Memory-Hard Puzzles in the Standard Model with Applications to Memory-Hard Functions and Resource-Bounded Locally Decodable Codes

Mohammad Hassan Ameri[1][0000−0002−2415−3285], Alexander R. Block[1][0000−0002−2632−763X], and Jeremiah Blocki[1⋆][0000−0002−5542−4674]

Purdue University {mameriek,block9,jblocki}@purdue.edu

Abstract. We formally introduce, define, and construct *memory-hard puzzles*. Intuitively, for a difficulty parameter t, a cryptographic puzzle is memory-hard if any parallel random access machine (PRAM) algorithm with "small" cumulative memory complexity ($\ll t^2$) cannot solve the puzzle; moreover, such puzzles should be both "easy" to generate and be solvable by a sequential RAM algorithm running in time t. Our definitions and constructions of memory-hard puzzles are in the standard model, assuming the existence of indistinguishability obfuscation ($i\mathcal{O}$) and one-way functions (OWFs), and additionally assuming the existence of a *memory-hard language*. Intuitively, a language is memory-hard if it is undecidable by any PRAM algorithm with "small" cumulative memory complexity, while a sequential RAM algorithm running in time t can decide the language. Our definitions and constructions of memory-hard objects are the first such definitions and constructions in the standard model without relying on idealized assumptions (such as random oracles).

We give two applications which highlight the utility of memory-hard puzzles. For our first application, we give a construction of a (one-time) *memory-hard function* (MHF) in the standard model, using memory-hard puzzles and additionally assuming $i\mathcal{O}$ and OWFs. For our second application, we show any cryptographic puzzle (e.g., memory-hard, time-lock) can be used to construct *resource-bounded locally decodable codes* (LDCs) in the standard model, answering an open question of Blocki, Kulkarni, and Zhou (ITC 2020). Resource-bounded LDCs achieve better rate and locality than their classical counterparts under the assumption that the adversarial channel is resource bounded (e.g., a low-depth circuit). Prior constructions of MHFs and resource-bounded LDCs required idealized primitives like random oracles.

1 Introduction

Memory-hardness is an important notion in the field of cryptography that is used to design egalitarian proofs of work and to protect low entropy secrets (e.g., passwords) against brute-force attacks. Over the last decade, there has been a

⋆ Corresponding author

rich line of both theoretical and applied work in constructing and analyzing memory-hard functions [43,7,16,2,4,6,8,28,3,5,25,34]. Ideally, one wants to prove that any algorithm evaluating the function (possibly on multiple distinct inputs) has high cumulative memory complexity (cmc) [7] (asymptotically equivalent to the notions of (amortized) Space-Time complexity and (amortized) Area-Time complexity in idealized models of computation [4]). Intuitively, the cmc of an algorithm A_f evaluating a function f on input x (denoted by $\mathsf{cmc}(A_f, x)$) is the summation of the amount of memory used by A_f during every step of the computation. Currently, security proofs for memory-hard objects rely on idealized assumptions such as the existence of random oracles [7,6,8,5] or other ideal objects such as ideal ciphers or permutations [34]. Informally, a function f is memory-hard if there is a sequential algorithm computing f in time t, but any parallel algorithm computing f (possibly on multiple distinct inputs) has high cmc, e.g., $t^{2-\varepsilon}$ for small constant $\varepsilon > 0$. An important open question is to construct provably secure memory-hard objects in the standard model.

In this work, we focus specifically on *memory-hard puzzles*. Cryptographic puzzles are cryptographic primitives that have two desirable properties: (1) for a target solution s, it should be "easy" to generate a puzzle Z with solution s; and (2) solving the puzzle Z to obtain solution s should be "difficult" for any algorithm $\mathcal{A}$ with "insufficient resources". Such puzzles have seen a wide range of applications, including using in cryptocurrency, handling junk mail, and constructing time-released encryption schemes [41,83,60,77]. For example, the well-known and studied notion of *time-lock puzzles* [83,29,44,73,19,74] requires that for difficulty parameter t and security parameter λ, a sequential (i.e., non-parallel) machine can generate a puzzle in time $\mathrm{poly}(\lambda, \log(t))$ and solve the puzzle in time $t \cdot \mathrm{poly}(\lambda)$, but requires that any parallel algorithm running in sequential time significantly less than t (i.e., any polynomial size circuit of depth smaller than t) cannot solve the puzzle, except with negligible probability (in the security parameter). In the context of memory-hard puzzles, we want to ensure that the puzzles are easy to generate, but that any algorithm solving the puzzle has high cmc. More concretely, we require that the puzzles can be generated (resp., solved) in time $\mathrm{poly}(\lambda, \log(t))$ (resp., $t \cdot \mathrm{poly}(\lambda)$) on a sequential machine while any algorithm solving the puzzle has cmc at least $t^{2-\varepsilon}$ for small constant $\varepsilon > 0$. We remark that any sequential machine solving the puzzle in time at most $t \cdot \mathrm{poly}(\lambda)$ will have cmc at most $t^2 \cdot \mathrm{poly}(\lambda)$ so a lower bound of $t^{2-\varepsilon}$ for the cmc of our puzzles would be nearly tight.

In this work, we ask the following questions:

Is it possible to construct memory-hard puzzles under standard cryptographic assumptions? If yes, what applications of memory-hard puzzles can we find?

1.1 Our Results

We formally introduce and define the notion of *memory-hard puzzles*. Inspired by time-lock puzzles and memory-hard functions, we define memory-hard puzzles *without* idealized assumptions. Intuitively, we say that a cryptographic puzzle

is memory-hard if any parallel random access machine (PRAM) algorithm with "small" cmc cannot solve the puzzles. This is in contrast with time-lock puzzles which require that any algorithm running in "small" sequential time (i.e., any low-depth circuit) cannot solve the puzzle. For both memory-hard and time-lock puzzles, the puzzles should be "easy" to generate; i.e., in sequential time $\mathrm{poly}(\lambda, \log(t))$.

Similar to the time-lock puzzle construction of Bitansky et al. [19], we construct memory-hard puzzles assuming the existence of a suitable *succinct randomized encoding scheme* [58,12,17,19,70,11,46], and the additional assumption that there exists a language which is "suitably" memory-hard. Towards this end, we formally introduce and define *memory-hard languages*: such languages, informally, require that (1) the language is decidable by a family of *uniformly succinct circuits*—succinct circuits which are computable by a uniform algorithm —of appropriate size; and (2) any PRAM algorithm deciding the language must have "large" cmc. We discuss the technical ideas behind our construction in Section 2.2 and present its memory-hardness in Theorems 2 and 3.

We stress that our construction does not rely on an explicit instance of a memory-hard language: the *existence* of such a language suffices to prove memory-hardness of the constructed puzzle, mirroring the construction of [19]. We use succinct randomized encoding scheme of Garg and Srinivasan [46], which is instantiated from indistinguishability obfuscation ($i\mathcal{O}$) for circuits and somewhere statistically binding hash functions [57,67,78].[1] We remark that our constructions are primarily of theoretical interest, as known constructions of randomized encodings rely on expensive primitives such as $i\mathcal{O}$ [13,45,67,69,14,1,33,59]. We make no claims about the practical efficiency of our constructions.

It is important to note that even if we defined memory-hard puzzles in an idealized model (e.g., the random oracle model), memory-hard functions do not directly yield memory-hard puzzles. Cryptographic puzzles stipulate that for parameters t and λ the puzzle generation algorithm needs to run in time $\mathrm{poly}(\lambda, \log(t))$. However, using a memory-hard function to generate a cryptographic puzzle would require the generation algorithm to compute the memory-hard function, which would yield a generation algorithm running in time (roughly) proportional to $t \cdot \mathrm{poly}(\lambda)$.

Application 1: Memory-Hard Functions. We demonstrate the power of memory-hard puzzles via two applications. For our first application, we use memory-hard puzzles to construct a (one-time secure) *memory-hard function* (MHF) in the standard model. As part of this construction, we formally define (one-time) memory-hard functions in the standard model, without idealized primitives; see Definitions 7 and 8. We emphasize that all prior constructions of memory-hard functions rely on idealized primitives such as random oracles

[1] Such hash functions generate a hashing key that statistically binds the i-th input bit. For example, a hash output y may have many different preimages, but all preimages have the same i-th bit. Construction of such hash functions exist under standard cryptographic assumptions such as DDH and LWE, among others [78].

[43,7,16,2,4,6,8,28,3,5,25] or ideal ciphers and permutations [34]. In fact, prior definitions of memory-hardness were with respect to an idealized model such as the parallel random oracle model, e.g., [7].

Recall that a function f is memory-hard if it can be computed by a sequential machine in time t (and thus uses space at most t), but any PRAM algorithm evaluating f (possibly on multiple distinct inputs) has large cumulative memory complexity (cmc); e.g., at least $t^{2-\varepsilon}$ for small constant $\varepsilon > 0$. One-time security stipulates that for any input x, any attacker with low cmc cannot distinguish between $(x, f(x))$ and (x, r) with non-negligible advantage when r is a uniformly random bit string.[2] Assuming the existence of indistinguishability obfuscation, puncturable pseudo-random functions, and memory-hard puzzles, we give a construction of one-time secure memory-hard functions. We discuss the technical ideas of our MHF construction in Section 2.3 and present its memory-hardness in Theorem 4.

We stress that, to the best of our knowledge, this is the *first* construction of a memory-hard function under standard cryptographic assumptions and the additional assumption that a memory-hard puzzle exists. Given our construction of a memory-hard puzzle, we construct memory-hard functions from standard cryptographic assumptions additionally assuming the existence of a memory-hard language. or ideal cipher and permutation models.

We also conjecture that our scheme is multi-time secure as well: if an attacker with low cmc, say some g, cannot compute $f(x)$ for given input x, then an attacker with cmc at most $m \cdot g$ cannot compute $f(x_i)$ for m distinct inputs $x_1, \ldots, x_m$. However, we are unable to formally prove this due to some technical barriers in the security proof. At a high level, this is due to the fact that allowing the attacker to have higher cmc (e.g., $m \cdot g$) eventually leads to an attacker with large enough cmc to simply solve the underlying memory-hard puzzle that is used in the MHF construction, thus allowing the adversary to distinguish instances of the MHF instance. See Section 2.3 for discussion.

Application 2: LDCs for Resource-Bounded Channels. We use cryptographic puzzles to construct efficient *locally decodable codes for resource-bounded channels* [26]. A (ℓ, δ, p)-locally decodable code (LDC) $C[K, k]$ over some alphabet Σ is an error-correcting code with encoding function $\mathsf{Enc}\colon \Sigma^k \to \{0,1\}^K$ and probabilistic decoding function $\mathsf{Dec}\colon \{1, \ldots, k\} \to \Sigma$ satisfying the following properties. For any message x, the decoder, when given oracle access to some $\tilde{y}$ such that $\Delta(\tilde{y}, \mathsf{Enc}(x)) \leqslant \delta K$, makes at most ℓ queries to its oracle and outputs x_i with probability at least p, where Δ is the Hamming distance. The *rate* of the code is k/K, the *locality* of the code is ℓ, the *error tolerance* is δ, and the *success probability* is p. Classically (i.e., the adversarial channel introducing errors is computationally unbounded), there is an undesirable trade-off between the rate k/K and locality, e.g., if $\ell = \text{polylog}(k)$ then $K \gg k$.

[2] Our one-time security definition differs from those in prior literature (e.g., [7,6]), and is, in fact, stronger. See Section 2.3 for discussion.

Modeling the adversarial channel as computationally unbounded may be overly pessimistic. Moreover, it has been argued that any real world communication channel can be reasonably modeled as a resource-bounded channel [71,26]. A *resource-bounded channel* is an adversarial channel that is assumed to have some constrained resource (e.g., the channel is a low-depth circuit), and a resource-bounded LDC is a LDC that is resilient to errors introduced by some class of resource-bounded channels $\mathbb{C}$. Arguably, error patterns (even random ones) encountered in nature can be modeled by some (not necessarily known) resource-bounded algorithm which simulates the same error pattern, and thus these channels are well-motivated by real world channels. For example, sending a message from Earth to Mars takes between (roughly) 3 and 22 minutes when traveling at the speed of light; this limits the depth of any computation that could be completed before the (corrupted) codeword is delivered. Furthermore, examining LDCs resilient against several resource-bounded channels has led to better trade-offs between the rate and locality than their classical counterparts [71,76,15,50,84,24]. Recently, Blocki, Kulkarni, and Zhou [26] constructed LDCs for resource-bounded channels with locality $\ell = \text{polylog}(k)$ and constant rate $k/K = \Theta(1)$, but their construction relies on random oracles.

We use cryptographic puzzles to modify the construction of [26] to obtain resource-bounded LDCs without random oracles. Given any cryptographic puzzle that is secure against some class of adversaries $\mathbb{C}$, we construct a locally decodable code for Hamming errors that is secure against the class $\mathbb{C}$, resolving an open problem of Blocki, Kulkarni, and Zhou [26]. We discuss our LDC construction in Section 2.4 and present its memory-hardness in Corollary 1. We can instantiate our LDC with any (concretely secure) cryptographic puzzle. In particular, the time-lock puzzles of Bitansky et al. [19] directly give us LDCs secure against small-depth channels, and our memory-hard puzzle construction gives us LDCs secure against any channel with low cmc. Our LDC construction for resource bounded Hamming channels can also be extended to resource-bounded insertion-deletion (InsDel) channels by leveraging recent "Hamming-to-InsDel" LDC compilers [80,21,20]. See discussion in Section 2.4 and Corollary 2.

Challenges in Defining Memory-Hardness. Defining the correct machine model and cost metric for memory-hard puzzles is surprisingly difficult. As PRAM algorithms and cmc are used extensively in the study of MHFs, it is natural to use the same machine model and cost metric. However, cmc introduces subtleties in the analysis of our memory-hard puzzle construction: like [19], we rely on parallel amplification in order to construct an adversary which breaks our memory-hard language assumption. While parallel amplification does not significantly increase the depth of a computation (the metric used by [19]), any amplification *directly increases* the cmc of an algorithm by a multiplicative factor proportional to the number of amplification procedures performed. This requires careful consideration in our security reductions.

One may also attempt to define memory-hard languages as languages with cmc at least $t^{2-\varepsilon}$, for small constant $\varepsilon > 0$, that are also decidable by single-tape

Turing machines (à la [19]) in time t, rather than by uniformly succinct circuit families. However, we demonstrate a major hurdle towards this definition. In particular, we show that any single-tape Turing machine running in time t can be simulated by any PRAM algorithm with cmc $O(t^{1.8} \cdot \log(t))$; see Section 2.1 for discussion and Theorem 1 for our formal theorem. Taking this approach, we could not hope obtain memory hard puzzles with cmc at least $t^{2-\varepsilon}$ for small ε as we can rule out the existence of memory-hard languages with $\mathsf{cmc} \gg t^{1.8}$. To contrast, under our uniformly succinct definition, we can provide a concrete candidate language with cmc plausibly as high as $t^{2-\varepsilon}$ such that the language is also decidable by a uniformly succinct circuit family of size $\widetilde{O}(t)$.[3] Furthermore, we show that our definition is essentially minimal, i.e., we can use memory-hard puzzles to construct memory-hard languages under the modest assumption that the puzzle solving algorithm is uniformly succinct; see discussion in Section 2.1 and Proposition 1.

1.2 Prior Work

Cryptographic puzzles are functions which require some specified amount of resources (e.g., time or space) to compute. Time-lock puzzles, introduced by Rivest, Shamir, and Wagner [83] extending the study of timed-released cryptography of May [75], are puzzles which require large sequential time to solve: any circuit solving the puzzle has large depth. [83] proposed a candidate time-lock puzzle based on the conjectured sequential hardness of exponentiation in RSA groups, and the proposed schemes of [29,44] are variants of this scheme. Mahmoody, Moran, and Vadhan [73] give a construction of *weak* time-lock puzzles in the random oracle model, where "weak" says that both a puzzle generator and puzzle solver require (roughly) the same amount of computation, whereas the standard definition of puzzles requires the puzzle generation algorithm to be much more efficient than the solving algorithm. Closer to our work, Bitansky et al. [19] construct time-lock puzzles using succinct randomized encodings, which can be instantiated from one-way functions, indistinguishability obfuscation, and other assumptions [46]. Recently, Malavolta and Thyagarajan [74] introduce and construct homomorphic time-lock puzzles: puzzles where one can compute functions over puzzle solutions without solving them. Continued exploration of indistinguishability obfuscation has pushed it closer and closer to being instantiated from well-founded cryptographic assumptions such as learning with errors [59].

Memory-hard functions (MHFs), introduced by Percival [81], have enjoyed rich lines of both theoretical and applied research in construction and analysis of these functions [34,7,8,16,43,2,4,5,6,28,3,25]. The security proofs of all prior MHF candidates rely on idealized assumptions (e.g., random oracles [7,6,8,5,27]) or other ideal objects (e.g., ideal ciphers or permutations [34]). The notion of *data-independent* MHFs—MHFs where the data-access pattern of computing the function, say, via a RAM program, is independent of the input—has also

[3] In fact, one can provably show that the cmc is $t^{2-\varepsilon}$ in the random oracle model.

been widely explored. Data-independent MHFs are attractive as they provide natural resistance to side-channel attacks. However, building data-independent memory-hard functions (iMHFs) comes at a cost: *any* iMHF has amortized space-time complexity at most $O(N^2 \cdot \text{loglog}(N)/\log(N))$ [2], while data-dependent MHFs were proved to have maximal complexity $\Omega(N^2)$ in the parallel random oracle model [6] (here, N is the run time of the honest sequential evaluation algorithm). Recently, Ameri, Blocki, and Zhou [10] introduced the notion of *computationally data-independent* memory-hard functions: MHFs which appear data-independent to a computationally bounded adversaries. This relaxation of data-independence allowed [10] to circumvent known barriers in the construction of data-independent MHFs as long as certain assumptions on the tiered memory architecture (RAM/cache) hold.

LDC constructions, like all code constructions, generally follow one of two channel models: the Hamming channel where worst-case bit-flip error patterns are introduced, and the Shannon channel where symbols are corrupted by an independent probabilistic process. Probabilistic channels may be too weak to capture natural phenomenon, while Hamming channels often limit achievable code constructions. For the Hamming channel, the channel is assumed to have unbounded power. Protecting against unbounded errors is desirable but often has undesirable trade-offs. For example, current constructions of LDCs with efficient (i.e., poly-time) encodings an obtain any constant rate $R < 1$, are robust to $\delta < (1 - R)$-fraction of errors, but have query complexity $2^{O(\sqrt{\log n \log\log n})}$ for codeword length n [65]. If one instead focuses on obtaining low query complexity, one can obtain schemes with codewords of length sub-exponential in the message size while using a constant number $q \geqslant 3$ queries [87,40,42]. These undesirable trade-offs have lead to a long line of work examining LDCs (and codes in general) with relaxed assumptions [71,76,15,50,84,24]. Two relaxations closely related to our work are due to Ostrovsky, Pandey, and Sahai [79] and Blocki, Kulkarni, and Zhou [26]. [79] introduce and construct *private* Hamming LDCs: locally decodable codes in the secret key setting, where the encoder and decoder share a secret key that is unknown to the (unbounded) channel. Blocki, Kulkarni, and Zhou [26] analyze Hamming LDCs in the context of *resource-bounded channels*. The LDC construction of [26] bootstraps off of the private Hamming LDC construction of [79], obtaining Hamming LDCs in the random oracle model assuming the existence of functions which are uncomputable by the channel.

While Hamming LDCs have enjoyed decades of research [61,86,40,42,62,65,66,87,88], the study of *insertion-deletion* LDCs (or InsDel LDCs) remains scarce. An InsDel LDC is a LDC that is resilient to adversarial insertion-deletion errors. In the non-LDC setting, there has been a rich line of research into insertion-deletion codes [68,64,51,55,48,49,56,54,32,37,36,38,53,52,85,35,39,47,72], and only recently have efficient InsDel codes with asymptotically good information rate and error tolerance been well-understood [54,52,53,47,72]. Ostrovsky and Paskin-Cherniavsky [80] and Block et al. [21] give a compiler which transforms any Hamming LDC into an InsDel LDC with a poly-logarithmic blow-up in the locality. Block and Blocki [20] extend the compiler of [21] to the private and resource-bounded

settings. Recently, Blocki et al. [22] give lower bounds for InsDel LDCs with constant locality: they show that (1) any 2-query InsDel LDC must have exponential rate; (2) 2-query linear InsDel LDCs do not exist; and (3) for any constant $q \geqslant 3$, a q-query InsDel LDC must have rate that is exponential in existing lower bounds for Hamming LDCs.

2 Technical Overview

Our construction of memory-hard puzzles relies on two key technical ingredients. First we require the existence of a language $\mathcal{L} \subseteq \{0,1\}^*$ that is suitably *memory-hard*. Given such a language, we additionally require *succinct randomized encodings* [17,70,46] for succinct circuits. With these two objects, we construct *memory-hard puzzles*. Both of our memory-hard objects are defined with respect to *parallel random access machine* (PRAM) algorithms and *cumulative memory complexity* (cmc). We say that an algorithm A is a *PRAM algorithm* if during each time-step of the computation, the algorithm has an internal state and can read from multiple positions from memory, perform a computation, then write to multiple positions in memory. Recall that $\mathsf{cmc}(A, x)$ is the summation of the memory used by $A(x)$ during every time step of the computation, and $\mathsf{cmc}(A, \lambda) = \max_{x:\,|x|=\lambda} \mathsf{cmc}(A, x)$. Moreover, for a function y, we say that $\mathsf{cmc}(A) < y$ if $\mathsf{cmc}(A, \lambda) < y(\lambda)$ for all $\lambda \in \mathbb{N}$. We note that even though we define cmc as a maximum, in all of our memory-hard definitions we quantify over all adversaries, and thus capture worst-case hardness.

We discuss the key ideas and present our main results in the remainder of this section. Section 2.1 presents our formal definition of memory-hard languages and a discussion on the plausibility and necessity of this assumption. Section 2.2 presents our formal definition of memory-hard puzzles and presents an overview of our construction assuming the existence of a memory-hard language and a succinct randomized encoding scheme. Section 2.3 presents an overview of our construction of a (one-time secure) memory-hard functions assuming the existence of indistinguishability obfuscation, one-way functions, and memory-hard puzzles. Finally, Section 2.4 presents our construction of resource-bounded locally decodable codes from any cryptographic puzzle.

2.1 Memory-Hard Languages

Our definition of memory-hard languages is inspired by the notion of non-parallelizing languages,[4] which are required by Bitansky et al. [19] to construct time-lock puzzles (also using succinct randomized encodings). We define our memory-hard languages with respect to a particular language class that requires the notion of *uniformly succinct circuits*. Informally, a circuit family $\{C_{t,\lambda}\}_{\lambda\in\mathbb{N}}$ is *succinct* if there exists a smaller circuit family $\{C'_{t,\lambda}\}_{\lambda\in\mathbb{N}}$ such that for every $t \in \mathbb{N}$: (1) $|C'_{t,\lambda}| = \mathrm{polylog}(|C_{t,\lambda}|)$; and (2) on input gate number g of $C_{t,\lambda}$

[4] Informally, a language is non-parallelizing if any polynomial sized circuit deciding the language has large depth.

the circuit $C'_{t,\lambda}(g)$ outputs the indices of the input gates of g and the function f_g computed by gate g. Furthermore, we say that a succinct circuit family is *uniformly succinct* if there additionally exists a sequential algorithm running in time $\mathrm{poly}(|C'_{t,\lambda}|)$ that outputs the description of the succinct circuit $C'_{t,\lambda}$ for every λ. We capture the formal definitions below, beginning with succinct circuits.

Definition 1 (Succinct Circuits [18,46]). *Let $C\colon \{0,1\}^n \to \{0,1\}^m$ be a circuit with $N-n$ binary gates. The gates of the circuit are numbered as follows. The input gates are given numbers $\{1,\ldots,n\}$. The intermediate gates are numbered $\{n+1, n+2, \ldots, N-m\}$ such that for any gate g with inputs from gates i and j, the label for g is bigger than i and j. The output gates are numbered $\{N-m+1,\ldots,N\}$. Each gate $g \in \{n+1,\ldots,N\}$ is described by a tuple $(i,j,f_g) \in [g-1]^2 \times \mathsf{GType}$ where the outputs of gates i and j serve as inputs to gate g and f_g denotes the functionality computed by gate g. Here, GType denotes the set of all binary functions $f : \{0,1\}^2 \to \{0,1\}$.*

We say that the circuit C is succinct *if there exists a circuit C^{sc} such that on input $g \in \{n+1, N\}$ outputs description (i,j,f_g) and $|C^{\mathsf{sc}}| < |C|$.*

For notational convenience, for any circuit C^{sc} that succinctly describes a larger circuit C, we define $\mathsf{FullCirc}(C^{\mathsf{sc}}) := C$ and $\mathsf{SuccCirc}(C) := C^{\mathsf{sc}}$. Next we give the definition uniformly succinct circuit families.

Definition 2 (Uniform Succinct Circuit Families). *We say that a circuit family $\{C_{t,\lambda}\}_{t,\lambda}$ is* succinctly describable *if there exists another circuit family $\{C^{\mathsf{sc}}_{t,\lambda}\}_{t,\lambda}$ such that $|C^{\mathsf{sc}}_{t,\lambda}| = \mathrm{polylog}(|C_{t,\lambda}|)$[5] and $\mathsf{FullCirc}(C^{\mathsf{sc}}_{t,\lambda}) = C_{t,\lambda}$ for every t,λ. Additionally, if there exists a PRAM algorithm A such that $A(t,\lambda)$ outputs $C^{\mathsf{sc}}_{t,\lambda}$ in time $\mathrm{poly}(|C^{\mathsf{sc}}_{t,\lambda}|)$ for every t,λ, then we say that $\{C_{t,\lambda}\}_{t,\lambda}$ is* uniformly succinct*.*

Given the notion of uniformly succinct circuits, we define our language class SC_t.

Definition 3 (Language Class SC_t). *Let t be a positive function. We define SC_t as the class of languages $\mathcal{L}$ decidable by a uniformly succinct circuit family $\{C_{t,\lambda}\}_\lambda$ such that there exists a polynomial p satisfying $|C_{t,\lambda}| \leqslant t \cdot p(\lambda, \log(t))$ for every λ and $t := t(\lambda)$.*

Given Definition 3, we define *memory-hard languages*. Intuitively, a language $\mathcal{L} \in \mathsf{SC}_t$ is memory-hard if any (PRAM) algorithm $\mathcal{B}$ that ε-decides $\mathcal{L}$ must have large cmc, where ε-decides here informally means that any probabilistic algorithm can decide the language $\mathcal{L}$ with advantage at least ε.

Definition 4 ((g,ε)-Memory Hard Language). *Let t be a positive function. A language $\mathcal{L} \in \mathsf{SC}_t$ is a (g,ε)-*memory hard language *if for every PRAM algorithm $\mathcal{B}$ with $\mathsf{cmc}(\mathcal{B},\lambda) < g(t(\lambda),\lambda)$, the algorithm $\mathcal{B}$ does not $\varepsilon(\lambda)$-decide $\mathcal{L}_\lambda$*

[5] For our purposes, we require the size of the succinct circuit to be poly-logarithmic in the size of the full circuit. One can easily replace this requirement with the requirement presented in Definition 1.

for every λ. *If* $\varepsilon(\lambda) = \mathsf{negl}(\lambda)$, *we say* $\mathcal{L}$ *is a* g-strong memory-hard language. *If* $\varepsilon(\lambda) \in (0, 1/2)$ *is a constant, we say* $\mathcal{L}$ *is a* (g, ε)-weakly memory-hard.

Note that one may define a weak memory-hard language with respect to $\varepsilon(\lambda) = 1/\operatorname{poly}(\lambda)$; however, this turns out to be essentially equivalent to $\varepsilon(\lambda) \in (0, 1/2)$. See the full version of our work [9] for a discussion. Moreover, our definition of memory-hard languages is essentially minimal, as one can construct memory-hard languages from memory-hard puzzles under the modest assumption that the puzzle solving algorithm is uniformly succinct. We prove the following proposition in the full version of our work [9].

Proposition 1. *Let* $\mathsf{Puz} = (\mathsf{Puz.Gen}, \mathsf{Puz.Sol})$ *be a* (g, ε)*-memory hard puzzle such that* $\mathsf{Puz.Sol}$ *is computable by a uniformly succinct circuit family* $\{C_{t,\lambda}\}_{t,\lambda}$ *of size* $|C_{t,\lambda}| \leqslant t \cdot \operatorname{poly}(\lambda, \log(t))$ *for every* λ *and difficulty parameter* $t := t(\lambda)$. *For language* $\mathcal{L}_{\mathsf{Puz}} := \{(Z, s) \colon s = \mathsf{Puz.Sol}(Z)\}$, *we have that* $\mathcal{L}_{\mathsf{Puz}} \in \mathsf{SC}_t$ *and is a* (g, ε)*-memory hard language.*

Plausibility of Memory-Hard Languages. We complement our definition of memory-hard languages by providing a concrete construction of a candidate memory-hard language. We define a language $\mathcal{L}_\lambda = \mathcal{L} \cap \{0,1\}^\lambda$ that is decidable by a uniformly succinct circuit $C_{t,\lambda}$ of size $t^2 \cdot \operatorname{polylog}(t)$. Our language relies on a hash function H, and under the idealized assumption that H is a random oracle, $\mathcal{L}_\lambda$ is provably memory-hard with cumulative memory complexity at least $t^2/\log(t)$.

Key to defining $\mathcal{L}_\lambda$ is a recent explicit construction of a depth-robust graph due to Blocki, Cinkoske, Lee, and Son [23]. Depth-robustness is a combinatorial property which is sufficient for constructing memory-hard functions in the parallel random oracle model [4]. Crucially, this graph is explicit and deterministic, and can be fully encoded by a uniformly succinct circuit. We remark that other randomized constructions of depth-robust graphs such the one used in the DRSample memory-hard function [3] cannot be used to construct memory-hard languages as the graphs are not uniformly succinct. We defer the reader to the full version of our work for more discussion [9].

We acknowledge that we only know how to prove our candidate language is memory-hard in the random oracle model or other idealized models of computation, which we are trying to avoid in our memory-hard puzzle construction. However, our memory-hard puzzle construction- does not require an explicit memory-hard language and our security proof holds as long as some memory-hard language exists. Thus, our goal is simply to establish a plausible candidate for such a language. We conjecture that our defined language will remain memory-hard when the random oracle is instantiated with a concrete cryptographic hash function such as SHA3. Proving that the conjecture holds in the standard model, however, would require major advances in the difficult field of complexity theory and circuit lower bounds. Moreover, assuming that all of our cryptographic assumptions hold, a concrete attack against our memory-hard puzzle construction would directly show that memory-hard languages do not exist, which is presumably a difficult problem in complexity theory.

PRAM Algorithms versus Turing Machines. One might try to define memory-hard languages to require they be decidable by a single-tape Turing machine rather than a PRAM algorithm. However, we show that if we require our memory-hard language to be decidable by a single-tape Turing machine in time $t = t(\lambda)$, then the language is only secure against PRAM algorithms with cmc *less than* $\widetilde{O}(t^{1.8})$. We show this by proving that any single-tape Turing machine running in time $t = t(\lambda)$ for λ-bit inputs can be simulated by a PRAM algorithm in time $O(t)$ using with space at most $O(t^{0.8} \cdot \log(t))$. As cmc is upper bounded by the maximum space of a computation times the maximum time of a computation, this implies that cmc is at most $O(t^{1.8} \cdot \log(t))$. We prove the following theorem in the full version of our work [9].

Theorem 1. *For any language $\mathcal{L}$ decidable in time $t(n)$ by a single-tape Turing machine for inputs of size n, there exists a constant $c > 0$ such that $\mathcal{L}$ is decidable by a PRAM algorithm with cmc at most $c \cdot t(n)^{1.8} \cdot \log(t(n))$.*

It is an interesting open question if such a reduction holds for multi-tape Turing machines; in particular, showing such a reduction for two-tape Turing machines would only strengthen our definition due to the reduction from multi-tape to two-tape Turing machines [82].

2.2 Memory-Hard Puzzles

We formally define *memory-hard puzzles*. Intuitively, a memory-hard puzzle is a cryptographic puzzle which requires any PRAM algorithm solving the puzzle to have large cmc. We give two flavors of memory-hard puzzles and begin with an asymptotically secure memory-hard puzzle.

Definition 5 (g-Memory Hard Puzzle). *A puzzle $\mathsf{Puz} = (\mathsf{Puz.Gen}, \mathsf{Puz.Sol})$ is a g-*memory hard puzzle *if there exists a polynomial t' such that for all polynomials $t > t'$ and for every PRAM algorithm $\mathcal{A}$ with $\mathsf{cmc}(\mathcal{A}) < y$ for the function $y(\lambda) := g(t(\lambda), \lambda)$, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$ and every pair $s_0, s_1 \in \{0,1\}^\lambda$ we have $|\Pr[\mathcal{A}(Z_b, Z_{1-b}, s_0, s_1) = b] - 1/2| \leqslant \mu(\lambda)$, where the probability is taken over $b \xleftarrow{\$} \{0,1\}$ and $Z_i \leftarrow \mathsf{Puz.Gen}(1^\lambda, t(\lambda), s_i)$ for $i \in \{0,1\}$.*

Note that for any difficulty parameter $t := t(\lambda)$ for security λ, we assume that $\mathsf{Puz.Sol}$ is computable in time $t \cdot \mathrm{poly}(\lambda)$ on a sequential RAM algorithm. This implies that there exists a PRAM algorithm A computing $\mathsf{Puz.Sol}$ has $\mathsf{cmc}(A, \lambda) \leqslant (t \cdot \mathrm{poly}(\lambda))^2 = t^2 \cdot \mathrm{poly}(\lambda)$. This yields an upper bound on the function g of Definition 5: take t to be any (large enough) polynomial. Then suitable values of g (ignoring $\mathrm{poly}(\lambda)$ factors) include $g = t^2/\log(t)$ or $g = t^{2-\theta}$ for small constant $\theta > 0$. In particular, we cannot expect to design g-memory hard puzzles for any function $g = \omega(t^2 \cdot \mathrm{poly}(\lambda))$ (by our definitions).

We complement Definition 5 with the following concrete security definition.

Definition 6 ((g, ε)-Memory Hard Puzzle). *A puzzle $\mathsf{Puz} = (\mathsf{Puz.Gen}, \mathsf{Puz.Sol})$ is a (g, ε)-*memory hard puzzle *if there exists a polynomial t' such that*

for all polynomials $t > t'$ and every PRAM algorithm $\mathcal{A}$ with $\mathsf{cmc}(\mathcal{A}) < y$ for $y(\lambda) := g(t(\lambda), \lambda)$, and for all $\lambda > 0$ and any pair $s_0, s_1 \in \{0,1\}^\lambda$, we have $|\Pr[\mathcal{A}(Z_b, Z_{1-b}, s_0, s_1) = b] - 1/2| \leqslant \varepsilon(\lambda)$, where the probability is taken over $b \xleftarrow{\$} \{0,1\}$ and $Z_i \leftarrow \mathsf{Puz.Gen}(1^\lambda, t(\lambda), s_i)$ for $i \in \{0,1\}$. If $\varepsilon(\lambda) = 1/\operatorname{poly}(\lambda)$, we say the puzzle is weakly memory-hard.

Similar to Definition 5, suitable values of g for Definition 6 include $g = t^2/\log(t)$ and $g = t^{2-\theta}$ for small constant $\theta > 0$, as any PRAM algorithm with cmc larger than $t^2 \cdot \operatorname{poly}(\lambda)$ can trivially break puzzles security simply by running the algorithm $\mathsf{Puz.Sol}$.

We note that in our security definition the adversary is given two puzzles Z_b, Z_{1-b} in random order along with both solutions s_0, s_1 (in the correct order). An alternate security definition would only give the adversary one puzzle, Z_b, and the solutions s_0, s_1. We remark that our security definition is at least as strong since the attacker can simply choose to ignore Z_{1-b}. It is an open question whether or not there is reduction in the other direction establishing tight concrete security guarantees. Thus, we choose to use the stronger definition.

We construct memory-hard puzzles by using succinct randomized encodings for succinct circuits and additionally assuming that a (suitable) memory-hard language exists. Informally, a succinct randomized encoding for succinct circuits consists of two algorithms $\mathsf{sRE.Enc}$ and $\mathsf{sRE.Dec}$ where $\widehat{C}_{x,G} \leftarrow \mathsf{sRE.Enc}(1^\lambda, C', x, G)$ takes as input a security parameter λ, a succinct circuit C' describing a larger circuit C with G gates and an input $x \in \{0,1\}^*$ and outputs a randomized encoding $\widehat{C}$ in time $\operatorname{poly}(|C'|, \lambda, \log(G), |x|)$. The decoding algorithm $\mathsf{sRE.Dec}(\widehat{C}_{x,G})$ outputs $C(x)$ in time at most $G \cdot \operatorname{poly}(\log(G), \lambda)$. Note that the running time requirement ensures $\mathsf{sRE.Enc}$ cannot simply compute $C(x)$. Intuitively, security implies that the encoding $\widehat{C}_{x,G}$ reveals nothing more than $C(x)$ to a computationally bounded attacker. Due to space constraints, we defer the formal definitions of both asymptotically secure and concretely secure succinct randomized encodings to the full version of our work [9].

We extend ideas from [19] to construct memory-hard puzzles from succinct randomized encodings; the formal construction is presented in the full version of our work [9]. The generation algorithm $\mathsf{Puz.Gen}(1^\lambda, t, s)$ first constructs a Turing machine $M_{s,t}$ that on any input runs for t steps then outputs s, where $t = t(\lambda)$ and $s \in \{0,1\}^\lambda$. This machine is then transformed into a succinct circuit $C'_{s,t}$ (via a transformation due to Pippenger and Fischer [82]), and then encodes this succinct circuit with our succinct randomized encoding; i.e., $Z = \mathsf{sRE.Enc}(1^\lambda, C'_{s,t}, 0^\lambda, G_{s,t})$. Here, $C'_{s,t}$ succinctly describes a larger circuit $C_{s,t}$ which is equivalent to $M_{s,t}$ (on inputs of size λ) and has $G_{s,t} := |C_{s,t}|$ gates. The puzzle solution algorithm simply runs the decoding procedure of the randomized encoding scheme; i.e., $\mathsf{Puz.Sol}(Z)$ outputs $s \leftarrow \mathsf{sRE.Dec}(Z)$.

Security is obtained via reduction to a suitable memory-hard language $\mathcal{L}$. If the security of the constructed puzzle is broken by an adversary $\mathcal{A}$ with small cmc, then we construct a new adversary $\mathcal{B}$ with small cmc which breaks the memory-hard language assumption by deciding whether $x \in \mathcal{L}$ with non-

negligible advantage. Suppose that $Z_0 \leftarrow \mathsf{Puz.Gen}(1^\lambda, t, s_0)$, $Z_1 \leftarrow \mathsf{Puz.Gen}(1^\lambda, t, s_1)$, b is a random bit, and $t := t(\lambda)$. If $\mathcal{A}(s_0, s_1, Z_b, Z_{1-b})$ can violate the MHP security and predict b with non-negligible probability, then we can construct an algorithm $\mathcal{B}$ with similar cmc that decides our memory-hard language. Algorithm $\mathcal{B}$ first constructs a uniformly succinct circuit $C_{a,a'}$ such that on any input x we have $C_{a,a'}(x) = a$ if $x \in \mathcal{L}$; otherwise $C_{a,a'}(x) = a'$ if $x \notin \mathcal{L}$. Our definition of memory-hard languages ensures that $C_{a,a'}$ is uniformly succinct and has size $G = t \cdot \mathrm{poly}(\lambda, \log(t))$. Let $C'_{a,a'}$ denote the smaller circuit that succinctly describes $C_{a,a'}$. The adversary computes $Z_i = \mathsf{sRE.Enc}(1^\lambda, C'_{s_i,s_{1-i}}, x, G)$ for $i \in \{0,1\}$, samples $b \xleftarrow{\$} \{0,1\}$, and obtains $b' \leftarrow \mathcal{A}(Z_b, Z_{1-b}, s_0, s_1)$. Our adversary $\mathcal{B}$ outputs 1 if $b = b'$ and 0 otherwise.

Observe that if $x \in \mathcal{L}$ then $\mathsf{Puz.Sol}(Z_0) = s_0$ and $\mathsf{Puz.Sol}(Z_1) = s_1$; otherwise if $x \notin \mathcal{L}$ then $\mathsf{Puz.Sol}(Z_0) = s_1$ and $\mathsf{Puz.Sol}(Z_1) = s_0$. By security of sRE, adversary $\mathcal{A}$ cannot distinguish between $Z_i = \mathsf{sRE.Enc}(1^\lambda, C'_{s_i,s_{1-i}}, x, G)$ and a puzzle generated with $\mathsf{Puz.Gen}$. Thus on input (Z_b, Z_{1-b}, s_0, s_1), the adversary $\mathcal{A}$ outputs $b' = b$ with non-negligible advantage. By our above observation, we have that $\mathcal{B}$ now (probabilistically) decides the memory-hard language $\mathcal{L}$ with non-negligible advantage. To obtain an adversary $\mathcal{B}'$ that deterministically decides $\mathcal{L}$, we use standard amplification techniques, along with the assumption of $\mathcal{B}'$ being a non-uniform algorithm (à la the argument for $\mathsf{BPP} \subset \mathsf{P}/\mathrm{poly}$). Whereas amplification—when performed in parallel—does not significantly increase the total computation depth, *any* amplification increases the cmc of an algorithm by a multiplicative factor proportional to the amount of amplification performed. Intuitively, this is because the cmc of an algorithm A is equal to the sum of the cmc of all sub-computations performed by A. We defer formal details to the full version of our work [9].

The memory-hardness of our construction relies on the particular succinct randomized encoding scheme used, and the existence of an appropriately memory-hard language. We again stress that the memory-hardness of our construction does not rely on an explicit instance of a memory-hard language, and the existence of such a language is sufficient for the above reduction to hold. We show that our construction satisfies two flavors of memory-hardness. First, given an asymptotically secure succinct randomized encoding scheme sRE and the existence of a *strong* memory-hard language, we show that there exists is an asymptotically secure memory-hard puzzle.

Theorem 2. *Let $t := t(\lambda)$ be a polynomial and let $g := g(t, \lambda)$ be a function. Let $\mathsf{sRE} = (\mathsf{sRE.Enc}, \mathsf{sRE.Dec})$ be a succinct randomized encoding scheme. If there exists a g'-strong memory-hard language $\mathcal{L} \in \mathsf{SC}_t$ for $g'(t, \lambda) := g + 2p_{\mathsf{sRE}}(\log(t), \lambda)^2 + 2p_{\mathsf{SC}}(\log(t), \log(\lambda))^2 + O(\lambda)$, then there exists a g-memory hard puzzle. Here, p_{sRE} and p_{SC} are fixed polynomials for the run-times of $\mathsf{sRE.Enc}$ and the uniform machine constructing the uniform succinct circuit of $\mathcal{L}$, respectively.*

To get a handle on Theorem 2, consider a large enough polynomial t such that $t \gg p_{\mathsf{sRE}}(\log(t), \lambda)$ and $t \gg p_{\mathsf{SC}}(\log(t), \log(\lambda))$. Then if there exists a g'-strong memory-hard language for $g'(t, \lambda) = t^2/\log(t)$, we obtain a g-memory

hard puzzle for $g(t,\lambda) = (1-o(1))\cdot g'(t,\lambda)$ (i.e., there is little loss in the memory-hardness of the constructed puzzle).

Next, assuming a concretely secure succinct randomized encoding scheme sRE and the existence of a *weak* memory-hard language, then there exists a weakly-secure memory-hard puzzle.

Theorem 3. *Let $t := t(\lambda)$ be a polynomial and let $g := g(t,\lambda)$ be a function. Let $\mathsf{sRE} = (\mathsf{sRE.Enc}, \mathsf{sRE.Dec})$ be a $(g, s, \varepsilon_{\mathsf{sRE}})$-secure succinct randomized encoding scheme for $g := g(t,\lambda)$ and $s(\lambda) := t \cdot \mathrm{poly}(\lambda, \log(t))$ such that p_{sRE} is a fixed polynomial for the runtime of $\mathsf{sRE.Enc}$. Let $\varepsilon := \varepsilon(\lambda) = 1/\mathrm{poly}(\lambda) > 3\varepsilon_{\mathsf{sRE}}(\lambda)$ be fixed. If there exists a $(g', \varepsilon_{\mathcal{L}})$-weakly memory-hard language $\mathcal{L} \in \mathsf{SC}_t$ for $g'(t,\lambda) := [g + 2p_{\mathsf{sRE}}(\log(t),\lambda)^2 + 2p_{\mathsf{SC}}(\log(t),\log(\lambda))^2 + O(\lambda)] \cdot \Theta(1/\varepsilon)$, and some constant $\varepsilon_{\mathcal{L}} \in (0, 1/2)$, then there exists a (g,ε)-weakly memory-hard puzzle. Here, p_{SC} is a fixed polynomial for the runtime of the uniform machine constructing the uniform succinct circuit for $\mathcal{L}$.*

Notice here we lose a factor of $\Theta(1/\varepsilon)$ when compared with Theorem 2. Concretely, using our same example from Theorem 2, if t is sufficiently large such that $t \gg p_{\mathsf{sRE}}(\log(t),\lambda)$ and $t \gg p_{\mathsf{SC}}(\log(t),\log(\lambda))$, and if $\varepsilon = 1/\lambda^2$, then for $g' = t^2/\log(t)$ we obtain a (g,ε)-weakly memory-hard puzzle for $g = g' \cdot \Theta(\lambda^2)$. This loss is due to the security reduction: our adversary performs amplification to boost the success probability of breaking the weakly memory-hard language assumption from ε to the constant $\varepsilon_{\mathcal{L}}$. To achieve constant $\varepsilon_{\mathcal{L}}$, one needs to amplify $\Theta(1/\varepsilon)$ times. As discussed previously, amplification directly incurs a multiplicative blow-up in the cmc complexity of a PRAM algorithm performing the amplification.

2.3 Memory-Hard Functions from Memory-Hard Puzzles

Using our new notion of memory-hard puzzles, we construct a one-time memory-hard function under standard cryptographic assumptions. To the best of our knowledge, this is the first such construction in the standard model; i.e., without random oracles [7] or other idealized primitives [34]. Recall that informally a function f is memory-hard if any PRAM algorithm computing f has large cmc. We define the *one-time* security of a memory-hard function f via the following game between an adversary and an honest challenger. First, before the game begins an input x is selected and provided to the challenger and the attacker. Second, the challenger computes $y_0 = f(x)$ and samples $y_1 \in \{0,1\}^\lambda$ and $b \xleftarrow{\$} \{0,1\}$ uniformly at random, and sends y_b. Then the attacker outputs a guess b' for b. We say that the adversary wins if $b' = b$, and say that f is (t,ε)-*one time secure* if for all inputs x and all attackers running in time $\leqslant t$ the probability that the attacker outputs the correct guess $b' = b$ is at most $\varepsilon(\lambda)$. Note that this definition differs from prior definitions in the literature (e.g., [7,6]), and is in fact stronger than requiring that an adversary with insufficient resources cannot compute the MHF. However, we remark that in the random oracle model, for random oracle H, any MHF f immediately yields a function $f'(x) = H(f(x))$ which

is indistinguishable from random to any adversary that cannot compute $f(x)$. We provide two definitions of one-time memory-hard functions in the standard model. First, we present a simplified definition of asymptotic security for MHFs (see [9] for the complete formal definition).

Definition 7 (One-Time g-MHF). *For a function $g(\cdot,\cdot)$, we say that a memory hard function* $\text{MHF} = (\text{MHF.}\mathsf{Setup}, \text{MHF.}\mathsf{Eval})$ *is* one-time g-memory hard *if there exists a polynomial t' such that for all polynomials $t(\lambda) > t'(\lambda)$ and every adversary A with $\mathsf{cmc}(A) < y$ for $y(\lambda) := g(t(\lambda), \lambda)$, there exists a negligible function $\mu(\lambda)$ such that for all $\lambda \in \mathbb{N}$ and every input $x \in \{0,1\}^\lambda$, we have $|\Pr[A(x, h_b, \mathsf{pp}) = b] - 1/2| \leqslant \mu(\lambda)$, where the probability is taken over $\mathsf{pp} \leftarrow \text{MHF.}\mathsf{Setup}(1^\lambda, t(\lambda))$, $b \xleftarrow{\$} \{0,1\}$, $h_0 \leftarrow \text{MHF.}\mathsf{Eval}(\mathsf{pp}, x)$, and $h_1 \xleftarrow{\$} \{0,1\}^\lambda$.*

We complement the above definition with a concrete security definition.

Definition 8 (One-time (g,ε)-MHF). *For a function $g(\cdot,\cdot)$, we say that a* $\text{MHF} = (\text{MHF.}\mathsf{Setup}, \text{MHF.}\mathsf{Eval})$ *is a* one-time (g,ε)-MHF *if there exists a polynomial t' such that for all polynomials $t(\lambda) > t'(\lambda)$ and every adversary A with area-time complexity $\mathsf{cmc}(A) < y$, where $y(\lambda) = g(t(\lambda), \lambda)$, and for all $\lambda > 0$ and $x \in \{0,1\}^\lambda$ we have $|\Pr[A(x, h_b, \mathsf{pp}) = b] - 1/2| \leqslant \varepsilon(\lambda)$, where the probability is taken over $\mathsf{pp} \leftarrow \text{MHF.}\mathsf{Setup}(1^\lambda, t(\lambda))$, $b \xleftarrow{\$} \{0,1\}$, $h_0 \leftarrow \text{MHF.}\mathsf{Eval}(x, \mathsf{pp})$ and a uniformly random string $h_1 \in \{0,1\}^\lambda$.*

We give a memory-hard function construction that relies on our new notion of memory-hard puzzles, and additionally uses indistinguishability obfuscation ($i\mathcal{O}$) for circuits and a family of puncturable pseudo-random functions (PPRFs) $\{F_i\}_i$ [30,63,31]. Informally, PPRFs are pseudo-random functions that allow one to "puncture" a key K at values $x_1, \ldots, x_k$, where the key K can be used to evaluate the function at any point $x \notin \{x_1, \ldots, x_k\}$ and hide the values of the function at the points $x_1, \ldots, x_k$.

We formally present our memory-hard function in the full version of our work [9] and provide a high-level overview of the construction here. During the setup phase we generate three PPRF keys K_1, K_2, and K_3 and obfuscate a program $\mathsf{prog}(\cdot,\cdot)$ which does the following. On input $(x, \perp)$, prog outputs a memory-hard puzzle $\text{Puz.}\mathsf{Gen}(1^\lambda, t(\lambda), s; r)$ with solution $s = F_{K_1}(x)$ using randomness $r = F_{K_2}(x)$. On input (x, s'), prog checks to see if $s' = F_{K_1}(x)$ and, if so, outputs $F_{K_3}(x)$; otherwise $\perp$. Given the public parameters $\mathsf{pp} = i\mathcal{O}(\mathsf{prog})$, we can evaluate the MHF as follows: (1) run $\mathsf{pp}(x, \perp) = i\mathcal{O}(\mathsf{prog}(x, \perp))$ to obtain a puzzle Z; (2) solve the puzzle Z to obtain $s = \text{Puz.}\mathsf{Sol}(Z)$; and (3) run $\mathsf{pp}(x, s) = i\mathcal{O}(\mathsf{prog})(x, s)$ to obtain the output $F_{K_3}(x)$. Intuitively, the construction is shown to be one-time memory-hard by appealing to the memory-hard puzzle security, PPRF security, and $i\mathcal{O}$ security.

We establish one-time memory-hardness by showing how to transform an MHF attacker $\mathcal{A}$ into a MHP attacker $\mathcal{B}$ with comparable cmc. Our reduction involves a sequence of hybrids H_0, H_1, H_2 and H_3. Hybrid H_0 is simply our above constructed function. In hybrid H_1 we puncture the PPRF keys $K_i\{x_0, x_1\}$ at target points x_0, x_1 and hard code the corresponding puzzles Z_0, Z_1 along with

their solutions—$i\mathcal{O}$ security implies that H_1 and H_0 are indistinguishable. In hybrid H_2 we rely on PPRF security to replace Z_0, Z_1 with randomly generated puzzles independent of the PPRF keys K_1, K_2 and hardcode the corresponding solutions s_0, s_1. Finally, in hybrid H_3 we rely on MHP security to break the relationship between s_i and Z_i; i.e., we flip a coin b' and hardcoded puzzles $Z'_0 = Z_{b'}$ and $Z'_1 = Z_{1-b'}$ while maintaining $s_i = \mathsf{Puz.Sol}(Z_i)$. In the final hybrid we can show that the attacker cannot win the MHF security game with non-negligible advantage.

Showing indistinguishability of H_2 and H_3 is the most interesting case. In fact, an attacker who can solve either puzzle Z_b or Z_{1-b} can potentially distinguish the two hybrids. Instead, we only argue that the hybrids are indistinguishable if the adversary has small area-time complexity. In particular, if an adversary with small cmc is able to distinguish between H_2 and H_3, then we construct an adversary with small cmc which breaks the memory-hard puzzle.

Our main result is that given a concretely secure PPRF family and a concretely secure $i\mathcal{O}$ scheme, if there exists a concretely secure memory-hard puzzle (Definition 6), then there exists a concretely secure memory-hard function.

Theorem 4. *Let $t := t(\lambda)$ be a polynomial and let $g := g(t, \lambda)$ be a function. Let $\mathcal{F}$ be a $(t_{\mathrm{PPRF}}, \varepsilon_{\mathrm{PPRF}})$-secure PPRF family and iO be a $(t_{i\mathcal{O}}, \varepsilon_{i\mathcal{O}})$-secure $i\mathcal{O}$ scheme. If there exists a $(g, \varepsilon_{\mathrm{MHP}})$-memory hard puzzle for $g \leqslant \min\{t_{\mathrm{PPRF}}(\lambda), t_{i\mathcal{O}}(\lambda)\}$, then there exists one-time $(g', \varepsilon_{\mathrm{MHF}})$-MHF for $g'(t, \lambda) = g(t, \lambda)/p(\log(t), \lambda)^2$, where $\varepsilon_{\mathrm{MHF}}(\lambda) = 2 \cdot \varepsilon_{\mathrm{MHP}}(\lambda) + 3 \cdot \varepsilon_{\mathrm{PPRF}}(\lambda) + \varepsilon_{i\mathcal{O}}(\lambda)$ and $p(\log(t), \lambda)$ is a fixed polynomial which depends on the efficiency of underlying puzzle and iO.*

To get a handle on Theorem 4, consider the following parameter settings. Let $\theta > 0$ be a small constant and suppose that t is suitably large such that $p(\log(t), \lambda)^2 = \Theta(t^c)$ for some suitably small constant $0 < c < \theta$. Then for $g(t, \lambda) = t^{2-\theta+c}$, $\varepsilon_{\mathrm{MHP}} = (1/6) \cdot 2^{-\lambda}$, $\varepsilon_{\mathrm{PPRF}} = (1/9) \cdot 2^{-\lambda}$, and $\varepsilon_{i\mathcal{O}} = (1/3) \cdot 2^{-\lambda}$,[6] our theorem yields a $(g', \varepsilon_{\mathrm{MHF}})$ for $g'(t, \lambda) = \Theta(t^{2-\theta})$ and $\varepsilon_{\mathrm{MHF}} = 1/2^{\lambda}$. Note that the exact parameters of the constructed MHF depend explicitly on the parameters of the underlying primitives used in the construction. Due to space constraints, we defer the formal definitions of concretely secure PPRF families and $i\mathcal{O}$ to the full version of our work [9].

Note that for any instantiation of $i\mathcal{O}$ that we are aware of, our construction is also a (computationally) *data-independent* MHF [10], i.e., the memory access pattern is (computationally) independent of the secret input x. This is a desirable and useful property that provides natural resistance to side-channel attacks.

Remark 1. One may attempt to construct memory-hard puzzles directly from memory-hard functions in a natural way. For example, for a memory-hard function f, one could define $\mathsf{Gen}(x) = r \| x \oplus f(r)$ for random $r \xleftarrow{\$} \{0,1\}^*$ and $\mathsf{Sol}(Z)$ such that first Z is parsed as $Z = r' \| y'$ and then returns $y' \oplus f(r')$. Clearly computing $\mathsf{Sol}(Z)$ is memory-hard, but $(\mathsf{Gen}, \mathsf{Sol})$ is not a cryptographic puzzle by our definitions since Gen violates the efficiency constraints of cryptographic puzzles.

[6] In this example, we assume sub-exponentially secure $i\mathcal{O}$.

Barriers to Proving Multi-Time Security. While we conjecture that our MHF construction achieves stronger multi-time security, we are unable to formally prove this. An interesting aspect of our final hybrid is that indistinguishability does not necessarily hold against an attacker with higher cmc who could trivially distinguish between (s_0, s_1, Z_0, Z_1) and (s_0, s_1, Z_1, Z_0) by solving the puzzles Z_0 and Z_1. However, once the cmc of the attacker is high enough to solve one puzzle, then we cannot rely on the MHP security for the indistinguishability of the final two hybrids. Proving multi-time security would involve proving that any attacker solving m distinct puzzles has cmc that scales linearly in the number of puzzles; i.e., any attacker with $\mathsf{cmc} = o(m \cdot g(t(\lambda)))$ will fail to solve all m puzzles. In particular, even though we expect the cmc of the attacker to be too small to solve all m puzzles, the cmc will become large enough to solve *at least one* puzzle, which allows the attacker to distinguish between the hybrids in our security reduction. See the full version [9] for more details.

2.4 Resource-Bounded LDCs from Cryptographic Puzzles

Recall that a resource-bounded LDC is a (ℓ, δ, p) locally decodable code that is resilient to δ-fraction of errors introduced by some channel in some adversarial class $\mathbb{C}$, where every $\mathcal{A} \in \mathbb{C}$ is assumed to have some resource constraint. For example, $\mathbb{C}$ can be a class of adversaries that are represented by low-depth circuits, or have small cumulative memory complexity. In more detail, security of resource-bounded LDCs requires that any adversary in the class $\mathbb{C}$ cannot corrupt an encoding $y = \mathsf{Enc}(x)$ to some $\tilde{y}$ such that (1) the distance between y and $\tilde{y}$ is at most $\delta \cdot |y|$; and (2) there exists an index i such that the decoder, when given $\tilde{y}$ as its oracle, outputs x_i with probability less than p.

We construct our resource-bounded LDC by modifying the construction of [26] to use cryptographic puzzles in place of random oracles. In particular, for algorithm class $\mathbb{C}$, if there exists a cryptographic puzzle that is unsolvable by any algorithm in $\mathbb{C}$, then we use this puzzle to construct a LDC secure against $\mathbb{C}$. See the full version of our work for the formal definitions of a $(\mathbb{C}, \varepsilon)$-hard puzzle and a $\mathbb{C}$-secure LDC [9].

Our construction crucially relies on a *private LDC* [79]. Private LDCs are LDCs that are additionally parameterized by a key generation algorithm Gen that on input 1^λ for security parameter λ outputs a shared secret key sk to *both* the encoding and decoding algorithm. Crucially, this secret key is hidden from the adversarial channel. See [79,26,9] for formal definitions.

We provide a high-level overview here of our LDC construction and defer the formal construction to the full version of our work [9]. Let $(\mathsf{Gen}, \mathsf{Enc}_\mathsf{p}, \mathsf{Dec}_\mathsf{p})$ be a private Hamming LDC. The encoder Enc_f, on input message x, samples random coins $s \in \{0,1\}^\lambda$ then generates cryptographic puzzle Z with solution s. The encoder then samples a secret key $\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda; s)$, where Gen uses random coins s, and encodes the message x as $Y_1 = \mathsf{Enc}_\mathsf{p}(x; \mathsf{sk})$. The puzzle Z is then encoded as Y_2 via some repetition code. The encoder then outputs $Y = Y_1 \circ Y_2$. This codeword is corrupted to some $\widetilde{Y}$, which can be parsed as $\widetilde{Y} = \widetilde{Y}_1 \circ \widetilde{Y}_2$.

The local decoder Dec_f, on input index i and given oracle access to $\widetilde{Y}$, first recovers the puzzle Z by querying $\widetilde{Y}_2$ and using the decoder of the repetition code (e.g., via random sampling with majority vote). Given s, the local decoder is able to generate the same secret key $\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda; s)$ and now runs the local decoder $\mathsf{Dec}_\mathsf{p}(i; \mathsf{sk})$. All queries made by $\mathsf{Dec}_\mathsf{p}(i; \mathsf{sk})$ are answered by querying $\widetilde{Y}_1$, and the decoder outputs $\mathsf{Dec}_\mathsf{p}(i; \mathsf{sk})$. The construction is secure against any class $\mathbb{C}$ for which there exist cryptographic puzzles that are secure against this class. For example, time-lock puzzles give an LDC that is secure against the class $\mathbb{C}$ of circuits of low-depth, and memory-hard puzzles give an LDC that is secure against the class $\mathbb{C}$ of PRAM algorithms with low cmc.

Security is established via a reduction to the cryptographic puzzle. Suppose there exists an adversary $A \in \mathbb{C}$ which can violate the security of our LDC. The reduction relies on a two-phase hybrid distinguishing argument [26]. Fix $(\mathsf{Enc}_\mathsf{f}, \mathsf{Dec}_\mathsf{f})$ to be the encoder and local decoder constructed above. We define two different encoders to be used in the hybrid arguments. First the encoder $\mathsf{Enc}_0 := \mathsf{Enc}_\mathsf{f}$ is defined to be exactly the same as our LDC encoder. Second, the encoder Enc_1 is defined to be identical to Enc_f, except with the following changes: (1) Enc_1 receives both a message x and some secret key sk as input; (2) Enc_1 encodes x as $Y_1 = \mathsf{Enc}_\mathsf{p}(x; \mathsf{sk})$; and (3) Enc_1 samples some $s' \xleftarrow{\$} \{0,1\}^\lambda$ that is uncorrelated with its input sk, computes puzzle $Z' \leftarrow \mathsf{Puz.Gen}(s')$, and computes Y_2 as the repetition encoding of Z'.

We now construct our attacker B which violates the security of the cryptographic puzzle as follows: B is given (Z_b, Z_{1-b}, s_0, s_1) for uniformly random bit b, where Z_i is a puzzle with solution s_i as input. Then B fixes a message x and encodes x as follows. (1) Using puzzle solution s_0, generate secret key $\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda, s_0)$. (2) Compute Y_2 as the encoding of Z_b (i.e., its first input) using the repetition code. (3) Compute $Y_1 \leftarrow \mathsf{Enc}_\mathsf{p}(x; \mathsf{sk})$. (4) Set $Y = Y_1 \circ Y_2$. With Y in hand, the adversary B simulates adversary A to obtain $\tilde{Y} = \tilde{Y}_1 \circ \tilde{Y}_2 \leftarrow A(x, Y)$. Finally, B outputs $b' \leftarrow \mathcal{D}(x, \mathsf{sk}, \tilde{Y}_1)$. Here, the distinguisher $\mathcal{D}$ is given $\tilde{Y}_1$, the secret key sk_0, and message x as input; additionally, it can simulate the local decoding algorithm Dec_p. In particular, the distinguisher $\mathcal{D}$ is defined as follows: (1) sample an index $i \xleftarrow{\$} \{1, \ldots, |x|\}$ uniformly at random; (2) compute $\tilde{x}_i \leftarrow \mathsf{Dec}_\mathsf{p}^{\tilde{Y}_1}(i; \mathsf{sk}_0)$; and (3) output $b' = 0$ if $x_i \neq \tilde{x}_i$ and $b' = 1$ otherwise.

Intuitively, if $b = 1$ then $Y_1 = \mathsf{Enc}_\mathsf{p}(s; \mathsf{sk}_0)$ where the secret key sk_0 is information theoretically hidden from A when the corrupted private-key codeword $\tilde{Y}_1 \leftarrow A(x, Y)$ is produced. Private key LDC security ensures that, except with negligible probability, $\mathsf{Dec}_\mathsf{p}^{\tilde{Y}_1}(i; \mathsf{sk}_0)$ will output the correct answer $\tilde{x}_i = x_i$ and $\mathcal{D}$ will output the correct answer $b' = 1$. On the other hand if $b = 0$ we have $Y = \mathsf{Enc}_0(x)$ and $\tilde{Y} \leftarrow A(x, Y)$ so that the probability that $\mathsf{Dec}_\mathsf{p}^{\tilde{Y}_1}(i; \mathsf{sk}_b)$ outputs the wrong answer $x_i \neq \tilde{x}_i$ will be non-negligible — at least $1/|x|$ times the advantage of A in the LDC security game. Thus, the probability that $\mathcal{D}$ outputs the correct answer $b' = 0$ is also non-negligible. It follows our adversary B outputs the correct bit $b = b'$ with non-negligible advantage violating security of the underlying memory hard puzzles. See [9] for formal details.

Our main result is that given any private Hamming LDC, if there exists a memory-hard puzzle, then there exists a resource-bounded LDC that is secure against the class of PRAM algorithms, where the parameters of the resource-bounded LDC are comparable to the parameters of the private LDC.

Corollary 1. *Let g be a function, let $\mathbb{C}(g) := \{\mathcal{A}\colon\ \mathcal{A}$ is a PRAM algorithm and $\mathsf{cmc}(\mathcal{A}) < g\}$, and let $C_{\mathsf{p}}[K_{\mathsf{p}}, k_{\mathsf{p}}, \lambda]$ be a $(\ell_{\mathsf{p}}, \delta_{\mathsf{p}}, p_{\mathsf{p}}, \varepsilon_{\mathsf{p}})$-private Hamming LDC. If there exists a (g, ε')-memory hard puzzle then there exists a $(\ell, \delta, \varepsilon)$-resource bounded LDC $C[\Omega(K_{\mathsf{p}}), k_{\mathsf{p}}]$ that is secure against the class $\mathbb{C}(g)$ with parameters $\ell = \Theta(\ell_{\mathsf{p}})$, $\delta = \Theta(1)$, $p = 1 - \mathsf{negl}(\lambda)$, and $\varepsilon = \Theta(\varepsilon_{\mathsf{p}} + \varepsilon')$.*

Actually, in the full version of our work [9], we prove a more general theorem which utilizes any private LDC in conjunction with a more general $(\mathbb{C}, \varepsilon)$-hard puzzle (i.e., the puzzle is secure against the class of adversaries $\mathbb{C}$, which allows us to construct a resource-bounded LDC that is secure against the class $\mathbb{C}$.

Resource-Bounded LDCs for Insertion-Deletion Errors in the Standard Model. Recently, Block and Blocki [20] proved that the so-called "Hamming-to-InsDel" compiler of Block et al. [21] extends to both the private Hamming LDC and resource-bounded Hamming LDC settings. That is, there exists a procedure which compiles any resource-bounded Hamming LDC to a resource-bounded LDC that is robust against *insertion-deletion errors* such that this compilation procedure preserves the underlying security of the Hamming LDC. We apply the result of Block and Blocki [20] to our construction and obtain the first construction of resource-bounded locally decodable code for insertion-deletion errors in the standard model. We remark that the prior construction presented in [20] was in the random oracle model.

Corollary 2. *Let $\mathbb{C}(g) = \{\mathcal{A}\colon\ \mathcal{A}$ is a PRAM algorithm and $\mathsf{cmc}(\mathcal{A}) < g\}$ and let $C_{\mathsf{p}}[K_{\mathsf{p}}, k_{\mathsf{p}}, \lambda]$ be a private Hamming LDC. If there exists a (g, ε')-memory hard puzzle and a $(\ell, \delta, p, \varepsilon)$ resource-bounded LDC that is secure against the class $\mathbb{C}(g)$, then there exists a $(\ell', \delta', p', \varepsilon'')$-$\mathsf{LDC}$ $C[n, k]$ for insertion-deletion errors against class $\mathbb{C}(g)$, where $\ell' = \ell \cdot O(\log^4(n))$, $\delta' = \Theta(\delta)$, $p' < p$, $\varepsilon'' = \varepsilon/(1 - \mathsf{negl}(n))$, $k = k_{\mathsf{p}}$, and $K = \Omega(K_{\mathsf{p}})$.*

Acknowledgments

Mohammad Hassan Ameri was supported in part by NSF award #1755708, by IARPA under the HECTOR program, and by a Summer Research Grant from Purdue University. Alexander R. Block was supported in part by NSF CCF #1910659. Jeremiah Blocki was supported in part by NSF CNS #1755708, NSF CNS #2047272, and NSF CCF #1910659 and by IARPA under the HECTOR program.

References

1. Agrawal, S., Pellet-Mary, A.: Indistinguishability obfuscation without maps: Attacks and fixes for noisy linear FE. In: EUROCRYPT (1). Lecture Notes in Computer Science, vol. 12105, pp. 110–140. Springer (2020)
2. Alwen, J., Blocki, J.: Efficiently computing data-independent memory-hard functions. In: CRYPTO (2). Lecture Notes in Computer Science, vol. 9815, pp. 241–271. Springer (2016)
3. Alwen, J., Blocki, J., Harsha, B.: Practical graphs for optimal side-channel resistant memory-hard functions. In: CCS. pp. 1001–1017. ACM (2017)
4. Alwen, J., Blocki, J., Pietrzak, K.: Depth-robust graphs and their cumulative memory complexity. In: EUROCRYPT (3). Lecture Notes in Computer Science, vol. 10212, pp. 3–32 (2017)
5. Alwen, J., Blocki, J., Pietrzak, K.: Sustained space complexity. In: EUROCRYPT (2). Lecture Notes in Computer Science, vol. 10821, pp. 99–130. Springer (2018)
6. Alwen, J., Chen, B., Pietrzak, K., Reyzin, L., Tessaro, S.: Scrypt is maximally memory-hard. In: EUROCRYPT (3). Lecture Notes in Computer Science, vol. 10212, pp. 33–62 (2017)
7. Alwen, J., Serbinenko, V.: High parallel complexity graphs and memory-hard functions. In: STOC. pp. 595–603. ACM (2015)
8. Alwen, J., Tackmann, B.: Moderately hard functions: Definition, instantiations, and applications. In: TCC (1). Lecture Notes in Computer Science, vol. 10677, pp. 493–526. Springer (2017)
9. Ameri, M.H., Block, A.R., Blocki, J.: Memory-hard puzzles in the standard model with applications to memory-hard functions and resource-bounded locally decodable codes. IACR Cryptol. ePrint Arch. p. 801 (2021)
10. Ameri, M.H., Blocki, J., Zhou, S.: Computationally data-independent memory hard functions. In: ITCS. LIPIcs, vol. 151, pp. 36:1–36:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
11. Applebaum, B.: Garbled circuits as randomized encodings of functions: a primer. In: Tutorials on the Foundations of Cryptography, pp. 1–44. Springer International Publishing (2017)
12. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography in nc^0. SIAM J. Comput. **36**(4), 845–888 (2006)
13. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. J. ACM **59**(2), 6:1–6:48 (2012)
14. Bartusek, J., Ishai, Y., Jain, A., Ma, F., Sahai, A., Zhandry, M.: Affine determinant programs: A framework for obfuscation and witness encryption. In: ITCS. LIPIcs, vol. 151, pp. 82:1–82:39. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
15. Ben-Sasson, E., Goldreich, O., Harsha, P., Sudan, M., Vadhan, S.P.: Robust pcps of proximity, shorter pcps, and applications to coding. SIAM J. Comput. **36**(4), 889–974 (2006)
16. Biryukov, A., Dinu, D., Khovratovich, D.: Argon2: New generation of memory-hard functions for password hashing and other applications. In: EuroS&P. pp. 292–302. IEEE (2016)
17. Bitansky, N., Garg, S., Lin, H., Pass, R., Telang, S.: Succinct randomized encodings and their applications. In: STOC. pp. 439–448. ACM (2015)
18. Bitansky, N., Garg, S., Telang, S.: Succinct randomized encodings and their applications. IACR Cryptol. ePrint Arch. p. 771 (2014)

19. Bitansky, N., Goldwasser, S., Jain, A., Paneth, O., Vaikuntanathan, V., Waters, B.: Time-lock puzzles from randomized encodings. In: ITCS. pp. 345–356. ACM (2016)
20. Block, A.R., Blocki, J.: Private and resource-bounded locally decodable codes for insertions and deletions. In: ISIT. pp. 1841–1846. IEEE (2021)
21. Block, A.R., Blocki, J., Grigorescu, E., Kulkarni, S., Zhu, M.: Locally decodable/correctable codes for insertions and deletions. In: FSTTCS. LIPIcs, vol. 182, pp. 16:1–16:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
22. Blocki, J., Cheng, K., Grigorescu, E., Li, X., Zheng, Y., Zhu, M.: Exponential lower bounds for locally decodable and correctable codes for insertions and deletions. In: FOCS. pp. 739–750. IEEE (2021)
23. Blocki, J., Cinkoske, M., Lee, S., Son, J.Y.: On explicit constructions of extremely depth robust graphs. In: STACS. LIPIcs, vol. 219, pp. 14:1–14:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022)
24. Blocki, J., Gandikota, V., Grigorescu, E., Zhou, S.: Relaxed locally correctable codes in computationally bounded channels. IEEE Trans. Inf. Theory **67**(7), 4338–4360 (2021)
25. Blocki, J., Harsha, B., Kang, S., Lee, S., Xing, L., Zhou, S.: Data-independent memory hard functions: New attacks and stronger constructions. In: CRYPTO (2). Lecture Notes in Computer Science, vol. 11693, pp. 573–607. Springer (2019)
26. Blocki, J., Kulkarni, S., Zhou, S.: On locally decodable codes in resource bounded channels. In: ITC. LIPIcs, vol. 163, pp. 16:1–16:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
27. Blocki, J., Ren, L., Zhou, S.: Bandwidth-hard functions: Reductions and lower bounds. In: CCS. pp. 1820–1836. ACM (2018)
28. Blocki, J., Zhou, S.: On the depth-robustness and cumulative pebbling cost of argon2i. In: TCC (1). Lecture Notes in Computer Science, vol. 10677, pp. 445–465. Springer (2017)
29. Boneh, D., Naor, M.: Timed commitments. In: CRYPTO. Lecture Notes in Computer Science, vol. 1880, pp. 236–254. Springer (2000)
30. Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: ASIACRYPT (2). Lecture Notes in Computer Science, vol. 8270, pp. 280–300. Springer (2013)
31. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: Public Key Cryptography. Lecture Notes in Computer Science, vol. 8383, pp. 501–519. Springer (2014)
32. Brakensiek, J., Guruswami, V., Zbarsky, S.: Efficient low-redundancy codes for correcting multiple deletions. IEEE Trans. Inf. Theory **64**(5), 3403–3410 (2018)
33. Brakerski, Z., Döttling, N., Garg, S., Malavolta, G.: Candidate io from homomorphic encryption schemes. In: EUROCRYPT (1). Lecture Notes in Computer Science, vol. 12105, pp. 79–109. Springer (2020)
34. Chen, B., Tessaro, S.: Memory-hard functions from cryptographic primitives. In: CRYPTO (2). Lecture Notes in Computer Science, vol. 11693, pp. 543–572. Springer (2019)
35. Cheng, K., Guruswami, V., Haeupler, B., Li, X.: Efficient linear and affine codes for correcting insertions/deletions. In: SODA. pp. 1–20. SIAM (2021)
36. Cheng, K., Haeupler, B., Li, X., Shahrasbi, A., Wu, K.: Synchronization strings: Highly efficient deterministic constructions over small alphabets. In: SODA. pp. 2185–2204. SIAM (2019)

37. Cheng, K., Jin, Z., Li, X., Wu, K.: Deterministic document exchange protocols, and almost optimal binary codes for edit errors. In: FOCS. pp. 200–211. IEEE Computer Society (2018)
38. Cheng, K., Jin, Z., Li, X., Wu, K.: Block edit errors with transpositions: Deterministic document exchange protocols and almost optimal binary codes. In: ICALP. LIPIcs, vol. 132, pp. 37:1–37:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
39. Cheng, K., Li, X.: Efficient document exchange and error correcting codes with asymmetric information. In: SODA. pp. 2424–2443. SIAM (2021)
40. Dvir, Z., Gopalan, P., Yekhanin, S.: Matching vector codes. SIAM J. Comput. **40**(4), 1154–1178 (2011)
41. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: CRYPTO. Lecture Notes in Computer Science, vol. 740, pp. 139–147. Springer (1992)
42. Efremenko, K.: 3-query locally decodable codes of subexponential length. SIAM J. Comput. **41**(6), 1694–1703 (2012)
43. Forler, C., Lucks, S., Wenzel, J.: Memory-demanding password scrambling. In: ASIACRYPT (2). Lecture Notes in Computer Science, vol. 8874, pp. 289–305. Springer (2014)
44. Garay, J.A., MacKenzie, P.D., Prabhakaran, M., Yang, K.: Resource fairness and composability of cryptographic protocols. J. Cryptol. **24**(4), 615–658 (2011)
45. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. SIAM J. Comput. **45**(3), 882–929 (2016)
46. Garg, S., Srinivasan, A.: A simple construction of io for turing machines. In: TCC (2). Lecture Notes in Computer Science, vol. 11240, pp. 425–454. Springer (2018)
47. Guruswami, V., Haeupler, B., Shahrasbi, A.: Optimally resilient codes for list-decoding from insertions and deletions. IEEE Trans. Inf. Theory **67**(12), 7837–7856 (2021)
48. Guruswami, V., Li, R.: Polynomial time decodable codes for the binary deletion channel. IEEE Trans. Inf. Theory **65**(4), 2171–2178 (2019)
49. Guruswami, V., Li, R.: Coding against deletions in oblivious and online models. IEEE Trans. Inf. Theory **66**(4), 2352–2374 (2020)
50. Guruswami, V., Smith, A.D.: Optimal rate code constructions for computationally simple channels. J. ACM **63**(4), 35:1–35:37 (2016)
51. Guruswami, V., Wang, C.: Deletion codes in the high-noise and high-rate regimes. IEEE Trans. Inf. Theory **63**(4), 1961–1970 (2017)
52. Haeupler, B.: Optimal document exchange and new codes for insertions and deletions. In: FOCS. pp. 334–347. IEEE Computer Society (2019)
53. Haeupler, B., Rubinstein, A., Shahrasbi, A.: Near-linear time insertion-deletion codes and $(1+\varepsilon)$-approximating edit distance via indexing. In: STOC. pp. 697–708. ACM (2019)
54. Haeupler, B., Shahrasbi, A.: Synchronization strings: explicit constructions, local decoding, and applications. In: STOC. pp. 841–854. ACM (2018)
55. Haeupler, B., Shahrasbi, A.: Synchronization strings: Codes for insertions and deletions approaching the singleton bound. J. ACM **68**(5), 36:1–36:39 (2021)
56. Haeupler, B., Shahrasbi, A., Sudan, M.: Synchronization strings: List decoding for insertions and deletions. In: ICALP. LIPIcs, vol. 107, pp. 76:1–76:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018)
57. Hubácek, P., Wichs, D.: On the communication complexity of secure function evaluation with long output. In: ITCS. pp. 163–172. ACM (2015)

58. Ishai, Y., Kushilevitz, E.: Randomizing polynomials: A new representation with applications to round-efficient secure computation. In: FOCS. pp. 294–304. IEEE Computer Society (2000)
59. Jain, A., Lin, H., Sahai, A.: Indistinguishability obfuscation from well-founded assumptions. In: STOC. pp. 60–73. ACM (2021)
60. Jakobsson, M., Juels, A.: Proofs of work and bread pudding protocols. In: Communications and Multimedia Security. IFIP Conference Proceedings, vol. 152, pp. 258–272. Kluwer (1999)
61. Katz, J., Trevisan, L.: On the efficiency of local decoding procedures for error-correcting codes. In: STOC. pp. 80–86. ACM (2000)
62. Kerenidis, I., de Wolf, R.: Exponential lower bound for 2-query locally decodable codes via a quantum argument. J. Comput. Syst. Sci. **69**(3), 395–420 (2004)
63. Kiayias, A., Papadopoulos, S., Triandopoulos, N., Zacharias, T.: Delegatable pseudorandom functions and applications. In: CCS. pp. 669–684. ACM (2013)
64. Kiwi, M., Loebl, M., Matoušek, J.: Expected length of the longest common subsequence for large alphabets. In: Farach-Colton, M. (ed.) LATIN 2004: Theoretical Informatics. pp. 302–311. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
65. Kopparty, S., Meir, O., Ron-Zewi, N., Saraf, S.: High-rate locally correctable and locally testable codes with sub-polynomial query complexity. J. ACM **64**(2), 11:1–11:42 (2017)
66. Kopparty, S., Saraf, S.: Guest column: Local testing and decoding of high-rate error-correcting codes. SIGACT News **47**(3), 46–66 (2016)
67. Koppula, V., Lewko, A.B., Waters, B.: Indistinguishability obfuscation for turing machines with unbounded memory. In: STOC. pp. 419–428. ACM (2015)
68. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics Doklady **10**(8), 707–710 (1966), doklady Akademii Nauk SSSR, V163 No4 845-848 1965
69. Lin, H., Matt, C.: Pseudo flawed-smudging generators and their application to indistinguishability obfuscation. IACR Cryptol. ePrint Arch. p. 646 (2018)
70. Lin, H., Pass, R., Seth, K., Telang, S.: Output-compressing randomized encodings and applications. In: TCC (A1). Lecture Notes in Computer Science, vol. 9562, pp. 96–124. Springer (2016)
71. Lipton, R.J.: A new approach to information theory. In: STACS. Lecture Notes in Computer Science, vol. 775, pp. 699–708. Springer (1994)
72. Liu, S., Tjuawinata, I., Xing, C.: On list decoding of insertion and deletion errors (2020)
73. Mahmoody, M., Moran, T., Vadhan, S.P.: Time-lock puzzles in the random oracle model. In: CRYPTO. Lecture Notes in Computer Science, vol. 6841, pp. 39–50. Springer (2011)
74. Malavolta, G., Thyagarajan, S.A.K.: Homomorphic time-lock puzzles and applications. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 11692, pp. 620–649. Springer (2019)
75. May., T.C.: Timed-release crypto
76. Micali, S., Peikert, C., Sudan, M., Wilson, D.A.: Optimal error correction against computationally bounded noise. In: TCC. Lecture Notes in Computer Science, vol. 3378, pp. 1–16. Springer (2005)
77. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system
78. Okamoto, T., Pietrzak, K., Waters, B., Wichs, D.: New realizations of somewhere statistically binding hashing and positional accumulators. In: ASIACRYPT (1). Lecture Notes in Computer Science, vol. 9452, pp. 121–145. Springer (2015)

79. Ostrovsky, R., Pandey, O., Sahai, A.: Private locally decodable codes. In: ICALP. Lecture Notes in Computer Science, vol. 4596, pp. 387–398. Springer (2007)
80. Ostrovsky, R., Paskin-Cherniavsky, A.: Locally decodable codes for edit distance. In: ICITS. Lecture Notes in Computer Science, vol. 9063, pp. 236–249. Springer (2015)
81. Percival, C.: Stronger key derivation via sequential memory-hard functions. In: BSDCan 2009 (2009)
82. Pippenger, N., Fischer, M.J.: Relations among complexity measures. J. ACM **26**(2), 361–381 (1979)
83. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Tech. rep., USA (1996)
84. Shaltiel, R., Silbak, J.: Explicit list-decodable codes with optimal rate for computationally bounded channels. Comput. Complex. **30**(1), 3 (2021)
85. Sima, J., Bruck, J.: Optimal k-deletion correcting codes. In: ISIT. pp. 847–851. IEEE (2019)
86. Sudan, M., Trevisan, L., Vadhan, S.P.: Pseudorandom generators without the XOR lemma. J. Comput. Syst. Sci. **62**(2), 236–266 (2001)
87. Yekhanin, S.: Towards 3-query locally decodable codes of subexponential length. J. ACM **55**(1), 1:1–1:16 (2008)
88. Yekhanin, S.: Locally decodable codes. Found. Trends Theor. Comput. Sci. **6**(3), 139–255 (2012)