB of raw data (from the 12 categories) and on 12 GB adversarial synthetic data. The efficiency is measured for von the Neumann extractor, Local Hash, and Trevisan's extractor. See the Supplementary Information for tables and figures showing this.

Empirical initial randomness generation. Seeded extraction, as in RRB, needs uniform random bits to start. All the randomness for the seeds in our experiments is obtained by the following method (which we call it *randomness bootstrapping*) in two phases: (i) obtain initial randomness ρ through (seedless) multiple-independent-source extraction, and (ii) use ρ for parameter estimation and run RRB to extract a longer string ρ_{long}, $|\rho_{long}| = 2^{\sqrt{|\hat{\rho}|/54}-7}$, where $\hat{\rho}$ is the part of ρ used as the seed of RRB during bootstrapping. By elementary information theory, ρ_{long} can be used instead of a uniformly random string.

Phase (i) is not known to have a streaming implementation, which is a not an issue since it only extracts from small samples. Start with 144 statistically independent compressed audio samples $\rho_1, \ldots, \rho_{144}$: each sample is 4 MB of high-quality (320 Kbps) compressed recording (MPEG2-layer3). Taken together, the samples last 4.1 hours. These samples are generated privately – without malicious adversarial control – using different independent sound-settings and sources. Partition the samples into 16 groups, each consisting of $9 = 3^2$ samples. Every ρ_i can be interpreted as a field element in GF[p], where $p = 2^{57885161} - 1$ is the largest known Mersenne prime and 4 MB $< \log_2 p$ bits. For the first group $\{\rho_1, \ldots, \rho_9\}$, compute $\rho^{(1)} = \rho' \cdot \rho'' + \rho'''$ where $\rho' = \rho_1 \cdot \rho_2 + \rho_3$, $\rho'' = \rho_4 \cdot \rho_5 + \rho_6$, and $\rho''' = \rho_7 \cdot \rho_8 + \rho_9$; which is a two-level recursion. In the same way, compute $\rho^{(2)}, \ldots, \rho^{(16)}$ and finally let $\rho = \rho^{(1)} + \ldots + \rho^{(16)}$, with all operations in GF[p]. We call this the BIWZ method due to the authors[32,33] who studied provable multi-source extraction based on the field operation $x \cdot y + z$.

Phase (ii) uses the 4 MB extracted by BIWZ out of which 3.99 MB are used in parameter estimation for compressed video. The remaining 10 KB are used to run RRB on 15 GB compressed video, which is generated and compressed privately, i.e., without adversarial control. Our hypothesis is that the estimated parameters are valid for RRB, i.e., n bits of compressed video contain min-entropy $n/2$ that can be extracted by RRB with effectiveness factor $\gamma = 1/32$. This hypothesis is verified experimentally. With the given seed and $\kappa = 1/2$, $\gamma = 1/32$, and $\varepsilon = 10^{-100}$, RRB extracts the final 9.375×10^8 random bits.

Empirical parameter estimation protocol. There are two crucial parameters for RRB: the min-entropy rate κ and the effectiveness factor γ. In theory, γ is determined by κ, n, ε. In practice, better, empirically validated values are estimated simultaneously for κ and γ. This works because in addition to min-entropy, κ induces the next-block-min-entropy guarantee for a fraction of γ blocks.

For every semantic data category, the following protocol estimates a pair of (κ, γ).

First, obtain a bit sequence s of size 1 GB by concatenating sampled $<$ 1 MB segments from the target data category. Then, compress s into s' using LZ77[34] ($s' = s$ if s is already compressed). Since the ideal compression has $|s'|$ equal to the Shannon entropy of s, the compression rate $\frac{|s'|}{|s|}$ is also an upper bound for the min-entropy rate. To obtain a lower bound for the min-entropy rate (required parameter for RRB), we start from $\kappa' = \frac{|s'|}{2|s|}$ and search inside $[0, \kappa']$. For min-entropy rate $\kappa \in \left\{\kappa', \frac{\kappa'}{2}, \frac{\kappa'}{4}, \frac{\kappa'}{8}, \frac{\kappa'}{16}\right\}$ and effectiveness factor $\gamma \in \left\{\gamma', \frac{\gamma'}{2}, \frac{\gamma'}{4}, \frac{\gamma'}{8}, \frac{\gamma'}{16}\right\}$, extract from s using RRB, with parameters κ, γ, and $\varepsilon = 10^{-20}$ and seed from the initial randomness. Apply NIST tests on the extracted bits for every (κ, γ) pair. If the amount of extracted bits is insufficient for NIST tests, then start over with an s twice as long. We call a pair of (κ_0, γ_0) *acceptable* if NIST fails with frequency at most 0.25% for every run of RRB with parameters $\kappa \le \kappa_0$ and $\gamma \le \gamma_0$. This 0.25% threshold is conservatively set slightly below the expected failure probability of NIST on ideal random inputs, which is 0.27%. If (κ_0, γ_0) is a correctly estimated lower bound, then every estimate (κ, γ) with $\kappa \le \kappa_0$ and $\gamma \le \gamma_0$ is also a correct lower bound. Hence, the extraction with (κ, γ) should be random and pass the NIST tests. We choose the acceptable pair (if any) that maximizes the output length.

There is strong intuition in support of the correct operation of this protocol. First, the random sampling for s preserves with high probability the min-entropy rate[35]. Second, an extractor cannot extract almost-uniform randomness if the source has min-entropy much lower than the estimated one. Finally, NIST tests exhibit a certain ability to detect non-uniformity. Verification of the estimated parameters is done by cross-validation.

Streaming realization of the RRB extractor. The streaming extractor uses $d + 2\log_2 n$ bits local memory and $3\left\lceil \log_2 \log_2 \frac{n}{\varepsilon} - 2\log_2 \kappa + 2\log_2 3 + 1\right\rceil$ passes over two streams, for input length n, min-entropy rate κ, error tolerance ε and seed length d. RRB is also parametrized by the effectiveness factor γ as shown below.

Given n, ε, and the estimated κ, γ, we initially set $k = \kappa n$, the output length $m = \frac{1}{2}\gamma\kappa n$, and the number of super-blocks $b = \frac{9}{2\kappa^2}\log_2 \frac{n}{\varepsilon}$. For convenience, n is padded to a power of 2, κ and γ are rounded down to an inverse power of 2, and b is rounded up to a power of 2. Hereafter, no further rounding is needed. Let σ_1 and σ_2 denote two read/write streams. The input sample $x \in \{0, 1\}^n$ is initially on σ_1. Obtain a seed y of length $d = b\left(1 + \frac{\kappa}{2} + \log_2 \frac{n}{b}\right)$ from the initially generated randomness, and store it in local memory. We interpret y as $y = (y_0, y_1, \ldots, y_b) \in \{0, 1\}^{(1+\frac{\kappa}{2})b} \times \left\{0, 1, 2, \ldots, \frac{n}{b} - 1\right\}^b$.

In Stage I, we partition the input into b super-blocks $x = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_b)$, where $|\overline{x}_i| = \frac{n}{b}$ for every $i \in \left\{1, 2, \ldots, \frac{n}{b}\right\}$. RRB reads x from σ_1 and writes $(\mathrm{shift}(\overline{x}_1, y_1), \ldots, \mathrm{shift}(\overline{x}_b, y_b))$ to σ_2, where every $\overline{x}_i = \left(\overline{x}_i[1], \ldots, \overline{x}_i\left[\frac{n}{b}\right]\right)$, $\mathrm{shift}(\overline{x}_i, y_i) = \left(\overline{x}_i[y_i + 1], \ldots, \overline{x}_i\left[\frac{n}{b}\right], \overline{x}_i[1], \ldots, \overline{x}_i[y_i]\right)$ denotes the cyclic shift of $\overline{x}_i$ with offset y_i. This can be done with 4 passes.

In Stage II, we compute the re-bucketing of $\mathrm{shift}(\overline{x}_1, y_1), \ldots, \mathrm{shift}(\overline{x}_b, y_b)$, which is stored on σ_2. The re-bucketing output is denoted by $(z_1, \ldots, z_{n/b})$, where every z_j collects the j-th bit from all shifted super-blocks, i.e., $z_j = (\mathrm{shift}(\overline{x}_1, y_1)[j], \ldots, \mathrm{shift}(\overline{x}_b, y_b)[j]) = (\overline{x}_1[y_1 + j], \ldots, \overline{x}_b[y_b + j])$. The re-bucketing of b super-blocks can be done with $\lceil \log_2 b \rceil$ iterations, where every iteration reduces the number of super-blocks by a factor of two by interlacing (with the help of σ_1) the first and second half of σ_2. In particular, the first iteration merges every pair of $\text{shift}(\overline{x}_i, y_i)$ and $\text{shift}(\overline{x}_{i+b/2}, y_{i+b/2})$ into a single super-block $\left(\text{shift}(\overline{x}_i, y_i)[1], \text{shift}(\overline{x}_{i+b/2}, y_{i+b/2})[1], \ldots, \text{shift}(\overline{x}_i, y_i)\left[\frac{n}{b}\right], \text{shift}(\overline{x}_{i+b/2}, y_{i+b/2})\left[\frac{n}{b}\right]\right)$, which consists of n/b blocks (i.e. $(\text{shift}(\overline{x}_i, y_i)[j], \text{shift}(\overline{x}_{i+b/2}, y_{i+b/2})[j])$ for $j = 1, 2, \ldots, n/b$) each of length 2. During the $\lceil \log_2 b \rceil$ many iterations, RRB spends $3\lceil \log_2 b \rceil$ passes to compute $(z_1, \ldots, z_{n/b})$ and store it on σ_1.

In the final stage, we output $(h(z_1), \ldots, h(z_{b_O}))$ to σ_2, where h: $\{0, 1\}^b \rightarrow \{0, 1\}^{\kappa b/2}$ is a hash function realized through a Toeplitz matrix specified by y_0 from the seed and $b_O = \gamma n/b$ the number of blocks used for the output. This m-bit-long output can be locally extracted with 2 passes.

Therefore, RRB extracts m bits with $3\lceil \log_2 b \rceil + 6 = 3\left\lceil \log_2 \log_2 \frac{n}{\varepsilon} - 2\log_2 \kappa + 2\log_2 3 + 1 \right\rceil$ passes. The local memory size is dominated by Stage I, which requires $d + 2\log_2 n$ bits to store the seed and two counters for head positions.

The above description is for the estimated (κ, γ). If there is theoretical knowledge for κ and the error tolerance ε is given, then RRB provably extracts $m = \Omega(n)$ bits that are ε-close to uniform with $b = O\left(\log \frac{n}{\varepsilon}\right)$ and $\gamma = \Omega(1)$. For instance, RRB provably works for $m \geq \left(1 - \frac{\alpha}{2}\right)\alpha^3 n/64$ and $b = 64\alpha^{-3} \log \frac{n}{\varepsilon}$, $\gamma = \alpha/4$, where $\alpha = \kappa^2/(6\log\kappa^{-1})$ is a constant.

Empirical statistical tests. Each statistical test measures one property of the uniform distribution by computing a P-value, which on ideal random inputs is uniformly distributed in [0, 1]. For each NIST test, subsequences are derived from the input sequence and P-values are computed for each subsequence. A significance level $\boldsymbol{\alpha} \in [0.0001, 0.01]$ is chosen such that a subsequence passes the test whenever P-value $\geq \boldsymbol{\alpha}$ and fails otherwise. If we think that NIST is testing ideal random inputs, then the proportion of passing subsequences has expectation $1 - \boldsymbol{\alpha}$, and the *acceptable range of proportions* is the confidence interval chosen within 3 standard deviations. Furthermore, a second-order P-value is calculated on the P-values of all subsequences via a χ^2-test. An input passes one NIST test if (i) the input induces an acceptable proportion and (ii) the second-order P-value ≥ 0.0001. An input passes one DIEHARD-test if P-value is in $[\boldsymbol{\alpha}, 1 - \boldsymbol{\alpha}]$.

We compare the statistical behavior of bits produced by our method with ideal random bits. For ideal random bit-sequences, $\boldsymbol{\alpha}$ is the *ideal failure rate*. Anything significantly lower or higher than this indicates non-uniform input. In our tests, we choose the largest suggested significance level $\boldsymbol{\alpha} = 0.01$; i.e., the hardest to pass the test. All tests on our extracted bits appear statistically identical to ideal randomness. See the Supplementary Information for details.

Experimental platform details. The performance of the streaming RRB, von Neumann extractor, and Local Hash is measured on a desktop PC, with Intel Core i5 3.2 GHz CPU, 8 GB RAM, two 1 terabyte (TB) hard drives and kernel version Darwin 14.0.0. The performance of Trevisan's extractor is measured on the same PC with the entire input and intermediate results stored in main memory. We use the following software platforms and libraries. TPIE[36] is the C++ library on top of which we implement all streaming algorithms – TPIE provides application-level streaming I/O interface to hard disks. For arbitrary precision integer and Galois field arithmetic we use GMP[37] and FGFAL[38]. Mathematica[39] is used for data processing, polynomial fitting, and plots. Source code is available upon request.

Conclusion. We introduced the study of big source extraction, proposed a novel method for achieving this, and demonstrated its feasibility in theory and practice. Big source extraction has immediate gains and poses new challenges, while opening directions in the intersection of randomness extraction, data stream computation, mathematics of computation and statistics, and quantum information. We refer the reader to the Supplementary Information for details of proofs and experiments.

References

1. Shaltiel, R. An introduction to randomness extractors. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 21–41 (Springer, 2011).
2. Shaltiel, R. *Current trends in theoretical computer science. The Challenge of the New Century. (book chapter)*, vol. Vol 1: Algorithms and Complexity (World Scientific, 2004).
3. Binder, K. & Heermann, D. *Monte Carlo Simulation in Statistical Physics: An Introduction* (Springer, 2010).
4. Allen, M. P. & Tildesley, D. J. *Computer Simulation of Liquids* (Oxford Science Publications, 1989).
5. Katz, J. & Lindell, Y. *Introduction to Modern Cryptography: Principles and Protocols* (Chapman & Hall/CRC, 2007).
6. Andrieu, C., De Freitas, N., Doucet, A. & Jordan, M. I. An introduction to MCMC for machine learning. *Machine learning* **50,** 5–43 (2003).
7. Motwani, R. & Raghavan, P. *Randomized Algorithms* (Cambridge University Press, 1995).
8. Bansal, M. Big data: Creating the power to move heaven and earth. *MIT Technology Review* (2014) (date of access: June 28, 2016). http://www.technologyreview.com/view/530371/big-data-creating-the-power-to-move-heaven-and-earth.
9. Wakefield, J. & Kerley, P. How "big data" is changing lives. *BBC (News Technology)* (2013) (date of access: June 28, 2016). http://www.bbc.com/news/technology-21535739.
10. Cox, I. J., Miller, M. L., Bloom, J. A. & Honsinger, C. *Digital watermarking*, vol. 53 (Springer, 2002).
11. Marangon, D. G., Vallone, G. & Villoresi, P. Random bits, true and unbiased, from atmospheric turbulence. *Sci. Rep.* **4** (2014).
12. Ma, X. *et al.* Postprocessing for quantum random-number generators: Entropy evaluation and randomness extraction. *Phys. Rev. A* **87,** 062327 (2013).
13. Pincus, S. & Singer, B. H. A recipe for randomness. *Proceedings of the National Academy of Sciences* **95,** 10367–10372 (1998).
14. Pincus, S. & Singer, B. H. A zoo of computable binary normal sequences. *Proceedings of the National Academy of Sciences* **109,** 19145–19150 (2012).
15. Pironio, S. *et al. Random numbers certified by bells theorem.* **464,** 1021–1024 (2010).
16. Seife, C. *New test sizes up randomness.* **5312,** 532 (1997).
17. Grohe, M., Hernich, A. & Schweikardt, N. Lower bounds for processing data with few random accesses to external memory. *Journal of the ACM* **56,** Art. 12, 58 (2009).

18. Bar-Yossef, Z., Reingold, O., Shaltiel, R. & Trevisan, L. Streaming computation of combinatorial objects. In *Conference on Computational Complexity (CCC)*, 133–142 (IEEE, 2002).
19. Trevisan, L. Construction of extractors using pseudo-random generators. In *Symposium on Theory Of Computing (STOC)*, 141–148. ACM (ACM, 1999).
20. Raz, R., Reingold, O. & Vadhan, S. Extracting all the randomness and reducing the error in trevisan's extractors. In *Symposium on Theory Of Computing (STOC)*, 149–158. ACM (ACM, 1999).
21. Valiant, G. & Valiant, P. Estimating the unseen: An n/log(n)-sample estimator for entropy and support size, shown optimal via new clts. In *Symposium on Theory Of Computing (STOC)*, 685–694 (ACM, 2011).
22. von Neumann, J. Various techniques in connection with random digits. *Applied Math Series* **12,** 36–38 (1951).
23. Um, M. *et al.* Experimental certification of random numbers via quantum contextuality. *Sci. Rep.* **3** (2013).
24. Pironio, S. *et al.* Random numbers certified by Bell's theorem. *Nature* **464,** 1021–1024 (2010).
25. Troyer, M. & Renner, M. A randomness extractor for the Quantis device. *ID Quantique Technical Paper on Randomness Extractor (date of access: June 28, 2016)* (2012).
26. Zuckerman, D. Simulating bpp using a general weak random source. *Algorithmica* **16,** 367–391 (1996).
27. Chor, B. & Goldreich, O. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing* **17,** 230–261 (1988).
28. Chung, K.-M., Mitzenmacher, M. & Vadhan, S. Why simple hash functions work: Exploiting the entropy in a data stream. *Theory of Computing* **9,** 897–945 (2013).
29. Rukhin, A., Soto, J., Nechvatal, J., Smid, M. & Barker, E. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Special Publication 800–22 (Revision 1a), National Institute of Standards and Technology (2010).
30. The marsaglia random number CDROM including the Diehard Battery of Tests of Randomness (date of access: June 28, 2016). http://www.stat.fsu.edu/pub/diehard/ (2008).
31. Soto, J. & Bassham, L. Randomness testing of the advanced encryption standard finalist candidates. *Tech. Rep.* NIST (NISTIR) 6483 (2000).
32. Barak, B., Impagliazzo, R. & Wigderson, A. Extracting randomness using few independent sources. *SIAM Journal on Computing* **36,** 1095–1118 (2006).
33. Zuckerman, D. General weak random sources. In *Foundations of Computer Science (FOCS)*, 534–543. IEEE (IEEE, 1990).
34. Ziv, J. & Lempel, A. A universal algorithm for sequential data compression. *IEEE Transactions on information theory* **23,** 337–343 (1977).
35. Nisan, N. & Zuckerman, D. Randomness is linear in space. *Journal of Computer and System Sciences* **52,** 43–52 (1996).
36. The templated portable i/o environment (date of access: June 28, 2016). http://madalgo.au.dk/tpie/ (2013).
37. The gnu multiple precision arithmetic library (date of access: June 28, 2016). https://gmplib.org (2014).
38. Fast galois field arithmetic library in c/c++ (date of access: June 28, 2016). http://web.eecs.utk.edu/plank/plank/papers/CS-07-593/ (2007).
39. Mathematica (date of access: June 28, 2016). http://www.wolfram.com/mathematica/ (2015).

Acknowledgements
We wish to thank Paul Beame, Matei David, Artur Ekert, Mile Gu, Yuval Ishai, Kihwan Kim, Eyal Kushilevitz, Fu Li, John Steinberger, Avi Wigderson, Jia Xu, Shenghao Yang, and Uri Zwick for discussions, corrections, and suggestions. The work is partially supported by National Natural Science Foundation of China (61222202, 61433014, 61502449, 61602440) and the China National Program for support of Top-notch Young Professionals.

Author Contributions
D.P.W. wrote the introductory part of the paper, whereas P.A.P. and G.Y. the rest. The figures were prepared by G.Y. All authors have reviewed the manuscript and confirmed the submission.

Additional Information
Supplementary information accompanies this paper at http://www.nature.com/srep

Competing financial interests: The authors declare no competing financial interests.

How to cite this article: Papakonstantinou, P. A. *et al.* True Randomness from Big Data. *Sci. Rep.* **6**, 33740; doi: 10.1038/srep33740 (2016).