
Proceedings Paper:
Dodd, Charles John, Farshim, Pooya, Shahandashti, Siamak F. et al. (2026) Multi-Instance Unrecoverability of iMHF-Based Password Hashing. In: 31st European Symposium on Research in Computer Security. 31st European Symposium on Research in Computer Security, 14-18 Sep 2026 , ITA. (In Press)

Multi-Instance Unrecoverability of iMHF-Based Password Hashing

Charles Dodd[1,2], Pooya Farshim[1,3], Siamak F. Shahandashti[2], and Karl Southern[1]

[1] Department of Computer Science, Durham University, Durham, UK
karl.southern@durham.ac.uk
[2] Department of Computer Science, University of York, York, UK
{charles.dodd,siamak.shahandashti}@york.ac.uk
[3] Input Output, Switzerland
pooya.farshim@iohk.io

Keywords: password hashing · secure password storage · memory-hard function · multi-instance security · cumulative memory complexity · pebbling complexity

Abstract. The study of memory-hard functions (MHFs) so far has mainly focused on providing provable guarantees on the expected minimum cumulative memory complexity (CMC) required per *evaluation* when amortized over multiple instances. Such results, however, do not provide any guarantees for the security of compromised password banks in the sense of passwords remaining *unrecoverable*. Indeed, a construction can be memory-hard while still leaking information about its input. We provide the first formal treatment of the unrecoverability of graph-based data-independent MHFs (iMHFs) in the multi-instance setting. Multi-instance security is the widely accepted security model when inputs have low entropy or are correlated, and require the adversarial effort to linearly scale with the number of instances broken.

To prove these results, we appropriately extend the ex-post-facto pebbling technique of Alwen and Serbinenko (STOC'15) and the unguessability reductions of Farshim and Tessaro (EUROCRYPT'21). We then use the resulting compatible frameworks to bound the number of guesses of adversaries with a given CMC in terms of the pebbling complexity of the graph underlying the iMHF. Combined with known lower bounds for the pebbling complexities of their graphs, we obtain, as corollaries, concrete unrecoverability bounds for the Argon2i, Catena, and Balloon hashing, showing in particular that adversarial advantage indeed scales linearly with the number of instances and the CMC of the adversary.

1 Introduction

Password-based cryptography leverages the prevalence of passwords to underpin multiple applications such as authentication and key derivation. Password storage in such applications is usually protected through salting and hashing,

i.e., rather than a password pw in plaintext, a pair $(sa, \mathsf{H}(pw||sa))$ is stored for a random salt sa. To slow down password-cracking attacks, H is replaced with a construction C^{H} built from H through *iteration*, as in the case of bcrypt [17] and PBKDF2 [16]. However, the ever-growing efficiency of specialized hardware such as ASICs in evaluating hash functions necessitates higher iteration counts that disproportionately affect honest users.

To address this issue, recommendations for password hashing [1] advocate the use of *memory-hard functions* (MHFs) designed to ensure that successful computation of the function requires expending large amounts of time and memory. This leverages the fact that dedicated hardware does not provide the same cost saving in scaling up memory as in computation. Candidates for MHFs include Scrypt [15], the first proposal for such a function in 2009 by Percival, and Argon2 [7] chosen through the Password Hashing Competition in 2015 [1].

MHFs come in two flavors, distinguished by their memory access pattern. In *data-dependent* MHFs (dMHFs), the memory access pattern changes with the input, whereas *data-independent* MHFs (iMHFs) have the same memory-access pattern for every input. The latter is important when the function is run on a secret input (such as passwords) in an insecure environment, as dMHF memory-access patterns can potentially allow the recovery of secrets/passwords via side-channel attacks [10]. Constructions of iMHFs are usually based on an underlying *directed acyclic graph* (DAG) that specifies how hash functions are combined to give the construction. Such iMHFs are called *graph-based* iMHFs. Examples include Argon2i [7], Catena [13], and Balloon hashing [9].

In practice, passwords may be weak, and even if stored under iterated hashing or an MHF C^{H}, they are still vulnerable to guessing attacks: an adversary with access to a leaked password database $(sa_i, \mathsf{C}^{\mathsf{H}}(pw_i||sa_i))$ for passwords pw_i, can attempt to recover them by guessing common passwords pw^*, computing $\mathsf{C}^{\mathsf{H}}(pw^*||sa_i)$ for each pw^*, and comparing them with the database entries. Hence, there is an inherent limit to the protection constructions can provide for *individual* passwords. Despite this, one still would like assurances that the effort needed to recover *multiple* passwords increases, ideally *linearly*, with the number of passwords recovered. The *multi-instance* (mi) security model, introduced by Bellare, Ristenpart, and Tessaro (BRT12) [5], formalizes this metric based on how the resources expended by the adversary grow with the number of "cracked" target hashes. This model was further studied by Farshim and Tessaro (FT21) [12] who proved the efficacy of salting against preprocessing attacks (e.g., rainbow tables [14]) in the multi-instance setting.

In light of ASIC-assisted attacks, measuring the complexity of attacks through query complexity is not sufficient. Percival [15] proposed maximum memory usage multiplied by time as an alternative metric. Subsequently, this metric was shown to provide insufficient guarantees against *amortization* [4].[4] The state-of-the-art metric for measuring memory cost, proposed by Alwen and Serbinenko

[4] The maximum memory usage can be high even if it is high only at the initial stages of the attacks, thus allowing reuse of freed memory for a second instance after the initial stage is over.

(AS15) [4], is the *cumulative memory complexity* (CMC): the *sum* of memory usage at each time step in the computation. Alwen and Serbinenko prove lower bounds on the CMC of graph-based iMHFs based on the *cumulative complexity* (CC) of the underlying graph, defined as the minimum *pebbling* complexity of the graph [4]. This allows them to prove the non-amortizability of MHF evaluations.

Although provable guarantees on the memory-hardness of graph-based iMHFs provide high confidence that *computation* would incur a certain minimum memory cost, they do not translate into provable *unrecoverability* guarantees for secure password storage using such iMHFs. To see this, given a well-designed MHF F^{H}, consider the construction $\mathsf{C}^{\mathsf{H}}(x) := \mathsf{F}^{\mathsf{H}}(x)||x$. This construction is memory-hard, as computing the output requires computing F^{H}. However, it does not provide any security for password storage as *recovering* the input is trivial. More generally, a construction $\mathsf{F}_1^{\mathsf{H}}(x)||\mathsf{F}_2^{\mathsf{H}}(x)$ will have a CMC of at least the maximum CMC of $\mathsf{F}_1^{\mathsf{H}}$ and $\mathsf{F}_2^{\mathsf{H}}$, but will only achieve an unrecoverability bound of at most the *minimum* unrecoverability bounds that $\mathsf{F}_1^{\mathsf{H}}$ and $\mathsf{F}_2^{\mathsf{H}}$ achieve.

Multi-instance unrecoverability is the natural security property expected from secure password storage protected by MHFs. However, to date there has been no formal unrecoverability analysis of MHFs in the literature.

Our contributions. Motivated by the prevalence of MHFs in password-based protocols, we provide the first formal treatment of multi-instance unrecoverability of graph-based iMHFs. We prove concrete upper bounds on the unrecoverability of passwords based on the unguessability of the underlying password distribution, the CMC of the adversary, and the CC of the graph underlying an iMHF. Towards this, we show how to combine the FT21 and AS15 frameworks to derive unrecoverability bounds for iMHFs. Our proofs bound the number of guesses of any adversary with a given CMC in terms of the CC of the underlying graph via an extension of the ex-post-facto pebbling argument of AS15. It then applies the unrecoverability-to-unguessability reduction of FT21 to derive our final bounds.

Along the way, we define *strong* memory-hardness (sMH), whereby an adversary may choose its iMHF inputs adaptively based on the iMHF's underlying hash function. This strengthening is required, along with extensions to the AS15 and FT21 frameworks, to enable our main result. We also show a separation between strong memory-hardness and memory-hardness by showing that a widely-used memory-hard function is not strongly memory hard. Despite this, we prove that *graph-based iMHFs* achieve strong memory hardness.

Future work. In this work we focus on the security of iMHFs, as such one open problem would be to extend our results to specific or generalized dMHFs.

2 Preliminaries

Notation. We denote the size of a set N by $|N|$, the length of a vector $\mathbf{x}$ by $|\mathbf{x}|$, and the set of its unique members by $\{\mathbf{x}\}$. For sampling a value v uniformly at random from a set N, we write $v \twoheadleftarrow N$. When sampling a value v randomly from

a distribution $\mathcal{D}$ (or by a randomized algorithm), we write $v \twoheadleftarrow \mathcal{D}$. For the set of integers from 1 to m we write $[m]$. We denote the set of functions with domain N and range M as $\mathrm{Fun}(N, M)$. An adversary $\mathcal{A}$ is a randomized algorithm, and $\mathcal{A}^{\mathcal{O}}$ denotes that it has access to an oracle $\mathcal{O}$. We write ε for the empty string.

2.1 Basic security definitions

The parallel random oracle model. The random oracle model (ROM) [6] is an idealized model of computation in which all parties, honest or otherwise, have access to a uniformly sampled function $\mathsf{H} \twoheadleftarrow \mathrm{Fun}(N, M)$. We consider adversaries in the *parallel random oracle model* (pROM) [4]. In this model, an algorithm can make batches of queries to the random oracle H in rounds, and receive the corresponding responses simultaneously. More precisely, a pROM algorithm $\mathcal{A}_*^{\mathsf{H}}$ takes an input x, and makes t rounds of batch oracle calls. In each round, $\mathcal{A}$ makes a batch of queries $\mathbf{q}$ and writes to a free state st. The batch of responses $\mathbf{y}$ along with st are subsequently given to $\mathcal{A}$, and $\mathcal{A}$ writes to a state σ_i any information passed to the following round. $\mathcal{A}$ may also append values to a special output register $\sigma_{\mathcal{O}}$. To append some value l to the output register, we assume $\mathcal{A}$ makes a special query of the form (l, out), where out is a special symbol. This will not be answered by the oracle, but recorded in the transcript. When $\mathcal{A}_*^{\mathsf{H}}$ terminates, $\sigma_{\mathcal{O}}$ is returned. The structure of a pROM algorithm is shown in Fig. 1 (top left).

Password samplers and salt generators. A password sampler $\mathcal{P}_m$ in BRT12/FT21 is defined to be a randomized algorithm that takes no input and outputs a vector of m passwords $\mathbf{pw}$ and possibly some leakage information z on the passwords. The leakage z models the information the adversary holds about the password distribution prior to its attack. Salts are generated by a salt-generator algorithm Gen. This is a randomized algorithm which takes as input a user index $i \in [0, m)$ and a counter $j \in [0, \ell)$, and outputs salts $\mathbf{sa}[i, j]$ (allowing a password to be reused across ℓ salts) from a space of size K, which we consider to be $[K]$.

Unguessability and salted unguessability. We briefly recall the notions of (salted) unguessability from [5,12]. We give the security game for salted unguessability in Fig. 1, and the standard unguessability game can be recovered by removing salting. Intuitively, guessability measures the adversary's ability to recover a full bank of passwords, sampled from a password distribution, using some leakage z on the passwords. The most basic game to address this notion is the Guess game from [5]. An adversary $\mathcal{A}$ in the Guess game is given leakage z, with access to test and corruption oracles Test and Cor. Queries to Test allow $\mathcal{A}$ to check password guesses, returning a win_i flag if the ith password is guessed correctly. We consider adversaries that are able to make at most c queries to Cor, i.e., they automatically "win" on c of the passwords.

For the salted setting, we again consider adversaries that are able to make up to c corruption queries to Cor. The Test oracle now takes password-salt pairs as queries. Each query $\mathcal{A}$ sends to Test is checked against each password-salt pair in the sampled set, and win flags are returned for each matching password

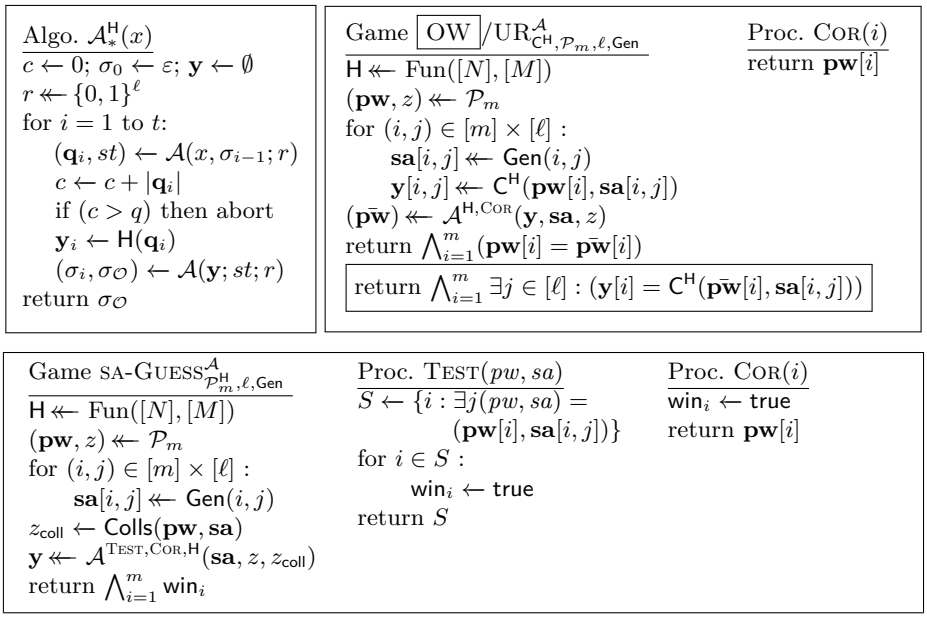

Fig. 1: Top left: Structure of an algorithm in pROM. Top right: UR and OW games w.r.t. a m-sampler $\mathcal{P}_m$ and salt generator Gen. Bottom: SA-GUESS game w.r.t. $\mathcal{P}_m$. Here z_{coll} denotes the collision pattern of password-salt pairs.

which has a matching salt. This models the fact that the adversary can verify the guess without repeating it for each salt. The collision pattern on password-salt pairs is given explicitly to the adversary by procedure $\mathsf{Colls}(\mathbf{pw}, \mathbf{sa})$. This takes the vector of m passwords and the $m \times \ell$ matrix of salts, and returns the password-salt collision pattern z_{coll}. This is an $m\ell \times m\ell$ matrix with entries $((i_1, j_1), (i_2, j_2))$ set to 1 only when $(\mathbf{pw}[i_1], \mathbf{sa}[i_1, j_1]) = (\mathbf{pw}[i_2], \mathbf{sa}[i_2, j_2])$.

Unrecoverability and one-wayness. We recall the notions of unrecoverability and one-wayness with relation to password hashing, the security games for both are given in Fig. 1. Unrecoverability models the hardness of recovering passwords given a password bank protected by a hash-based construction. We consider the unrecoverability game for a general hash-based construction C^{H}. In this game, a password sampler $\mathcal{P}_m$ outputs m passwords along with some leakage z. The salt generator Gen outputs the salt matrix $\mathbf{sa}$, which along with the password vector $\mathbf{pw}$ is used as input to C^{H}, generating the challenge vector $\mathbf{y}$. The adversary is given $\mathbf{y}$ along with $\mathbf{sa}$ and the leakage z. For this game, the collision pattern of password-salt pairs is public and does not need to be provided to $\mathcal{A}$.

2.2 Graph-based iMHFs and cumulative memory complexity

Let $\mathcal{G} = (V, E)$ be a directed acyclic graph (DAG), where the node set V is the set of integers 1 to n, referred to as node indices. Let $\mathbf{Pa}(i)$ (resp. $\mathbf{Ch}(i)$) be the

set of parents (resp. children) of the node i, with $\mathbf{Pa}_j(i)$ (resp. $\mathbf{Ch}_j(i)$) being the j-th parent (resp. child) ordered by node index. Nodes with no parents or children are called *source* and *sink* nodes, respectively. Let δ_i be in-degree of the ith node and $\delta := \max_i\{\delta_i\}$. We consider iMHFs which use the labeling scheme from AS15 [4]. We reproduce the definition of the labeling here, and extend it to allow for salting. To allow salting, we extend the initial label l to be drawn from $L \times [K]$ rather than L (i.e., l would take the form (pw, sa)).

(H, l)-labeling of graphs [4]. Let $\mathcal{G} = (V, E)$ be a DAG with maximum in-degree δ, L be an arbitrary label set, $[K]$ be a salt set of size K, and $\mathbb{H} := \mathrm{Fun}(V \times L^\delta \times [K], L)$. For a function $\mathsf{H} \in \mathbb{H}$ and a label $l \in L \times [K]$, the (H, l)-labeling of $\mathcal{G}$ is a mapping $\mathbf{lab} : V \mapsto L$ defined recursively for all v by:

$$\mathbf{lab}(v) := \begin{cases} \mathsf{H}(v, l) & : \mathrm{indeg}(v) = 0\,; \\ \mathsf{H}(v, \mathbf{lab}(\mathbf{Pa}_1(v)), \ldots, \mathbf{lab}(\mathbf{Pa}_d(v))) & : 0 < \mathrm{indeg}(v) = d \leq \delta\,. \end{cases}$$

There can be at most $\delta + 1$ inputs to H. We assume an injective padding function is applied to the inputs to give exactly $\delta + 2$ inputs before H is applied to compute the label. We, however, omit this padding function for clarity. We use $\mathbf{pre\text{-}lab}(v)$ to refer to the H inputs which define v's label, i.e., $\mathbf{pre\text{-}lab}(v) = (v, l)$ if $\mathrm{indeg}(v) = 0$, and $\mathbf{pre\text{-}lab}(v) := (v, \mathbf{lab}(\mathbf{Pa}_1(v)), \ldots, \mathbf{lab}(\mathbf{Pa}_d(v))$ otherwise. Note that as l is a password-salt pair, a salt is only included when $\mathrm{indeg}(v) = 0$. Using this labeling definition, a graph-based iMHF C^H is defined as follows.

Graph-based iMHF [4]. Let $\mathcal{G} = (V, E)$ be a DAG on n vertices, with maximum in-degree δ, a single source node, and a single sink node v_s. Let L be an arbitrary label set, $[K]$ be the salt set, and $\mathbb{H} = \mathrm{Fun}(V \times L^\delta \times [K], L)$. The *graph functions* (of $\mathcal{G}$ and $\mathbb{H}$) are the members of the family of oracle functions $\mathsf{C} = \mathsf{C}^{\mathbb{H}}_{\mathcal{G}}$, indexed by functions in $\mathbb{H}$, mapping $L \times [K]$ to L. For input $l \in L \times [K]$, the value of $\mathsf{C}^\mathsf{H} \in \mathsf{C}$ is defined as $\mathsf{C}^\mathsf{H}(l) := \mathbf{lab}(v_s)$, where $\mathbf{lab}$ is the (H, l)-labeling of $\mathcal{G}$.

We primarily consider the labeling set $L := [N]$, but the results hold for an arbitrary labeling set L where $|L| = N$ that can be efficiently mapped to $[N]$. We follow the notation used in [4] for denoting multiple instances of a graph. That is, for a DAG $\mathcal{G} = (V, E)$ and a positive integer m, we use $\mathcal{G}^{\times m}$ (which we call the tensor graph) to denote the disjoint union of m copies of $\mathcal{G}$, where each node has a unique index of the form (v, k) for $v \in V$ and $k \in [m]$. We denote constructions based on $\mathcal{G}^{\times m}$ and random oracle H by $\mathsf{C}^\mathsf{H}_{\times m}$. Note that $\mathsf{C}^\mathsf{H}_{\times m}$ is defined by a labeling on a graph with distinct node indices.

In order to model the memory requirements for an adversary to compute an iMHF, one usually analyzes an adversary playing the *(black) pebbling game* on the underlying graph. In the pebbling game, an adversary is challenged with placing pebbles on the sink nodes of a DAG, while following some simple rules as follows. The pebbling occurs in rounds. In each round, an adversary can add a single pebble and remove any number of pebbles from the graph. The source nodes for the graph can be pebbled at any time. For all other nodes, an adversary can only place a pebble on that node if all of its parents had pebbles on them

in the previous round. We consider the *parallel* pebbling game where batches of pebbles can be added (or removed) in each round.

Parallel pebbling. Formally, given a DAG $\mathcal{G} = (V, E)$, the parallel (black) pebbling of $\mathcal{G}$ is defined as a sequence of configurations $P = (P_0, \ldots, P_t)$, where $P_i \subseteq V$. A pebbling with starting and target sets $S, T \subset V$ is *legal* if the following three conditions hold: (1) $P_0 = \emptyset$; (2) a node can only be added when its predecessors have a pebble on, or if it is a starting node, i.e., $\forall i \in [t], \forall x \in P_i \setminus P_{i-1} : (\forall y \in \mathbf{Pa}(x) : y \in P_{i-1}) \vee (x \in S)$, where there is no limit on the number of pebbles that can be placed in a configuration; and (3) all target nodes are pebbled at some round, i.e., $\forall x \in T,\ \exists z \leq t : x \in P_z$. A pebbling is *complete* if S and T are the set of source and sink nodes, respectively. Note that by (3), all nodes in T are pebbled at some point in time.

Cumulative pebbling complexity. Given a parallel pebbling adversary $\mathcal{A}$ that has at most q pebbles to place across at most t rounds, we define the cumulative pebbling complexity (CPC) of $\mathcal{A}$'s pebbling of $\mathcal{G}$ as $\mathrm{CPC}(\mathcal{A}) := \sum_{i=0}^{t-1} |P_i|$. We define the q-CPC of $\mathcal{G}$ as the minimum CPC of any q-pebble adversary's legal and complete pebbling of $\mathcal{G}$, i.e., let $S[q]$ be the set of all q-pebble adversaries, then $\mathrm{CPC}_q(\mathcal{G}) := \min_{\mathcal{A} \in S[q]}(\mathrm{CPC}(\mathcal{A}))$. Above a sufficiently large q (e.g., $q > |V|^2$), $\mathrm{CPC}_q(\mathcal{A})$ will not decrease, as an adversary with more pebbles can always use a sufficient number of them. Hence, for sufficiently large q, CPC is independent of q. We define this minimum as the *cumulative complexity* of the graph $\mathrm{CC}(\mathcal{G})$.

Cumulative memory complexity. We consider memory-constrained adversaries in pROM, which make batches of queries to the random oracle in rounds. Keeping track of the state size (which measures the memory stored between rounds), allows for an analysis of the total memory used by a parallel adversary across its execution time. We define the cumulative memory complexity of $\mathcal{A}$ with respect to H, input x, and coins r as $\mathrm{CMC}(\mathcal{A}, x, \mathsf{H}, r) := \sum_{i=0}^{t} |\sigma_i|$, where σ_i is the state after the ith round as defined in Section 2.1. For our applications, the input to $\mathcal{A}$ is an empty string and so we use $\mathrm{CMC}(\mathcal{A})$, leaving both H and r implicit. Following [4], we define the cumulative memory complexity of $\mathcal{A}$ with respect to x as $\mathrm{CMC}(\mathcal{A}, x) := \mathbb{E}_{\mathsf{H},r}\left[\mathrm{CMC}(\mathcal{A}, x, \mathsf{H}, r)\right]$, where the expectation is taken over the choice of random oracle H and random coins r. Since our proofs are written with respect to an H, we use the former of these two equations, rather than needing to take the expectation over all H.

Single-sink vs. multi-sink graphs. Current metrics for CMC assume that a full computation of a construction has to be done, and that a partial computation is not useful to an adversary. For the purposes of guessing passwords, this assumption is false. Consider a function F^{H} with a single sink which has high CMC, and define a construction $\mathsf{C}^{\mathsf{H}} := \mathsf{F}^{\mathsf{H}}(x)||\mathsf{H}(x)$ with two sinks. Now, C^{H} still has a high CMC. However, to check whether a guessed password x' is correct, it is sufficient to consider the second sink only since $\mathsf{H}(x) \neq \mathsf{H}(x')$ is sufficient to ascertain that $x \neq x'$. Hence, the adversary only needs to make a single hash query for such

a check rather than comparing the full computation of $\mathsf{C}^{\mathsf{H}}(x')$ with $\mathsf{C}^{\mathsf{H}}(x)$. Note that this holds not only for unrecoverability, but even for finding collisions in that it is easy to verify non-collision without a full computation. Whilst the example we have given is slightly contrived, it is clear that for any multi-sink construction the CMC required to compute any sink could be much lower than the CMC required to compute all sinks. We therefore only consider constructions with a single sink. We note that all major graph based iMHFs, including Argon2i [7], Catena [13] and Balloon hashing [9], adhere to this restriction.

3 Strong Memory Hardness

In existing hash-independent definition of memory-hard functions, a parallel adversary chooses the input x to the function at the beginning, and the function is deemed memory hard if the adversary requires a large amount of CMC to correctly evaluate the function (with high probability). Importantly, the cost of evaluations must not be amortizable across multiple instances. In this setting, the adversary is restricted to choosing the input to the function *independently* of the random oracle. The security game for the standard definition is presented as the m-MH game in Fig. 2 (left). This game is for two-stage adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$. With no access to H, $\mathcal{A}_0$ picks a vector $\mathbf{x}$ of m distinct inputs. Then $\mathcal{A}_1$, given $\mathbf{x}$ as input, makes batch queries to H, and outputs a vector of evaluations $\mathbf{y}$. Adversary $\mathcal{A}$ wins if for each input value x_i in $\mathbf{x}$, the corresponding output y_i in $\mathbf{y}$ satisfies $\mathsf{C}^{\mathsf{H}}(x_i) = y_i$. More precisely, the construction C^{H} is strongly (m, q, ϵ, μ)-memory hard if for all q-query adversaries $\mathcal{A}$ with CMC($\mathcal{A}$) at most μ, adversary $\mathcal{A}$ wins the m-MH game with probability at most ϵ.

We introduce a stronger notion, the *strong* MH game sMH, that extends adversarial capabilities in two orthogonal ways. First, it grants the adversary oracle access whilst choosing inputs. Second, it allows the adversary to output an *unordered* set of q inputs and q outputs, where only m are required to be correct MHF computations. This game is formalized in Fig. 2 (right). We define $\mathcal{A}$'s advantage in the sMH game as $\mathbf{Adv}^{(q,m)\text{-sMH}}_{\mathsf{C}^{\mathsf{H}}}(\mathcal{A}) := \Pr[(q,m)\text{-sMH}^{\mathcal{A}}_{\mathsf{C}^{\mathsf{H}}}]$. Strong (m, q, ϵ, μ)-memory hardness is defined analogously.

This definition is strictly stronger than memory-hardness: an MHF with high (even optimal) CMC can have fast evaluations, i.e., a non-optimal strong memory hardness. To prove this separation, we show in the full version of this paper [11, Appendix A] that whilst Scrypt is proven to be maximally memory-hard [3], this is not the case in the strong memory-hardness game.

We extend the AS15 results and show that single-sink single-source *graph-based* MHFs are indeed strongly memory hard. To this end, we adapt the ex-post-facto pebbling argument. We first give an overview of how memory hardness is established in the hash-independent setting.

The hash-independent case. AS15 proves a lower bound on the amortized CMC of evaluating m instances of an iMHF based on a graph $\mathcal{G}$, with probability at least ϵ. The bound shows that the best adversary evaluating the function

Game m-MH$^{\mathcal{A}}_{\mathsf{C}^{\mathsf{H}}}$	Game (q,m)-sMH$^{\mathcal{A}}_{\mathsf{C}^{\mathsf{H}}}$
$\mathsf{H} \twoheadleftarrow \mathrm{Fun}([N],[M])$	$\mathsf{H} \twoheadleftarrow \mathrm{Fun}([N],[M])$
$\mathbf{x} \twoheadleftarrow \mathcal{A}_0()$	$(\mathbf{x},\mathbf{y}) \twoheadleftarrow \mathcal{A}^{\mathsf{H}}()$
if $(\lvert\{\mathbf{x}\}\rvert \neq m)$:	if $(\lvert\{\mathbf{x}\}\rvert \geq q) \vee (\lvert\{\mathbf{y}\}\rvert \geq q)$:
return 0	return 0
$\mathbf{y} \twoheadleftarrow \mathcal{A}_1^{\mathsf{H}}(\mathbf{x})$	for x in $\mathbf{x}$ and y in $\mathbf{y}$:
return $(\mathbf{y} = \mathsf{C}^{\mathsf{H}}(\mathbf{x}))$	if $y = \mathsf{C}^{\mathsf{H}}(x)$:
	$S \leftarrow S \cup \{(x,y)\}$
	return $\lvert S\rvert \geq m$

Fig. 2: The hash-independent (left), and the strong (right) memory hardness games. In the unsalted setting, $\mathbf{x}$ is a vector of passwords pw_i. In the salted setting, $\mathbf{x}$ is a vector of tuples of the form (pw_i, sa_i).

(with high probability) will have CMC close to any adversary which simply pebbles each instance of the graph. This result is proved by applying the results of three auxiliary lemmas to the definition of the amortized cost. The first ([4, Lemma 11]) calculates the expected adversarial savings made by using collisions between the labeling of the instances. This bounds the cost of computing m possibly colliding instances by the cost of fewer non-colliding instances. The m non-colliding instances are then treated as a single instance of $\mathcal{G}^{\times m}$, whose CC is lower bounded in [4, Lemma 12]. The CMC of an adversary computing a single instance of $\mathcal{G}^{\times m}$ is then bounded, roughly, by $\mathrm{CC}(\mathcal{G}^{\times m})$. This is done by analyzing an adversary's queries, showing that a legal and complete pebbling of the underlying graph can be extracted from any correct evaluation. The third result ([4, Lemma 3]) proves CC of two node-disjoint graphs is additive. The bound for the one-off cost of $\mathcal{G}^{\times m}$ can then be given in terms of that of $\mathcal{G}$. Together, in [4, Theorem 3], these lemmas bound the amortized cost in $\mathrm{CC}(\mathcal{G})$.

Extension to strong memory-hardness. In our proof, we extend the technique in [4, Lemma 12]. By omitting the final step of [4, Lemma 12] where a lower bound on the cost is derived, we obtain an upper bound on the advantage. Extending the first steps of the AS15 proof, we give a reduction from an adversary in the (q,m)-sMH game to a prediction game, in which an adversary predicts the output of the random oracle on an unqueried input. This prediction argument allows for the analysis of both the legality and the complexity of an extracted pebbling, and gives the final bound in terms of the CC of the underlying graph.

Theorem 1 (Amortized strong memory-hardness). *Let $q,t,m,n,\delta,K,N \in \mathbb{N}$. Let $\mathcal{A}$ be a q-query t-time adversary with cumulative memory complexity* $\mathrm{CMC}(\mathcal{A})$ *and $\mathcal{G}$ be a DAG. Let C^{H} be the iMHF based on $\mathcal{G}$ in the $(nN^{\delta}K, N)$-RO model as defined in Section 2.2. Then*

$$\mathbf{Adv}^{(q,m)\text{-sMH}}_{\mathsf{C}^{\mathsf{H}}}(\mathcal{A}) \leq \frac{2q^2}{N} + 2^{-\gamma} \ ,$$

where $\gamma := \big(m \cdot \mathrm{CC}(\mathcal{G})[\log N - \log(qm)] - \mathrm{CMC}(\mathcal{A})\big)/t$.

Intuitively, γ captures the gap between the expected CMC of an adversary computing m instances following the canonical pebbling algorithm (where for each pebble, the adversary queries the hash to compute the next label) and the actual CMC of $\mathcal{A}$ averaged over t steps. Thus, γ becomes large as the adversarial CMC decreases beyond the minimum required for the construction. We give the full proof in the full version of this paper [11], but give an outline in Appendix A.

In our extension, we have to account for adversaries that choose inputs in a hash-dependent way, and compute multiple instances of the construction. Note that if an illegal pebble placement occurs in the extracted pebbling, then a "bad" query in the transcript contains a correct response to some other "target" random oracle query, before the query is made. In the weaker memory hardness game, a predictor, given the index of this bad query as a hint, can recompute the response to the target query from the input.

Crucially, the adaptivity of picking inputs in the strong memory-hardness game means legality is *not* defined until $\mathcal{A}$ completes its run and the inputs become known. Thus in our reduction, we give an extended hint, to allow the predictor to run $\mathcal{A}$ safely without querying the point for which it makes the prediction. The extension to the hint contains the position in the transcript, if there, of the query for which the adversary will predict the output.

We make a further extension in order to analyze the pebbling complexity, where again we reduce to a prediction game. This time, to run $\mathcal{A}$ safely, the predictor must detect for which instances $\mathcal{A}$'s queries are correct without necessarily knowing the input to the instance. This again requires an extension to the hint, that contains the location in the transcript of the starting queries for each of the instances the adversary needs to detect. These extensions allow us to give an upper bound on the advantage an adversary has in computing these graph-based iMHFs, showing that they are also *strongly* memory-hard.

4 Unrecoverability of Memory-Hard Functions

We now consider multi-instance unrecoverability of graph-based iMHFs. Our result establishes security bound in terms of the adversary's CMC (as opposed to query complexity), and rely on adversarial upper bounds in computing an iMHF on m distinct hash-dependent inputs established in the previous section.

Theorem 2 (UR security of C^{H}). *Let $P, q, n, m, c, t, \delta, \ell \in \mathbb{N}$, $\mathcal{P}_m$ be a m-sampler, $\mathsf{Gen} = [K]$, $\mathcal{G}$ be a DAG on n vertices, and C^{H} be the iMHF based on the graph $\mathcal{G}$ in the $(nN^{\delta}K, N)$-RO model as defined in Section 2.2. Then for any q-query, t-time, c-corruption adversary $\mathcal{A}$ in the* UR *game with respect to $\mathcal{P}_m, \mathsf{Gen}$ and C^{H}, there exists a P-query, c-corruption adversary $\mathcal{B}$ against* SA-GUESS *s.t.*

$$\mathbf{Adv}^{\mathrm{UR}}_{\mathcal{P}_m,\ell,\mathsf{Gen},\mathsf{C}^{\mathsf{H}}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{SA-GUESS}}_{\mathcal{P}_m,\ell,\mathsf{Gen}}(\mathcal{B}) + 2^{-\gamma} + \frac{(2q_{\mathcal{P}} + nm\ell + 4q)^2}{N} + \frac{m\ell}{K} \ ,$$

where $\gamma = [(P+1)\mathrm{CC}(\mathcal{G})[\log N - \log(qm)] - (\mathrm{CMC}(\mathcal{A}) + \Delta)]/t$, and $\Delta = \mathrm{CMC}(\mathcal{P}_m) + mt \log N$.

Setting $P = (\mathrm{CMC}(\mathcal{A})+\Delta)/\mathrm{CC}(\mathcal{G})(\log N/qm)$ minimizes the right hand side of the bound, as we show in the full version of this paper [11, Appendix C].

We start with the multi-instance unrecoverability game and then move to a game that lazily samples the RO, and sets a bad flag and aborts on collisions in outputs. By aborting on collisions, we stop the adversary from choosing distinct inputs that collide early on in the computation, making it easier to amortize the computation. We then transition to a game that sets a bad flag and aborts if the adversary makes a query for an index > 1 that had already been made during the challenge generation phase, but where $\mathcal{A}$ has not computed the chain up to the node i. This ensures that an adversary cannot start the computation midway through and then try to recover a valid starting point. From here, we go through a series of game transitions that lead to the game no longer storing intermediate points generated during the challenge generation phase. Finally, we transition to a game where we abort if the adversary computes C^{H} on more than P distinct inputs, which we justify via a reduction to the $(q, P+1)$-sMH game defined in Section 3. Having bounded the number of full computations the adversary can perform, we finally reduce to the sa-Guess game, where the adversary is able to make P queries to the Test oracle. By reducing to sa-Guess, we only need to account for full computations on inputs the adversary chooses, as only the final H query in a full computation is forwarded to Test. Note that in FT21, which considers the monolithic random oracle, every H query is forwarded to Test.

Proof. We prove the theorem via a sequence of games as follows.

G_0: This game is defined to be the UR game with respect to $\mathcal{P}_m, \mathsf{Gen}$ and C^{H}. Thus, $\Pr[\mathrm{G}_0] = \mathbf{Adv}^{\mathrm{UR}}_{\mathcal{P}_m,\ell,\mathsf{Gen},\mathsf{C}^{\mathsf{H}}}(\mathcal{A})$.

G_1: This game lazily samples the random oracle. This game is identical to G_0: $\Pr[\mathrm{G}_1] = \Pr[\mathrm{G}_0]$.

G_2: This game sets a bad flag bad_1 whenever the output point sampled for any oracle $i < n$ has already been used as part of an input to the oracle indexed with any $j \in \mathbf{Ch}(i)$ or there is a collision in the hash outputs. (This event ensures that no two computations on distinct inputs collide – note that this includes computations from the challenge generation phase.) The game then aborts after bad_1 is set. Games G_1 and G_2 are identical until bad_1. As the outputs are sampled randomly, and the game (that is the adversary and the challenge generation phase) make a total of at most $2q + mn\ell$ queries to the oracles, we have $\Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_2] \leq \Pr[\mathsf{bad}_1] \leq \frac{(nm\ell+2q)^2}{N}$.

G_3: This game sets bad_2 when $\mathcal{A}$ makes a query containing a label generated during the challenge phase, without first completing the computation from the corresponding input, and then aborts after bad_2 is set. The check for computation from an input to a point within the computation is done using a procedure called findChain. Given a label y and a point v, findChain checks if there exists a computation chain from an initial input up to the point v and then either returns the input point or returns $\perp$. On input (y, v), findChain runs as follows, it parses the input y into $v, \mathbf{lab}(\mathbf{Pa}_1(v)), \ldots, \mathbf{lab}(\mathbf{Pa}_d(v))$, for each $\mathbf{lab}(\mathbf{Pa}_i(v))$ it looks in the table for a query that returned $\mathbf{lab}(\mathbf{Pa}_i(v))$.

Due to the bad events in the previous games for each $\mathbf{lab}(\mathbf{Pa}_i(v))$ there will be a at most one inverse y'. If no inverse exists then the function returns $\perp$, otherwise it recursively applies FINDCHAIN on $(y', \mathbf{Pa}_i(v))$. When FINDCHAIN is called on $(y, 1)$, i.e., the first hash in the computation, if it has an inverse in the table, FINDCHAIN has found the initial input to the construction and returns that input. Formally, for each password-salt pair (pw, sa), the queries made by the game to evaluate $\mathsf{C}^{\mathsf{H}}(pw, sa)$ are assigned to each node v a label l. If $\mathcal{A}$ makes a query containing the label l, without fully computing the $\mathsf{C}^{\mathsf{H}}(pw, sa)$ up to the corresponding node v, then bad_2 is set. We show in the challenge point prediction claim below that $\Pr[\mathrm{G}_3] - \Pr[\mathrm{G}_2] \leq \Pr[\mathsf{bad}_2] \leq m\ell q/N$.

G_4: In this game, when the adversary makes a query for the final node, the game uses FINDCHAIN (described in G_3), to check if the query completes a full computation starting from a password-salt pair $(\mathbf{pw}[i], \mathbf{sa}[i, j])$. If it finds such a computation, it responds with the corresponding challenge value, else it samples a fresh random value. The bad_2 flag ensures that $\mathcal{A}$ only labels points from the challenge generation correctly via a full computation from an input to that point. The bad_1 flag ensures $\mathcal{A}$ only computes in the forward direction. Together these flags ensure that the game will stay consistent with the challenge values $\mathbf{y}$, because any correct query $\mathcal{A}$ makes to label a sink node will be the result of a full chain, and so the game can respond with the right challenge point. Thus $\Pr[\mathrm{G}_4] = \Pr[\mathrm{G}_3]$.

G_5: In this game, we no longer store the intermediate points generated in the challenge phase, and if during the online phase $\mathcal{A}$ evaluates the construction on a correct password-salt pair the points are freshly sampled. Until a query from $\mathcal{A}$ resamples these intermediate points, the values are information theoretically hidden from the adversary. Thus: $\Pr[\mathrm{G}_5] = \Pr[\mathrm{G}_4]$.

G_6: This game sets a bad flag bad_3 whenever the adversary fully computes C^{H} on more than P unique inputs, and then aborts when bad_3 is set. Games G_6 and G_5 are identical until bad_3. We show, via a reduction to a strong $(q, P+1)$-sMH game, in the bounded chain completion claim below that $\Pr[\mathrm{G}_6] - \Pr[\mathrm{G}_5] \leq \Pr[\mathsf{bad}_3] \leq \mathbf{Adv}^{(q,P+1)\text{-sMH}}_{\mathsf{Gen},\mathsf{C}^{\mathsf{H}}}(\mathcal{B})$. We conclude the proof by showing in the G_6 to SA-GUESS reduction claim below that the advantage of any adversary in G_6 is bounded by one in the SA-GUESS.

Claim (Collisions). For the parameters above, we have

$$\Pr[\mathcal{A} \text{ sets } \mathsf{bad}_1 \text{ in } \mathrm{G}_2] \leq \frac{(nm\ell + 2q)^2}{N} \ .$$

The proof of this claim follows from a simple collision bound of any of the hash outputs colliding and is omitted for brevity.

Claim (Challenge point prediction). For the parameters above, we have

$$\Pr[\mathsf{bad}_2 \text{ is set in } \mathrm{G}_3] \leq \frac{m\ell q}{N} \ .$$

The proof of this claim follows from a simple collision bound of any of the $m\ell$ challenge points colliding with any of the queries of $\mathcal{A}$ and is omitted for brevity.

Claim (Bounded chain completion). For the parameters above:

$$\Pr[\mathcal{A} \text{ sets } \mathsf{bad}_3 \text{ in } \mathrm{G}_6] \leq \mathbf{Adv}^{(q,P+1)\text{-sMH}}_{\mathsf{C}^{\mathsf{H}}}(\mathcal{B}) \leq 2^{-\gamma} + \frac{2q^2}{N} \ ,$$

where $\gamma := ((P+1) \cdot \mathrm{CC}(\mathcal{G})(\log N - \log(qm) - \mathrm{CMC}(\mathcal{B}))/t$, and $\mathrm{CMC}(\mathcal{B}) = \mathrm{CMC}(\mathcal{A}) + mt \log N$.

Proof. Given an adversary $\mathcal{A}$ as defined in the theorem statement, we construct a q-query adversary $\mathcal{B}$ against $(q, P+1)$-sMH, where $\mathrm{CMC}(\mathcal{B}) = \mathrm{CMC}(\mathcal{A}) + mt \log(N)$. Algorithm $\mathcal{B}$, with access to H, samples a random password vector and leakage z from $\mathcal{P}_m$, samples $m\ell$ salt vectors from Gen and generates a random challenge output vector $\mathbf{y}$ which it sends to $\mathcal{A}$, along with the salt vectors and leakage z. $\mathcal{B}$ then runs $\mathcal{A}$ and forwards each of $\mathcal{A}$'s queries to its oracle.

When $\mathcal{A}$ makes a query with the index of the source node, i.e., a query of the form $(1, l)$, algorithm $\mathcal{B}$ appends the input l to its special output register $\sigma_{\mathcal{O}}$. When $\mathcal{A}$ makes a query to label the sink node n, i.e., a query of the form (n, l), $\mathcal{B}$ appends the response $\mathsf{H}(n, l)$ to the register $\sigma_{\mathcal{O}}$. For any Cor query $\mathcal{A}$ makes, $\mathcal{B}$ responds with the corresponding password. When $\mathcal{A}$ terminates, $\mathcal{B}$ returns $\sigma_{\mathcal{O}}$, which contains at most q values.

To handle $\mathcal{A}$'s corruption queries, $\mathcal{B}$ stores the password vector $\mathbf{pw}$. This increases $\mathrm{CMC}(\mathcal{B})$ by $mt \log(N)$. As algorithm $\mathcal{B}$ recognizes each input or output query as it receives it, and appends stored values to $\sigma_{\mathcal{O}}$ immediately, it runs $\mathcal{A}$ with no extra CMC cost. If $\mathcal{A}$ triggers bad_3, then there exists a set of $(P+1)$ correct input-output pairs in $\sigma_{\mathcal{O}}$, and $\mathcal{B}$ wins the $(q, P+1)$-sMH game. □

Claim (G_6 to sa-Guess *reduction).* Let $P, q, n, m, c \in \mathbb{N}$, $\mathcal{G}$ a graph on n vertices with maximum in-degree δ, and C^{H} the iMHF based on $\mathcal{G}$ in the $(nN^{\delta}K, N)$-RO model defined in Section 2.2. Then, for any m-sampler $\mathcal{P}_m$, any unpredictable salt generator Gen, and any q-query c-corruption adversary $\mathcal{A}$ against G_6, there exists a P-query, c-corruption adversary $\mathcal{B}$ against sa-Guess s.t.

$$\mathbf{Adv}^{\mathrm{G}_6}_{\mathcal{P}_m,\ell,\mathsf{Gen},\mathsf{C}^{\mathsf{H}}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{sa-Guess}}_{\mathcal{P}_m,\ell,\mathsf{Gen}}(\mathcal{B}) \ .$$

Proof. Given an adversary $\mathcal{A}$ as in the theorem statement in G_6, we construct an adversary $\mathcal{B}$ against sa-Guess as follows. We assume, without loss of generality, that $\mathcal{A}$ only returns a value x if it has computed $\mathsf{C}^{\mathsf{H}}(x)$. $\mathcal{B}$ receives the salt vector $\mathbf{sa}$, leakage z, and generates a challenge vector $\mathbf{y}$ following the rules of G_6. Algorithm $\mathcal{B}$ runs $\mathcal{A}$ on $\mathbf{y}$ and z and for all $\mathcal{A}$'s queries with node indices from 1 to $n-1$, $\mathcal{B}$ answers by lazily sampling the random oracle, if any of $\mathcal{B}$'s answers lead to a collision between computations with distinct start points, then $\mathcal{B}$ aborts according to the rules corresponding to bad_1 of G_2. For any queries that $\mathcal{A}$ makes to the oracle for the final node, i.e., of the form $(n, \ell_1, \ldots, \ell_\delta)$, algorithm $\mathcal{B}$ checks if past queries form a complete computation of the construction on some input (x, sa) using findChain (described in G_3). If it does, $\mathcal{B}$ queries (x, sa) to Test.

If TEST returns true, $\mathcal{B}$ responds to $\mathcal{A}$'s query with the value $y \in \mathbf{y}$ corresponding to (x, sa), else it lazily samples the random oracle output. As we have a single sink, any query to the sink node must correlate to a full computation, note that if there were multiple sink nodes then a query to TEST must be made any time a sink node is reached, regardless of if the other sinks had been reached yet – i.e. regardless of if the whole computation had been made. $\mathcal{A}$ could have computed the function for at most P unique start points, otherwise bad_2 would have been triggered and the game would have aborted. As $\mathcal{B}$ will have queried each of the P start points (on which $\mathcal{A}$ computed C^{H}) to its TEST oracle, and $\mathcal{A}$ only returns values on which it has performed full computations, then any (x', sa') that $\mathcal{A}$ returns will have been queried to TEST by $\mathcal{B}$. Note that throughout the proof, even with a single challenge, if the adversary correctly "guesses" the password but doesn't fully complete the computation, we do not set a bad flag. As $\mathcal{B}$ will have returned y if TEST returns true, then whenever $\mathcal{A}$ wins, $\mathcal{B}$ will also win. □

Combining the game transitions above gives

$$\begin{aligned}\Pr[\mathrm{G}_0] &= \Pr[\mathrm{G}_6] + \sum_{i=0}^{5}(\Pr[\mathrm{G}_i] - \Pr[\mathrm{G}_{i+1}]) \\ &\leq \mathbf{Adv}^{\text{SA-GUESS}}_{\mathcal{P}_m,\ell,\mathsf{Gen}}(P, c) + \frac{2q^2}{N} + 2^{-\gamma} + \frac{m\ell q}{N} + \frac{(nm\ell + 2q)^2}{N} \\ &\leq \mathbf{Adv}^{\text{SA-GUESS}}_{\mathcal{P}_m,\ell,\mathsf{Gen}}(P, c) + 2^{-\gamma} + \frac{(nm\ell + 4q)^2}{N} \ ,\end{aligned}$$

where the last inequality is due to $(nm\ell + 4q)^2 \geq 2q^2 + m\ell q + (nm\ell + 2q)^2$ and the definitions of γ and Δ as per the statement of Theorem 2. □

One-wayness. Password-storage security can also be analyzed with respect to a standard one-wayness security notion where the adversary's goal is to recover *any* input as long as it produces the same construction output as that of the actual password. Moving between unrecoverability and one-wayness can be achieved in a straightforward manner up to the collision resistance of the construction, which can be shown to be $\mathcal{O}(q^2)/N$ for any q-query adversary, i.e., we get:

$$\mathbf{Adv}^{\text{OW}}_{\mathcal{P}_m,\ell,\mathsf{Gen},\mathsf{C}^{\mathsf{H}}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{UR}}_{\mathcal{P}_m,\ell,\mathsf{Gen},\mathsf{C}^{\mathsf{H}}}(\mathcal{B}) + \frac{\mathcal{O}(q^2)}{N} \ .$$

This bound is sufficient for password-hashing as passwords are typically low entropy and thus the UR bound dominates the term due to hash collisions. This means that even if the format of the passwords is not checked and any value is accepted as an input by a password-based system, security remains intact.

Interpretation. Consider a setting without corruptions, i.e., $c = 0$. Then, the dominating term in the bound in Theorem 2 is $\mathbf{Adv}^{\text{SA-GUESS}}_{\mathcal{P}_m,\ell,\mathsf{Gen}}(P)$. As above, we set $P = (\text{CMC}(\mathcal{A}) + \Delta)/\text{CC}(\mathcal{G})(\log N/qm)$, which gives $2^{-\gamma} = (qm/N)^{\text{CC}(\mathcal{G})/t}$. Assuming a uniform salt generator, and moving from SA-GUESS to GUESS by

applying [12, Thm 3], we can bound the dominating term by $\mathbf{Adv}^{\text{GUESS}}_{\mathcal{P}_m}(P) + m^2\ell^2/K$. Applying [12, Thm 2], bounds this by $\binom{P}{m}\mathbf{Adv}^{\text{GUESS}}_{\mathcal{P}_m}(m)$. For a password sampler $\mathcal{P}_m$ that outputs m uniform passwords from a dictionary of size D, the advantage of a P-query adversary in the GUESS game is $\binom{P}{m}/D^m \approx \left(\frac{P}{mD}\right)^m$. For a general graph based iMHF, and a password sampler whose query complexity is at most linear in $\text{CMC}(\mathcal{A})$, we can upper bound the advantage as

$$\mathbf{Adv}^{\text{UR}}_{\mathcal{P}_m,\ell,\mathsf{Gen},\mathsf{C}^{\mathsf{H}}}(P) \leq \tilde{\mathcal{O}}\left(\left(\frac{\text{CMC}(\mathcal{A})}{\text{CC}(\mathcal{G})mD}\right)^m\right) .$$

This matches the intuition that for salted passwords, the adversary has to spread its CMC across all m instances, using CMC/m per instance. So for each instance it can compute the construction $\text{CMC}/m\text{CC}(\mathcal{G})$ times, each of which is a correct guess with probability $1/D$. As the adversary needs to guess correctly for m independent instances, we have the bound above. Hence, the adversarial effort to recover the passwords scales linearly in both the CC of the underlying graph and the number of passwords. Note that neither ℓ nor K appear in this simplified bound, as they only contribute to a small additive term. In Table 1 we compute upper and lower bounds for $\mathbf{Adv}^{\text{UR}}_{\mathcal{P}_m,\ell,\mathsf{Gen},\mathsf{C}^{\mathsf{H}}}(\mathcal{A})$, where C^{H} is set to different iMHFs, incorporating current bounds on cumulative complexity of the underlying graphs for each iMHF taken from [8,2].

iMHF	Lower bound	Upper bound
ARGON2I	$\tilde{\Omega}\left(\left(\frac{\text{CMC}(\mathcal{A})}{n^{1.767}mD}\right)^m\right)$	$\tilde{\mathcal{O}}\left(\left(\frac{\text{CMC}(\mathcal{A})}{n^{1.75}mD}\right)^m\right)$
CATENA	$\tilde{\Omega}\left(\left(\frac{\text{CMC}(\mathcal{A})}{n^{1.625}mD}\right)^m\right)$	$\tilde{\mathcal{O}}\left(\left(\frac{\text{CMC}(\mathcal{A})}{n^{1.5}mD}\right)^m\right)$
BALLOON	$\tilde{\Omega}\left(\left(\frac{\text{CMC}(\mathcal{A})}{n^{1.625}mD}\right)^m\right)$	$\tilde{\mathcal{O}}\left(\left(\frac{\text{CMC}(\mathcal{A})}{n^{1.5}mD}\right)^m\right)$

Table 1: Bounds on the unrecoverability advantage for iMHF constructions

Acknowledgments. Pooya Farshim was supported in part by EPSRC grant EP/V034065/1.

References

1. Password hashing competition, `https://www.password-hashing.net/`
2. Alwen, J., Blocki, J., Pietrzak, K.: Depth-robust graphs and their cumulative memory complexity. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part III. LNCS, vol. 10212, pp. 3–32. Springer, Cham (Apr / May 2017). `https://doi.org/10.1007/978-3-319-56617-7_1`
3. Alwen, J., Chen, B., Pietrzak, K., Reyzin, L., Tessaro, S.: Scrypt is maximally memory-hard. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part III. LNCS, vol. 10212, pp. 33–62. Springer, Cham (Apr / May 2017). `https://doi.org/10.1007/978-3-319-56617-7_2`

4. Alwen, J., Serbinenko, V.: High parallel complexity graphs and memory-hard functions. In: Servedio, R.A., Rubinfeld, R. (eds.) 47th ACM STOC. pp. 595–603. ACM Press (Jun 2015). https://doi.org/10.1145/2746539.2746622
5. Bellare, M., Ristenpart, T., Tessaro, S.: Multi-instance security and its application to password-based cryptography. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 312–329. Springer, Berlin, Heidelberg (Aug 2012). https://doi.org/10.1007/978-3-642-32009-5_19
6. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93. pp. 62–73. ACM Press (Nov 1993). https://doi.org/10.1145/168588.168596
7. Biryukov, A., Dinu, D., Khovratovich, D.: Argon2: New generation of memory-hard functions for password hashing and other applications. In: IEEE European Symposium on Security and Privacy (EuroS&P). pp. 292–302 (2016)
8. Blocki, J., Zhou, S.: On the depth-robustness and cumulative pebbling cost of Argon2i. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 445–465. Springer, Cham (Nov 2017). https://doi.org/10.1007/978-3-319-70500-2_15
9. Boneh, D., Corrigan-Gibbs, H., Schechter, S.E.: Balloon hashing: A memory-hard function providing provable protection against sequential attacks. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 220–248. Springer, Berlin, Heidelberg (Dec 2016). https://doi.org/10.1007/978-3-662-53887-6_8
10. Bonneau, J., Mironov, I.: Cache-collision timing attacks against AES. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 201–215. Springer, Berlin, Heidelberg (Oct 2006). https://doi.org/10.1007/11894063_16
11. Dodd, C., Farshim, P., Shahandashti, S.F., Southern, K.: Multi-instance unrecoverability of iMHF-based password hashing. Cryptology ePrint Archive, Report 2026/018 (2026), https://eprint.iacr.org/2026/018
12. Farshim, P., Tessaro, S.: Password hashing and preprocessing. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part II. LNCS, vol. 12697, pp. 64–91. Springer, Cham (Oct 2021). https://doi.org/10.1007/978-3-030-77886-6_3
13. Forler, C., Lucks, S., Wenzel, J.: Catena: A memory-consuming password scrambler. Cryptology ePrint Archive, Report 2013/525 (2013), https://eprint.iacr.org/2013/525
14. Oechslin, P.: Making a faster cryptanalytic time-memory trade-off. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 617–630. Springer, Berlin, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4_36
15. Percival, C., Josefsson, S.: The scrypt Password-Based Key Derivation Function. RFC 7914, IETF (2016), https://www.rfc-editor.org/info/rfc7914
16. PKCS #5: Password-based cryptography specification. RSA Data Security, Inc. (Sep 2000), version 2.0
17. Provos, N., Mazières, D.: A future-adaptable password scheme. In: Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference, June 6-11, 1999, Monterey, California, USA. pp. 81–91. USENIX (1999)

A Proof of Theorem 1

We first recall two lemmas from [4] used in our proof. The first lemma concerns the additive property for the cumulative complexity of two graphs [4, Lemma 3],

a consequence of this is that for any m and DAG $\mathcal{G}$, the minimum cost to pebble m instances of $\mathcal{G}$ is given by $\mathrm{CC}(\mathcal{G}^{\times m}) = m \cdot \mathrm{CC}(\mathcal{G})$. The second lemma concerns the predictor bound [4, Lemma 1], stating that, given a hint, a response from a random oracle can be successfully predicted with probability bounded by the size of the hint space divided by the size of the range of the random oracle.

Lemma 1 (Additive property of $\mathrm{CC}(\mathcal{G})$ [4]). *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two node-disjoint DAGs and let $\mathcal{G}$ be their disjoint union then, $\mathrm{CC}(\mathcal{G}) = \mathrm{CC}(\mathcal{G}_1) + \mathrm{CC}(\mathcal{G}_2)$.*

Lemma 2 (Predictor bound [4]). *Let $B = b_1, ..., b_u$ be a sequence of random bits. Let $\mathcal{P}'$ be a randomized procedure which gets a hint $h \in \mathrm{H}$, and can adaptively query any of the bits of B by submitting an index i and receiving b_i. At the end of the execution, $\mathcal{P}'$ outputs a subset $S \subseteq [u]$ of $|S| = k$ indices which were not queried, along with guesses for all $\{b_i | i \in S\}$. Then the probability (over the choice of B and randomness of $\mathcal{P}'$) that there exists some $h \in \mathrm{H}$ for which $\mathcal{P}'(h)$ outputs* all correct guesses *is at most $|\mathrm{H}|/2^k$.*

We now recall Theorem 1 and present an overview of the proof. We will leave the proofs of the individual claims to the full version of the paper [11].

Theorem 1 (Amortized strong memory-hardness). *Let $q, t, m, n, \delta, K, N \in \mathbb{N}$. Let $\mathcal{A}$ be a q-query t-time adversary with cumulative memory complexity $\mathrm{CMC}(\mathcal{A})$ and $\mathcal{G}$ be a DAG. Let C^{H} be the iMHF based on $\mathcal{G}$ in the $(nN^{\delta}K, N)$-RO model as defined in Section 2.2. Then*

$$\mathbf{Adv}^{(q,m)\text{-sMH}}_{\mathsf{C}^{\mathsf{H}}}(\mathcal{A}) \leq \frac{2q^2}{N} + 2^{-\gamma} \ ,$$

where $\gamma := \big(m \cdot \mathrm{CC}(\mathcal{G})[\log N - \log(qm)] - \mathrm{CMC}(\mathcal{A})\big)/t$.

In [4] the proof in the single-instance hash-independent setting proceeds via two claims. The first claims that a pebbling P, extracted from a transcript via the ex-post-facto pebbling extraction algorithm, is legal with high probability. The second claims that, with high probability, the total number of pebbles in the extracted P is upper bounded by an essentially linear function of $\mathrm{CMC}(\mathcal{A})$. Both claims rely on a certain "prediction argument", which, roughly speaking, states that a transcript giving rise to an illegal pebbling can be used to predict the output of a random oracle on a point without querying it. For an overview of how we adapt the techniques for our sMH game, see the main body in Section 3.

Proof. Fix all variables and graph as in the statement of the lemma. Let $\mathcal{A}$ be a q-query adversary in the (q, m)-sMH game with respect to C^{H}.

Transcript. An attack transcript T is a tuple $(\mathbf{q}_1, \ldots, \mathbf{q}_t, \sigma_O)$, where each $\mathbf{q}_i$ is a round of queries $(q_{i,1}, q_{i,2}, \ldots)$. Each $q_{i,j}$ is either of the form $(x_{i,j}, y_{i,j})$, where $x_{i,j}$ is a hash input and $y_{i,j}$ is a hash output, or $((v_{i,j}, \ell_{i,j}, \mathit{out}), \perp)$, where $v_{i,j}$ is a node, and $\ell_{i,j}$ is a labeling function output. Here, $\sigma_O = (\mathbf{x}, \mathbf{y})$ is the special output register that corresponds to the final output of the adversary.

Given a transcript of $\mathcal{A}$, we adapt the ex-post-facto pebbling algorithm of [4] to extract pebbling P of the graph $\mathcal{G}^{\times m}$ *even in a hash-dependent setting.*

Overview. Before extracting a pebbling, we first determine, from T, a vector $\mathbf{x}^*$ of m inputs for which $\mathcal{A}$ correctly computed the output of C^{H}. As we consider adversaries which may output more values than they correctly compute, we first extract the subset of inputs relevant to the pebbling. For each $k \in [m]$, the input $\mathbf{x}^*[k]$ defines a labeling of $\mathcal{G}$, and using these m computations, we extract a pebbling of the graph $\mathcal{G}^{\times m}$. A pebbling $P = (P_0, \ldots, P_t)$ is extracted by a two-step procedure in which we process each $\mathbf{q}_i$ to calculate P_i by first combining all nodes that are *correctly labeled* by queries in $\mathbf{q}_i$ with the previous set P_{i-1}. Importantly, the node to be added to the pebbling set needs to be calculated from the index-input pair for which the label is correct. That is, if a node $v \in \mathcal{G}$, is correctly labeled with respect to an input $\mathbf{x}^*[k]$, the node (v, k) will be added to the set P_i. Next, we only keep *necessary nodes* in P_i (these are nodes whose labels are used in a later step and not recomputed in the meantime) and remove all others. We start by defining a (hash) function corresponding to a transcript and node-labels computed with respect to it, before formalizing these notions.

Pre-label of a node. Given $(\mathbf{q}_1, \ldots, \mathbf{q}_t)$ we define a function H where $H(x_{ij}) := y_{ij}$ if $(x_{ij}, y_{ij}) \in \mathbf{q}_i$ for some i, and otherwise $H(x) :=\perp$. We also set $H(\perp) :=\perp$.[5] As we are dealing with different labelings of the same graph, we need to extend the notation for labels and pre-labels. For each $k \in [m]$ let $\mathbf{lab}_k$ and $\mathbf{pre\text{-}lab}_k$ be the labeling and pre-labeling functions as defined in Section 2.2 with respect to H and $l := \mathbf{x}^*[k]$ (recovered from the transcript). Recall that a pre-label $\mathbf{pre\text{-}lab}_k(v)$ for some node v takes the form $(v, \mathbf{lab}_k(\mathbf{Pa}_1(v)), \ldots, \mathbf{lab}_k(\mathbf{Pa}_\delta(v)))$, where $\mathbf{Pa}$, as before, is the parent function. In what follows, all labeling and pre-labeling are performed with respect to the H and $\mathbf{x}^*$ extracted from the transcript. We make H implicit, but specify k.

Correct call. Intuitively, a correct call for a node in $\mathcal{G}$ is the pre-label of the node. Formally, a query (x, y) in the transcript is called correct (for some input $\mathbf{x}^*[k]$) if there exists a non-sink node v in $\mathcal{G}$ s.t. $x = \mathbf{pre\text{-}lab}_k(v)$, or there exists a sink node v in the graph s.t. $x = (v, \mathbf{lab}_k(v), \mathit{out})$. We call v the output-node corresponding to (x, y) and any parents of v in $\mathbf{Pa}(v)$ an input-node for (x, y).

Necessary node. Intuitively, a (pebbled) node is called unnecessary if the pebble on it is not used in a subsequent round to pebble a child. Formally, for any $k \in [m]$, and any node v in graph $\mathcal{G}$, the node $(v, k) \in P_i$ is called necessary in a round i if there exists a batch of queries $\mathbf{q}_j$ in a later round $j > i$ that contains a correct oracle call with respect to input $\mathbf{x}^*[k]$, where node v in (graph $\mathcal{G}$) is an *input-node*, but there is no batch $\mathbf{q}_p$ with $i < p < j$ that contains a correct *oracle call* for v. Note that we are translating from a node in the graph $\mathcal{G}^{\times m}$ to a node in $\mathcal{G}$, so correctness is defined by the kth input in $\mathbf{x}^*$.

Extraction. We now define the ex-post-facto pebbling-extraction algorithm. Given a transcript T, the extraction algorithm sets $P_0 := \emptyset$, and defines $P_1, \ldots, P_t$ by

[5] We assume T satisfies the unique-output-value property expected from a function.

applying the following three rules to the each batch $\mathbf{q}_i$ in the order that they appear in the transcript.

1. Add all nodes in P_{i-1} to P_i.
2. For each query $x_{i,j}$ in the current batch, if $x_{i,j}$ is a correct call for some node v and input $\mathbf{x}^*[k]$, add (v, k) to P_i.[6]
3. For each (v, k) in P_i that is not necessary, remove it from P_i

Note that we say a node is *placed* in P (or pebbled) at some step i if it is added to P_i in step 2 above. This allows us to distinguish between the nodes inherited from a previous step and ones added due to a correct oracle call. We provide a pseudo-code description of the ex-post-facto pebbling extraction in the full version of the paper [11]. With the pebbling extraction defined, we can analyze a pebbling extracted from a hash-dependent adversary. To this end, we prove three claims. To prove the first (legality), we make a similar prediction argument to that made in [4], adapting the predictor to be suitable for hash-dependent adversaries by extending the predictor's hint, which replaces a q/N term by q^2/N. For the second (complexity), we again adapt the predictor's hint, this time to account for complications introduced by an adversary computing multiple instances of the construction. In particular, the predictor must know for which instance oracle calls are correct, and we pass this information for a certain set of queries, which replaces a $\log(q)$ term with $\log(mq)$ in the final bound. The final claim (collision free transcript) follows from a standard collision argument.

Claim (Pebbling legality). For a fixed q-query pROM adversary $\mathcal{A}$, with fixed random coins and a fixed H, let T be a winning transcript corresponding to $\mathcal{A}$ against (q, m)-sMH. Then the pebbling P extracted from T is legal with probability at least $1 - q^2/N$, over the choice of H and the coins of $\mathcal{A}$.

Accounting for hash-dependence and multiple instances. As described above, when referring to labels and pre-labels we will specify an index $k \in [m]$ to point to the input $\mathbf{x}^*[\mathbf{k}]$ for which a label or pre-label is correct. Additionally, as P is a pebbling for $\mathcal{G}^{\times m}$, we translate the node indices (for queries labeling distinct instances of a construction), into node indices for (distinct subgraphs of) $\mathcal{G}^{\times m}$. To extend the argument to a hash-dependent setting, we account for the fact that the adversary $\mathcal{A}$ is free to choose the input value x^*. As $\mathcal{A}$ may not decide upon x^* until its final step, we need to extend the hint to ensure that the predictor is still able to find a correct call for u and *avoid* forwarding the query to its oracle. To do this, we also include in the hint the index of the first correct call for u (if there is one). The additional index required increases the hint space by a factor of q, giving rise to the q^2/N bound from the claim.

Claim (Pebbling complexity). Let T be a winning transcript of a q-query pROM adversary $\mathcal{A}$, with fixed random coins and a fixed H, against (q, m)-sMH. Let P be the pebbling extracted from the transcript. Then for any $\lambda \geq 0$:

$$\Pr\left[\exists i \in [t] : |P_i| > \frac{|\sigma_i| + \lambda}{\log N - \log(qm)}\right] < t2^{-\lambda}$$

[6] This translates the labels $\mathcal{A}$ computes for nodes in $\mathcal{G}$ to pebbles for the graph $\mathcal{G}^{\times m}$.

over the choice of H and the coins of $\mathcal{A}$, where σ_i is the state (see Section 2.1).

Changes for multiple instances. An analogous result was proven in [4], for a hash-independent adversary computing a single instance of a construction. Here we adapt the argument to allow for adversaries computing multiple instances of the construction. In particular, the predictor needs to know *for which instance* any given query is correct. Without knowing this, much like in the previous claim, the predictor cannot avoid querying a point for which it would later make a prediction. We provide this information in the hint, which results in a $\log(qm)$ term replacing a $\log(q)$ term in our final bound. This claim and its proof are the same regardless of whether the adversary is hash-independent or not, since it bounds the amount of memory required to store each pebbling set P_i, which intuitively is unaffected by the adversary not being hash-independent.

Next, we bound the probability of a collision in the labels $\mathcal{A}$ computes, and then move onto bound the probability $\mathcal{A}$ wins given there are no collisions.

Claim (Collision-free transcript). Let $\mathcal{A}$ be a (q, m)-sMH adversary, and bad be the event that $\mathcal{A}$ finds a collision in any two labelings $\mathbf{lab}_j$ and $\mathbf{lab}_k$ for some $j, k \in [m]$. Note that the $\mathcal{A}$'s advantage never decreases when bad occurs, and

$$\mathbf{Adv}^{(q,m)\text{-sMH}}_{\mathsf{C}^{\mathsf{H}}}(\mathcal{A}) \leq \Pr[\mathsf{bad}] + \Pr[\mathcal{A} \text{ wins } (q,m)\text{-sMH}_{\mathsf{C}^{\mathsf{H}}} | \neg \mathsf{bad}] \ .$$

The above three claims allow us to complete the proof for Theorem 1. For any q-query adversary $\mathcal{A}$ in the (q, m)-sMH game w.r.t. C^{H}, that produces a collision-free transcript T, let P be the ex-post-facto pebbling of $\mathcal{G}^{\times m}$ extracted from T. Let $\mathcal{W}$ be the event that $\mathcal{A}$ wins the game, and assume $\Pr[\mathcal{W}] \geq \epsilon$. Let $\bar{\epsilon} := \log(\epsilon - q^2/N)$, t be the number of steps in P, and $\mathcal{C}$ be the event that P is a legal pebbling *and* that

$$\sum_{i=0}^{t-1} |P_i| \leq \frac{t\bar{\epsilon} + \sum_{i=0}^{t-1} |\sigma_i|}{\log(N) - \log(qm)} \ .$$

By claim 1 (legality), claim 2 (complexity), and an application of the union bound, we have shown that $\Pr[\mathcal{C}] > 1 - (q^2/N) - 2^{-\bar{\epsilon}} = 1 - \epsilon$. Therefore, the probability that $\mathcal{W}$ and $\mathcal{C}$ occur simultaneously is positive, and so an adversary whose transcript satisfies $\mathcal{C}$ can win with probability greater than ϵ. As such a pebbling is legal and complete, the cumulative pebbling cost of P must by definition be lower bounded by $\mathrm{CC}(\mathcal{G}^{\times m})$. Therefore

$$\mathrm{CC}(\mathcal{G}^{\times m}) \leq \frac{-t \log(\epsilon - q^2/N) + \mathrm{CMC}(\mathcal{A})}{\log(N) - \log(qm)} \ .$$

Applying Lemma 1, and rearranging, we obtain the bound for ϵ, and by collecting terms we arrive at the statement in the lemma. □