Cryptography with Streaming Algorithms*

Periklis A. Papakonstantinou and Guang Yang

Institute for Theoretical Computer Science,
Tsinghua University,
Beijing 100084, China

Abstract. We put forth the question of whether cryptography is feasible using streaming devices. We give constructions and prove lower bounds. In streaming cryptography (not to be confused with stream-ciphers) everything—the keys, the messages, and the seeds—are huge compared to the internal memory of the device. These streaming algorithms have small internal memory size and make a constant number of passes over big data maintained in a constant number of read/write external tapes. Typically, the internal memory size is $O(\log n)$ and we use 2 external tapes; whereas 1 tape is provably insufficient. In this setting we cannot compute instances of popular intractability assumptions. Nevertheless, we base cryptography on these assumptions by employing non-black-box techniques, and study its limitations.

We introduce new techniques to obtain unconditional lower bounds showing that no super-linear stretch pseudorandom generator exists, and no Public Key Encryption (PKE) exists with private-keys of size sub-linear in the plaintext length.

For possibility results, assuming the existence of one-way functions computable in NC^1—e.g. factoring, lattice assumptions—we obtain streaming algorithms computing one-way functions and pseudorandom generators. Given the Learning With Errors (LWE) assumption we construct PKE where both the encryption and decryption are streaming algorithms. The starting point of our work is the groundbreaking work of Applebaum-Ishai-Kushilevitz on Cryptography in NC^0. In the end, our developments are technically orthogonal to their work; e.g. there is a PKE where the decryption is a streaming algorithm, whereas no PKE decryption can be in NC^0.

Keywords: streaming, lower bound, big data, randomized encoding, non-black-box, PRG, PKE.

1 Introduction

In most cryptosystems the keys can be assumed to reside in a local memory provided with unlimited access. What if access to the keys is not for free? Suppose

* This work was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61350110536, 61361136003.

J.A. Garay and R. Gennaro (Eds.): CRYPTO 2014, Part II, LNCS 8617, pp. 55–70, 2014.

that the key is very long and, together with everything else in the input, is stored as a stream that can be sequentially scanned only a few times. Is it possible to compute cryptographically secure functions in this way?

More formally, we consider the possibility of cryptography against arbitrary polynomial time adversaries, who are as powerful as usual, using a (less powerful) streaming algorithm which has access to a bounded internal memory, and to external read/write tapes (*RW streams*) where we quantify on the number of passes over them. These RW streams are commonly thought (see e.g. [19,17] and references within) to correspond to hard disk drives or other sequentially accessed buffers. The question of cryptography using streaming devices is motivated in practice in settings where the keys and messages are huge (e.g. authenticating big data), whereas theoretically it falls in the fundamental study of cryptography with rudimentary resources.

Below, we give an overview of our results, then we discuss related work and several subtleties of streaming cryptography, and finally we compare to previous work in randomized encodings.

Our results. For the rest of this paper and unless mentioned otherwise, a *streaming algorithm* has 2 RW streams (external tapes), over which it makes $O(1)$ many passes, and it uses $O(\log n)$ internal memory size[1] for input length n. These are optimal parameters, within constants, under which streaming cryptography may exist (we show that 1 read/write stream is not sufficient). We devise streaming constructions of private- and public-key primitives by synthesizing various previous works along with new techniques necessary for streaming. This possibility is quite unexpected for distinct reasons. We also introduce technically novel machinery and study the limitations of private- and public-key cryptography in this setting.

Impossibility results. We show the impossibility of super-linear stretch streaming pseudorandom generator, and we also obtain a linear (in the security parameter) lower bound on the key-size of streaming PKE. The proof technique is inspired by [19,17]. However, a cryptography lower bound is *not for a specific function* as per usual streaming lower bounds. It must hold for all cryptographically secure functions realizing the same primitive.

Possibility results. Given that one-way functions exist in NC^1, e.g. based on factoring or lattice assumptions, we construct one-way functions and pseudorandom generators by streaming algorithms that use: internal memory of size $O(\log n)$, 2 RW streams (one contains the input and the other one is auxiliary), 5 passes in total for one-way functions and 7 for pseudorandom generators.

Starting from the Learning With Errors (LWE) assumption [8] and based on the constructions in [8,7] we construct an Indistinguishable under Chosen-Plaintext Attack (IND-CPA) secure (or semantically secure [21]) Public-Key

[1] Logarithmic memory size precludes uninteresting trivialities that can happen for size $\omega(\log n)$, when one assumes the existence of very hard functions. In this case, in principle the question is not about streaming.

Encryption (PKE), where both the encryption and the decryption are streaming algorithms. This is mainly a feasibility result. Improved key-lengths, and a CCA-secure PKE are very interesting open questions.

For the existence of streaming one-way functions and pseudorandom generators, the assumption can be relaxed to existence of one-way functions in $\mathsf{Logspace}$. For PKE we rely on LWE, a concrete intractability assumption.

Relation to previous work. The construction of streaming one-way functions from an arbitrary one-way function in NC^1 relies on Barrington's characterization [20]. In particular, computing a boolean NC^1 function reduces in some very local sense to computing the product $\sigma_1\sigma_2\cdots\sigma_m$ of permutations from $\mathsf{Sym}(5)$, the symmetric group over 5 letters. Kilian [23] encoded $\sigma = \sigma_1\cdots\sigma_m$ as $\hat{\sigma} = \langle\sigma_1 r_1, r_1^{-1}\sigma_2 r_2, \ldots, r_{m-1}^{-1}\sigma_m\rangle$ for uniformly random chosen r_i's. Then, given $\hat{\sigma}$ we can efficiently "decode" σ, whereas no additional information about $\sigma_1, \cdots, \sigma_m$ can be extracted. The seminal work of Applebaum-Ishai-Kushilevitz [10,9,11,3] provides a paradigm for dealing with general forms of similar *randomized encodings*.

Constructing a pseudorandom generator is more complex. One could have tried to implement as a streaming algorithm the steps of the celebrated [14] construction. Indeed, such a streaming pseudorandom generator is *non-trivially* achievable given certain entropy parameters of hashed values. But, there is no obvious way to streaming-compute these values, neither can it be circumvented by creating many copies as in [14]. We instead adapt [5,16] that bypass this obstacle and also buys us efficiency over [14]. Both [5] and [16] can be non-trivially modified into streaming algorithms. We use [5] because it is simpler and gives better parameters.

Regarding Public Key Encryption systems, we base our construction on [8,7], where the original constructions are not streaming computable.

Why streaming cryptography is not immediate from concrete hardness assumptions, such as lattice assumptions? By modifying [17] we see that multiplying a matrix by a vector requires $\Omega(\log n)$ many passes if the number of streams is constant and the internal memory logarithmic. This limitation is circumvented by taking randomized encodings of NC^1 computations of such functions (these are non-black-box constructions since the computation itself is encoded). We note that [12] ruled out families of black-box streaming cryptography constructions and it conjectured impossibility of streaming cryptography with a constant number of passes, which we refute in this paper. Thus, the possibility of streaming cryptography is unexpected.

One more reason that makes streaming cryptography counter-intuitive is that no single-stream algorithm with internal memory size $O(\log n)$ and $O(1)$-many passes computes a one-way function. However, by adding a second stream we can bring the number of passes down to a constant, and this strongly contrasts folk wisdom[2] in streaming computation.

[2] It was believed that for common types of functions, if when adding a second tape helps then permuting the input in the single-stream model will help as well. But a permuted one-way function is also one-way.

The multiple read/write (RW) stream model we consider here is closely related to the reversals-parameterized Turing Machines [2,18], except that we only make forward scans. To the best of our knowledge, before our work for $o(\log n)$ many passes in the literature there were only lower bounds, e.g. [19,17]. This multiple stream model generalizes the single-stream model, aka "online model", which has been scrutinized for quite a while. In the study of randomness [15] gives lower bounds in the single-stream model as well as constructions for online universal hash functions, extractors, and condensers. Also, a restricted form of the single-stream model with read-only (RO) input was studied in [13,6].

Table 1. Cryptography with Streaming Algorithms vs Cryptography in NC^0

	Streaming Model	NC^0
one-way function pseudorandom generator (PRG)	✓	✓
PKE (**Enc** & **Dec**)	✓	×
linear-stretch PRG	**?**	from Alekhnovitch's assumption [22]
super-linear-stretch PRG	×	**?**

Streaming Cryptography vs NC^0 and Locality. Streaming cryptography and [10,9,1,11,3] rely on randomized encodings, but they are incomparable in a number of places. There are obvious things streaming algorithms can do (e.g. sample almost uniformly from $\mathsf{Sym}(5)$) but NC^0 cannot, whereas, generally, NC^0 functions with underlying dependency graphs of $\mathsf{poly}(n)$ treewidth cannot be computed by streaming algorithms. This holds in particular for circuits associated with cellular automata (CA) as they appear in [3], where the treewidth is $\Omega(\sqrt{n})$. Furthermore, there are concrete technical separations between streaming and highly parallel cryptography. For example, IND-CPA secure PKE with streaming encryption and decryption exists whereas no NC^0 decryption is possible (and for AC^0 is still open). The CAs constructions are based on the concrete DRLC assumption (see Section 7), whereas even for [1] (these CAs make a single step that makes them a special case of constant input/output locality circuits [1]) it is impossible to start from general encodings. Our streaming private-key primitives are from generic assumptions.

2 Preliminaries

We use capital bold letters, e.g. $\mathbf{A}$, to denote matrices, and use lower case bold letters, e.g. $\mathbf{x}$, for column vectors, and correspondingly $\mathbf{x}^T$ for row vectors. Let $\mathbb{Z}_q := \mathbb{Z}/q\mathbb{Z} = \{0, 1, 2, \cdots, q-1\}$ be the ring of integers with addition and multiplication modulo q.

Probability distributions are denoted by calligraphic letters, e.g. $\mathcal{D}$. We use $x \leftarrow \mathcal{D}$ to denote that x is sampled from $\mathcal{D}$, and $x \in_R S$ when x is sampled uniformly from the set S. $\mathcal{U}_n$ denotes the uniform distribution over $\{0,1\}^n$ and $\mathcal{U}_S$ is the uniform distribution over the set S.

In this work all complexity classes are function classes (but we prefer to write e.g. NC^0 instead of FNC^0). $\mathsf{Logspace}$ denotes the set of all functions computable by a Turing Machine (transducer) with a read-only input, $O(\log n)$ large working tape, and a write-only output tape.

For $i \in \mathbb{Z}^{\geq 0}$, NC^i denotes the set of functions computable by families of poly-size boolean circuits with constant fan-in gates and $O(\log^i n)$ depth for input length n. A family of circuits is (log-space) uniform if there is a (log-space) Turing machine such that on input 1^n, $n \in \mathbb{Z}^{\geq 0}$, it generates the description of the corresponding circuit in the family.

An (s, p, t) *streaming algorithm* is a Turing machine that has internal memory $s(n)$, t-many unbounded external RW streams which we can scan from left to right for $p(n)$ passes. Unless mentioned otherwise for a *streaming algorithm* $s(n) = O(\log n)$, $t = 2$, and $p = O(1)$. A function is *streaming computable* if it can be computed by a streaming algorithm. An *oblivious* streaming algorithm is one where the head movement and the internal memory depend only on the time step.

Private and public key primitives. $f : \{0,1\}^* \to \{0,1\}^*$ is a (T, ϵ)-*secure one-way function* for $T = T(n), \epsilon = \epsilon(n)$ if f is polynomial time computable and for all sufficiently large n, for every time T randomized algorithm $\mathcal{A}$, we have $\Pr_{y \leftarrow f(\mathcal{U}_n)}[f\big(\mathcal{A}(y)\big) = y] < \epsilon$.

A polynomial time computable function $G : \{0,1\}^* \to \{0,1\}^*$ is a (T, ϵ)-*pseudorandom generator* if $\forall x, |G(x)| > |x|$ and $G(\mathcal{U}_n)$ is (T, ϵ)-*pseudorandom*, i.e. for sufficiently large n and for every time T randomized algorithm $\mathcal{D}$, there is $\big|\Pr[\mathcal{D}(G(\mathcal{U}_n)) = 1] - \Pr[\mathcal{D}(\mathcal{U}_{|G(1^n)|}) = 1]\big| < \epsilon$. For simplicity, we omit (T, ϵ) for computational security, i.e. when $T = n^{\omega(1)}, \epsilon = n^{-\omega(1)}$.

A *public-key encryption (PKE) system* consists of three polynomial time algorithms **KeyGen**, **Enc**, **Dec**, for key-generation, encryption and decryption respectively, where the key-generation and the encryption are probabilistic. (i) **KeyGen** takes the security parameter 1^n as input and it generates a public encryption key $\mathcal{PK}$ and a private decryption key $\mathcal{SK}$; (ii) **Enc** outputs a ciphertext c on input $(\mathcal{PK}, m)$, for every m drawn from the message space; (iii) **Dec** takes $(\mathcal{SK}, c)$ as input to decrypt m from c with overwhelming probability over the random choices of **Enc**. We say that a PKE system is streaming computable if both **Enc** and **Dec** are streaming algorithms.

In a PKE system, IND-CPA security is defined in the following security experiment as a game between a Challenger and an Adversary:

- The challenger runs **KeyGen** and uses its random choices to generate a public $\mathcal{PK}$ and a private $\mathcal{SK}$ key, and reveals the $\mathcal{PK}$ to the adversary.
- The adversary chooses two equal-length messages $\mathbf{x}_0$ and $\mathbf{x}_1$, and sends them to the challenger.
- The challenger flips an unbiased coin $b \in_R \{0,1\}$, computes $\mathbf{c} = \mathbf{Enc}(\mathcal{PK}, \mathbf{x}_b)$ and gives the ciphertext $\mathbf{c}$ to the adversary.
- The adversary outputs $b' \in \{0,1\}$ based on $\mathcal{PK}$ and $\mathbf{c}$, and it wins if and only if $b' = b$.

Here the adversary is a probabilistic polynomial time algorithm. The PKE is *IND-CPA secure* if for every $k \in \mathbb{R}$, we have $\Pr[b' = b] < \frac{1}{2} + \frac{1}{N^k}$, when $N = |\mathbf{x}_i|$ is sufficiently large.

Randomized Encoding. For a function $f : \{0,1\}^n \to \{0,1\}^m$, the function $\hat{f} : \{0,1\}^n \times \{0,1\}^\rho \to \{0,1\}^{m'}$ is a *randomized encoding* of f if the following conditions hold

1. For every $x \in \{0,1\}^n$, the output distribution $\hat{f}(x,\mathcal{U}_\rho)$ uniquely determines a $f(x)$, i.e. $\hat{f}(x,r) \neq \hat{f}(x',r')$ for any r, r', as long as $f(x) \neq f(x')$.
2. The output distribution is fully determined by the encoded value $f(x)$, i.e. if $f(x) = f(x')$ then $\hat{f}(x,\mathcal{U}_\rho)$ and $\hat{f}(x',\mathcal{U}_\rho)$ are identically.
3. $|\rho| = \mathsf{poly}(n)$ and there are $\mathsf{poly}(n)$-time algorithms to decode $f(x)$ from any sample in $\hat{f}(x,\mathcal{U}_\rho)$, and to sample from $\hat{f}(x,\mathcal{U}_\rho)$ when given $f(x)$.

Intuitively, (1) means that $\hat{f}(x,r)$ contains all information about $f(x)$, and (2) asserts that $\hat{f}(x,\mathcal{U}_\rho)$ reveals no extra information about x other than the value of $f(x)$. Putting these two together we have that if f is a one-way function then $\hat{f}$ is also one-way [10].

3 Warm-Up: How to Construct Streaming One-Way Functions?

We present a generic compiler (Section 3.2) that maps every $f \in \mathsf{NC}^1$ to its streaming randomized encoding $\hat{f}$. Due to a very useful coincidence regarding the specific encoding we use, and after a little "massaging" we get $\hat{f}$ computable with 2 streams (the reader is encouraged to think ahead to see where the issue is). Corollary 1 immediately follows by [10].

Theorem 1. *Every function $f \in \mathsf{NC}^1$ has a randomized encoding function $\hat{f}$ which is oblivious streaming computable with 5 passes.*

Corollary 1. *A streaming one-way function exists if a one-way function exists in* Logspace.

Here is an advanced remark. The construction in the proof of Theorem 1 relies on a specific randomized encoding that also causes a polynomial blow-up compared to the regular output of Barrington (see below). unavoidable (for this technique) and why the AIK encoding [10] cannot be used.

3.1 Background: NC^1 to Width-5 Branching Programs

Let us now recall the definition of a bounded-width permutation branching program.

Definition 1. *A width-w* permutation branching program *is a sequence of $m = m(n)$ instructions $B_n = (s_1, \langle j_1, \sigma_1, \tau_1\rangle) \cdots (s_m, \langle j_m, \sigma_m, \tau_m\rangle)$, where for every $1 \le i \le m$, $j_i \in \{1, \cdots, n\}$, $\sigma_i, \tau_i \in \mathsf{Sym}(w)$. Here $\mathsf{Sym}(w)$ refers to the group of permutations over $[w] = \{1, 2, \cdots, w\}$. On input $x = (x_1, \cdots, x_n) \in \{0,1\}^n$, B_n is evaluated as $B_n(x) = s_1 \cdot s_2 \cdot \cdots \cdot s_m$, where $s_i = \sigma_i$ if $x_{j_i} = 1$ and $s_i = \tau_i$ if $x_{j_i} = 0$.*

A function $f : \{0,1\}^n \to \{0,1\}$ is recognized by B_n if there exists a cycle $\theta \in \mathsf{Sym}(w)$, such that $\forall x \in \{0,1\}^n$, $B_n(x) = \theta$ when $f(x) = 1$, and $B_n(x) = e$ is the identity permutation when $f(x) = 0$.

Everything in the following theorem holds as well for log-space uniform branching programs.

Theorem 2 (Barrington's Theorem [20]). *Any boolean function f computable by a family of depth d and fan-in 2 circuits can be recognized by a family of width-5 permutation branching programs for $m \le 4^d$. In particular, $m = \mathsf{poly}(n)$ for $f \in \mathsf{NC}^1$ and input length n.*

Thus, evaluating $f : \{0,1\}^n \to \{0,1\}$ on input x reduces to deciding whether $B_n(x) = s_1 s_2 \cdots s_m$ is the identity. Define $\hat{f} : \{0,1\}^n \times \mathsf{Sym}(5)^{m-1} \to \mathsf{Sym}(5)^m$ as $\hat{f}(x; r) = \langle \pi_1, \cdots, \pi_m\rangle = \langle s_1 r_1, r_1^{-1} s_2 r_2, \cdots, r_{m-2}^{-1} s_{m-1} r_{m-1}, r_{m-1}^{-1} s_m\rangle$, where $r_i \in_R \mathsf{Sym}(5)$, s_i follows the i-th instruction in B_n and m is the length of B_n. Then, $\hat{f}$ is a randomized encoding of f, since $\langle \pi_1, \cdots, \pi_m\rangle$ uniformly distributes over $\mathsf{Sym}(5)^m$ conditioned on $\pi_1 \pi_2 \cdots \pi_m = s_1 s_2 \cdots s_m$.

We define **Sample** : $\{0,1\}^q \to \mathsf{Sym}(5)$ to be the algorithm that samples $r_i \in_R \mathsf{Sym}(5)$ within statistical distance $2^{-\Omega(q)}$ using $q = q(n)$ (read-once) random bits. Then, every permutation in $\mathsf{Sym}(5)$ is identified by its unique binary ID from $\{0,1\}^7$. Thus, $\hat{f}$ is represented in binary as $\widehat{f} : \{0,1\}^n \times (\{0,1\}^q)^{m-1} \to \left(\{0,1\}^7\right)^m$ that induces a loss of at most $2^{-\Omega(q(n))}$ in the output distribution. It remains to make $\widehat{f}$ streaming computable. The issue is that non-consecutive s_i's may be arbitrarily associated with the same input bit, so we must do something about this.

3.2 Streaming Computable Randomized Encoding

Our streaming algorithm is based on the following observations:

- fixing any poly-time invertible permutation ψ over $\{1, \ldots, m\}$, $g(x; r) = \langle \pi_{\psi(1)}, \cdots, \pi_{\psi(m)}\rangle$ is a one-way function as long as $\hat{f}$ is a one-way function, recalling that $\hat{f}(x; r) = \langle \pi_1, \cdots, \pi_m\rangle$;
- a permutation branching program (e.g. B_n) recognizes exactly the same function after inserting *dummy instructions* like $(s, < j, e, e >)$; that is, $(s_1, \langle j_1, \sigma_1, \tau_1\rangle) \cdots (s_m, \langle j_m, \sigma_m, \tau_m\rangle)$ recognizes exactly the same function as $(s_1, \langle j_1, \sigma_1, \tau_1\rangle) \cdots (s, < j, e, e >) \cdots (s_m, \langle j_m, \sigma_m, \tau_m\rangle)$.

Due to space limitations we omit the (not hard) full proof of Theorem 1. Here is a sufficiently detailed outline. By the second observation,we may replace

in B_n, the length m branching program that recognizes f, the first instruction $\big(s_1, \langle j_1, \sigma_1, \tau_1\rangle\big)$ with $\big(s'_1, \langle 1, e, e\rangle\big) \cdots \big(s'_{j_1}, \langle j_1, \sigma_1, \tau_1\rangle\big) \cdots \big(s'_n, \langle n, e, e\rangle\big)$, i.e. $s'_{j_1} = s_1$, whereas $s'_i = e$ for $i \neq j_1$, so that $s'_1 \cdots s'_n = s_1$ for every input. The advantage of the new instructions is that $\forall i \in \{1, 2, \ldots, n\}$, s'_i depends on exactly x_i. With similar tricks for $s_2, \cdots, s_m$, we get a length $mn = \mathsf{poly}(n)$ new branching program B'_n and $s'_1, \cdots, s'_{mn}$ with oblivious input dependency. In what follows, we use B_n and s_i instead of B'_n and s'_i for simplicity.

In a single pass, we compute the s_i's in the order: $s_1, s_{n+1}, \cdots, s_{mn-n+1}$, $s_2, s_{n+2}, \cdots, s_{mn-n+2}, \cdots\cdots, s_n, s_{2n}, \cdots, s_{mn}$ (sorted by their dependency on x, which coincide the subscripts modular n). Then, for $f : \{0,1\}^n \to \{0,1\}$, we apply the first observation to construct the oblivious streaming computable randomized encoding $\widehat{f} : \{0,1\}^n \times (\{0,1\}^q)^{mn-1} \to \big(\{0,1\}^7\big)^{mn}$ as follows

$$\widehat{f}(x; y_1, \ldots, y_{mn-1}) =$$
$$\Big\langle s_1 r_1,\ r_n^{-1} s_{n+1} r_{n+1}, \cdots, r_{mn-n}^{-1} s_{mn-n+1} r_{mn-n+1} \cdots, \cdots, r_{mn-1}^{-1} s_{mn} \Big\rangle$$

where $r_i = \mathbf{Sample}(y_i)$, r_i^{-1} is the inverse of r_i, and s_i is a function of $x_{(i \mod n)}$ for $i = 1, 2, \cdots, mn$.

When $f(x) = \langle f_1(x), f_2(x), \cdots, f_{\ell(n)}(x)\rangle$ has $\ell(n)$ output bits, we design $\hat{f} = \langle \hat{f}_1, \cdots, \hat{f}_{\ell(n)}\rangle$, which consists of an individual randomized encoding $\hat{f}_i$ one for each f_i. It is not too hard to globally rearrange the output bits and obtain the final streaming computable function $\widehat{f}$.

4 Streaming Pseudorandom Generators

The encoding in Section 3.2 does not preserve pseudorandomness, simply because $2^7 \nmid |\mathsf{Sym}(5)|$. In fact, Barrington's theorem holds also for the non-solvable $\mathsf{Sym}(w)$, $w \geq 5$ but there is no $k \in \mathbb{Z}$ such that $2^k | (w!)$. Yet, we provide a rather technical adaptation of [5] to build streaming pseudorandom generators from any streaming one-way function f.

Theorem 3. *Let $f : \{0,1\}^n \to \{0,1\}^m$ be a streaming one-way function. Then, there is a streaming computable pseudorandom generator G requiring 2 additional passes to the streaming algorithm for $\langle f^{(1)}, \cdots, f^{(\ell t)}\rangle$, for ℓ, t defined as below*

Moreover, if $m = O(n)$, then $\ell = \frac{n}{\log n}, t = O(n^2 \log^2 n)$, and the seed length of G is $O(n^6 \log^3 n)$.

In fact, the construction in Section 3.2 gives an oblivious streaming one-way function, which implies that evaluating polynomial many copies of f does not need more passes than f.

There are four steps in the [5] construction: 1) next-block pseudoentropy generation; 2) entropy equalization; 3) converting Shannon entropy to min-entropy and amplifying the gap; 4) randomness extraction. The first three steps remain intact for our streaming algorithm, whereas the fourth has to be modified.

In a nutshell, the first step constructs the generator $G_{nb}(\mathbf{s}) = \langle f(\mathbf{s}), \mathbf{s}\rangle$ for a random seed $s \leftarrow \mathcal{U}_n$. For notation convenience, let $f : \{0,1\}^n \to \{0,1\}^{m-n}$, so that $G_{nb} : \{0,1\}^n \to \{0,1\}^m$. The second step concatenates the outputs of G_{nb} on ℓ independent seeds to get $z^{(1)}, \ldots, z^{(\ell)}$, and randomly shifts (by discarding from the head and the tail) blocks to convert total entropy into the entropy in individual blocks, via $EQ : \{0,1\}^{\log m} \times (\{0,1\}^m)^\ell \to \{0,1\}^{m_\ell}$ for $m_\ell = m(\ell-1)$ and

$$EQ\left(j, z^{(1)}, \cdots, z^{(\ell)}\right) := \left\langle z_j^{(1)}, \cdots, z_m^{(1)}, \cdots, z_1^{(\ell)}, \cdots, z_{j-1}^{(\ell)} \right\rangle$$

Let $\mathcal{X} := EQ\left(J, G_{nb}(\mathcal{U}_n^{(1)}), \cdots, G_{nb}(\mathcal{U}_n^{(\ell)})\right)$ for $J = \mathcal{U}_{\log m}$, the third step concatenates t independent copies of $\mathcal{X}$ within each block (we do this step virtually by allowing non-consecutive cells in one block), i.e. $\mathcal{Y} = \left\langle \left(\mathcal{X}_1^{(1)}, \cdots, \mathcal{X}_1^{(t)}\right), \cdots, \left(\mathcal{X}_{m_\ell}^{(1)}, \cdots, \mathcal{X}_{m_\ell}^{(t)}\right) \right\rangle$, thus every block has high pseudo-min-entropy conditioned on previous blocks. The details for these steps can be found in [5,16].

Now, let us give an informal but accurate description of the part that has to be non-trivially modified for the streaming construction. The fourth step requires evaluating a single random universal hash function $\mathbf{H} : \{0,1\}^t \to \{0,1\}^{\alpha_t - \log^2 n}$ on every block of $\mathcal{Y}$ to extract randomness, where α_t is the next-block pseudo-min-entropy of each block. This step is difficult since streaming algorithms cannot re-read the code of $\mathbf{H}$. To that end, we prove that each block can be associated with a different linear hash function $\mathbf{H}^{(i)} \in \{0,1\}^{t \times (\alpha_t - \log^2 n)}$. Then, in the streaming algorithm we use $\widehat{\mathbf{H}^{(i)}}$, the randomized encoding of $\mathbf{H}^{(i)}$, to extract randomness. Let $\mathbf{H}_j^{(i)}$ be the j-th row of $\mathbf{H}^{(i)}$, and $\mathbf{R}_j^{(i)}$ the j-th row of the random input $\mathbf{R}^{(i)} \in \{0,1\}^{(t-1)\times(\alpha_t - \log^2 n)}$, then $\widehat{\mathbf{H}^{(i)}}\left(\left(\mathcal{X}_i^{(1)}, \cdots, \mathcal{X}_i^{(t)}\right), \mathbf{R}^{(i)}\right) = \left\langle \mathcal{X}_i^{(1)}\mathbf{H}_1^{(i)} + \mathbf{R}_1^{(i)}, \ldots, \mathcal{X}_i^{(j)}\mathbf{H}_j^{(i)} + \mathbf{R}_{j-1}^{(i)} + \mathbf{R}_j^{(i)}, \ldots, \mathcal{X}_i^{(t)}\mathbf{H}_t^{(i)} + \mathbf{R}_{t-1}^{(i)} \right\rangle$ where all the additions are modular 2. Combining the four steps, G appears as follows

$$\begin{aligned}
&G\left(\mathbf{J}^{(1)}, \cdots, \mathbf{J}^{(t)}, \mathcal{U}_n^{(1)}, \cdots, \mathcal{U}_n^{(t\ell)}; \mathbf{H}; \mathbf{R}\right) \\
=&\Big\langle \mathbf{H},\ \mathcal{X}_1^{(1)}\mathbf{H}_1^{(1)} + \mathbf{R}_1^{(1)}, && \cdots, \mathcal{X}_{m_\ell}^{(1)}\mathbf{H}_1^{(m_\ell)} + \mathbf{R}_1^{(m_\ell)}, \\
&\cdots, \mathcal{X}_1^{(j)}\mathbf{H}_j^{(1)} + \mathbf{R}_{j-1}^{(1)} + \mathbf{R}_j^{(1)}, && \cdots, \mathcal{X}_{m_\ell}^{(j)}\mathbf{H}_j^{(m_\ell)} + \mathbf{R}_{j-1}^{(m_\ell)} + \mathbf{R}_j^{(m_\ell)}, \\
&\cdots, \mathcal{X}_1^{(t)}\mathbf{H}_t^{(1)} + \mathbf{R}_{t-1}^{(1)}, && \cdots, \mathcal{X}_{m_\ell}^{(t)}\mathbf{H}_t^{(m_\ell)} + \mathbf{R}_{t-1}^{(m_\ell)} \Big\rangle
\end{aligned}$$

Note that we use a family of linear hash functions because it is not clear how to implement with streaming algorithms the description-succinct hash family in [5,16]. This causes loss in efficiency (which contrasts the purpose of [5,16]), but here we strive for a streaming feasibility result which is not at all obvious how to get. Theorems 1 and 3, and [10] yield:

Corollary 2. *If there is a one-way function in* Logspace*, then there exists a pseudorandom generator which is streaming computable with* 7 *passes.*

5 Limitations for Super-Linear Stretch Pseudorandom Generators

We devise a new lower bounding methodology, which a central technical contribution of this work. We first use this to show that streaming computable super-linear stretch pseudorandom generators do not exist. Note that $n^{1-\varepsilon}$ stretch is easy to achieve by running in parallel $n^{1-\varepsilon}$ copies of a single-bit stretch pseudorandom generator on independent seeds.

Theorem 4. *Suppose $\ell(n) = \omega(n)$ and $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ is a pseudorandom generator. Then, no streaming algorithm can compute G.*

We prove Theorem 4 by analyzing the information flow in the computation, and by partitioning appropriately the output into blocks, we upper bound the entropy transferred to each output block from the input. Intuitively, a single block cannot collect much entropy and therefore it cannot induce large stretch.

Our proof makes use of the following observation and the concept of a dependency graph originally introduced in [19,17] (in fact, [24]). We tailor them for cryptographic applications to partition the computation into $p + 1$ phases corresponding to p passes.

Observation 5 ([24]). *When a tape cell is written, its content only depends on the internal memory and the t cells currently being scanned by the heads of the external streams. Moreover, those t cells are written before this pass, since no cell can be visited twice before making a new pass.*

5.1 Dependency Graphs and Dependency Trees

First, we provide the definition of dependency graph. Due to space limitations we omit concrete examples and diagrams of dependency graphs and trees.

Definition 2. *Fix a streaming algorithm G which on input x it makes $\leq p$ passes over t external tapes. The* dependency graph*, denoted by $\Gamma(x)$, is a directed graph with $p + 1$ levels. Each beginning of a new pass (at any tape) is associated with a distinct level. The i-th level in $\Gamma(x)$ contains all nodes labeled (v, i) if the tape cell v* has ever been visited *before the i-th pass begins. We assume that all input cells are* written *at the beginning. (e.g. $\{(v, 1)|$ v is an input cell$\}$ contains exactly all the nodes in level 1, and $\{(v, p+1)|$ v is written in the computation of G on input x $\}$ for level $p+1$.) $\Gamma(x)$ has edges $(u, i) \to (v, i+1)$ iff there is a head reading (u, i) when $(v, i+1)$ is being written. Furthermore, there is always an edge $(u, i) \to (u, i+1)$ as long as (u, i) is in $\Gamma(x)$.*

In the dependency graph, each level represents a single phase in the computation. Therefore the nodes (except for those at level 1) have in-degree at most t, while all edges are heading to the next level. Intuitively, those directed edges depict the information flow excluding the internal memory.

We also remark it possible that old passes are not yet finished when a new pass begins. In this case, old passes will be processed in the new level.

5.2 Overview of the Lower-Bound

We first introduce the definition of *blocks*. Intuitively, blocks are used to package the entropy from the input, and the dependency of blocks describes the information flow during the computation, except for the information carried in the bounded internal memory. By analyzing the dependency of blocks we have devised an elegant information-theoretic way of upper bounding the amount of entropy in part of the output.

Definition 3. *A* block *is an equivalence class consisting of all nodes corresponding to tape cells at the same level on the same tape such that they depend on exactly the same set of blocks at the previous level. Specifically, an* input block *refers to a set of nodes at the first level corresponding to consecutive tape cells on the input tape.*

Note that if two cells, from the same tape and the same level, have the same dependency on blocks at the previous level, then any cell in between would have exactly the same dependency, because the dependency changes "monotonically". Therefore, our partition of blocks is well-defined such that every block consists of only consecutive tape cells.

Then, we partition the input x into b input blocks as $x = (x_1, x_2, \cdots, x_b)$, and use a corollary of Proposition 3.1 in [19] to bound the number of blocks.

Proposition 1. *Partition x into b input blocks and let $\Gamma(x)$ be the dependency graph. Then, the number of blocks at level i in $\Gamma(x)$ is bounded $\leq (b+1)t^{i-1}$, where t denotes the number of tapes.*

Due to space limitation, here we give only a proof sketch of Theorem 4.

Proof (Proof Sketch of Theorem 4). By Proposition 1, there are at most $(b+1)t^p = O(b)$ blocks at level $p+1$ because both t, p are constants.

We consider $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ where $\ell(n) = \omega(n)$. Partition the input equally into $b = \lceil n/\log n \rceil$ blocks, so that each input block has length $\leq \log n$. Recalling that there are $O(b)$ output blocks, there is an output block v with $\ell(n)/O(b) = \omega(\log n)$ many bits in expectation.

However, every output block depends on $O(1)$ input blocks when t, p are both constants. That is, the block v has expected length $\omega(\log n)$, while it only receives $O(\log n)$ bits of entropy from $O(1)$ input blocks plus another $O(\log n)$ bits from the internal memory. This immediately suggests an advised distinguisher $\mathcal{D}_A$:

$\mathcal{D}_A$: DISTINGUISHING $G(\mathcal{U}_n)$ FROM $\mathcal{U}_{\ell(n)}$ (with advice A):

1 For all $z \in A$, check whether z is a sub-string of the input;
2 If find any $z \in A$ in the input, output 1;
3 Otherwise output 0.

The advice A is a $\mathsf{poly}(n)$ long list containing all strings that are sufficiently long (i.e. $\Omega(\ell(n)/b) = \omega(\log n)$) and could appear in the block v. $\mathcal{D}_A\big(G(\mathcal{U}_n)\big) = 1$

if the block v is sufficiently long to be captured by A, which happens with probability $\Omega(1/b)$ by Markov's inequality. On the other hand, $\Pr[\mathcal{D}_A(\mathcal{U}_{\ell(n)}) = 1] < \mathsf{poly}(n) \cdot \ell(n) \cdot 2^{-\Omega(\ell(n)/b)} = 2^{-\omega(n)}$. Therefore

$$\Pr\Big[\mathcal{D}_A\big(G(\mathcal{U}_n)\big) = 1\Big] - \Pr\Big[\mathcal{D}_A(\mathcal{U}_{\ell(n)}) = 1\Big] \geq \Omega(\frac{1}{b}) - 2^{-\omega(n)} = \Omega(\log n/n)$$

For a uniform distinguisher, the advice A is efficiently generated as follows: enumerate every input block and internal memory on every possible dependency tree of v, then simulate the computation of v, and add only sufficiently long output substrings to the list. Although suffering from a polynomial blow-up than optimal, such advice A suffices for the above argument of $\mathcal{D}_A$.

6 Public-Key Encryption in the Streaming Model

We construct an IND-CPA secure PKE system based on Regev's LWE assumption together with the PKE construction in [8], where the encryption and the decryption algorithms are streaming algorithms, henceforth called *streaming PKE*. The private keys contain a good deal of redundancy. We also show that large private-keys are necessary. A lower bound on the length of the private key is given in Theorem 7 (using the technique introduced in Section 5), though there is still a gap with our construction.

Theorem 6. *Given the decision-LWE assumption (Assumption 1), the construction in Section 6.1 is an IND-CPA secure PKE. Moreover, both the encryption and the decryption algorithms are streaming computable.*

Theorem 7. *For every IND-CPA secure PKE whose decryption scheme is a streaming algorithm, the private-key has length $\Omega(N)$, where N is the length of the plaintext.*

The main challenge of a streaming PKE is the decryption algorithm. The techniques we developed so far do not apply, because the decryption algorithm should output exactly the plaintext rather than any code.

We construct our streaming PKE based on the decision-LWE assumption. The intuition of such assumption is exposited in [8], which also gives reductions from worst-case lattice problems (by now these lattice assumptions and reductions are common place).

Definition 4 (LWE problem). *Let $q = q(n) \leq \mathsf{poly}(n)$, consider a list of equations $b_i = \langle \mathbf{s}, \mathbf{a}_i \rangle + e_i \pmod q$ for $i = 1, 2, \cdots, \mathsf{poly}(n)$, where $\mathbf{s} \in \mathbb{Z}_q^n$, $\mathbf{a}_i \in_R \mathbb{Z}_q^n$ and $b_i \in \mathbb{Z}_q$. If furthermore $e_i \in \mathbb{Z}_q$ follows a discrete Gaussian distribution*[3] *with parameter α, we denote by* search-$\mathsf{LWE}_{q,\alpha}$ *the problem of recovering $\mathbf{s}$ from such equations. In* decision-$\mathsf{LWE}_{q,\alpha}$ *the goal is to distinguish $(\mathbf{a}, \langle \mathbf{s}, \mathbf{a} \rangle + e \pmod q)$ from $\mathcal{U}_{\mathbb{Z}_q^{n+1}}$ with non-negligible advantage, when both $\mathbf{s}, \mathbf{a} \in_R \mathbb{Z}_q^n$.*

[3] A discrete Gaussian distribution over $\mathbb{Z}_q$ is defined by $D_{\mathbb{Z}_q,\alpha}(x) = \rho_\alpha(x/q)/\rho_\alpha(\mathbb{Z}_q)$, where $\rho_\alpha(x) = \sum_{k=-\infty}^{\infty} \alpha^{-1} \exp(-\pi(\frac{x+k}{\alpha})^2)$ follows a continuous Gaussian distribution, and $\rho_\alpha(\mathbb{Z}_q) = \sum_{x \in \mathbb{Z}_q} \rho_\alpha(x/q)$.

Assumption 1 (cf. [8,7]). *When $\alpha \geq 2\sqrt{n}$, search-$\mathsf{LWE}_{q,\alpha}$ cannot be solved in probabilistic polynomial time with non-negligible probability. If $\alpha \geq \omega(\sqrt{n}\log n)$ then decision-$\mathsf{LWE}_{q,\alpha}$ cannot be solved in probabilistic polynomial time with non-negligible advantage.*

6.1 The Construction

In our construction the public and private keys are "streaming useable" forms of the following two matrices: $\mathbf{A}$ and a random matrix $\mathbf{D}$. Matrix $\mathbf{A}$ is statistically close to uniform, and at the same time orthogonal to $\begin{bmatrix}\mathbf{I}\\ \mathbf{D}\end{bmatrix}$. The latter consists of short vectors which cannot be retrieved from a uniformly random matrix (this is the lattice hardness assumption).

KeyGen: Pick a matrix $\mathbf{D} \in \mathbb{Z}_p^{(m-w)\times w}$ uniformly at random from $\{0, \pm 1\}^{(m-w)\times w}$. Uniformly at random pick $\overline{\mathbf{A}} \in \mathbb{Z}_q^{n\times(m-w)}$, and compute $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ as $\mathbf{A} = [-\overline{\mathbf{A}}\mathbf{D} \mid \overline{\mathbf{A}}] \mod q$. Let $\begin{bmatrix}\mathbf{I}\\ \mathbf{D}\end{bmatrix} = [\mathbf{d}_1, \cdots, \mathbf{d}_w]$. Here $k = \lceil 2\log n \rceil, q = 2^k, m = 3nk, w = nk$, for the security parameter n.

Output N copies of $\mathbf{A}$ as the public key, and nN copies of $\mathbf{d}_1, \cdots, \mathbf{d}_w$ as the private key. Each copy of $\mathbf{A}$ is written in row-first order, i.e. $(a_{11}, a_{12}, \cdots, a_{1m}, a_{21}, \cdots, a_{2m}, \ \cdots, \ a_{n1}, \cdots, a_{nm})$.

Enc: On input $\mathbf{x} = (x^{(1)}, \cdots, x^{(N)}) \in \{0,1\}^N$, for $i = 1, 2, \cdots, N$, uniformly choose $\mathbf{s}_i \in_R \mathbb{Z}_q^n$ and $\mathbf{x}_i \in_R \{qx^{(i)}/2\} \times \mathbb{Z}_q^{w-1}$.

Sample $\mathbf{e}_i \in \mathbb{Z}_q^m$ for $i = 1, 2, \cdots, N$, where each entry $e_{ij} \sim \mathcal{D}_\alpha$ follows the discrete Gauss distribution with mean 0 and standard deviation α, for $j = 1, 2, \cdots, m$.

For every $i = 1, 2, \cdots, N$, sequentially output y_i, where y_i is a randomized encoding of $\mathbf{s}_i^T\mathbf{A} + \mathbf{e}_i^T + (\mathbf{x}_i^T, \mathbf{0}) \mod q$. That is, for $\mathbf{R} \in_R \mathbb{Z}_q^{(n-1)\times m}$, realizing $\mathbf{e}_i^T, (\mathbf{x}_i^T, \mathbf{0})$ as $1 \times m$ row vectors, and recalling that $\mathbf{A}$ is an $n \times m$ matrix, we define y_i is the row-first order of $\mathbf{Y}_i$ as follows

$$\mathbf{Y}_i = \begin{bmatrix} s_{i1} & & \\ & \ddots & \\ & & s_{in} \end{bmatrix} \cdot \mathbf{A} + \begin{bmatrix} \mathbf{R} \\ (\mathbf{x}^T, \mathbf{0}) \end{bmatrix} + \begin{bmatrix} \mathbf{e}_i^T \\ -\mathbf{R} \end{bmatrix}$$

Dec: Given the ciphertext $\{y_i\}_{i=1,2,\cdots,N}$ and the decryption key nN copies of $\mathbf{d}_1$. We compute $b = [1\ 1\ \cdots\ 1]_{1\times n}\mathbf{Y}_i\mathbf{d}_1 \mod q$ and output $x^{(i)} = \lfloor 2b/q + 1/2 \rfloor \mod 2$ for every $i = 1, \cdots, N$.

Comparison with [8]. The above construction is similar to the PKE construction in [8]. We borrow from [7] the key generation and encryption algorithms which enable us to turn them into streaming computable encryption/decryption. Note that [7], unlike us, achieves a CCA-secure PKE. Currently, we do not know how to perform ciphertext validity checks (as in e.g. [7]) in a streaming fashion.

This Public-Key Encryption scheme is statistically correct and IND-CPA secure, and it has both encryption and decryption in a streaming fashion.

7 Conclusions and Some Remarks on Practical Constructions

Our work leaves open the possibility of streaming cryptography for a number of popular private and public-key primitives. As a next step we propose to study the streaming possibility for the following cryptographic primitives: (i) linear-stretch pseudorandom generators, (ii) CCA-secure PKE, (iii) signature schemes, and (iv) message authentication.

It is also open whether the number of passes we achieve (see Table 2 below) are optimal, and also simultaneously improve the seed-efficiency of streaming pseudorandom generators from NC^1 one-way functions. For example, our generic streaming one-way function is done with 5 passes, whereas when starting from a concrete assumption (see below) we can do it with 4, which is optimal.

Some remarks on practicality. Randomized encodings generally use huge amounts of randomness (typically $\Omega(n^4)$) for input length n, and thus our generic compilers can be understood as feasibility results. In practice, starting from concrete intractability assumptions we can do much better. Here is a practical example which in fact resembles a lot the one in [3] (but a few model-specific differences – our model is not two dimensional but things are arranged similarly).

Assumption 2 (Decoding Random Linear Codes (DRLC)). *A* random linear code f_{code} *is defined as* $f_{code} : (\mathbf{A}, \mathbf{x}, \mathbf{e}) \mapsto (\mathbf{A}, \mathbf{Ax} + \mathbf{e})$, *where* $\mathbf{A} \in \mathsf{GF}(2)^{m \times n}$, $\mathbf{x} \in \mathsf{GF}(2)^n$, $\mathbf{e} \in \mathsf{GF}(2)^m$. *Choose positive constants* κ, ϵ, δ *such that* $\kappa = \frac{n}{m} < 1 - H_2\big((1+\epsilon)\delta\big)$, *where* $H_2(p) = -p\log_2 p - (1-p)\log_2(1-p)$ *for* $p < 1/2$ *and* $H_2(p) = 1$ *otherwise. If* $\mathbf{A}, \mathbf{x}$ *are chosen uniformly at random, while* $\mathbf{e}$ *has at most* $\frac{\delta m}{2}$ *one-entries, then* f_{code} *is a one-way function.*

Theorem 8. *Suppose that the DRLC assumption holds true. Then, there exists a one-way function* F *computable by a streaming algorithm with* 2 *streams,* 4 *passes and* $O(\log n)$ *internal memory. Furthermore, if the DRLC input is of size* N *the corresponding input size for* F *is* $n \leq 2N$.

Proof (Construction outline). Suppose the random bits $(r_{11}, r_{21}, \cdots, r_{mn})$ are given on the extra stream (this is without loss of generality/not necessary), and parse the input stream as $(x_1, a_{11}, a_{21}, \cdots, a_{m1}, \cdots, x_n, a_{1n}, \cdots, a_{mn}, e_1, \cdots, e_m)$.

In the first pass (over two streams) we compute $(a_{11}x_1 + r_{11}, \cdots, a_{m1}x_1 + r_{m1}, a_{12}x_2 + r_{12}, \cdots, a_{m2}x_2 + r_{m2}, \cdots, a_{1n}x_n + r_{1n}, \cdots, a_{mn}x_n + r_{mn}, e_1, \cdots, e_m)$.

In the next pass we compute $(a_{11}x_1 + r_{11}, \cdots, a_{m1}x_1 + r_{m1}, a_{12}x_2 + r_{12} - r_{11}, \cdots, a_{m2}x_2 + r_{m2} - r_{m1}, \cdots, a_{1n}x_n + r_{1n} - r_{1(n-1)}, \cdots, a_{mn}x_n + r_{mn} - r_{m(n-1)}, e_1 - r_{1n}, \cdots, e_m - r_{mn})$. Thus, a randomized encoding of $\mathbf{Ax} + \mathbf{e}$ is computed with 4 passes over 2 streams.

Table 2. OWF & PRG from any OWF in Logspace; PKE from LWE

	# of passes	external tapes
one-way function	5	1 RO & 1 RW
pseudorandom generator	7	2 RW
	15	1 RO & 1 RW
PKE **Enc**	5	1 RO & 1 RW
PKE **Dec**	2	key & cipher in different tapes

We conclude with a note on the practicality of the multi-stream model. One physical analog of a stream is a hard-disk or a disk-array. Although it makes sense to think of physical disks to be of size $2n$ or $3n$, for an input of length n, under no stretch of imagination n^3 is reasonable size. For more than one stream we believe that this *stream-size* parameter should be added to the other parameters: number of streams, number of passes, internal memory size. In this paper all constructions make ≤ 9 passes and the stream size never exceeds $2 \times$ input length. In practice, though the stream size is even more important and in the sense that perhaps we might be able to tolerate slightly super-constant many passes given that the stream size stays linear throughout the computation.

References

1. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography with constant input locality. Journal of Cryptology, 429–469; In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 92–110. Springer, Heidelberg (2007)
2. Chen, J., Yap, C.-K.: Reversal complexity. SIAM Journal on Computing 20(4), 622–638 (1991)
3. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography by Cellular Automata or How Fast Can Complexity Emerge in Nature? In: ICS, pp. 1–19 (2010)
4. Impagliazzo, R., Levin, L.A., Luby, M.: In: Symposium on Theory of Computing (STOC), pp. 12–24 (1989)
5. Vadhan, S.P., Zheng, C.J.: Characterizing pseudoentropy and simplifying pseudorandom generator constructions. In: Symposium on Theory of Computing (STOC), pp. 817–836 (2012)
6. Yu, X., Yung, M.: Space Lower-Bounds for Pseudorandom-Generators. In: Structure in Complexity Theory Conference, pp. 186–197 (1994)
7. Micciancio, D., Peikert, C.: Trapdoors for lattices: Simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 700–718. Springer, Heidelberg (2012)
8. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Symposium on Theory of Computing (STOC), pp. 84–93 (2005)
9. Applebaum, B., Ishai, Y., Kushilevitz, E.: Computationally Private Randomizing Polynomials and Their Applications. Computational Complexity 15(2), 115–162 (2006)
10. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography in NC^0. SIAM Journal of Computing (SICOMP) 36(4), 845–888 (2006)

11. Applebaum, B., Ishai, Y., Kushilevitz, E.: On pseudorandom generators with linear stretch in NC^0. Computational Complexity 17(1), 38–69 (2008)
12. Bronson, J., Juma, A., Papakonstantinou, P.A.: Limits on the stretch of non-adaptive constructions of pseudo-random generators. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 504–521. Springer, Heidelberg (2011)
13. Kharitonov, M., Goldberg, A.V., Yung, M.: Lower Bounds for Pseudorandom Number Generators. In: Foundations of Computer Science (FOCS), pp. 242–247 (1989)
14. Håstad, J., Impagliazzo, R., Levin, L.A., Luby, M.: A Pseudorandom Generator from any One-way Function. SIAM Journal of Computing (SICOMP) 28(4), 1364–1396 (1999)
15. Bar-Yossef, Z., Reingold, O., Shaltiel, R., Trevisan, L.: Streaming Computation of Combinatorial Objects. In: Annual IEEE Conference on Computational Complexity (CCC), vol. 17 (2002)
16. Haitner, I., Reingold, O., Vadhan, S.: Efficiency improvements in constructing pseudorandom generators from one-way functions. In: Symposium on Theory of Computing (STOC), pp. 437–446 (2010)
17. Grohe, M., Hernich, A., Schweikardt, N.: Lower bounds for processing data with few random accesses to external memory. Journal of the ACM 56(3): Art. 12, 58 (2009)
18. Hernich, A., Schweikardt, N.: Reversal complexity revisited. Theoretical Computer Science 401(1-3), 191–205 (2008)
19. Beame, P., Huynh, T.: The Value of Multiple Read/Write Streams for Approximating Frequency Moments. ACM Transactions on Computation Theory 3(2), 6 (2012)
20. Barrington, D.A.: Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. Journal of Computer and System Sciences 38(1), 150–164 (1989)
21. Goldwasser, S., Micali, S.: Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information. In: Symposium on Theory of Computing (STOC), pp. 365–377 (1982)
22. Alekhnovich, M.: More on average case vs approximation complexity. In: Foundations of Computer Science (FOCS), pp. 298–307 (2003)
23. Kilian, J.: Founding cryptography on oblivious transfer. In: Symposium on Theory of Computing (STOC), pp. 20–31 (1988)
24. Grohe, M., Schweikardt, N.: Lower bounds for sorting with few random accesses to external memory. In: Symposium on Principles of Database Systems (PODS), pp. 238–249 (2005)