Hash-Dependent Passwords and Memory Hard Functions

Charles Dodd

PhD

University of York

Department of Computer Science

September 2025

Abstract

Hash rates provided by specialized hardware such as ASICs render schemes based on iterated hashing ineffective in protecting password storage, and hence memory-hard functions (MHFs) are recommended for modern password storage. The study of MHFs so far has mainly focused on providing provable guarantees on their minimum expected time–memory complexity cost per evaluation. However, this does not tell us to what degree passwords remain unrecoverable if password banks protected using MHFs are compromised. This thesis aims to answer this question in the context of up to date password security models. This means proving unrecoverability bounds for a large class of MHFs: graph-based data-independent MHFs (iMHFs). Specifically, we prove upper bounds on the *multi-instance* unrecoverability of any graph-based iMHF, based on the unguessability of stored passwords, the number of hash evaluations it carries out, and crucially the cumulative memory complexity of the adversary.

To prove this result, we extend the standard definitions of both guessability, and of memory-hardness to strictly stronger *hash-dependent* settings. We prove reductions from unrecoverability to unguessability in the hash-dependent setting and show that, unlike the hash-independent setting, unpredictable salting is essential in the hash-dependent setting. We then prove that, in contrast to general MHFs, graph-based iMHFs are memory-hard in the hash-dependent setting.

In the last part of the thesis, the analysis is extended to prove indifferentiability results for iMHFs in a large class of multi-stage games, i.e., games which follow a sample, salt and use structure. We show that in this class of games, an iMHF is as secure as a random oracle. Importantly, we give a treatment of adversaries with access to preprocessing information.

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. The work was undertaken as part of a research project with my collaborators: Siamak Shahandashti, Pooya Farshim, and Karl Southern, who all advised on the editing, use of proof techniques, and direction of research. This work has not previously been presented for a degree or other qualification at this University or elsewhere. A version of some of the material in Chapters 3 to 5 is published on crypto e-print [28] and will appear at ESORICS 2026, and a version of Chapter 6 has been submitted to TCC 2026; both were co-authored with the same collaborators as above. All sources are acknowledged as references.

Acknowledgements

Firstly, I would like to express deep gratitude to my supervisors, who have been instrumental in both my academic development and the completion of this thesis.

I am grateful to Dr. Pooya Farshim for recruiting me as a PhD student, and giving me the opportunity to embark on this research project. His patient guidance in teaching me both the fundamental techniques of cryptography and the art of research has been invaluable. I was continually impressed by his remarkable foresight in anticipating potential pitfalls in research directions, often steering us away from unproductive paths before significant time was lost. His generosity with time throughout this process, especially during the challenging moments when progress seemed slow, has made this work possible.

I would like to extend my sincere thanks to Dr. Siamak Shahandashti for welcoming me so warmly into the York Computer Science department. His willingness to undertake substantial administrative work on my behalf has been extraordinary and greatly appreciated. Throughout, his helpful advice has guided me through both academic and practical challenges. I am particularly grateful for his keen eye in spotting opportunities that have enriched my research experience, and for his thoughtful supervision that has helped shape both this thesis and my development as a researcher. Without the combination of both their expertise and support this thesis would not have been possible.

Mostly, I want to thank my wife Matty and daughter Sylvie, who have been a continuous source of motivation.

1 Introduction

1.1 Problems and Motivation

Password storage. Passwords remain a prevalent form of secret entropy as they can be generated and memorized by users. Password-based cryptography leverages the prevalence of passwords to underpin multiple applications such as authentication and key derivation. Interacting with these systems has become an everyday feature of modern life, and many inherently security-critical applications are protected by passwords. Even so, being user generated, passwords are chosen predictably. Though users do appear to choose stronger, i.e. less predictable, passwords to protect finances, stronger passwords are more likely to be reused across accounts [8, 27]. The most basic protection is to apply a cryptographic hash function H to each password pw, and store the output. When the user logs in, the hash of their password is checked against the database, so access to the server doesn't give direct access to the passwords. However, where the bank of hashes is leaked, adversaries can check guesses for predictable passwords against the hashes. That is, for a known hash H, an adversary can guess a common password pw, calculate $\mathsf{H}(pw)$ and check their guess against the bank.

Protections. There is a limited amount of defense one can hope to achieve here, but the main strategy is to increase the cost of the adversary's guesses. A common approach in such applications is to protect through salting, i.e., rather than a hashed password $\mathsf{H}(pw)$, a pair of the form $(sa, \mathsf{H}(pw||sa))$ is stored for a random salt sa. Intuitively, the addition of the salt attempts to ensure an adversary cannot recognise the recurrence of a common password in the bank. So even if the same password appears twice, the adversary will still have to make the guesses again to crack both passwords..

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 [51] and PBKDF2 [50]. These constructions are designed to make bulk computations prohibitively costly, while incurring only a small cost on authenticating honest users. The computational cost of such constructions can be controlled

by an *iteration count*. However, the ever-growing efficiency of specialized hardware such as ASICs in evaluating hash functions necessitates higher counts that disproportionately affect honest users. The iteration count needed to sufficiently slow down adversaries with dedicated hardware noticeably slows the login time for users computing a single hash.

To address this issue, recommendations for password hashing [48] advocate the use of *memory-hard functions* (MHFs) that are 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 it does for computation.

Candidates for MHFs include Scrypt [49], the first proposal for such a function in 2009 by Percival, and Argon2 [14] chosen through the Password Hashing Competition in 2015 [48]. MHFs appear in two flavours, distinguished by the function's 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. This is important when the function is run on a secret input in an insecure environment, as dMHF memory access patterns could allow side-channel attacks [21]. Constructions of iMHFs are usually based on an underlying *directed acyclic graph* that basically specifies how hash functions are combined to give the construction. Such iMHFs are referred to as *graph-based* iMHFs. Examples include Argon2i [14], Catena [32], and Balloon hash [20].

An illustration. A good example to give intuition for how these data-independent graph-based constructions are designed is the structure of Catena [34]. A directed acyclic graph (DAG) is chosen, here based on a kind of graph called a bit-reversal graph. See Fig. 1.1, for an illustration of a simple construction based on the bit-reversal graph, taken from the specification of Catena [34]. The construction is evaluated by hashing the input to label the first node (v_0^1), and all other nodes are labelled by hashing the labels of their parent nodes. The output of the construction is the label of the final node in the DAG. The actual graph used by Catena has many layers, as both the length of the top chain and the number of layers are tweak-able parameters for the construction. Such constructions are data-independent because the graph does not depend on the input, i.e., the graph is fixed from the outset. Other, data-dependent, constructions like Scrypt [49] build the graph on the fly, using node labels to determine

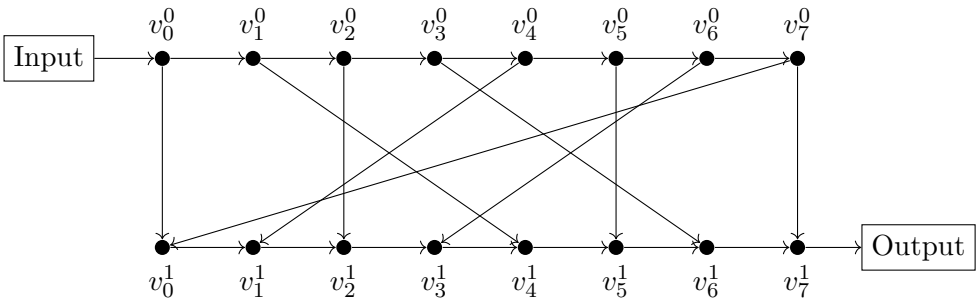

Figure 1.1: Illustration of the bit-reversal graph for graph-based iMHF Catena.

the edges, potentially leaking information about the input in the structure of the graph itself.

Multi-instance security. Note that due to the prevalence of weak passwords, even if they are stored salted, and under iterated hashing or MHFs, they are vulnerable to guessing attacks. An adversary with access to a leaked password database gets a list of password hashes $(sa_i, \mathsf{C}^{\mathsf{H}}(pw_i||sa_i))$ for different user passwords pw_i. To recover a password, the adversary can guess common passwords, say pw^*, compute $\mathsf{C}^{\mathsf{H}}(pw^*||sa_i)$ for each pw^* and compare it with each $\mathsf{C}^{\mathsf{H}}(pw_i||sa_i)$ with a good probability of success. Hence, there is an inherent limit to the protection any such construction is able to provide for *individual* passwords. Despite this, one still would like assurances that the effort needed to recover *multiple* passwords increases, and ideally scales linearly, with the number of passwords recovered. The *multi-instance* (mi) security model, introduced by Bellare et. al. (BRT12) [10], formalizes this and requires multiple instances of a hashing method to be broken by an adversary, capturing how the resources expended by the adversary grow with the number of target hashes that are cracked. Note that without a multi-instance security model, the effect of salting passwords is not captured. This model was further studied by Farshim and Tessaro (FT21) [31], where the authors proved the efficacy of salting in defeating pre-processing attacks, e.g., rainbow tables [46], in the multi-instance setting. Recent work [22] has studied public key encryption in a multi-instance setting to investigate security against mass surveillance, including an investigation into different notions of multi-instance security for a PKE setting.

Cost metrics. To formally analyze the security of password datasets protected by iterated hashing or MHFs, we need to be able to measure the time and memory requirements of such constructions. The original cost metric proposed by Percival, the maximum memory usage multiplied by time, has been shown to provide insufficient guarantees against *amortization* [5]. Specifically, the maximum memory usage can be amortized across many evaluations by sharing memory across multiple CPUs. The state-of-the-art metric for measuring memory cost, proposed by Alwen and Serbinenko (AS15) [5], is the *cumulative memory complexity* (CMC): the *sum* of memory usage at each time step in the computation.

Capturing the non-amortizability of a construction guarantees that the effort required to compute the function scales essentially linearly with the number of instances computed. Alwen and Serbinenko proved 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 [5].

Alwen and Blocki [2] developed a general attack on graph-based iMHFs with a fixed in-degree, and concluded that no such iMHF can be maximally memory hard. That is, no such iMHF with n nodes and a hash-digest length w can achieve a required cumulative memory complexity of $\Omega(n^2)$. Maximum memory hardness can still be achieved by dMHFs, Scrypt in particular achieves the maximum memory hardness bounds [4]. Since then the literature has seen further theoretical developments [3, 24, 16] and practical schemes [20].

Unrecoverability. The natural security property expected from a password storage system is that given a bank of leaked protected passwords, e.g., those of the form $(sa, \mathsf{C}^{\mathsf{H}}(pw||sa))$, it is hard to *recover* a significant number of the original passwords. This property can be modeled as *multi-instance unrecoverability* which captures how the advantage of an adversary recovering original passwords relates to the number of passwords recovered and the resources the adversary needs to expend in terms of time and memory. Although provable guarantees on the memory-hardness of graph-based iMHFs provide high confidence that computation would incur a certain minimum memory cost, it is not clear how, or if, they would translate into provable unrecoverability guarantees for password storage using graph-based iMHFs. Consider a hash-based iMHF C^{H}, then we can define another function MHF, such that for any input x, we let $MHF(x) := \mathsf{C}^{\mathsf{H}}(x)||x$. Clearly if C^{H} is memory-hard, then so is MHF.

But given any output y of MHF, it is trivial to recover the input x such that $MHF(x) = y$. To date, there has been no direct analysis of the unrecoverability of passwords stored under MHFs in the literature. These are the significant gaps that the first main results in this thesis will address.

Hash dependence. Existing frameworks in the literature fall short of providing the required tools for analyzing the unrecoverability of passwords protected by constructions such as iterated hashing and graph-based iMHFs. The AS15 framework enables the analysis of the CMC of such constructions, but does not support translating these analyses to unrecoverability of passwords protected by such constructions. On the other hand, while the FT21 framework (itself a generalization of BRT12) allows the analysis of the unrecoverability of monolithic password hashing with respect to the query complexity of the adversary and the inherent guessability of the passwords, it does not account for the CMC of the adversary and lacks the mechanics to support the analysis of more complex constructions. This is yet another gap that we address to enable our main result.

The AS15 and FT21 frameworks can both be seen as being in a *hash-independent* setting. That is, the definitions of unrecoverability and unguessability in FT21, and that of memory-hardness in AS15, both assume that passwords are chosen *independently* of the hash functions used in the constructions. This is too strong of an assumption and neither corresponds to the diversity of practices in choosing passwords, e.g., passwords chosen by password managers, nor conforms to the nature of the random oracle model in which all entities are assumed to have access to the random oracle, including *password samplers* that model passwords selected according to specific distributions.

Indeed, several existing composition methods derive passwords that *are* explicitly dependent on hash functions. Pronounceable password generators compose passwords using hash-based operating system PRFs, e.g., on Linux, `pwgen` [1] and `secpwgen` [58] both use `/dev/(u)random`, a hash-based PRF (also see [57, 59]). Many organizations enforce or strongly recommend system-assigned passwords that are typically based on hash-based PRFs as well. Hash-based password managers, e.g., Password Multiplier [37], `pwdHash` [55], and Passpet [60], generate domain-specific passwords by hashing a user-chosen password along with the domain information, to make password-reuse safer (also see [35, 56]).

First set of contributions

The first main contributions of this thesis are in three areas.

- *Practical*: This is the first formal treatment of the single and multi-instance unrecoverability of graph-based iMHFs. Importantly the unrecoverability bounds are based on the unguessability of the underlying passwords, the cumulative memory complexity of the adversaries, as well as the cumulative (pebbling) complexity of the underlying graph.

- *Technical*: To prove the above bounds requires combining the FT21 and AS15 frameworks to enable the analysis of the unrecoverability of graph-based iMHFs. Specifically, this requires bounding the the number of guesses of any adversary with a given cumulative memory complexity in terms of the cumulative (pebbling) complexity of the underlying graph, via the ex-post-facto pebbling argument of AS15. Roughly speaking, this ex-post-facto argument takes a parallel algorithm evaluating a graph-based function and builds one that pebbles the underlying graph in a parallel graph pebbling game. Via a reduction argument, one can use known properties of the graph to deduce properties of the original algorithm. By then applying an unrecoverability to unguessability reduction, via an extended version of FT21 techniques, we derive the final bound.

- *Definitional*: To enable the above treatment, this work enhances the definitions of unguessability (of FT21), unrecoverability, and memory hardness (of AS15) to a more general *hash-dependent* setting, and show analogous theorems to those of FT21 and A515 in this setting. Along the way, while generalizing the FT21 framework, we give an attack that demonstrates a clear separation between the hash-dependent and hash-independent settings.

Broader security notions. After providing a positive answer to the question 'are passwords secure when stored under an iMHF?', we raise the natural question 'in what other security games are iMHFs a secure replacement for a hash function?'. In fact, cryptographic schemes that are instantiated with hash functions are generally proven secure in the random oracle model (ROM). In this model, all parties are given oracle access to a random function, which essentially captures the ideal functionality of a hash function.

However, this black-box treatment of the hash function becomes difficult to motivate when cryptographic schemes are instantiated using concrete hash functions constructed with another primitive as a building block. Aside from iMHFs, one prominent example is the Merkle-Damgård construction, which builds a hash function from an underlying compression function. Using such a hash function, which is a mode of operation for a compression function, causes a disparity between the real world and the security model. A scheme may be proven secure in the ROM but in the real world, all parties also have access to the underlying primitive. So while there may be a security proof when the hash function is modelled as a random oracle, in practice, the construction has a structure that makes it trivially distinguishable from the random oracle. Where this is the case, a security proof in the ROM doesn't apply. Examples, like the length-extension attacks on the Merkle-Damgård construction [29], illustrate the need to give all parties access to the underlying primitive as well as the construction.

One solution to this is to prove security for constructions in each scheme on a case-by-case basis. This would mean a new security proof for every application of an iMHF, even in schemes which are known to be secure in the ROM. Maurer, Renner, and Holenstein [42] introduced a better solution, which allows for composability. In their *indifferentiability* framework, given a public primitive f, the goal is to build another primitive F using a construction C^f. For example, given a hash function H, and a hash-based construction C^{H}, we want to know under what conditions C^{H} can securely replace a random oracle RO.

The indifferentiability framework. Maurer, Renner, and Holenstein [42] formalized the conditions under which a construction can securely stand in for a random oracle in the following way: if there exists a simulator $\mathcal{S}$ such that every distinguisher interacting with the systems (C^f, f) and $(F, \mathcal{S}^F)$, finds them indistinguishable.

Intuitively they define an indistinguishability game, for a distinguisher $\mathcal{D}$, that gets access to two oracles: one for the construction, and one for the building-block primitive. The distinguisher then has to distinguish between two worlds. In the 'real' world, any query x to the construction or primitive oracles are answered by $\mathsf{C}^f(x)$ and $f(x)$ respectively. In the 'ideal' world, the construction queries are answered by $F(x)$, where F is the ideal primitive or functionality against which the construction is being measured, e.g., a random oracle RO. The queries to the primitive oracle

are answered by a simulator $\mathcal{S}$, which has oracle access to F. One can show that the construction C^f is indifferentiable from F, if one can build a $\mathcal{S}$ such that it simulates the primitive f using queries to F.

This is, at first, counter-intuitive. However, the composition theorem in [42] clarifies why this is a sufficient condition. Informally, one can show that any game $\mathcal{G}^{\mathsf{C}^f}$ running adversary $\mathcal{A}^f$ can be viewed as a distinguisher $\mathcal{D}^{\mathsf{C}^f,f}$. Thus, by an indifferentiability result, one guarantees there is a simulator $\mathcal{S}$ such that the real and ideal world are indistinguishable from the point of view of the composed game and adversary pair. Then for any attacker $\mathcal{A}^f$ in the real world, there is an attacker $\mathcal{B}^{\mathsf{RO}} := \mathcal{A}^{\mathcal{S}^{\mathsf{RO}}}$ in the ideal world with a similar advantage. And so, if this indifferentiability property holds for some construction C^f, it can securely replace F.

Limits of composition. Unfortunately, in [52] it was shown that the reach of the composition theorem is limited to *single-stage* games. The authors use the example of hash-based storage auditing to show that even a construction which is indifferentiable from a random oracle is insecure in the scheme. They show that the problem arises from the multiple stages of the game. A single-stage game is best understood as a counterpart to a multi-stage game. In a multi-stage game the adversary is split into multiple stages, e.g., $(\mathcal{A}_1, \mathcal{A}_2)$. In a single-stage game the adversary stays whole. For example, the ordinary (hash-independent) password unrecoverability game is a single-stage game. There, the game samples some passwords, hashes them, and gives the challenges to the adversary, who attempts to recover them using queries to the hash. In a hash-dependent unrecoverability game, both the sampler and adversary make queries to the hash, but there is a state passed between them via the game. As the game is split into two, the ordinary composition theorem no longer applies. The simulator which makes up part of the ideal world adversary is not split. Intuitively, one needs a separate simulator for each stage of the adversary. For single-stage games, Alwen and Tackmann introduced a notion of indifferentiability which allows one to prove resource-aware bounds [7]. Using this definition, they were able to argue for the single-stage indifferentiability of graph-based iMHFs, à la AS15 (with a slight modification).

Multi-stage games. In a multi-stage setting, the distinguisher is essentially split into stages, possibly passing along some state. Ristenpart et al. [53] showed that as long as the simulator is stateless, the composition theorem will cover multi-stage security games. However, this is a

difficult requirement to meet, and proofs for many constructions require the simulator to keep state. Another technique which gives composition in multi-stage games is reset indifferentiability [41]. In this setting the distinguisher can reset the simulator at will, essentially wiping its state. The authors of [41] proved multiple impossibility results for multi-stage indifferentiability, in particular they showed that no domain extender is multi-stage indifferentiable from a random oracle. A more attainable requirement was proposed by Mittelbach [45], who showed that if the hash-based construction and security game in question are shown to satisfy a certain property called *unsplittablility*, the indifferentiability composition theorem will apply. This unsplittablility requirement ensures that the probability of computations being completed across multiple stages of the game is small. Intuitively, from the point of view of the simulation, this makes the different stages independent, and nullifies the attack used in [41]. Unfortunately, as well as having a large definitional overhead – making it difficult to extend modularly – the unsplittability notion is only defined for a limited set of hash-based constructions. Indeed, the result was proven before the introduction of the aforementioned Password Hashing Competition [48], which perhaps explains why it does not cover MHFs.

Preprocessing adversaries. There is another large class of games which does not fall under the range of either the single-stage, or standard multi-stage definitions – games with 'auxiliary input'. In these games, the first stage adversary is given unbounded access to the random oracle, as opposed to just bounded query access, and outputs some auxiliary information for the second-stage adversary to receive as input. Intuitively, these games model preprocessing attacks on the hash functions, and importantly for this work (in light of the earlier sections) there exist such attacks on passwords. For example, the attacks on unrecoverability of passwords using Hellman tables [38] or rainbow tables [46]. In attacks like these, an adversary preprocesses the hash function, building an advice string (or table) to inform later attacks. To account for these kinds of attacks, one requires the model to grant an adversary unbounded access to the underlying oracle.

One approach, introduced by Coretti et al. [25], is to consider a simplified model in which the preprocessing information only fixes some inputs and outputs of the random oracle, and the rest of the values are resampled. Intuitively, even with unbounded access to the input-output points of the random oracle, a preprocessing adversary can only communicate some

of those. From the point of view of an online adversary, the rest of the points are still random. While other techniques, such as the compression arguments used by [36], can provide tighter bounds, the benefit of the bit-fixing approach is that it allows one to use the established proof techniques of the random oracle model. Recent work [23] has given some conditions under which one can say a construction that is indifferentiable from a random oracle remains so in the presence of preprocessing information. The requirement is called perfect reset indifferentiability, which in practice requires a stateless simulator which is able to successfully simulate for distinguishers with unbounded queries. However, such a requirement is difficult to meet, and the examples given by the authors do not cover the more complicated graph-based constructions. Indeed, the proof was limited to a single round of the sponge construction.

C-amplification. Clearly, the query complexity of the simulator is important to the security bounds. In order to interpret indifferentiability bounds for some construction, one should relate the number of queries made by the simulator to the number of complete computations made by the distinguisher. As the aim is to show that for any attacker in the real world, there is a similar attacker in the ideal world, we should aim to guarantee the attack in the ideal world makes roughly as many queries as the number of full C computations made by the attacker in the real world. The authors of [11] call this a c-amplifying simulator. This is related to the indifferentiability framework of [7], which gives the simulator limited computational resources.

The approach of this thesis. This work takes a different approach, and defines a multi-stage indifferentiability notion which gives composition for a broad class of multi-stage games, which we refer to as *sample, salt and use (SSU)* games. Intuitively, this class of games captures the standard game structure of sampling some inputs, salting them, and then using the hash of these salt-input pairs in a security game. The sampler is given access to the underlying primitive, making the notion multi-stage. Games which fall under this class include an auxiliary-input, hash-dependent version of a KDF security game [40, 30]. Notably, this also includes the hash-dependent unrecoverability notions introduced in the first sections of the thesis.

Second set of contributions. The final chapter of the thesis contributes in the following ways:

- We give a formal treatment of multi-stage indifferentiability games with preprocessing adversaries for graph-based constructions. We are able to show that iMHF constructions, based on graphs with at most one source and sink, are indifferentiable from a random oracle in a wide class of games we call 'sample, salt & use' (SSU) games.

- Using a modular, game-based approach, we define a condition under which an indifferentiability result can be lifted to indifferentiability in our multi-stage setting. By restricting the composition, we give conditions for which the c-amplification of the simulators may be preserved.

- To prove the above results requires developing new techniques that bridge single-stage and multi-stage security analysis. Specifically, this requires establishing conditions under which standard indifferentiability results extend to multi-stage settings, developing a restricted indifferentiability notion that preserves c-amplification properties of simulators, and proving composition theorems for restricted classes of SSU games.

Taken together, these contributions extend the literature by accounting for multi-stage, and preprocessing games simultaneously, and giving the conditions for efficient simulators.

Blocki, Holman, and Lee [17] introduced the parallel reversible pebbling game to study the reversible space-time complexity of iMHF graphs - modelling quantum adversaries. They extended the pebbling attack of Alwen and Blocki to the parallel reversible game, and showed that an efficient classical pebbling implies an efficient reversible pebbling [18].

1.1.1 Structure of the thesis

In order to avoid an overwhelming number of definitions early on, the preliminaries section will introduce only those notions which are common to all the later sections. Definitions that are more specific to certain sections will be introduced where they are used. Additionally, further discussions of the literature are left to appear nearer to the relevant sections. This again allows the technical definitions to appear in better context, and for better readability in general.

Chapter 2 covers the basic notation and security definitions, and Chapter 3 covers our contributions generalizing the FT21 framework. In Chapter 4,

we extend the AS15 framework and define hash-dependent memory hardness. Using these results, we prove multi-instance unrecoverability bounds for graph-based iMHFs in Chapter 5. Finally, in Chapter 6 we depart from the frameworks for proving unrecoverability, to more general indifferentiability notions. We define some multi-stage indifferentiability games, to prove security bounds and a composition result for a general class of games. We end this chapter by giving some restrictions on the notions which will allow for more efficient simulators.

2 Preliminaries

This first chapter is given as a repository of the important definitions used throughout the paper. To avoid front-loading definitions and security notions which are not used until the final chapter, only the essentials are given in these preliminaries. To aid reading, the later definitions appear in-line, and in their proper context.

We start with a brief overview of some important notation specific to the area. A more extensive overview is given in Table 2.1 which may be a useful reference for the later chapters, especially Chapter 6. We then cover the main security games, used in Chapters 3 to 5.

2.1 Notation

We denote the size of a set N by $|N|$ and the length of a vector $\mathbf{x}$ by $|\mathbf{x}|$. For sampling a value v uniformly at random from a finite set N, we write $v \twoheadleftarrow N$. When sampling a value v randomly from a distribution $\mathcal{W}$ (or by a randomized algorithm), we write $v \twoheadleftarrow \mathcal{W}$. We use the symbol $\leftarrow$ for a deterministic assignment. 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. We say an algorithm that makes at most q queries to its oracle is a q-query algorithm. We describe a two-stage algorithm that makes at most q oracle queries in the first stage, and q' queries in the second, as a (q, q')-query algorithm. For an algorithm $\mathcal{A}$ with unbounded access to $\mathcal{O}$, we write $\mathcal{A}(\mathcal{O})$. To denote a game $\mathcal{G}$ that runs an algorithm $\mathcal{A}^{\mathcal{O}}$, we write $\mathcal{G}^{\mathcal{A}^{\mathcal{O}}}$. As a slight abuse of notation, if the game $\mathcal{G}$ also has access to some oracle $\mathcal{O}'$, we write $\mathcal{G}^{\mathcal{O}', \mathcal{A}^{\mathcal{O}}}$. We use the notation $\overline{\mathcal{A}} = (\mathcal{A}_i, \ldots, \mathcal{A}_n)$ to denote a sequence of algorithms. In this thesis, we consider constructions C^{H} built from a primitive H, modelled as a random oracle. The constructions we consider come with a canonical algorithm for evaluating them honestly.

Notation	Description
$\lvert N \rvert$	Size of a set N
$\lvert \mathbf{x} \rvert$	Length of a vector $\mathbf{x}$
$v \twoheadleftarrow N$	Sampling a value v uniformly at random from a finite set N
$v \twoheadleftarrow \mathcal{W}$	Sampling a value v randomly from a distribution $\mathcal{W}$ (or by a randomized algorithm)
$v \leftarrow$	Deterministic assignment
$[m]$	Set of integers from 1 to m
$\mathrm{Fun}(N, M)$	Set of functions with domain N and range M
$\mathcal{A}$	A randomized algorithm (adversary)
$\Pr[\ldots]$	The probability that some game outputs 1.
Adv	The probability that the adversary wins the game.
$\mathcal{A}^{\mathcal{O}}$	Algorithm $\mathcal{A}$ with access to oracle $\mathcal{O}$
ε	Empty string
q-query algorithm	Algorithm that makes at most q queries to its oracle
(q, q')-query algorithm	Two-stage algorithm making at most q oracle queries in first stage, q' in second
$\mathcal{A}(\mathcal{O})$	Algorithm $\mathcal{A}$ with unbounded access to oracle $\mathcal{O}$
$\mathcal{G}^{\mathcal{A}^{\mathcal{O}}}$	Game $\mathcal{G}$ that runs algorithm $\mathcal{A}^{\mathcal{O}}$
$\mathcal{G}^{\mathcal{O}', \mathcal{A}^{\mathcal{O}}}$	Game $\mathcal{G}$ with access to oracle $\mathcal{O}'$ that runs algorithm $\mathcal{A}^{\mathcal{O}}$
$\overline{\mathcal{A}} = (\mathcal{A}_i, \ldots, \mathcal{A}_n)$	Sequence of algorithms
$\mathcal{D}$	Distinguisher
$\mathcal{S}$	Simulator
T_i	Number of queries to the i-th primitive oracle
T_{Const}	Number of queries to the construction oracle
T_{Gen}	Number of queries to the primitive, induced by the construction queries
C^{H}	Construction built from primitive H (modelled as random oracle)
st_i	A state, generated by, and passed from, the i-th game
st_a	A state generated for an adversary
$\mathcal{G}_i$	The i-th game
$\mathbf{pw}, \mathbf{sa}, \mathbf{y}$	Password, salt, and challenge vectors
$\perp$	A null return or aborting a process
Gen	Salt generator
$T[x]$	The entry in the function table for input x

Table 2.1: Notation used throughout this thesis.

2.1.1 The parallel random oracle model

The random oracle model (ROM) [12] is an idealized model of computation in which all parties, honest or otherwise, have access to a uniformly sampled function $\mathsf{H} \twoheadleftarrow \mathrm{Fun}(N, M)$. That is, at the outset of any game in the ROM, a function H is sampled uniformly at random from the space of functions with domain and range N and M respectively. Since, the image of the particular function is unknown to all parties (until receiving responses from queries), a standard technique to analyse these games is to treat each response to a new query as sampled uniformly at random. We consider adversaries in the *parallel random oracle model* (pROM) [5]. 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 some input z, and makes t rounds of batch oracle calls. In each round, $\mathcal{A}$ makes a batch of queries $\mathbf{q}$ and writes to a state st. This state st is "free" in the sense that the space it uses is not counted towards the space-time complexity of the algorithm $\mathcal{A}_*$. This state captures any memory used within a round, and models the fact that the algorithm is not reset in-between making queries and receiving the responses.

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 the algorithm terminates, the contents of $\sigma_{\mathcal{O}}$ are returned. Essentially, the special output register allows the algorithm to "bank" its output early, and so not dedicate space to storing a part of its output, calculated early, for the rest of the computation. See Fig. 2.1 for the structure of a pROM algorithm.

2.2 Basic security definitions

We now describe our basic security metrics. We begin by motivating and describing the notion of a hash-dependent password sampler. Then, we use this to extend the security games from BRT12 and FT21 to a hash-dependent setting.

Following FT21, a *password sampler* $\mathcal{P}_m$ is defined to be a randomized algorithm which takes no input, and outputs a vector $\mathbf{pw}$ of m passwords and possibly some leakage on the passwords z. This leakage models any

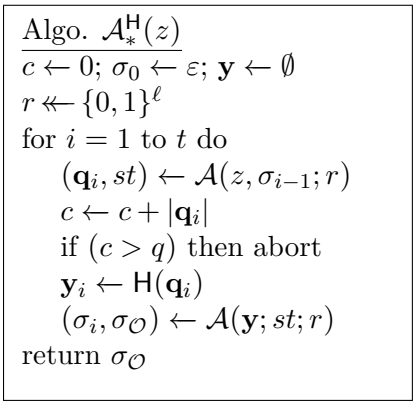

Figure 2.1: Structure of an algorithm in pROM.

information the adversary can obtain before the attack about the password distribution.

Hash-dependent password sampler. The hash-dependend notion is put to technical use to prove the unrecoverability of MHFs. Even in the hash-independent setting, where password samplers *do not* have access to the hash function, the adversary in the unrecoverability game evaluates inputs whilst having query access to the hash. Such adversaries can choose to evaluate the construction on inputs based on the output of the hash. We use the hash-dependent notion to derive security bounds which account for this.

We define a hash-dependent m-password sampler $\mathcal{P}^{\mathsf{H}}_m$ to be a randomized algorithm that takes no input, and outputs a vector $\mathbf{pw}$ of m passwords and some leakage on the passwords z, whilst having oracle access to H.

To demonstrate the effect the notion of a hash-dependent sampler has on the unrecoverability of passwords, in Section 3.3, we present such a sampler which is susceptible to an attack in the UR game, but permits only negligible success probability against any Guess adversary.

Salts generators. 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 a salt $\mathbf{sa}[i, j]$ from a space of size K. We denote a generator sampling uniformly at random from a space of size K as $\mathsf{Gen} = [K]$. In the case no salting is used, Gen outputs m empty salts. We denote this latter case as $\mathsf{Gen} = \epsilon$ and we assume $\ell = 1$.

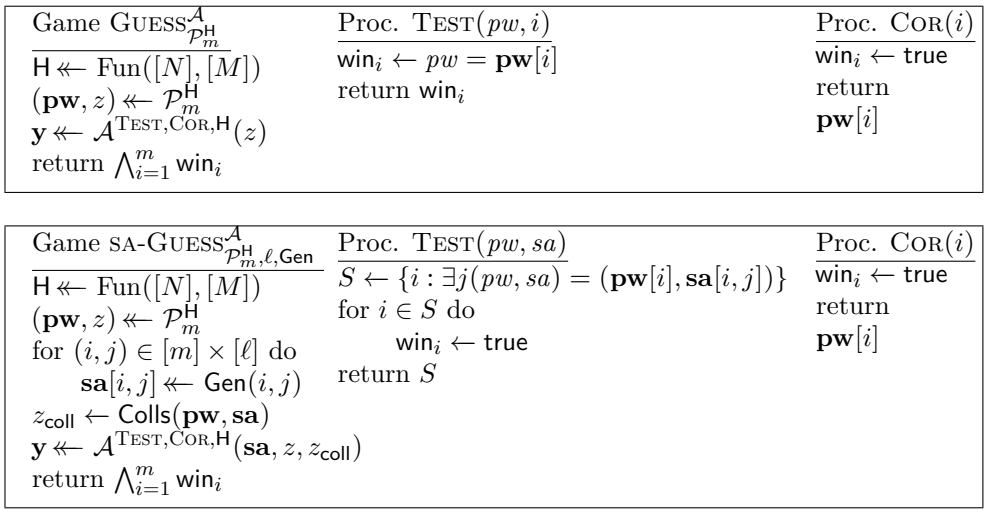

Figure 2.2: The hash-dependent GUESS and SA-GUESS games with respect to $\mathcal{P}^{\mathsf{H}}_m$.

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 $(\mathbf{pw}[i_1], \mathbf{sa}[i_1, j_1]) = (\mathbf{pw}[i_2], \mathbf{sa}[i_2, j_2])$. The construction C^{H} is applied to each password–salt pair with the results given to the adversary as a challenge vector.

Hash-dependent unguessability. Intuitively, guessability measures the adversary's ability to recover a full bank of passwords, sampled from a general password distribution, using some leakage z on the passwords. The most basic game to address this notion is the GUESS game from [10]. An adversary $\mathcal{A}$ in the GUESS game is given leakage z, with access to test and corruption oracles TEST and COR respectively. Queries to TEST allow $\mathcal{A}$ to check password guesses, returning a win_i flag when the ith password is guessed correctly. We consider adversaries that are able to make up to c corruption queries to COR, i.e., they automatically "win" on c of the passwords.

We extend this game to a hash-dependent setting, shown in Fig. 2.2. In this setting password samplers $\mathcal{P}^{\mathsf{H}}_m$ have oracle access to H allowing them to sample hash-dependent passwords. We say password samplers with at

most $q_{\mathcal{P}}$ queries to H are $q_{\mathcal{P}}$-query samplers. Along with the TEST and COR oracles, we give $\mathcal{A}$ oracle access to H and consider adversaries with q queries to H, T queries to TEST and c queries to COR. Formally, we refer to such an adversary as a (q, T, c)-query adversary $\mathcal{A}$ and for a $q_{\mathcal{P}}$-query password sampler $\mathcal{P}_m^{\mathsf{H}}$ in $\text{GUESS}_{\mathcal{P}_m^{\mathsf{H}}}^{\mathcal{A}}$, we define the advantage of adversary $\mathcal{A}$ (in the GUESS game with respect to sampler $\mathcal{P}^{\mathsf{H}_m}$ as

$$\mathbf{Adv}_{\mathcal{P}_m^{\mathsf{H}}}^{\text{GUESS}}(\mathcal{A}) := \Pr\left[\text{GUESS}_{\mathcal{P}_m^{\mathsf{H}}}^{\mathcal{A}}\right].$$

And so the advantage of the adversary $\mathcal{A}$ in the GUESS game, with respect to the password sampler $\mathcal{P}_m^{\mathsf{H}}$, is the probability that the adversary $\mathcal{A}$ wins the game. If the password sampler makes no queries to the oracle H, then any of $\mathcal{A}$'s queries to H have no effect on the advantage, and we recover the standard GUESS game from FT21. To analyze the security of salted password banks we extend the SA-GUESS game from FT21 to a hash-dependent setting, given in Fig. 2.2.

Salted unguessability. 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 which has a matching salt. This models the fact that the adversary can verify the guess without repeating it for each salt. For a (q, T, c)-query adversary $\mathcal{A}$, a $q_{\mathcal{P}}$-query password sampler $\mathcal{P}_m^{\mathsf{H}}$, and a salt generator Gen we define

$$\mathbf{Adv}_{\mathcal{P}_m^{\mathsf{H}},\ell,\mathsf{Gen}}^{\text{SA-GUESS}}(\mathcal{A}) := \Pr\left[\text{SA-GUESS}_{\mathcal{P}_m^{\mathsf{H}},\ell,\mathsf{Gen}}^{\mathcal{A}}\right].$$

Similar to GUESS, if $\mathcal{P}_m^{\mathsf{H}}$ makes no queries to H, we recover the hash-independent SA-GUESS game.

Hash-dependent unrecoverability and one-wayness. We also extend the unrecoverability and one-wayness security games studied in FT21 to a hash-dependent setting. The unrecoverability of passwords models the hardness of recovering passwords given a password bank protected by a hash-based construction. We consider the unrecoverability game for some 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 at input to C^{H}, generating the challenge values $\mathbf{y}$. The adversary

is given the challenge vector $\mathbf{y}$ along with the salts $\mathbf{sa}$ and the leakage z. Note that for this game, the collision pattern is public, and we do not need to give it to $\mathcal{A}$. The adversary $\mathcal{A}$ wins if it recovers all m of the original passwords. To extend this setting to a hash-dependent setting, we grant $\mathcal{P}_m$ oracle access to H, which we denote $\mathcal{P}_m^{\mathsf{H}}$.

Formally, for some hash-based construction C^{H}, password sampler $\mathcal{P}_m^{\mathsf{H}}$ and salt generator Gen, we define the advantage of adversary $\mathcal{A}$ in $\mathrm{UR}^{\mathcal{A}}_{\mathsf{C}^{\mathsf{H}},\mathcal{P}_m^{\mathsf{H}},\ell,\mathsf{Gen}}$ as

$$\mathbf{Adv}^{\mathrm{UR}}_{\mathsf{C}^{\mathsf{H}},\mathcal{P}_m^{\mathsf{H}},\ell,\mathsf{Gen}}(\mathcal{A}) := \Pr\left[\mathrm{UR}^{\mathcal{A}}_{\mathsf{C}^{\mathsf{H}},\mathcal{P}_m^{\mathsf{H}},\ell,\mathsf{Gen}}\right].$$

When we write the advantage in the UR game where Gen outputs the empty string ϵ, and $\ell = 1$, we leave these parameters blank, i.e. we write $\mathbf{Adv}^{\mathrm{UR}}_{\mathsf{C}^{\mathsf{H}},\mathcal{P}_m^{\mathsf{H}}}(\mathcal{A})$. The hash-dependent unrecoverability game is presented in Fig. 2.3.

We also present an extension of the one-wayness security game to a hash-dependent setting. The one-wayness security game is similar to the unrecoverability game, with a different winning condition. The adversary is not required to recover the password vector $\mathbf{pw}$, but instead needs to provide a preimage for each challenge point, with respect to the construction C^{H} and one of the challenge point's salts. The extended OW game gives the password sampler oracle access to H. This game is presented in Fig. 2.3 with the box to indicate the change from the UR game.

Formally, for some hash-based construction C^{H}, password sampler $\mathcal{P}_m^{\mathsf{H}}$ and salt generator Gen, we define the advantage of adversary $\mathcal{A}$ in $\mathrm{OW}^{\mathcal{A}}_{\mathsf{C}^{\mathsf{H}},\mathcal{P}_m^{\mathsf{H}},\ell,\mathsf{Gen}}$ as

$$\mathbf{Adv}^{\mathrm{OW}}_{\mathsf{C}^{\mathsf{H}},\mathcal{P}_m^{\mathsf{H}},\ell,\mathsf{Gen}}(\mathcal{A}) := \Pr\left[\mathrm{OW}^{\mathcal{A}}_{\mathsf{C}^{\mathsf{H}},\mathcal{P}_m^{\mathsf{H}},\ell,\mathsf{Gen}}\right].$$

As in the Guess and sa-Guess games, if $\mathcal{P}_m^{\mathsf{H}}$ makes no oracle queries, the above definitions reduce to those of FT21.

2.2.1 Graph-based MHFs

Let $\mathcal{G} = (V, E)$ be a directed acyclic graph (DAG), where the node set V is the set of integers 1 to n. 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 jth parent (resp. child) ordered by node index. Let δ_i be in-degree of the ith node, and let $\delta := \max_i\{\delta_i\}$. In this thesis, we consider iMHFs which use the labeling scheme from AS15 [5]. We reproduce the definition of that labeling here, and extend it to allow for salting. To allow salting, we

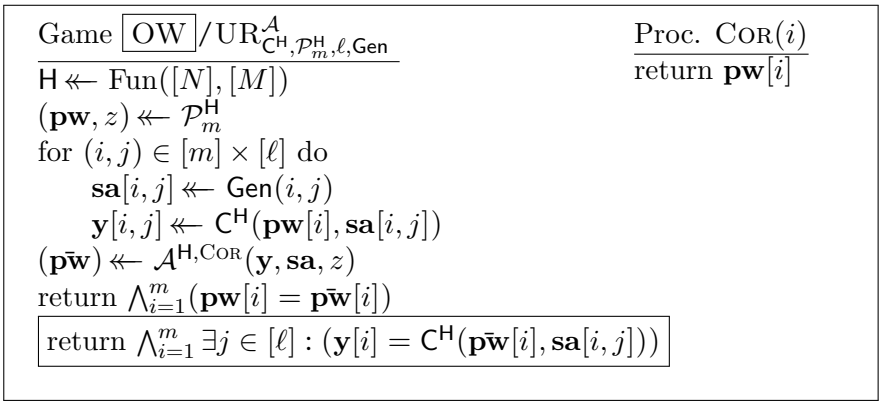

Figure 2.3: The UR and OW games with respect to a hash-dependent m-sampler $\mathcal{P}^{\mathsf{H}}_m$ and salt generator Gen.

extend the initial label l to be drawn from $L \times K$ rather than L (i.e., l would take the form (pw, sa)).

Definition 1 ((H, l)-labeling of graphs [5])**.** *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:*

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

Where there are fewer than $\delta + 2$ inputs to H, we pad the inputs with ε, assuming $\varepsilon \in L$ and $\varepsilon \in [K]$. For brevity, we use $\mathbf{pre\text{-}lab}(v)$ to refer to the H input which defines the label of node v, 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 a salt is only included when $\mathrm{indeg}(v) = 0$.

Using the labeling of a graph, we can describe graph-based iMHFs in the random oracle model. A construction C is formed from a graph $\mathcal{G}$ using the labeling from Definition 5, and when instantiated with the random oracle H, it forms a construction we denote as C^{H}. Computing C^{H} on $l = (x, sa)$, denoted as $\mathsf{C}^{\mathsf{H}}(l)$, will give an (H, l)-labeling of $\mathcal{G}$. We recall the formal definition below.

Definition 2 (Graph-based iMHF [5])**.** *Let $\mathcal{G} = (V, E)$ be a DAG with maximum in-degree δ and sink nodes $\{v_1, \ldots, v_z\}$ for some positive integer*

z. Let L be an arbitrary label set, $[K]$ be a salt set of size K, and $\mathbb{H} = \mathrm{Fun}(V \times L^{\delta} \times [K], L)$. The graph functions *(of $\mathcal{G}$ and $\mathbb{H}$) are the members of the family of oracle functions $\mathsf{C} = \mathsf{C}_{\mathcal{G}}^{\mathbb{H}}$, indexed by functions in $\mathbb{H}$, mapping $L \times [K]$ to L^z. For input $l \in L \times [K]$, the value of $\mathsf{C}^{\mathsf{H}} \in \mathsf{C}$ is defined as $\mathsf{C}^{\mathsf{H}}(l) := (\mathbf{lab}(v_1), \ldots, \mathbf{lab}(v_z))$, where v_i are arranged in lexicographic order and* **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 where $|L| = N$. We follow the notation used in [5] for denoting multiple instances of a graph. That is, for a DAG $\mathcal{G} = (V, E)$ and some positive integer m, we use $\mathcal{G}^{\times m}$ (which we call the tensor graph) to denote the disjoint union of m copies of $\mathcal{G}$. Note that each node index in $\mathcal{G}^{\times m}$ is unique. We denote constructions based on $\mathcal{G}^{\times m}$ and random oracle H by $\mathsf{C}_{\times m}^{\mathsf{H}}$. Note that $\mathsf{C}_{\times m}^{\mathsf{H}}$ is defined by a labeling on a graph with distinct node indices.

2.2.2 Cumulative pebbling and memory complexities

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 black pebbling game an adversary is challenged with placing pebbles on the sink nodes of some DAG, while following some simple rules as follows. The pebbling occurs in rounds, in which an adversary can add a single pebble and remove any number of pebbles from the graph per round. 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. The adversary can remove pebbles from the graph at any point. We consider the *parallel* pebbling game where batches of pebbles can be added (or removed) in rounds.

Pebbling. Formally, given a DAG $\mathcal{G} = (V, E)$, the parallel (black) pebbling game is defined as follows: A parallel pebbling of a DAG $\mathcal{G}$ with source nodes S and target nodes T is a sequence of configurations $P = (P_0, \ldots, P_t)$, where $P_0 = \emptyset$ and $P_i \subseteq V$. A pebbling is *legal* if the following two conditions hold: (1) a node can only be added when its predecessors have a pebble on, or it is a source node, i.e. $\forall i \in [t], \forall x \in P_i \setminus P_{i-1}, \forall y \in \mathbf{Pa}(x) : y \in P_{i-1}$; and (2) for every target node, there is a round (not necessarily the last) where it is pebbled, i.e. $\forall x \in T,\ \exists z \leq t : x \in P_z$. A pebbling is *complete* if T is the set of sink nodes.

Cumulative pebbling complexity. Given a parallel pebbling adversary $\mathcal{A}$ that places at most q pebbles in at most t rounds, and produces a complete pebbling, 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 a graph as the minimum CPC of any q-pebble adversary's legal pebbling of the graph, i.e. $\mathrm{CPC}_q(\mathcal{G}) := \min_{\mathcal{A}}(\mathrm{CPC}(\mathcal{A}))$. Above a sufficiently large q (e.g., if $\mathcal{G}$ has n nodes, for $q > n^2$), $\mathrm{CPC}(\mathcal{A})$ will not decrease, as an adversary with more available pebbles can always use fewer. Hence, for sufficiently large q, CPC is independent of q. We define this minimum as the *cumulative complexity* of the graph $\mathrm{CC}(\mathcal{G})$. We recall the following [5, Lemma 3].

Lemma 1 (Additive property of graph cumulative complexity [5])**.** *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be a pair of node-disjoint DAGs and let $\mathcal{G}$ be their disjoint union then,* $\mathrm{CC}(\mathcal{G}) = \mathrm{CC}(\mathcal{G}_1) + \mathrm{CC}(\mathcal{G}_2)$.

Consequently, for any positive integer m and any DAG $\mathcal{G}$, the minimum cost to pebble m instances of $\mathcal{G}$ is given by $\mathrm{CC}(\mathcal{G}^{\times m}) = m \cdot \mathrm{CC}(\mathcal{G})$.

Cumulative memory complexity. In the first chapters of this thesis, we will directly consider memory-constrained adversaries in pROM, which make batches of queries to the random oracle in rounds. Keeping track of the size of the state (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 2.1.1. When the input x is the empty string, as in the case for an adversary with no input that we consider later, we use $\mathrm{CMC}(\mathcal{A})$. 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.

Complexity lower-bounds for graph-based iMHFs. In [6], the authors used this machinery to prove lower bounds on the CMC of a parallel algorithm computing m instances of a graph-based iMHF from Definition 5. They do not formally define a security game, but rather use a metric more appropriate to cryptanalysis.

For any construction C^{H}, a natural number q and a real number $\epsilon \in [0, 1]$ they define $\mathsf{comp}_{q,\epsilon}(\mathsf{C}^{\mathsf{H}})$ to be the minimum CMC for a q-query parallel algorithm to correctly evaluate C^{H} on some input, with probability greater than ϵ. Here we recall the lower-bound they prove the following lower bound for graph-based iMHFs C^{H} based on the graph $\mathcal{G}$:

$$\mathsf{comp}_{q,\epsilon}(\mathsf{C}^{\mathsf{H}}) \geq \frac{\epsilon \mathrm{CC}(\mathcal{G})(\log M - \log q)}{\epsilon\bar{\epsilon} + 1},$$

where $\bar{\epsilon} := -\log\left(\epsilon - q2^{-\log M}\right)$.

Importantly, they limit their analysis to a setting where all inputs to the iMHF are chosen in advance of any queries to the hash.

3 Hash-dependent Unguessability

As mentioned in the introduction, to facilitate the first main contributions of this thesis one needs to integrate the frameworks of Farshim and Tessaro (FT21) [31] (itself an extension of BRT12 [10]) and of Alwen and Serbinenko (AS15) [5]. This chapter extends the former framework, laying the groundwork for Theorem 4. First, we give the intuition behind the extension of the framework.

3.1 Extending the Unguessability framework

Hash-dependent unguessability and unrecoverability. In the preliminaries, we extended the basic security notions from FT21 (and BRT12) to a hash-dependent setting. In this chapter, we will commence with an analysis of them. Whilst hash-dependent passwords have been considered before [9, 10], there has been little direct analysis of this setting. However, there is good reason to consider such samplers, and the reader can refer to the introduction for real-world examples. The first notion we extend is unguessability, defined as the advantage of an adversary in an interactive password guessing game. This notion captures how the inherent difficulty of guessing multiple passwords sampled from a password distribution grows with the number of passwords guessed. This provides a baseline for the multi-instance security of the constructions we consider. The guessing game comes in two flavors: Guess and sa-Guess, corresponding to unsalted and salted usage of passwords. In both, selection of passwords from a likely non-uniform password distribution is modeled through sampling by a *password sampler* $\mathcal{P}$.

Essentially in both Guess and sa-Guess, the password sampler $\mathcal{P}$ samples m passwords (for m users) and the adversary tries to guess the passwords without getting any information about them, other than some leakage on the underlying distribution. In sa-Guess, after passwords are sampled, a *salt generator* is used to produce salt values that are given to the adversary, modelling the use of salts, possibly known by the adversary, alongside passwords. The adversary is given access to a Test oracle and can check at each invocation of the oracle in Guess (respectively

SA-GUESS) whether a guess for the password (respectively password–salt pair) of a specific user is correct or not. The adversary wins if it correctly guesses all the passwords.[1] The adversary's resource usage is measured by the number of guesses it makes, i.e., the number of TEST queries, denoted by T, and its success probability is denoted by $\mathbf{Adv}^{\text{GUESS}}_{\mathcal{P}}(\mathcal{A})$ and $\mathbf{Adv}^{\text{SA-GUESS}}_{\mathcal{P}}(\mathcal{A})$, accordingly. Both GUESS and SA-GUESS measure the inherent (un)guessability of the passwords independently of the strength of any hash-based construction applied to the passwords, and depend only on the password sampler (and potentially the salt generator).

We extend the above definitions by introducing *hash-dependent password samplers* $\mathcal{P}^{\mathsf{H}}$, which may query the random oracle H before outputting passwords. Assuming passwords are chosen hash-independently is a strong assumption, as the hash function is public and there are many password assistance technologies which help users choose better passwords. For example, passwords may be generated by hashing some master secret, or samplers may reject a generated password if its hash is stored in some published database (e.g., à la haveibeenpwned.com). Additionally, and in line with the random oracle model, we grant the adversary the ability to query H as well as TEST, and count the number of H queries q as additional resources the adversary uses. $\mathbf{Adv}^{\text{GUESS}}_{\mathcal{P}^{\mathsf{H}}}(\mathcal{A})$ and $\mathbf{Adv}^{\text{SA-GUESS}}_{\mathcal{P}^{\mathsf{H}}}(\mathcal{A})$ denote advantages in these generalized games. The pseudocode, along with a detailed description of each of the games is presented in Fig. 2.2. The use of hash-dependent password samplers naturally gives rise to more generalized definitions of unrecoverability and one-wayness for hash-based constructions. We use $\mathbf{Adv}^{\text{UR}}_{\mathcal{P}^{\mathsf{H}},\mathsf{C}^{\mathsf{H}}}(\mathcal{A})$ and $\mathbf{Adv}^{\text{OW}}_{\mathcal{P}^{\mathsf{H}},\mathsf{C}^{\mathsf{H}}}(\mathcal{A})$ to denote the corresponding hash-dependent advantages. The pseudocode for these is presented in Fig. 2.3.

Admissible samplers. In this thesis, we limit the consideration to computationally bounded password samplers by insisting on an upper-bound for their query complexity. That is, we consider $q_{\mathcal{P}}$ samplers, making at most $q_{\mathcal{P}}$ queries to the random oracle H before outputting the password vector. We also restrict our consideration to samplers that are limited to querying H, e.g., they cannot "fix points" in the table of H. We argue that both of these are reasonable restrictions as, in addition to being in line with the philosophy of the ROM (which is core to the security model we are considering), it more closely reflects practice. This is because the

[1] The games allow corruptions as well, but for a readable presentation of the intuition, we limit the consideration to the no-corruption case.

hash dependent algorithms which sample passwords, in practice, are not adversarial in the traditional sense. Indeed, for most user passwords the hash-dependent part of the sampling is likely to be engineered for efficiency, as ultimately it will be run on company servers. While here we model samplers as adversarial, this is to upper bound the effect of any weakness they introduce to the security of password-based security games. Thus it makes sense to consider an upper limit on the number of hash evaluations these algorithms make.

Known relations. In the hash-independent setting there are already the following known relations between the unguessability and the unrecoverability of passwords (proven in [30]). We state them informally below one by one. Firstly, a bound relating a T query adversary to a $m - c$ query adversary. Intuitively, this is an adversary who gets only one guess per password.

$$\mathbf{Adv}^{\text{GUESS}}_{\mathcal{P}_m}(\mathcal{A}) \leq \binom{T}{m-c} \mathbf{Adv}^{\text{GUESS}}_{\mathcal{P}_m}(\mathcal{B}) \ .$$

Next, a bound relating the salted guess game to the guess game, where the term collected under negl is from collisions in the salts:

$$\mathbf{Adv}^{\text{SA-GUESS}}_{\mathcal{P}_m,\ell,\mathsf{Gen}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{GUESS}}_{\mathcal{P}_m}(\mathcal{B}) + \text{negl} \ .$$

For predictable, low-entropy, or small salts, this negl term may be considerable. In practice however, high entropy (and so likely distinct) salts are of primary interest. And finally, they relate unrecoverability and salted guess

$$\mathbf{Adv}^{\text{UR}}_{\mathsf{H},\mathcal{P}_m,\ell,\mathsf{Gen}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{SA-GUESS}}_{\mathcal{P}_m,\ell,\mathsf{Gen}}(\mathcal{B}).$$

Chapter Overview. Having elevated the definitions of (plain and salted) unguessability, unrecoverability, and one-wayness to the hash-dependent setting, a natural question arises as to whether the known relations between these notions in the hash-independent framework of FT21 would still hold in the more general hash-dependent setting. The answer, we will show, is nuanced. We prove that the relations do hold in the hash-dependent setting in the presence of uniform salting and do not hold if no salts are used.

To demonstrate the relationships that hold, we prove two main lemmas. First, a reduction from salted unguessability to unguessability, specifically that when uniform salt generation is used, the advantage of a

SA-GUESS adversary $\mathcal{A}$ can be bounded by the advantage of a GUESS adversary $\mathcal{B}$, with the same query and time complexity, and up to an additive factor coll representing the probability of collision between salts, i.e. $\mathbf{Adv}^{\text{SA-GUESS}}_{\mathcal{P}^{\mathsf{H}}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{GUESS}}_{\mathcal{P}^{\mathsf{H}}}(\mathcal{B}) + \mathsf{coll}$. Second, a reduction from unrecoverability to salted unguessability, specifically that when uniform salt generation is used, the advantage of a UR adversary $\mathcal{A}$ can be bounded by the advantage of a SA-GUESS adversary $\mathcal{B}$, with the same query and time complexity, and up to an additive factor in the order of the proportion of the H domain values $\mathcal{P}^{\mathsf{H}}$ queries cover which will be negligible in practice, i.e. $\mathbf{Adv}^{\text{UR}}_{\mathcal{P}^{\mathsf{H}}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{SA-GUESS}}_{\mathcal{P}^{\mathsf{H}}}(\mathcal{B}) + \text{negl}$. Both results are similar to their counterpart in FT21 with the exception of the negligible term in the second lemma. The proofs follow similar strategies to those of their hash-independent counterparts in FT21, however our proofs need to deal with H queries by the adversary and crucially the password sampler.

To show the separation in the unsalted case, we construct a 'bad' hash-dependent password sampler that generates passwords that are hard to guess but easy to recover from their hashes. In particular, the bad sampler samples uniform passwords, queries H on each of them, and only outputs those for which the hash 'leaks' the password, e.g., when the password is a prefix of its hash. For such a bad sampler, we prove that while any adversary $\mathcal{A}$ would have negligible GUESS advantage $\mathbf{Adv}^{\text{GUESS}}_{\mathcal{P}^{\mathsf{H}}}(\mathcal{A})$, there exists an adversary $\mathcal{B}$ with respect to the same H and $\mathcal{P}^{\mathsf{H}}$ that has a UR advantage $\mathbf{Adv}^{\text{UR}}_{\mathsf{H},\mathcal{P}^{\mathsf{H}}}(\mathcal{B}) \geq 1/2$.

3.2 Reductions

Here we present three reductions, generalizing analogous theorems in the FT21 framework to the hash-dependent setting. Essentially, these results extend the bounds in the more restricted framework, with the exception that due to an attack presented in Theorem 1, the passwords in the UR and OW games need to be salted. The proofs use similar arguments to those of FT21, but keep track of hash queries made by the password sampler and adversary.

Lemma 2 (Hash-dependent unguessability)**.** *Let $m, q, q_{\mathcal{P}}, T, c \in \mathbb{N}$, and $\mathcal{P}^{\mathsf{H}}_m$ be a hash-dependent m sampler. Then, for any (q, T, c)-adversary $\mathcal{A}$ against* GUESS*, there exists a $(q, m-c, c)$-adversary $\mathcal{B}$ against* GUESS *s.t.*

$$\mathbf{Adv}^{\text{GUESS}}_{\mathcal{P}^{\mathsf{H}}_m}(\mathcal{A}) \leq \binom{T}{m-c} \mathbf{Adv}^{\text{GUESS}}_{\mathcal{P}^{\mathsf{H}}_m}(\mathcal{B}) \ .$$

To prove this, we extend the argument of [31] by accounting for the q queries to H which are relayed in the reduction.

Proof. Given some (q, T, c)-adversary $\mathcal{A}$ in the dGuess game with respect to a password sampler $\mathcal{P}_m^{\mathsf{H}}$ that makes at most $q_{\mathcal{P}}$ queries, we build a $(q, m-c, c)$-adversary $\mathcal{B}$. Assume that $\mathcal{A}$ makes exactly c corruption queries and exactly $m-c$ of the test queries result in a win flag being set to true. The adversary $\mathcal{B}$ runs $\mathcal{A}$ as follows. First, $\mathcal{B}$ predicts which of the $m-c$ queries to Test are correct, and decides at the outset that these are the Test queries it will forward to its own Test oracle. For each of these queries $\mathcal{B}$ will respond to $\mathcal{A}$ with true. There are $\binom{T}{m-c}$ choices for these queries. The adversary $\mathcal{B}$ relays all of $\mathcal{A}$'s random oracle and corruption queries to its own random oracle H and corruption oracle respectively. Whenever $\mathcal{B}$ correctly guesses the $m-c$ queries, $\mathcal{A}$ is run in exactly the environment it expects. The adversary $\mathcal{B}$ then wins whenever $\mathcal{A}$ wins. □

Next, we lift [31, Theorem 3] to include hash-dependent password samplers. To do this, we import the following definition from FT21, which captures the probability of a collision in the $m\ell$ salts. For a salt sampler Gen

$$\mathsf{coll}_{\mathsf{Gen}}(m, \ell) := \Pr[\exists (i,j) \neq (i',j') \in [m] \times [\ell] : \mathsf{Gen}(i,j) = \mathsf{Gen}(i',j')].$$

With this, we can prove the following lemma.

Lemma 3 (sa-Guess to Guess)**.** *Let $\ell, T, c \in \mathbb{N}$. Then, for any $q_{\mathcal{P}}$-query m-sampler $\mathcal{P}_m^{\mathsf{H}}$, any salt generator Gen, and any (q, T, c)-adversary $\mathcal{A}$ against* sa-Guess*, there exists a (q, T, c)-adversary $\mathcal{B}$ against* Guess *s.t.*

$$\mathbf{Adv}^{\text{sa-Guess}}_{\mathcal{P}_m^{\mathsf{H}},\ell,\mathsf{Gen}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{Guess}}_{\mathcal{P}_m^{\mathsf{H}}}(\mathcal{B}) + \mathsf{coll}_{\mathsf{Gen}}(m, \ell) \ .$$

To prove this, we show that as long as the salts do not collide, the reduction follows easily and our extension to the hash-dependent setting relays all queries to H.

Proof. Given a (q, T, c)-adversary $\mathcal{A}$ against sa-Guess we build a (q, T, c)-adversary $\mathcal{B}$ against Guess as follows. $\mathcal{B}$ receives the leakage z from $\mathcal{P}_m^{\mathsf{H}}$, and randomly samples $m\ell$ salts, and aborts if they are not all distinct. By definition, this happens with probability at most $\mathsf{coll}_{\mathsf{Gen}}(m, \ell)$. As all the salts are unique, $\mathcal{B}$ sets z_{coll} to be the identity matrix. Algorithm $\mathcal{B}$ then runs $\mathcal{A}$ on the input $(\mathbf{sa}, z, z_{\mathsf{coll}})$, and responds to any queries to Cor or H from $\mathcal{A}$ using its own respective oracles. When $\mathcal{A}$ queries (pw, sa) to

TEST, $\mathcal{B}$ checks if sa is in $\mathbf{sa}$, responding to the query with $\perp$ if it isn't, and calculating i, j such that $sa = \mathbf{sa}[i, j]$ if it is. $\mathcal{B}$ then queries (pw, i) to its TEST oracle and returns $\{i\}$ if TEST returns true, otherwise it returns the empty set. □

We now present a reduction from UR to SA-GUESS, which is the analogue of [31, Theorem 6].

Lemma 4 (UR to SA-GUESS)**.** *Let $q, q_{\mathcal{P}}, c, m, \ell, K \in \mathbb{N}$, and $\mathsf{Gen} = [K]$. Then, for any $q_{\mathcal{P}}$-query m-sampler $\mathcal{P}_m^{\mathsf{H}}$ and any (q, c)-adversary $\mathcal{A}$ against* UR*, there exists a (q, q, c)-adversary $\mathcal{B}$ against* SA-GUESS *s.t.*

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

Intuitively, if the password sampler queries are different to the queries made in the challenge generation phase, each response from the random oracle will be independent and the challenge values will leak no information to the adversary. With unpredictable salts, this will occur with high probability.

Proof. We prove this lemma by starting with the UR game, making a single game hop to some game G_1, and then finally do a direct reduction from G_1 to SA-GUESS. Let G_0 be the UR game with respect to parameters from the theorem statement and $\mathcal{A}$ be an adversary in this game.

Let G_1 be a modified game where after $\mathcal{P}_m^{\mathsf{H}}$ has sampled the passwords, the game switches to using a second random oracle H' that is defined as follows: for any query (in the challenge generation or online phase) of the form (pw, sa) where $pw \in \mathbf{pw}$ and $sa \in \mathbf{sa}$, H' samples a value randomly, for any other query H' is consistent with H.

Let bad be the event that $\mathcal{P}_m^{\mathsf{H}}$ queries a password-salt pair (pw, sa), which is then queried again after the passwords have been sampled. If $\mathcal{P}_m^{\mathsf{H}}$ doesn't query any such password-salt pair, then the value of $\mathsf{H}(pw, sa)$ remains free and the responses from H' are identically distributed to those from H. All other queries that are made to H and H' will be responded to identically.

Then G_0 and G_1 are identical up until bad, and $\Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1] \leq \Pr[\mathsf{bad}]$. We now bound the probability of bad. There are at most $m\ell$ password-salt pairs, and each salt is uniformly sampled from a space of size K. Each query made by $\mathcal{P}_m^{\mathsf{H}}$ will match a password-salt pair with probability at most $m\ell/K$. As the sampler $\mathcal{P}_m^{\mathsf{H}}$ makes at most $q_{\mathcal{P}}$ queries to H, by the union bound, the probability of bad is at most $q_{\mathcal{P}} m\ell/K$.

We now reduce from G_1 to SA-GUESS. Given a (q, c)-adversary $\mathcal{A}$ against G_1, we build a (q, q, c)-adversary $\mathcal{B}$ against SA-GUESS. Algorithm $\mathcal{B}$ receives $\mathbf{sa}$, the leakage z, and collision pattern z_{coll}. Algorithm $\mathcal{B}$ then simulates H' by generating a random challenge vector $\mathbf{y}$ of length $m\ell$, consistent with z_{coll}. Then $\mathcal{B}$ runs $\mathcal{A}$ on the input $(\mathbf{y}, \mathbf{sa}, z)$, handling its hash queries as follows. It forwards all of $\mathcal{A}$'s hash queries to its TEST oracle. If TEST returns a non-empty set of indices, algorithm $\mathcal{B}$ returns a challenge hash value corresponding to one of the password-salt pairs for the i-th password for any i in the set of indices. (Recall challenge hash values are chosen to be compatible with the collision pattern.) If TEST returns the empty set, then $\mathcal{B}$ relays the query to its hash oracle H, and returns the response. Algorithm $\mathcal{B}$ relays all corruption queries to its own COR oracle.

Therefore $\mathcal{B}$ recognizes all queries of the form (pw, sa) for $pw \in \mathbf{pw}$ and $sa \in \mathbf{sa}$, by the use of TEST and responds with the challenge values. All other queries are relayed to H, and so the simulation of the random oracle is consistent with the rules of G_1.

Assuming that $\mathcal{A}$ makes at most q queries to its hash oracle and at most c queries to its COR oracle, algorithm $\mathcal{B}$ makes at most q queries to its TEST oracle, at most q queries to H, and c queries to COR. □

Dependence on parameters. Here we make the loss in advantage or query count explicit for each lemma. In Lemma 2 the bound depends on the fact that the adversary we reduce to has fewer TEST queries than the adversary in the guess game. In particular we reduce the T query adversary to a $m - c$ query adversary. This is a natural metric, because it is the same number of "guesses" are there are remaining passwords. This comes at the cost of the factor of $\binom{T}{m-c}$ in the security.

In Lemma 3 the adversaries, while playing different games, have the same parameters. The security loss comes from the number of collisions in the salts. As the probability of collision increases, the security guarantee weakens. In practice salts are sampled uniformly, and this term will be negligible.

In Lemma 4 the adversaries once again have the same number of queries to the random oracle H, though the SA-GUESS adversary is given access to the TEST oracle too. Here there is a security loss from the $q_{\mathcal{P}} m\ell/K$ term. We can summarise the parameter changes with the following statement:

Corollary 1. *Let $q, q_{\mathcal{P}}, c, m, \ell, K \in \mathbb{N}$, and $\mathsf{Gen} = [K]$. Then, for any $q_{\mathcal{P}}$-query m-sampler $\mathcal{P}_m^{\mathsf{H}}$ and any (q, c)-adversary $\mathcal{A}$ against* UR*, there exists a (q, q, c)-adversary $\mathcal{B}$ against* Guess *s.t.*

$$\mathbf{Adv}^{\mathrm{UR}}_{\mathsf{H},\mathcal{P}_m^{\mathsf{H}},\ell,\mathsf{Gen}}(\mathcal{A}) \leq \binom{q}{m-c}\mathbf{Adv}^{\mathrm{Guess}}_{\mathcal{P}_m^{\mathsf{H}}}(\mathcal{B}) + \mathsf{coll}_{\mathsf{Gen}}(m,\ell) + \frac{q_{\mathcal{P}} m \ell}{K} \ .$$

A few remarks. Note that this result relies on the unpredictability of the salting algorithm, the analogous results in FT21 for known distinct salts or for no salts *do not* extend to the hash-dependent password setting, as there is no way of ensuring that the password sampler has not already queried the random oracle with a value that the adversary may also query. Indeed, in the section below, we outline an attack in Theorem 1 ruling out reductions in the unsalted case.

Note also that the bound in Lemma 4 doesn't depend on the number of queries made by the adversary. So when $q_{\mathcal{P}} = 0$, i.e., when the password sampler makes no queries to its random oracle, we recover the same bounds as that given in FT21 for hash-independent password samplers. The resulting reduction is the same as that in FT21 as the bad event is never triggered and the hash function H' is never involved; furthermore there are no hash queries.

3.3 A "Bad" Password Sampler

Here we define the bad sampler before presenting an attack. This will confirm two things: firstly, that hash-dependent samplers really do affect the security; and secondly, that salting must be unpredictable to achieve meaningful security.

Bad sampler. Let $\mathcal{P}_1^{\mathsf{H}}$ be a hash-dependent password sampler that samples a password as follows. The sampler $\mathcal{P}_1^{\mathsf{H}}$ picks uniformly random passwords from the set of integers up to 2^l, each with a length of l bits. For each candidate password pw, the sampler uses a query to H to check whether the first l bits of $\mathsf{H}(pw)$ match the l bits of pw. If the bits match, then $\mathcal{P}_1^{\mathsf{H}}$ returns the password pw with an empty leakage z, otherwise it continues to check candidate passwords. If $\mathcal{P}_1^{\mathsf{H}}$ uses all its queries and does not find such a password, it returns a uniformly random password along with empty leakage z.

Intuition and motivation. Intuitively, such a sampler makes the UR game trivial. The password that the UR adversary has to guess is leaked in the hash $\mathsf{H}(pw)$. Thus, as we show in Theorem 1, the adversary can win the game without any queries by simply submitting the challenge. Importantly, even though the GUESS game is also hash-dependent, the adversary's advantage actually remains low. This is because the adversary in the GUESS game does *not* see the hashed passwords; it is simply guessing blindly. While the GUESS game is supposed to be an abstraction of the UR game, the bad sampler shows that hash-dependence breaks the relation. By modelling the password distribution via a sampler we are able to give it hash access, treat it adversarially, and reason about the effect on password-based security.

Note that assuming the adversary has enough queries to succeed, the expected runtime is 2^ℓ queries. Thus the feasibility of this particular counter example depends heavily on a short password length.

Theorem 1. *Let $T, q, N, M, l \in \mathbb{N}$ and $l < \log M$. There exists a 2^l-query sampler $\mathcal{P}_1^{\mathsf{H}}$ in the (N, M)-RO model that outputs l-bit passwords s.t. for any $(q, T, 0)$-adversary $\mathcal{A}$ against* GUESS*:*

$$\mathbf{Adv}^{\text{GUESS}}_{\mathcal{P}_1^{\mathsf{H}}}(\mathcal{A}) \leq \frac{q+T}{2^l} \ ,$$

and there exists a $(0, 0)$-adversary $\mathcal{B}$ against UR *w.r.t. the same H and $\mathcal{P}_1^{\mathsf{H}}$ s.t.*

$$\mathbf{Adv}^{\text{UR}}_{\mathsf{H},\mathcal{P}_1^{\mathsf{H}}}(\mathcal{B}) \geq 1/2 \ .$$

Proof. Consider the password sampler described above. To prove the first claim, we upper bound the probability of an adversary $\mathcal{A}$ finding even one password pw such that the first l bits of $\mathsf{H}(pw)$ match pw. With each new query pw^* to H or TEST, the probability that the first l bits of $\mathsf{H}(pw^*)$ match pw^* is $1/2^l$. With q queries to H and T queries to TEST, the probability at least one guess satisfies this requirement is at most $(q+T)/2^l$.

To prove the second claim, we show that an adversary in the UR game can recover the password whenever $\mathcal{P}_1^{\mathsf{H}}$ succeeds. Let $\mathcal{B}$ be an adversary in the UR game with respect to the password sampler $\mathcal{P}_1^{\mathsf{H}}$ described above. Then $\mathcal{B}$ receives a challenge y, and submits the first l bits of y as its guess for the password pw sampled by $\mathcal{P}_1^{\mathsf{H}}$. Whenever $\mathcal{P}_1^{\mathsf{H}}$ succeeds, $\mathcal{B}$ wins. We lower bound the probability that $\mathcal{P}_1^{\mathsf{H}}$ succeeds as follows.

$\mathcal{P}_1^{\mathsf{H}}$ makes at most 2^l queries. We say a query q succeeds if the first l bits of $\mathsf{H}(q)$ match the l bits of q. The probability that each query succeeds is

2^{-l}. The probability that at least one query succeeds is $1-(1-2^{-l})^{2^l} \geq 1-1/e \geq 1/2$. □

Extension to a trade-off. While this example suffices to show a separation between the notions, we can extend the sampler to use fewer queries if we allow it to return more passwords. That is, let $\mathcal{P}_m^{\mathsf{H}}$ sample m passwords similarly, but leak $l' = l/m$ bits of the first password in the starting l' bits of each of the m challenges. Then the entirety of the first password is leaked across the starting bits of the m challenges.

Suppose that the sampler spends $2^{l/m}$ queries on each of the m passwords, for a total of $m2^{l/m} < 2^l$ queries. For each query the sampler makes, the probability it succeeds in leaking the l/m bits is now $2^{-l/m}$. As the sampler performs $2^{l/m}$ queries per l/m bit substring of the first password, the probability that the sampler can leak it is $1-(1-2^{-l/m})^{2^{l/m}} \geq 1-1/e \geq 1/2$. The probability that the full password is leaked is lower bounded by $1/2^m$. This offers a trade-off between the number of queries the sampler is expected to make, and the number of passwords the adversary needs to guess.

Remark. Note that the attack demonstrates why the result in [31, Theorem 6] does not extend simply to hash-dependent password samplers. In fact, it clarifies that *unpredictable* salting is necessary to protect hash-dependent passwords. A predictable salt can be prepended to the candidate password when checking for a 'bad' password, as the sampler can predict which salt will be used.

3.4 Conclusion

In this chapter we extended the ungessability framework of FT21 to allow for password samplers which have access to the random oracle. The framework relates the unrecoverability bounds to the unguessability of the underlying password distribution. We showed that by accounting for the sampler's queries, we can lift each of the reductions in the framework. Importantly, we show that for security with hash-dependent password samplers one needs to salt passwords unpredictably.

This extends the FT21 results by introducing the random oracle H to each of the security games in the reductions. This meant proving the reductions while accounting for the additional oracle and queries. This showed that the relations between Guess and sa-Guess remain unchanged, but the

reduction from UR to SA-GUESS introduced a new term. Essentially this term, $q_{\mathcal{P}} m\ell / K$, formalises the weakness introduced by hash-dependent samplers; it closes the gap opened by the bad sampler counter example.

Having made the necessary extensions, we can move on to analysing the unrecoverability of passwords stored under iMHFs, knowing that we will be able to reduce the advantage down to the guessing games of [30].

4 Hash-dependent Memory Hardness

Hash-dependent memory hardness. Throughout this chapter, we work in the *parallel random oracle model* (pROM) in which algorithms make batches of oracle queries in rounds, storing information about the responses in some state passed between rounds. The *cumulative memory complexity* (CMC) of a parallel algorithm is the expected cumulative sum of the lengths of its states in all rounds. The memory cost $\mathrm{CMC}_{q,\epsilon}(\mathsf{C}^{\mathsf{H}})$ of evaluating a hash-based construction is then defined as the minimum CMC of any q-query parallel algorithm evaluating C^{H} on any input x with success probability at least ϵ. Analogously, for a graph $\mathcal{G}$, the *cumulative complexity* (CC) of the graph is defined as the minimum cumulative sum of the number of *pebbles* required in *pebbling* the graph.

The AS15 framework focused on lower bounding the required CMC of algorithms evaluating MHFs in terms of the CC of the underlying graphs. Intuitively, a lower bound on the CMC required to evaluate C^{H} gives an upper bound on the number of evaluations an adversary can achieve. However, bounds given in the standard definition of CMC only apply for evaluations where the inputs on which C^{H} is evaluated are chosen independently of the hash. This limitation renders such lower bounds inapplicable to proving security against pROM adversaries with oracle access to H which can pick input values dependent on H.

It is relatively straightforward to see why the hash-dependent setting is strictly stronger than the hash-independent setting as any MHF can be converted to one that behaves identically except for a specific hash-dependent input, e.g., $\mathsf{H}(0)$, where it returns 0. The modified construction remains memory-hard in the hash-independent setting as the adversary has a negligible probability of guessing what $\mathsf{H}(0)$ is at the onset, but is trivially not memory-hard in the hash-dependent setting as no resources need to be expended to compute $(\mathsf{H}(0), 0)$ as a valid pair of input–output. A natural question that would arise is whether the AS15 results, which lower-bound the required CMC of a parallel algorithm evaluating a construction C^{H}, hold in this strictly stronger setting. We provide a positive answer to this question.

4.1 From Hash-independent to Hash-dependent Memory Hardness

In the existing hash-independent definition of memory-hard functions, the 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. 4.1 (left). This game is for two stage adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$. Without access to H, adversary $\mathcal{A}_0$ picks a vector $\mathbf{x}$ of m distinct inputs. Then $\mathcal{A}_1$ takes $\mathbf{x}$ as input, and makes batch queries to H, finally outputting $\mathbf{y}$. Adversary $\mathcal{A}$ wins if for each input value x_i in $\mathbf{x}$, the corresponding output y_i in $\mathbf{y}$ satisfies $\mathsf{C}^{\mathsf{H}}(x_i) = y_i$. Specifically, if $\mathrm{CMC}(\mathcal{A})$ is at least μ for all q-query adversaries $\mathcal{A}$ that win the m-MH game with probability at least ϵ, then the construction C^{H} is (m, q, ϵ, μ)-memory hard in the hash-independent setting.

We extend this definition of memory hardness to adversaries that have oracle access whilst choosing the inputs by introducing the dependent memory-hardness game m-dMH in Fig. 4.1 (middle). In this game, the parallel adversary $\mathcal{A}$ is given access to the random oracle H from the outset, and returns a pair of length m vectors $(\mathbf{x}, \mathbf{y})$. As above, $\mathcal{A}$ wins the game if $\mathsf{C}^{\mathsf{H}}(x_i) = y_i$ for each input-output pair (x_i, y_i). We recover the hash-independent m-MH game when the adversary $\mathcal{A}$ consists of two phases $(\mathcal{A}_0, \mathcal{A}_1)$ as described above. Note that the m-dMH game is sufficient to describe a proof-of-work type of game. If the return statement is amended to require that $\mathsf{C}^{\mathsf{H}}(\mathbf{x})$ also satisfies some unlikely property (like leading zeros), we recover a proof-of-work type property.

The parallel adversary $\mathcal{A}$ is given access to the random oracle H, and returns the vectors $(\mathbf{x}, \mathbf{y})$, each of length m. The adversary wins the game if $\mathsf{C}^{\mathsf{H}}(x_i) = y_i$ for each (x_i, y_i). (For salted constructions, the inputs in $\mathbf{x}$ will be of the form (x_i, sa_i) where sa_i is some salt chosen by $\mathcal{A}$. We define adversary $\mathcal{A}$'s advantage in the dependent memory-hardness game as

$$\mathbf{Adv}^{m\text{-dMH}}_{\mathsf{C}^{\mathsf{H}}}(\mathcal{A}) := \Pr\left[m\text{-dMH}^{\mathcal{A}}_{\mathsf{C}^{\mathsf{H}}}\right].$$

For technical reasons, we also introduce a third stronger notion, the *strengthened* MH game sMH, which allows the adversary to output an

unordered set of q input-output pairs, where only m correspond to correct MHF computations. This game is formalized in Fig. 4.1 (right). Note that the m-dMH game is a special case of the sMH game, and the advantage bounds we prove in this game also apply to the m-dMH game. We define adversary $\mathcal{A}$'s advantage in the strong memory-hardness game as

$$\mathbf{Adv}^{(q,m)\text{-sMH}}_{\mathsf{C}^{\mathsf{H}}}(\mathcal{A}) := \Pr\Big[(q,m)\text{-sMH}^{\mathcal{A}}_{\mathsf{C}^{\mathsf{H}}}\Big].$$

However, by relaxing some of the m-dMH game structure, and giving the adversary more capabilities, we both strengthen the notion and make it more flexible. We put this flexibility to use in the next chapter in the proof of Theorem 4, in which we give a reduction to an instance of the sMH game.

Intuition for the changes in the hash-dependent games. Intuitively, both the sMH and m-dMH games allow the adversary to:

- Make queries to (and see the responses from) the random oracle H while choosing the m inputs. This allows it to use its H queries to find candidate inputs which require the least CMC to compute (e.g. the graphs may share node labels).
- Store a state between rounds, and thus adapt the strategy based on the responses. For example, the adversary may use the
- Use the same state for all instances. That is, the instances do not need to be computed in series, and so queries and responses generated for the computation of one instance can be reused in any of the others.

Memory hard construction definition for the hash-dependent setting. For an integer m, we say $\mathcal{A}$ ϵ-computes m hash-dependent instances of C^{H} if $\mathbf{Adv}^{m\text{-dMH}}_{\mathsf{C}^{\mathsf{H}}}(\mathcal{A}) \geq \epsilon$. Given a construction C^{H}, we define its *hash-dependent* (q,ϵ)-dCMC as the CMC of the "best" q-query algorithm that computes C^{H} on some (typically the easiest) input with probability at least ϵ. That is,

$$\text{dCMC}_{q,\epsilon}(\mathsf{C}^{\mathsf{H}}) := \min_{\mathcal{A}\in S[q,\epsilon,\mathsf{C}^{\mathsf{H}}]}\{\text{CMC}(\mathcal{A})\}\ ,$$

where $S[q,\epsilon,\mathsf{C}^{\mathsf{H}}]$ is the set of all q-query algorithms $\mathcal{A}$ in pROM with advantage at least ϵ in the 1-dMH game against C^{H}. Similarly, for $m \in \mathbb{N}$,

we define the (m, q, ϵ)-dCMC of C^{H} as the CMC of the best q-query algorithm that computes C^{H} on at most m (possibly oracle-dependent) instances with probability at least ϵ, where the effort is scaled by the number of instances computed. In other words,

$$\mathrm{dCMC}_{m,q,\epsilon}(\mathsf{C}^{\mathsf{H}}) := \min_{\bar{m}\in[m]} \min_{\mathcal{A}\in S[q,\epsilon,\bar{m},\mathsf{C}^{\mathsf{H}}]} \{\mathrm{CMC}(\mathcal{A})/\bar{m}\} \ ,$$

where $S[q, \epsilon, \bar{m}, \mathsf{C}^{\mathsf{H}}]$ is the set of all q-query algorithms $\mathcal{A}$ in pROM that have advantage at least ϵ in the $\bar{m}$-dMH game against C^{H}.

Definition 3 (Memory-hard construction)**.** *Let m and q be natural numbers, and let μ and ϵ be positive real numbers. A construction C^{H} is* (m, q, ϵ, μ)-memory-hard *if its* (m, q, ϵ)-dCMC *is at least μ.*

Note that this is a translation of the memory hardness definition from [6] to the hash-dependent -dMH game. Also, note that when $m = 1$, we recover the single-instance notion defined above. Intuitively, these definitions describe the average cumulative memory cost *per instance*, for the best adversary succeeding with probability at least ϵ. Measuring the cost per instance allows us to capture the *non-amortizability* of a construction. Informally, any savings in CMC will be captured by this metric, even if they are the result of sharing memory across computations for (possibly many) different instances.

Importantly, this technical definition still takes the cryptanalytic approach. Throughout this paper we will use the approach more natural to cryptography, i.e., we will focus on the advantage in the security game. As such, when we prove memory hardness in the hash-dependent setting, we prove an upper bound on the advantage of an adversary in our hash-dependent game. The reasoning behind this is that the

We build on the results of AS15 and show that, in contrast to general MHFs, *graph-based* MHFs are indeed memory hard even if the input instances are picked in a hash-dependent way. To this end, we will adapt the ex-post-facto pebbling argument. Before doing so, we will further clarify the motivation for our m-dMH game by giving as examples an intuitive hash-dependent notion that does not suffice for our purposes.

4.1.1 Motivation for our definition

In Fig. 4.2 we present the pseudocode for two rejected m-dMH game candidates. By taking the security game implied by the cumulative complexity measure in AS15, and giving the first-stage adversary oracle access to the

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

Figure 4.1: The hash-independent (left), the dependent (middle), 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).

hash, one arrives at the structure which is similar to both of these candidates. In the *salt-split*-dMH game (Candidate 1), the salt is sampled before the first stage adversary runs. We opt to ignore this requirement in the m-dMH game for a technical reason. In any proof which gives a reduction from one game G to the m-dMH game, we would like the adversary to be able to pick the salt. This is because the adversary $(\mathcal{A}_0, \mathcal{A}_1)$ will need to simulate the game G in question.

Removing this requirement from Candidate 1 gives Candidate 2, the *split-m*-dMH game. In this game, the adversary is still forced to commit to a set of inputs before computation begins. While this is the intuitive extension of the m-MH game in Fig. 4.1, it actually doesn't quite capture the setting we wish to consider. By splitting the adversary into stages $\mathcal{A}_0$ and $\mathcal{A}_1$, we closely follow the structure of the m-MH game, but restrict the final-stage adversary without profit. Indeed, a fully adaptive adversary should be able to choose inputs to the construct even at its final step. Since the motivation behind this game is as a *metric*, we would like it to apply to adversaries in as wide a range of games as possible, including adversaries that choose their candidate inputs in the final step. For example, consider a simple adversary $\mathcal{A}$ making guesses in the UR game, with respect to the construction C^{H}. After expending all oracle queries to compute many instances of C^{H}, and comparing each of the outputs to the challenge vector, if $\mathcal{A}$ still has not recovered a password in $\mathbf{pw}$, then $\mathcal{A}$ may submit a final

Candidate 1: *salt-split*-dMH$^{\mathcal{A}}_{\mathsf{C}^{\mathsf{H}}}$	Candidate 2: *split*-dMH$^{\mathcal{A}}_{\mathsf{C}^{\mathsf{H}}}$
$\mathsf{H} \twoheadleftarrow \mathrm{Fun}([N],[M])$	$\mathsf{H} \twoheadleftarrow \mathrm{Fun}([N],[M])$
$\mathbf{sa} \twoheadleftarrow \mathsf{Gen}$	$\mathbf{x} \twoheadleftarrow \mathcal{A}_0^{\mathsf{H}}()$
$\mathbf{x} \twoheadleftarrow \mathcal{A}_0^{\mathsf{H}}(\mathbf{sa})$	if $(\lvert\{\mathbf{x}\}\rvert \neq m)$: return 0
if $(\lvert\{\mathbf{x}\}\rvert \neq m)$: return 0	$\mathbf{y} \twoheadleftarrow \mathcal{A}_1^{\mathsf{H}}(\mathbf{x})$
$\mathbf{y} \twoheadleftarrow \mathcal{A}_1^{\mathsf{H}}(\mathbf{sa},\mathbf{x})$	return $(\mathbf{y} = \mathsf{C}^{\mathsf{H}}(\mathbf{x}))$
return $(\mathbf{y} = \mathsf{C}^{\mathsf{H}}(\mathbf{sa},\mathbf{x}))$	

Figure 4.2: Candidate 1 (left): in this game the salt sampler is specified and the adversary chooses inputs dependent on the hash and the salt sampled. Candidate 2 (right): in this game the salting is removed but adversary is still forced to commit to a set of inputs before the computation begins.

guess vector $\mathbf{x}$, without first computing $\mathsf{C}^{\mathsf{H}}(\mathbf{x})$. If we fix the CMC of each of the adversarial stages, then the *split*-dMH game does not capture this attack.

By removing the split in the adversary, we end up with the m-dMH game. This game is intuitive, and captures both the multi-instance and hash-dependent aspects of the CMC metric we wish to define. As mentioned above, the proofs we provide to bound the advantage in this game are given for an even stronger notion. This is essentially so that we may use the result in two separate areas of the thesis.

Firstly, we use the upper-bound on advantage to derive a lower-bound on the expected cost. This allows us to confirm that the graph-based iMHFs of AS15 remain memory hard in a hash-dependent setting. However, we also put the bound to use in Theorem 4. There we need to give a reduction to a memory-hardness notion, and require the added flexibility in the sMH game.

To achieve this first goal – the lower-bound on cost – we do not need the strengthened sMH game. But then we would need a separate proof, with similar steps, as part of Theorem 4. Therefore, we opt to prove the bound once, in the stronger game, allowing for a more modular approach.

4.1.2 Technical Approach

Before showing that the graph-based iMHFs of AS15 satisfy our stronger notion of memory hardness, we recall the results in the hash-independent setting, to highlight the differences.

The hash-independent case. In [5], the authors prove 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 (with high probability) will have cumulative memory cost close to adversaries which simply pebble 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 [5, 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 a larger construction based on the tensor graph $\mathcal{G}^{\times m}$, whose cumulative complexity is lower bounded in [5, Lemma 12]. The CMC of an adversary computing a single instance of this construction is then bounded, roughly, by the cumulative complexity of the underlying tensor graph. 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, reproduced in Lemma 1, proves the cumulative complexities of two graphs are additive. The bound for the one-off cost of the large graph can then be given in terms of that of the smaller graph underlying the construction. Together, in [5, Theorem 3], these lemmas bound the amortized cost in terms of the cumulative pebbling complexity of the corresponding graph.

Extension to stronger adversaries. To extend the above results, we account for adversarial strategies afforded by the additional adaptivity. In this setting, the adversary can use information from the oracle queries to pick inputs which are known to have collisions in the labeling, reducing the amortized cost. Therefore, calculating the expected fraction of non-colliding instances, which we do in Lemma 6, requires additional analysis.

Although Alwen and Serbinenko [5] focus on the CMC cost, this is arguably better motivated from a cryptanalytic perspective. Here, we instead focus on the advantage an adversary with a fixed CMC has in the (q, m)-sMH game, which better conforms to establishing security. We then derive the expected CMC cost via a sequence of corollaries (see Section 4.3).

First, we prove an upper bound on the advantage that an adversary has in computing m instances in Theorem 2 below. The extension to hash-dependent adversaries requires an analysis of a similar implicit pebbling, while also accounting for adversaries which choose inputs mid-evaluation.

Additionally, to account for adversaries that compute multiple instances, we directly analyse an implicit pebbling of the tensor graph $\mathcal{G}^{\times m}$. This gives an upper bound on the advantage an adversary has in computing these graph-based iMHFs. As a corollary, we prove a lower bound on the amortized cost in this general setting, roughly similar to the bound in the hash-independent setting, showing that they are also strongly memory-hard. For this corollary, we take a modular approach, and first bound the cost of computing multiple instances in terms of the cost of computing a single larger instance.

Comparison with AS15. The required extensions described above do not change the core strategy of the proof, namely using the transcript from the evaluation of a graph-based construction (by a parallel algorithm), to extract a pebbling of the underlying graph. Indeed, we also inherit the method to reason about the pebbling's complexity, i.e., bounding the probability of certain bad events with the predictor bound in Lemma 5. It is the analysis of the bad events, (in the proofs of Claim 1 and Claim 2), which needs to be adapted.

A detailed description of the adaptations to the AS15 method, is given in the paragraphs marked "Changes for Hash-dependence" and "Changes for multiple instances" which appear before the proofs of each of the claims. These give precise descriptions as they appear after the relevant notions are introduced (in the paragraphs following the overview).

As some intuition for where hash-independence is used in the corresponding claims from AS15, consider the transcripts of two parallel algorithms, computing hash-independent and hash-dependent respectively. The hash-independent algorithms choose the input at the outset, whereas the hash-dependent counterparts may not submit the inputs until their final step. This difference in behaviour, when reflected in the transcripts, allows for a straightforward analysis of the pebbling legality in the hash-independent case. Similarly, to see where the AS15 argument puts the single instance setting to use, consider the process of inferring a pebbling from a computation. Where there is a single instance, there is no ambiguity for which instance any given query may be attempting to pebble. The added ambiguity of multiple instances set hurdles for the later stages of the argument – the details of which are highlighted in the aforementioned paragraphs.

4.2 Advantage in the strong memory hardness game

We make use of the following lemma in the proofs involving the pebbling extraction algorithm. The intuition behind the lemma statement is that an algorithm, given a hint, can successfully predict a response from a random oracle with probability bound by the size of the hint space divided by the size of range of the random oracle.

Lemma 5 (Predictor bound (Lemma 1 in [5]))**.** *Let $B = b_1, ..., b_u$ be a sequence of random bits. Let $\mathcal{P}'$ be a randomized procedure that 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 $S \subseteq \{1, ..., u\}$ of $|S| = k$ indices which were not previously queried, along with guesses for all the bits $\{b_i | i \in S\}$. 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 $|\mathrm{H}|/2^k$.*

Now we give the statement of Theorem 2 and present the proof. Recall that we make the following assumptions on the structure of the underlying graphs: that there is a maximum indegree of two; it is directed and acyclic; there is a single source node and a single sink node.

Theorem 2 (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 Definition 6. Then*

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

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

Here, γ essentially captures the difference in the average CMC used per round by $\mathcal{A}$ and average CMC required per round to evaluate m instances of C^{H}. So γ becomes large as the adversarial CMC decreases beyond the minimum required for the construction.

In [5] 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.

To put these claims to use in a multi-instance hash-dependent setting we instead give a reduction from an adversary in the the (q, m)-sMH game to a predictor algorithm, taking care to account for adversaries that choose inputs in a hash-dependent way, and compute multiple instances of the construction. Our analysis relies on extracting a pebbling of $\mathcal{G}^{\times m}$ from an adversary that computes C^{H} on m inputs, and therefore requires alterations to the pebbling extraction algorithm to translate the node indices appropriately. Note that the following argument is independent of whether the construction is evaluated with a salt or not. Any salt used in evaluating the construction (and thus defining the labeling of the graph) is chosen by the adversary. Therefore, without loss of generality, we can consider the input to be a salt-input pair. We refer to the body of the proof for an overview.

Proof. Fix all variables and the 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 itself a round of queries $(q_{i,1}, q_{i,2}, \ldots)$ consisting of $q_{i,j}$ 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 of the form $((v_{i,j}, \ell_{i,j}, \mathit{out}), \perp)$, 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 extraction algorithm of [5] to extract pebbling P of the graph $\mathcal{G}^{\times m}$ *even in a hash-dependent setting.*

Overview. We start with a high-level overview. Before extracting a pebbling, we first determine, from the transcript, a vector $\mathbf{x}^*$ of m inputs for which $\mathcal{A}$ correctly computed the output of the construction. As we consider adversaries which may output many more values than they correctly compute, we first extract the subset of inputs which are relevant to the pebbling. For each $k \in [m]$, the input $\mathbf{x}^*[k]$ will define a labeling of the graph $\mathcal{G}$, and using these m computations, we extract a pebbling of the tensor graph $\mathcal{G}^{\times m}$. A pebbling $P = (P_0, P_1, \ldots, P_t)$ 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 some 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 $H(\perp) :=\perp$.[1] 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 Definition 5 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 that node. Formally, a query (x, y) in the transcript is called correct (for some input $\mathbf{x}^*[k]$) if there exists some non-sink node v in the graph $\mathcal{G}$ such that $x = \mathbf{pre\text{-}lab}_k(v)$, or there exists a sink node v in the graph such that $x = (v, \mathbf{lab}_k(v), \mathit{out})$. 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). For the description of the procedure in pseudocode, see Fig. 4.3.

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

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

translating from a node in the graph $\mathcal{G}^{\times m}$ to a node in $\mathcal{G}$, so correctness is defined by the $k-$th input in $\mathbf{x}^*$. A description of the procedure is given in Fig. 4.3.

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 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.[2]
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 which are inherited from a previous step and ones which are added due to a correct oracle call. See Fig. 4.3 for the pseudocode describing the ex-post-facto pebbling extraction. With the pebbling extraction defined, we can analyze a pebbling extracted from a hash-dependent adversary. We now prove two claims. To prove the first (legality) we make a similar prediction argument to that made in [5], 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.

Claim 1 (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 $\mathcal{A}$.*

An analogous result was proven in [5] for a pebbling of $\mathcal{G}$ extracted from a hash-independent q-query adversary computing a single instance. In such a setting, illegal pebble placement must correspond to an adversary

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

Extraction$(\mathcal{G}, T)$:

0. $((\mathbf{q}_1, \ldots, \mathbf{q}_t), \mathbf{x}^*) \leftarrow \text{parse}(T)$
1. $P_0, P_1, \ldots, P_t \leftarrow \emptyset$
2. for $i = 1$ to t do
3. $\quad P_i \leftarrow P_{i-1}$
4. $\quad$ for $(x_{ij}, y_{ij}) \in \mathbf{q}_i$ do
5. $\quad\quad$ for $k \in [m]$ do
6. $\quad\quad$ if isCorrect$(x_{ij}, k, \mathcal{G}, T)$ then
7. $\quad\quad\quad (v, *) \leftarrow \text{parse}(x_{ij})$
8. $\quad\quad\quad P_i \leftarrow P_i \cup \{(v, k)\}$
9. $\quad$ for $(v, k) \in P_i$ do
10. $\quad\quad$ if $\neg$isNecessary$((v, k), i, \mathcal{G}, T)$ then
11. $\quad\quad\quad P_i \leftarrow P_i \setminus \{(v, k)\}$
12. return $(P_0, P_1, \ldots, P_t)$

isCorrect$(x, k, \mathcal{G}, T)$:

15. $((\mathbf{q}_1, \ldots, \mathbf{q}_t), \mathbf{x}^*) \leftarrow \text{parse}(T)$
16. $(v, *) \leftarrow \text{parse}(x)$
17. $(V, E) \leftarrow \text{parse}(\mathcal{G})$
18. if $v \in V$ then
19. $\quad$ if $x = \mathbf{pre\text{-}lab}_k(v)$:
20. $\quad\quad$ return true
21. if $x = ((v, \mathbf{lab}_k(v)), out)$ then
22. $\quad$ return true
23. return False

isNecessary$((v, k), i, \mathcal{G}, T)$:

25. for $j = i + 1$ to t do
26. $\quad$ for $(u, k) \in \mathbf{Ch}(v)$ do
27. $\quad\quad$ for $x_{j,j'} \in \mathbf{q}_j$ do
28. $\quad\quad\quad$ if $(u, *) = x_{j,j'}$ and isCorrect$(x_{j,j'}, k, \mathcal{G}, T)$:
29. $\quad\quad\quad\quad$ for $\ell = i + 1$ to $j - 1$ do
30. $\quad\quad\quad\quad\quad$ for $x_{\ell,\ell'} \in \mathbf{q}_\ell$ do
31. $\quad\quad\quad\quad\quad\quad$ if $(v, *) = x_{\ell,\ell'}$ and isCorrect$(x_{\ell,\ell'}, k, \mathcal{G}, T)$:
32. $\quad\quad\quad\quad\quad\quad\quad$ return False
33. $\quad\quad\quad$ return true
34. return False

Figure 4.3: pseudocode for the algorithm to extract a pebbling of $\mathcal{G}^{\times m}$ from a transcript of the sMH game.

predicting the output to the random oracle. To see how, recall that a node v is added to pebbling step P_i if a correct call has been made in the i-th round of queries. Any correct call for some node v is a query containing $\mathbf{pre\text{-}lab}(v)$, and thus the labels for the parents of v.[3] If the corresponding node was pebbled illegally, one of these labels in $\mathbf{pre\text{-}lab}(v)$ (say for node u) has not been received as a response from the oracle. Such an adversary can be reduced to a predicting algorithm, with success probability bound by Lemma 5. The predictor outputs a guess $\mathbf{lab}(u)$ for the input $\mathbf{pre\text{-}lab}(u)$. For hash-independent adversaries, the hint the predictor receives is the query index $i \in [q]$ of the call which illegally pebbles v. This gives a bound of q/N.

Changes for 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 $\mathcal{G}^{\times m}$, when adding pebbles we translate the node indices (for queries labeling distinct instances of a construction), into node indices for (distinct subgraphs of) $\mathcal{G}^{\times m}$.

Changes for hash-dependence. 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.

Proof. Let bad be the event that P is an illegal pebbling. That is, there is step $P_i \in P$ such that for some $(v,k) \in P_i$ there exists a parent node $\mathbf{Pa}_j((v,k))$, such that $\mathbf{Pa}_j((v,k)) \notin P_{i-1}$. Any node (v,k) from $\mathcal{G}^{\times m}$ is only placed in P if there is a query (in the transcript) of the form $\mathbf{pre\text{-}lab}_k(v)$ where v is the corresponding node in $\mathcal{G}$. If (v,k) is placed illegally, then there is some bad parent node $(u,k) := \mathbf{Pa}_j((v,k))$ such that for some $k \in [m]$, the label $\mathbf{lab}_k(u)$ is contained in $\mathbf{pre\text{-}lab}_k(v)$, though no *correct call* (with respect to the input $\mathbf{x}^*[k]$) was previously made for u in $\mathcal{G}$. Note that if there were a previous correct call for u, the ex-post-facto pebbling algorithm would have kept (u,k) in P_{i-1}.

[3] In the single-instance hash-independent setting, there is a single pre-chosen input x to the construction, so **pre-lab** and **lab** are both defined with respect to x.

We next claim that $\Pr[\mathsf{bad}]$ is upper bounded by q^2/N. Let $\mathcal{A}$ be an adversary that triggers event bad. We convert $\mathcal{A}$ to an algorithm $\mathcal{B}$ against the prediction game which takes a hint, which is dependent on $\mathcal{A}$, from a space of size q^2. We next apply Lemma 5, to obtain the upper bound. The reduction (which is no longer ex-post-facto) proceeds as follows.

Hint. The hint function is defined as a pair (i, ℓ) where the first entry is the index $i \in [q]$ of the oracle call for the first node (v, k) as defined above (which causes the illegal pebble placement in P). The second is the index $\ell \in [q]$ of the first correct oracle call for the bad parent (u, k) of (v, k) for which there was no *previous* correct oracle call. If there is no correct call for (u, k), the predictor gets the hint (i, i), and if there is no illegal pebble, then the predictor gets the hint $(\perp, \perp)$.

One difference here with that in [5] is the increased hint space. This is because in our setting the predictor does not get access to the input $\mathbf{x}^*$ from the outset – $\mathcal{A}$ outputs $\mathbf{x}^*$ as it finishes. The predictor therefore also needs to receive the index of the query corresponding to the first correct oracle call for the bad parent (u, k) so that it can compute $\mathbf{pre\text{-}lab}_k(u)$. The increase in the hint space changes the bound from q/N to q^2/N.

Algorithm $\mathcal{B}$ receives a hint (i, ℓ) as defined above and runs $\mathcal{A}$ as follows, aborting if the hint is of the form $(\perp, \perp)$ or is not of the correct form. $\mathcal{B}$ is aiming to predict the value of $\mathsf{H}(\mathbf{pre\text{-}lab}_k(u))$ without querying $\mathbf{pre\text{-}lab}_k(u)$ to the oracle.

Reduction. $\mathcal{B}$ forwards all oracle calls of $\mathcal{A}$ to H up to and including the i-th query. This query is correct, by the definition of the hint function, and so it contains the labels of the parents of v in $\mathcal{G}$, with respect to $\mathbf{x}^*[k]$ for some $k \in [m]$. A correct call for at least one parent, say u, is never queried to H. After receiving the i-th query $\mathcal{B}$ extracts $\mathbf{lab}_k(u)$ from $\mathbf{pre\text{-}lab}_k(v)$. $\mathcal{B}$ then resumes running $\mathcal{A}$ under one of the following cases.

Case 1: If the hint is of the form (i, ℓ) where $i \neq \ell$, then $\mathcal{B}$ forwards all of $\mathcal{A}$'s oracle calls to H until the ℓ-th query and records $\mathbf{pre\text{-}lab}_k(u)$.

Case 2: If the hint is of the form (i, i), then there is no later correct call for u and $\mathcal{B}$ runs $\mathcal{A}$ until it terminates, allowing $\mathcal{B}$ to record $\mathbf{x}^*[k]$ from the adversary's output. $\mathcal{B}$ then uses $\mathbf{x}^*[k]$ to recompute $\mathbf{pre\text{-}lab}_k(u)$, making queries to H as needed.

In both cases, $\mathcal{B}$ has the values for $\mathbf{lab}_k(u)$ and $\mathbf{pre\text{-}lab}_k(u)$, and did not forward $\mathbf{pre\text{-}lab}_k(u)$ to the random oracle. $\mathcal{B}$ now returns $\mathbf{lab}_k(u)$ as its guess for the bits of $\mathsf{H}(\mathbf{pre\text{-}lab}_k(u))$.

If $\mathcal{A}$ triggers the event bad, then $\mathcal{B}$ wins the prediction game. As the hint is from a space of size q^2, and the prediction is in a space of size N, then by Lemma 5 the probability of $\mathcal{B}$ winning the prediction game, and therefore the probability of bad being triggered, is upper bounded by q^2/N. The probability that P is legal is therefore at least $1 - q^2/N$. □ □

Claim 2 (Pebbling complexity)**.** *Let T be a winning transcript corresponding to 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 $\lambda \geq 0$:*

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

over the choice of H and the coins of $\mathcal{A}$. Where σ_i is the state as defined in Section 2.1.1.

Changes for multiple instances. An analogous result was proven in [5], 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.

As each node $(v, k) \in P_i$ is necessary, there is a later round in which there is a correct oracle call (with respect to some $\mathbf{x}^*[k]$) containing $\mathbf{lab}_k(v)$ as a label for one of the input nodes – call this a critical call for v. As there are no correct calls for v with respect to $\mathbf{x}^*[k]$ in the meantime, the adversary making this oracle call must be able to find $\mathbf{lab}_k(v)$ using only the state σ_i. Intuitively the size of σ_i must limit the number of nodes in P_i. This intuition is formalized by constructing an algorithm $\mathcal{B}$ which predicts the label of each node (v, k) in the set P_i with respect to the $(\mathbf{x}^*[k], \mathsf{H})$-

labeling of $\mathcal{G}$. The predictor will get a hint from a hint function, and has access to H.

Proof. Let bad_i be the event that for a fixed round $i \in [t]$ and some $\lambda \geq 0$, the size of P_i exceeds the bound in the claim. We claim that $\Pr[\mathsf{bad}_i]$ is upper bounded by $2^{-\lambda}$. Let $\mathcal{A}$ be an adversary which triggers the event bad_i. Given $\mathcal{A}$ we build an algorithm $\mathcal{B}$ against the prediction game, which takes a hint from a space of size at most $q^{|P_i|}2^{|\sigma_i|}$ and successfully predicts, for *each* $(v,k) \in P_i$ the response of oracle query $\mathbf{pre\text{-}lab}_k(v)$, predicting a total of $|P_i| \log N$ bits. Note that k does not have to be the same for each v. Recall a *k-critical call* for node $(v,k) \in P_i$ is a correct oracle call (with respect to $\mathbf{x}^*[k]$) containing $\mathbf{lab}_k(v)$ as a label for one of its input nodes. Intuitively, this is the oracle call that makes (v,k) necessary at step i. The reduction proceeds as follows.

Hint. The hint function is defined as a pair (J, σ_i) where σ_i is the adversary's i-th state and the set $J = \{(j_1,k_1),\ldots,(j_r,k_r)\} \in [q]^r \times [m]^r$ is the set of indices of the critical calls for each $(v,k) \in P_i$, and the indices of the instances for which they are correct. A pair (j,k) means the j-th query from $\mathcal{A}$ is a critical call, correct with respect to $\mathbf{x}^*[k]$.

Algorithm $\mathcal{B}$ receives the hint (as defined above) and runs $\mathcal{A}$ with input σ_i. $\mathcal{B}$ is aiming to predict, for each $(v,k) \in P_i$, the values of $\mathsf{H}(\mathbf{pre\text{-}lab}_k(v))$.

Reduction. $\mathcal{B}$ records all queries from $\mathcal{A}$, and forwards any calls for nodes not in P_i to the oracle, relaying the response. For all critical calls, $\mathcal{B}$ records the labels of the input nodes, along with the instance for which they are correct (these contain the prediction values used later). For all queries of the form $(v, l_1, \ldots, l_\delta)$,[4] $\mathcal{B}$ makes the following *recursive* check for correctness, starting by checking if, for any $k \in [m]$ such that $(v,k) \in P_i$ the query $(v, l_1, \ldots, l_\delta)$ is correct for v. For any query $x_{i,j}$, $\mathcal{B}$ can check if it is correct for some node $w \in \mathcal{G}$, with respect to some $\mathbf{x}^*[k]$, with the following steps

- Check if $x_{i,j}$ is a critical call for some node in P_i (in which case the index of the query appears in J). Then $x_{i,j}$ is correct, as all critical calls are correct by definition. Moreover, the hint contains the instance for which the call is correct.

[4] We are considering queries which would label v for any of the m inputs.

- Check, recursively, if there are previous correct calls for all parents of w, whose labels match the pre-label in the query. This search ends in at least one critical call. Referring to the hint, $\mathcal{B}$ finds the index for which the call is correct.

If $x_{i,j} = (v, l_1, \ldots, l_\delta)$ is a correct call for $(v, k) \in P_i$, and $\mathbf{lab}_k(v)$ is already recorded, $\mathcal{B}$ responds with $\mathbf{lab}_k(v)$, or else it forwards the call to the oracle and relays the response. If $x_{i,j}$ was a correct call for v with respect to $\mathbf{x}^*[k]$, then $\mathcal{B}$ will record the pair $((v, l_1, \ldots, l_\delta), \mathbf{lab}_k(v))$ to output later. Once $\mathcal{A}$ is run, $\mathcal{B}$ has $\mathbf{lab}_k(v)$ for all $(v, k) \in P_i$ without querying $\mathbf{pre\text{-}lab}_k(v)$ to the oracle. $\mathcal{B}$ can then submit $\mathbf{lab}_k(v)$ as its prediction for $\mathsf{H}(\mathbf{pre\text{-}lab}_k(v))$ for each $(v, k) \in P_i$, correctly predicting the response of the random oracle.

Analysis. As the size of the hint space is $q^r m^r 2^{|\sigma_i|}$, and the space of bits being guessed is $N2^{|P_i|}$, then by Lemma 5 $\mathcal{B}$ can do this with probability at most $q^r m^r 2^{|\sigma_i|} / N2^{|P_i|}$.

$$\Pr[\mathcal{B} \text{ wins}] \leq \frac{q^r m^r \ 2^{|\sigma_i|}}{N \ 2^{|P_i|}} \leq \frac{2^{|P_i| \log(qm)} \ 2^{|\sigma_i|}}{2^{|P_i| \log(N)}} = 2^{-|P_i|(\log(N) - \log(qm)) + |\sigma_i|} \leq 2^{-\lambda}.$$

Where the second inequality is from $r \leq |P_i|$, and the final one from $-|P_i| < \frac{-|\sigma_i| - \lambda}{\log(N) - \log(qm)}$. This gives $\Pr[\mathsf{bad}_i] \leq \Pr[\mathcal{B} \text{ wins}] \leq 2^{-\lambda}$. Applying the union bound over all time steps gives

$$\Pr[\mathsf{bad}] \leq \sum_i \Pr[\mathsf{bad}_i] \leq t2^{-\lambda}.$$

□

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 3 (Collision free transcript)**.** *Let $\mathcal{A}$ be a q-query adversary in the (q, m)-*sMH *game. Let* 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 there is such a collision, and*

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

To upper-bound the probability of a bad, let $\mathbf{lab}_j(v)$ and $\mathbf{lab}_k(v)$ be the labels for a node v, then a collision in node labels corresponds to a collision

in two hash queries to the random oracle H. Namely, if $\mathbf{lab}_j(v) = \mathbf{lab}_k(v)$, then $\mathsf{H}(v, \ldots, \mathbf{lab}_j(\mathbf{Pa}_\delta(v))) = \mathsf{H}(v, \ldots, \mathbf{lab}_k(\mathbf{Pa}_\delta(v)))$. For each node v, let q_v be the number of queries $\mathcal{A}$ makes, beginning with index v. Then, as $\mathcal{A}$ labeled these two nodes, the probability that these two labels collide is at most q_v^2/N. Thus, the probability that $\mathcal{A}$ finds a collision in the labels of any node is at most

$$\frac{1}{N}\sum_{1}^{n} q_i^2 \leq \frac{1}{N}\left(\sum_{1}^{n} q_i\right)^2 \leq \frac{q^2}{N}\,.$$

The above three claims allow us to complete the proof for Theorem 2. The final steps in the analysis closely follow the proof in [5] for the hash-independent setting.

For any q-query adversary $\mathcal{A}$ in the (q, m)-sMH game, with respect to 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. □

Now we restate and prove Corollary 2, to lower bound the minimum expected CMC needed by a hash-dependent adversary computing a single instance of C^{H}.

Corollary 2 (One-off cost)**.** *Let all parameters be as above, then for* $\epsilon \in (q/N, 1]$*,* $q < \sqrt{N}$ *and* $\bar{\epsilon} := -\log(\epsilon - q^2/N)$*:*

$$\mathrm{dCMC}_{q,\epsilon}(\mathsf{C}^{\mathsf{H}}) \geq \frac{\epsilon \cdot \mathrm{CC}(\mathcal{G})(\log N - \log q)}{\epsilon\bar{\epsilon} + 1}\,.$$

Proof. Let $m = q' = 1$, then we recover the single instance 1-dMH game, and the proof of Theorem 2 gives

$$\mathrm{CC}(\mathcal{G}) \leq \frac{t\bar{\epsilon} + \sum_{i=0}^{t-1} |\sigma_i|}{\log(N) - \log(q)} \ .$$

And so $\sum_{i=0}^{t-1} |\sigma_i| > \mathrm{CC}(\mathcal{G})(\log N - \log q) - t\bar{\epsilon}$. Assuming that the adversary incurs no cost for failing to compute the iMHF, then the expected cumulative memory cost for an adversary which wins with probability at least ϵ is at least $\epsilon(\mathrm{CC}(\mathcal{G})(\log N - \log q) - t\bar{\epsilon})$.

The cost of an adversary must also be at least t (as at least one bit is stored per round). As $\epsilon(\mathrm{CC}(\mathcal{G})(\log N - \log q) - t\bar{\epsilon})$ is monotonically decreasing in t we can set $\epsilon(\mathrm{CC}(\mathcal{G})(\log N - \log q) - t_0\bar{\epsilon}) = t_0$ and solve for t_0. This gives the bound from the corollary statement.

□

4.3 Meeting a stronger CMC notion

Formalising the metric. For an integer m, we say $\mathcal{A}$ ϵ-computes m instances of C^{H} if $\mathbf{Adv}^{m\text{-dMH}}_{\mathsf{C}^{\mathsf{H}}}(\mathcal{A}) \geq \epsilon$. Given a construction C^{H}, we define its *strong* (q, ϵ)-dCMC as the CMC of the "best" q-query algorithm that computes C^{H} on some (typically the easiest) input with probability at least ϵ. That is,

$$\mathrm{dCMC}_{q,\epsilon}(\mathsf{C}^{\mathsf{H}}) := \min_{\mathcal{A} \in S[q,\epsilon,\mathsf{C}^{\mathsf{H}}]} \{\mathrm{CMC}(\mathcal{A})\} \ ,$$

where $S[q, \epsilon, \mathsf{C}^{\mathsf{H}}]$ is the set of all q-query algorithms $\mathcal{A}$ in pROM with advantage at least ϵ in the 1-dMH game against C^{H}. Similarly, for $m \in \mathbb{N}$, we define the (m, q, ϵ)-dCMC of C^{H} as the CMC of the best q-query algorithm that computes C^{H} on at most m (possibly hash-dependent) instances with probability at least ϵ, where the effort is scaled by the number of instances computed. In other words,

$$\mathrm{dCMC}_{m,q,\epsilon}(\mathsf{C}^{\mathsf{H}}) := \min_{\bar{m} \in [m]} \min_{\mathcal{A} \in S[q,\epsilon,\bar{m},\mathsf{C}^{\mathsf{H}}]} \{\mathrm{CMC}(\mathcal{A})/\bar{m}\} \ ,$$

where $S[q, \epsilon, \bar{m}, \mathsf{C}^{\mathsf{H}}]$ is the set of all q-query algorithms $\mathcal{A}$ in pROM that have advantage at least ϵ in the $\bar{m}$-dMH game against C^{H}. We say that a construction C is (m, q, ϵ, μ)-memory-hard if its (m, q, ϵ)-CMC is at least μ. Note that when $m = 1$, we recover the single-instance notion defined above. Intuitively, these definitions describe the average cumulative memory cost

per instance, for the best adversary succeeding with probability at least ϵ. Measuring the cost per instance allows us to capture the *non-amortizability* of a construction. Informally, any savings in CMC will be captured by this metric, even if they are the result of sharing memory across computations for (possibly many) different instances.

We present two corollaries, lower bounding the cost of computing the construction. These follow from the proof of Theorem 2 given in Section 4.2. First, we obtain a lower bound on the *cost* of computing m instances in terms of the computational cost of a single larger instance. In the hash-independent case, collisions in graph labels are not under the control of the adversary, whereas adversaries with access to H can use oracle queries to search for collisions and pick inputs *depending on* collisions in the hash. We account for adversaries which may use the additional adaptivity to choose inputs which result in collisions. This gives a factor of q^2 to the expected fraction of instances with collisions. In the hash-independent setting, the bound is independent of adversarial queries, as the expected number of collisions is fixed by the total number of nodes.

Lemma 6 (Cost of collisions)**.** *Let all parameters be as above, let* $\epsilon \in (q/N, 1]$ *be a real number and* $\beta(m) := \max\{0, 1 - mq^2/N\}$*. Then,*

$$m \cdot \mathrm{dCMC}_{m,q,\epsilon}(\mathsf{C}^{\mathsf{H}}) \geq \mathrm{dCMC}_{q,\epsilon}(\mathsf{C}^{\mathsf{H}}_{\times m\beta(m)}) \ .$$

Proof. Following from Theorem 2, the probability $\mathcal{A}$ finds a collision in the labels for any node is at most q^2/N. Therefore, when computing m instances, the expected number of collisions is at most $(m-1)mq^2/2N \leq m^2q^2/N$. Let u be the number of non-colliding instances. If $\mathcal{A}$ can compute an instance for free with every collision, then the expected value of u is at least $m - m^2q^2/N = m\beta(m)$. Thus the total cost of computing m instances (in which node labels may appear in multiple instances) is at least the cost of computing $m\beta(m)$ instances in which all nodes have different labels. We now show how the expected number of non-colliding instances relates to the definitions of dCMC:

$$\begin{aligned}
m \cdot \mathrm{dCMC}_{m,q,\epsilon}(\mathsf{C}^{\mathsf{H}}) &:= \min_{\bar{m}\in[m]} \min_{\mathcal{A}\in S[q,\epsilon,\bar{m},\mathsf{C}^{\mathsf{H}}]} \{m \cdot \mathrm{CMC}(\mathcal{A})/\bar{m}\} \\
&= \min_{\bar{m}\in[m]} \{ \min_{\mathcal{A}\in S[q,\epsilon,\bar{m},\mathsf{C}^{\mathsf{H}}]} \{m \cdot \mathbb{E}_{\mathsf{H},r}[\mathrm{CMC}(\mathcal{A},\mathsf{H},r)]/\bar{m}\}\} \\
&\geq \min_{\mathcal{A}\in S[q,\epsilon,\mathsf{C}^{\mathsf{H}}_{\times m\beta(m)}]} \{\mathrm{CMC}(\mathcal{A})\} \\
&=: \mathrm{dCMC}_{q,\epsilon}(\mathsf{C}^{\mathsf{H}}_{\times m\beta(m)}) \ ,
\end{aligned}$$

where the inequality follows from the argument above, noting that both the CMC metric and $\beta(m)$ are expectations taken over the choice of H. □

Next, we recall Corollary 2 (proved in Section 4.2) which gives a lower bound on the minimum expected CMC needed for a hash-dependent adversary to compute a single instance of a construction C^{H}. The bound we got is the same as that for hash-dependent adversaries, up to a small collision factor.

Corollary 2 (One-off cost)**.** *Let all parameters be as above, then for* $\epsilon \in (q/N, 1]$*,* $q < \sqrt{N}$ *and* $\bar{\epsilon} := -\log(\epsilon - q^2/N)$*:*

$$\mathrm{dCMC}_{q,\epsilon}(\mathsf{C}^{\mathsf{H}}) \geq \frac{\epsilon \cdot \mathrm{CC}(\mathcal{G})(\log N - \log q)}{\epsilon\bar{\epsilon} + 1} \ .$$

Next, we use the above two lemmas along with the additive property of a graph's cumulative complexity to derive a lower bound on the dCMC of graph-based iMHFs. This extends the non-amortizability of such iMHFs to the hash-dependent setting.

Theorem 3 (Amortized cost)**.** *Let all parameters be as above and let* ϵ *be a real number. Let* $\beta(m) := \max\{0, 1 - mq^2/N\}$ *and suppose* $\epsilon \in (q/N, 1]$ *and* $q < \sqrt{N}$*, let* $\bar{\epsilon} := -\log(\epsilon - q^2/N)$*. Then,*

$$\mathrm{dCMC}_{m,q,\epsilon}(\mathsf{C}^{\mathsf{H}}) \geq \frac{\beta(m)\epsilon \cdot \mathrm{CC}(\mathcal{G})(\log N - \log q)}{\epsilon\bar{\epsilon} + 1} \ .$$

Proof. Following the analogous proof for hash-independent adversaries in [5], we combine our hash-dependent results to reach a similar bound. Concretely,

$$\begin{aligned}
\mathrm{dCMC}_{m,q,\epsilon}(\mathsf{C}^{\mathsf{H}}) &= \min_{\bar{m}\in[m]} \min_{\mathcal{A}\in S[q,\epsilon,\bar{m},\mathsf{C}^{\mathsf{H}}]} \{\mathrm{CMC}(\mathcal{A})/\bar{m}\} && \text{(by definition)} \\
&\geq \min_{\mathcal{A}\in S[q,\epsilon,\mathsf{C}^{\mathsf{H}}_{\times m\beta(m)}]} \{\mathrm{CMC}(\mathcal{A})/m\} && \text{(by Lemma 6)} \\
&\geq \frac{\epsilon \cdot \mathrm{CC}(\mathcal{G}^{\times m\beta(m)})(\log N - \log q)}{m(\epsilon\bar{\epsilon} + 1)} && \text{(by Corollary 2)} \\
&\geq \frac{\beta(m)\epsilon \cdot \mathrm{CC}(\mathcal{G})(\log N - \log q)}{\epsilon\bar{\epsilon} + 1} \ . && \text{(by Lemma 1)}
\end{aligned}$$

□

Example application. We can apply the bound from Theorem 3 to the Argon2i family of MHFs. For an instantiation of Argon2i with n nodes, the tightest bounds on the cumulative complexity of the underlying graph are that of [19], giving a lower bound of $\tilde{\Omega}(n^{1.75})$. Assuming the round function of Argon2i behaves as a random oracle, we let C^{H} be Argon2i with n nodes. Applying our bound, and keeping parameters as above, we derive the minimum dCMC required to compute m instances with probability at least ϵ:

$$\mathrm{dCMC}_{m,q,\epsilon}(\mathsf{C}^{\mathsf{H}}) \geq \frac{\beta(m)\epsilon \cdot n^{1.75}(\log N - \log q)}{\epsilon\bar{\epsilon} + 1} \ .$$

Therefore, any q-query algorithm computing m instances of Argon2i essentially needs the CMC required to pebble the underlying graph with the hash digests.

Comparison to the hash-independent bounds. It is natural to ask how the bound in Theorem 3 bound compares to the analogous hash-independent case. Let $\mathrm{CMC}_{m,q,\epsilon}(\mathsf{C}^{\mathsf{H}})$ denote the expected cost for a parallel adversary to compute m instances of the construction C^{H} with success probability at least ϵ, in the hash-independent setting, i.e., in the m-MH game. Let $\beta(m) := \max\{0, 1 - mn/N\}$, $\epsilon \in (q/N, 1]$ and $q < \sqrt{N}$, let $\bar{\epsilon} := -\log(\epsilon - q/N)$, then from AS15 we have that

$$\mathrm{CMC}_{m,q,\epsilon}(\mathsf{C}^{\mathsf{H}}) \geq \frac{\beta(m)\epsilon \cdot \mathrm{CC}(\mathcal{G})(\log N - \log q)}{\epsilon\bar{\epsilon} + 1} \ .$$

Importantly, both the hash-dependent and hash-independent bounds are similar, as both are essentially linear in the pebbling complexity of the underlying graph.

The differences are in the parameters $\beta(m)$, and $\bar{\epsilon}$. In our stronger setting, the mn/N term is replaced with a mq^2/N term. Intuitively, this captures the additional collision probability available to an adversary that picks inputs adaptively. The change to $\bar{\epsilon}$ is a technical consequence from the extension to the hint we give in the proof of Theorem 2.

4.4 Conclusion

In this chapter, we have extended the notion of memory hardness to account for adversaries which pick their inputs adaptively. With this strengthened definition, we then asked whether the graph-based data-independent MHFs

of AS15 were still memory hard. By giving an adapted pebbling reduction, we were able to answer this question in the affirmative. The indexing of graph-based iMHFs is enough to ensure memory hardness even against more adaptive adversaries, and the lower-bounds we derive on the cost of computing m instances show that the computation cannot be amortized effectively.

This is good news, both for our stronger definitions and the constructions we consider. For a large class of iMHFs we have proven good upper bounds on the advantage in the security game, and good lower bounds on the cost. This confirms that the stronger hash-dependent setting is still useful, and the current model for graph-based iMHFs is secure even in a multi-instance setting. Additionally, we can be content with the paradigm for building graph-based iMHFs. As the bounds are essentially linear in the CC of the underlying graph, it suffices to pick a graph with high CC.

Looking forward, we will put the main result of this chapter to work in the proof of Theorem 4. Having bound the advantage of an adversary in the *stronger* sMH game in Theorem 2, we will make use of all of the adaptivity it affords the adversary.

5 Unrecoverability of MHFs

In this chapter we will use the main results of Chapters 3 and 4, to prove a multi-instance unrecoverability bound for hash-dependent passwords stored under the class of memory hard functions in Definition 5. This result is the culmination of extending both the unguessability framework of FT21 and the pebbling reduction framework of AS15, allowing us to upper-bound the advantage of a parallel adversary in the hash-dependent UR game in terms of the adversary's cumulative memory complexity and the unguessability of the password distribution, even when the distribution is dependent on the underlying hash function. As the entirety of this chapter is dedicated to the proof of Theorem 4, we give some intuition to the main steps in an overview.

5.1 Proof overview

In the proof we use the unpredictability of the salts to argue that the queries of the password sampler are independent from the queries made in the challenge generation. We prove this by showing that the unpredictability due to salting "propagates" through the construction, so that its queries are unpredictable from the sampler's point of view. Specifically, salting only the inputs is sufficient for this (as opposed to salting all nodes of the graph.)

We then show that outside of the full computations of C^{H}, from the point of view of the adversary, the outputs to the construction are random and independent of values for the underlying hash function. To do this, after ruling out collisions in the outputs of the random oracle, we consider whether the adversary can predict the label of a vertex generated during the preparation of challenges without performing a computation of the MHF in the *forward* direction. Then, we introduce a FindChain procedure that detects forward computations and aborts the game if a vertex label is correctly computed but FindChain does not detect it. If such a case arises, there is a correctly labeled vertex for which the adversary does not label at least one of its parents. We can rule this case out, as the adversary cannot predict the output of the random oracle. This enforces

MHF evaluations to be done in the forward direction only, and puts us in a position to bound the number of full computations via a reduction to the memory-hardness of the MHF.

In this reduction, we make full use of the adversarial adaptivity in the sMH game. Note that by inspecting the node indices in the adversary's H queries, we can detect both inputs and outputs to the construction, i.e., queries with the index of an input node, and responses which label the output node. However, detecting *which* inputs correspond to which outputs blows up the CMC of the adversary in the memory hardness game: to match input-output pairs and submit them later, one needs to store at least one label for each of the inputs. As a q-query adversary can start up to q computations, neither storing input-output pairs, nor recomputing them can be done efficiently. The strong memory hardness game is critical here, as the adversary has the flexibility to provide *sets* of up to q candidate valid inputs and outputs, of which m must be successful, rather than ordered vectors. As the sets are unordered, the adversary in the sMH game does not have to keep track of the correspondence, and the reduction doesn't blow up the CMC.

The final step of the proof is a reduction to the sa-Guess game, à la FT21, with modifications to account for the non-monolithic nature of the construction. In keeping the simulated hash function compatible with the challenge values, we need to place a query to the Test oracle if and only if a full chain computation arises. This is because when a full chain starts with one of the challenge password-salt pairs we need to pick the correct challenge value as the response to the hash query corresponding to the label of the sink node. The number of Test queries in this reduction is equal to the number of *full chains* that the adversary computes, which was bounded in the previous step. Note that a Test query needs to be placed for *every* sink node. With multiple sink nodes the adversary can use a reduced CMC to verify a guess. Hence, the iMHF constructions that we consider are those with a *single* sink node.

Assumptions on salt generation, and the password distribution. With a reduction to sa-Guess, our advantage bound inherits the collision term on the generated salts. As discussed in Chapter 3, for strong security guarantees we require the salts to be approximately uniformly and independently distributed. As this will be needed anyway, we assume a uniform salt generator Gen, sampling independent salts from a space K. Although one could consider low entropy salts, (or even adversarially chosen salts),

independently and uniformly distributed salts are more strongly motivated by practice.

For the password distribution, we do not make these assumptions. In fact we want to account for the setting where passwords are correlated, and not sampled independently. These bounds will show that high entropy salts are sufficient to derive good security bounds in terms of adversarial CMC, and the unguessability of the passwords.

5.2 Unrecoverability of Memory-Hard Functions

Having examined how to bound the advantage of an adversary trying to compute a graph-based iMHF on m distinct inputs, we now consider the multi-instance unrecoverability of graph-based iMHFs. We prove the first bounds in terms of the adversary's CMC, as opposed to just query complexity. Additionally, our results account for non-uniform and even hash-dependent password distributions. In Section 5.3, we provide an extension of this main theorem to one-wayness security. But here, we start with a formal statement of our result.

Theorem 4 (UR security of C^{H})**.** *Let $P, q, n, m, c, t, \delta, q_{\mathcal{P}}, \ell \in \mathbb{N}$, $\mathcal{P}_m^{\mathsf{H}}$ be a $q_{\mathcal{P}}$-query m-sampler, $\mathsf{Gen} = [K]$ (see assumptions above), $\mathcal{G}$ 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 Definition 5. Then for any q-query, t-time, c-corruption adversary $\mathcal{A}$ in the* UR *game with respect to $\mathcal{P}_m^{\mathsf{H}}, \mathsf{Gen}$ and C^{H}, there exists a P-query, c-corruption adversary $\mathcal{B}$ against* SA-Guess *s.t.*

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

where $\gamma = [(P+1)\mathrm{CC}(\mathcal{G})[\log N - \log((q+q_{\mathcal{P}})m)] - (\mathrm{CMC}(\mathcal{A}) + \Delta)]/t'$, $t' = t + q_{\mathcal{P}}$, and $\Delta = \mathrm{CMC}(\mathcal{P}_m^{\mathsf{H}}) + mt \log N$.

Intuitively, γ captures the gap between the CMC we expect from an adversary computing $(P+1)$ 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. Furthermore, similarly to Lemma 4, if the password sampler is hash-independent (i.e., when $q_{\mathcal{P}} = 0$), the bound is meaningful for any K. Otherwise, K needs to be sufficiently large for salting to be effective.

P is chosen to minimize $\mathbf{Adv}^{\text{SA-Guess}}_{\mathcal{P}_m^{\mathsf{H}},\ell,\mathsf{Gen}}(\mathcal{B}) + 2^{-\gamma} + \frac{(2q_{\mathcal{P}}+nm\ell+4q)^2}{N} + \frac{q_{\mathcal{P}}m\ell}{K}$. Since only the first two terms depend on P, it is sufficient to choose P

to minimize the sum $\mathbf{Adv}^{\text{SA-GUESS}}_{\mathcal{P}^{\mathsf{H}}_m,\ell,\mathsf{Gen}}(\mathcal{B}) + 2^{-\gamma}$. Note that $\mathbf{Adv}^{\text{SA-GUESS}}_{\mathcal{P}^{\mathsf{H}}_m,\ell,\mathsf{Gen}}(\mathcal{B})$ is non-decreasing in P as having more queries can only increase $\mathcal{B}$'s advantage, and $2^{-\gamma}$ is decreasing in P as the larger P is, the larger γ is by definition. The minimum for the sum occurs when $\mathbf{Adv}^{\text{SA-GUESS}}_{\mathcal{P}^{\mathsf{H}}_m,\ell,\mathsf{Gen}}(\mathcal{B})$ becomes the dominating term. Intuitively, this is the point where the adversary can't compute C^{H} with high probability. As we cannot give a closed form for the exact P that minimizes the sum, we instead provide bounds on P, leveraging the fact that $(1/N)^{m-c} \leq \mathbf{Adv}^{\text{SA-GUESS}}_{\mathcal{P}^{\mathsf{H}}_m,\ell,\mathsf{Gen}}(\mathcal{B}) \leq 1$.

We first lower bound P by finding the smallest P s.t. $2^{-\gamma} < 1$. This occurs when $\gamma > 0$, i.e. $((P+1)\text{CC}(\mathcal{G})(\log N - \log((q+q_{\mathcal{P}})m)) - (\text{CMC}(\mathcal{A}) + \Delta)/t' > 0$, specifically

$$P > \frac{(\text{CMC}(\mathcal{A}) + \Delta)}{\text{CC}(\mathcal{G})(\log N - \log((q+q_{\mathcal{P}})m))} - 1 \ .$$

We then upper bound P by finding the largest P such that $2^{-\gamma} > (\frac{1}{N})^{m-c}$. This occurs when $-\gamma > -\log N$ i.e. $(P+1)\text{CC}(\mathcal{G})(\log N - \log((q+q_{\mathcal{P}})m)) - (\text{CMC}(\mathcal{A}) + \Delta) < (m-c)t' \log N$, which in turn rearranges to give

$$P < \frac{(\text{CMC}(\mathcal{A}) + \Delta) + (m-c)t' \log N}{\text{CC}(\mathcal{G})(\log N - \log((q+q_{\mathcal{P}})m))} - 1 \ .$$

Combining the above gives us bounds on the optimal value P of

$$\frac{(\text{CMC}(\mathcal{A}) + \Delta)}{\text{CC}(\mathcal{G})(\log N - \log((q+q_{\mathcal{P}})m))} < P + 1 < \frac{(\text{CMC}(\mathcal{A}) + \Delta) + (m-c)t' \log N}{\text{CC}(\mathcal{G})(\log N - \log((q+q_{\mathcal{P}})m))} \ .$$

Having set the value of P, we now go back to the proof.

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 $\mathcal{P}^{\mathsf{H}}_m, \mathsf{Gen}$ and C^{H}. Thus, $\Pr[\text{G}_0] = \mathbf{Adv}^{\text{UR}}_{\mathcal{P}^{\mathsf{H}}_m,\ell,\mathsf{Gen},\mathsf{C}^{\mathsf{H}}}(\mathcal{A})$.

G_1: Let G_1 be a game where a bad flag, bad_1, is set if $\mathcal{P}^{\mathsf{H}}_m$ queries some x to H which is queried again during the challenge generation phase, after the bad flag is set we then abort. This ensures that the queries of $\mathcal{P}^{\mathsf{H}}_m$ are independent of the queries made during challenge generation. G_0 and G_1 are identical until bad, and an unpredictability argument (see the salt unpredictability claim below) shows that $\Pr[\mathsf{bad}_1] \leq q_{\mathcal{P}} m\ell/K$, where $q_{\mathcal{P}}$ is an upper bound on the number of queries of

the password sampler and ℓ is the number of salts per password: $\Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1] \leq \Pr[\mathsf{bad}_1] \leq \frac{q_{\mathcal{P}} m \ell}{K} + \frac{q_{\mathcal{P}} m \ell}{N}$. This is essentially due to the fact that salts are unpredictable, which in turn implies, in an inductive manner, that hash queries to compute the vertex labels of the graph, which involve hashing the parent labels, are also unpredictable.

G_2: This game lazily samples the random oracle. This game is identical to G_1: $\Pr[\mathrm{G}_2] = \Pr[\mathrm{G}_1]$.

G_3: This game sets a bad flag bad_2 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_2 is set. Games G_2 and G_3 are identical until bad_2. As the outputs are sampled randomly, and the game (that is the adversary and the challenge generation phase) make a total of at most $q + mn$ queries to the oracles, we have $\Pr[\mathrm{G}_2] - \Pr[\mathrm{G}_3] \leq \Pr[\mathsf{bad}_2] \leq \frac{(q_{\mathcal{P}} + nm\ell + 2q)^2}{N}$.

G_4: This game sets bad_3 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_3 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 $\mathsf{C}^{\mathsf{H}}(pw, sa)$ assign 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_3 is set. We show in the challenge point prediction claim below that $\Pr[\mathrm{G}_4] - \Pr[\mathrm{G}_3] \leq \Pr[\mathsf{bad}_3] \leq m\ell q/N$.

G_5: In this game, when the adversary makes a query for the final node, the game uses FINDCHAIN (described in G_4), 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_3 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_2 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 $\Pr[\mathrm{G}_5] = \Pr[\mathrm{G}_4]$.

G_6: 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: $\Pr[\mathrm{G}_6] = \Pr[\mathrm{G}_5]$.

G_7: This game sets a bad flag bad_4 whenever the adversary fully computes C^{H} on more than P unique inputs, and then aborts when bad_4 is set. Games G_7 and G_6 are identical until bad_4. We show, via a reduction to a strong $(q + q_{\mathcal{P}}, P + 1)$-sMH game, in the bounded chain completion claim below that $\Pr[\mathrm{G}_7] - \Pr[\mathrm{G}_6] \leq \Pr[\mathsf{bad}_4] \leq \mathbf{Adv}^{(q+q_{\mathcal{P}},P+1)\text{-sMH}}_{\mathsf{Gen},\mathsf{C}^{\mathsf{H}}}(\mathcal{B})$. We conclude the proof by showing in the G_7 to SA-GUESS reduction claim below that the advantage of any adversary in G_7 is bounded by one in the SA-GUESS.

Claim 4 (Salt unpredictability)**.** *For the parameters that we have above, we have that,*

$$\Pr[\mathsf{bad}_1 \textit{ is set in } \mathrm{G}_2] \leq \frac{q_{\mathcal{P}} m\ell}{K} + \frac{q_{\mathcal{P}} m\ell}{N} \ .$$

Proof. For $\mathcal{P}^{\mathsf{H}}_m$ to make a query that is then repeated during challenge generation it must have been of one of two forms: either it is at the start of the computation, i.e. of the form $(1, pw, sa)$ or it is midway through the computation i.e. (i, x). We first bound the probability of a query of the first

form being made. As the salts are generated after the password sampler produces a password, and the salts are unpredictable, the probability of any of $\mathcal{P}_m^{\mathsf{H}}$'s $q_{\mathcal{P}}$ queries picking one of the $m\ell$ salts that Gen samples afterwards is at most $q_{\mathcal{P}} m\ell / K$. Finally we bound the probability of a query of the second form being made. As a query of the first form has not been made (otherwise bad_1 would have already been set), and therefore the value of $\mathsf{H}(1, pw, sa)$ is unpredictable. Let $q_{\mathcal{P}_i}$ denote the number of queries the password sampler makes to the random oracle with index i. The probability of $\mathcal{P}_m^{\mathsf{H}}$ guessing this value – and therefore being able to make a query $\mathsf{H}(2, x)$ that was made during challenge generation – is at most $q_{\mathcal{P}2} m\ell / N$. This argument follows for each node on the graph, and thus the probability of a query $\mathsf{H}(i, x)$ being made for any $i > 1$ can be upper-bounded by $\sum_{i=2}^{n} q_{\mathcal{P}_i} m\ell / N \leq \frac{q_{\mathcal{P}} m\ell}{N}$. Combining both bounds gives $\Pr[\mathsf{bad}_1 \text{ is set in } \mathrm{G}_1] \leq \frac{q_{\mathcal{P}} m\ell}{K} + \frac{q_{\mathcal{P}} m\ell}{N}$.

□

Claim 5 (Collisions)**.** *For parameters as above, we have*

$$\Pr[\mathcal{A} \textit{ sets } \mathsf{bad}_2 \textit{ in } \mathrm{G}_3] \leq \frac{(q_{\mathcal{P}} + nm\ell + 2q)^2}{N} \ .$$

Proof. There are two events that trigger bad_2, the first is if the response to a (new) query collides with a previous query that has been made, the second is if the response to a (new) query collides with the output of a previous query. The first event only happens if the adversary makes a query relating to some node $i > 1$, without having made the query to the parent nodes. Let q_i represent the number of queries that are made to the oracle indexed with node i. The probability that the response of a query for node i collides with a query made to any of its children is at most $\frac{1}{N}(q_i \sum_j q_{\mathbf{Ch}_j(i)})$. Taking a union bound over i, we get the bound $\frac{1}{N} \sum_{i=1}^{n} \left(q_i \sum_{j=1} q_{\mathbf{Ch}_j(i)} \right)$. Note that $\forall i, \sum_j q_{\mathbf{Ch}_j(i)} \leq \sum_{i=1}^{n} q_i \leq q$. This gives

$$\frac{1}{N} \sum_{i=1}^{n} \left(q_i \sum_j q_{\mathbf{Ch}_j(i)} \right) \leq \frac{1}{N} \sum_{i=1}^{n} q \cdot q_i = \frac{q}{N} \sum_{i=1}^{n} q_i \leq \frac{q^2}{N}.$$

To bound the second event, we consider the total number of queries that are made. Between the password sampler, the challenge generation phase and the adversary there are a total of $q_{\mathcal{P}} + nm\ell + q$ queries. A standard collision argument upperbounds the probability of any of these queries

colliding (on distinct inputs) as $\frac{(q_{\mathcal{P}}+nm\ell+q)^2}{N}$.
Thus

$$\Pr[\mathcal{A} \text{ sets } \mathsf{bad}_2 \text{ in } \mathrm{G}_3] \leq \frac{(q_{\mathcal{P}}+nm\ell+q)^2+q^2}{N} \leq \frac{(q_{\mathcal{P}}+nm\ell+2q)^2}{N} \ .$$

□

Claim 6 (Challenge point prediction)**.** *For the parameters that we have above, we have that,*

$$\Pr[\mathsf{bad}_3 \textit{ is set in } \mathrm{G}_4] \leq \frac{m\ell q}{N} \ .$$

Proof. For each node $i \in [1, n]$, the challenge generation phase creates $m\ell$ labels. As there are no collisions in the queries, each label is unique and sampled uniformly at random. Consider the labels for the node with index $n-1$. In the online phase, suppose $\mathcal{A}$ does not compute the construction from an input to node $n-1$, but still makes a query containing the label for this node with respect to one of the $m\ell$ inputs. As there are no collisions in $\mathcal{A}$'s queries, $\mathcal{A}$ either obtained the label by making a query with the labels for the *parents* of node $n-1$, with respect to the same input, or guessed the output of an unqueried point. The probability $\mathcal{A}$ can guess the label for the unqueried point i (with respect to any of the $m\ell$ inputs) is at most $q_i m\ell/N$. By induction over the node parents, we have $\Pr[\mathsf{bad} \text{ is set in } \mathrm{G}_4] \leq \sum_{i=1}^{n} q_i m\ell/N \leq \frac{m\ell q}{N}$. □

Claim 7 (Bounded chain completion)**.** *For the parameters that we have above:*

$$\Pr[\mathcal{A} \textit{ sets } \mathsf{bad}_3 \textit{ in } \mathrm{G}_7] \leq \mathbf{Adv}_{\mathsf{C}^{\mathsf{H}}}^{(q+q_{\mathcal{P}},P+1)\text{-sMH}}(\mathcal{B}) \leq 2^{-\gamma} + \frac{2q^2}{N} \ ,$$

where $\gamma := ((P+1) \cdot \mathrm{CC}(\mathcal{G})[\log(N) - \log((q+q_{\mathcal{P}})m] - \mathrm{CMC}(\mathcal{B}))/t'$*, we have* $t' = t + q_{\mathcal{P}}$*, and* $\mathrm{CMC}(\mathcal{B}) = \mathrm{CMC}(\mathcal{A}) + q_{\mathcal{P}} + mt\log(N)$*.*

Intuitively an adversary that triggers bad_4, and so fully computes the construction on at least $(P+1)$ inputs, can win a $(P+1)$-instance memory hardness game. We give a reduction to an adversary in the $(q+q_{\mathcal{P}}, P+1)$-sMH game (defined in Section 4.1) and bound the advantage using Theorem 2.

Proof. Given an adversary $\mathcal{A}$ as defined in the theorem statement, we construct a $(q+q_{\mathcal{P}})$-query adversary $\mathcal{B}$ against $(q+q_{\mathcal{P}}, P+1)$-sMH, where

$\mathrm{CMC}(\mathcal{B}) = \mathrm{CMC}(\mathcal{A}) + q_{\mathcal{P}} + mt\log(N)$. Algorithm $\mathcal{B}$, with access to H, samples a random password vector and leakage z from $\mathcal{P}_m^{\mathsf{H}}$, 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., any 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), algorithm $\mathcal{B}$ 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, algorithm $\mathcal{B}$ returns $\sigma_{\mathcal{O}}$, which contains at most q values.

In order 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_4, then there exists a set of $(P+1)$ correct input-output pairs in $\sigma_{\mathcal{O}}$, and $\mathcal{B}$ wins the $(q + q_{\mathcal{P}}, P+1)$-sMH game. □

Claim 8 (G_7 to sa-Guess reduction)**.** *Let P, q, n, m, c be integers, $\mathcal{G}$ be a graph on n vertices with maximum in-degree δ, and C^{H} be the iMHF based on the graph $\mathcal{G}$ as defined in Definition 5 in the $(nN^{\delta}K, N)$-RO model. Then for any m-sampler $\mathcal{P}_m^{\mathsf{H}}$ that makes at most $q_{\mathcal{P}}$ random oracle queries, any unpredictable salt generator Gen, and any q-query, c-corruption adversary $\mathcal{A}$ against G_7, there exists a P-query, c-corruption adversary $\mathcal{B}$ against* sa-Guess *such that*

$$\mathbf{Adv}^{\mathrm{G}_7}_{\mathcal{P}_m^{\mathsf{H}},\ell,\mathsf{Gen},\mathsf{C}^{\mathsf{H}}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{sa-Guess}}_{\mathcal{P}_m^{\mathsf{H}},\ell,\mathsf{Gen}}(\mathcal{B}) \ .$$

Proof. Given an adversary $\mathcal{A}$ as in the theorem statement in G_7, 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 $\mathsf{C}^{\mathsf{H}}(x)$.[1] $\mathcal{B}$ receives the salt vector $\mathbf{sa}$, leakage z, and generates a challenge vector $\mathbf{y}$ following the rules of G_7. If $\mathbf{sa}$ contains a salt sa s.t. $\mathcal{P}_m^{\mathsf{H}}$ had made the query $(*, sa)$, then the oracle H' is used (according to the rules corresponding

[1] For any $\mathcal{A}$ that returns x without computing $\mathsf{C}^{\mathsf{H}}(x)$ we can create an $\mathcal{A}'$ which runs $\mathcal{A}$ and computes $\mathsf{C}^{\mathsf{H}}(x)$ before returning x, and where $\mathrm{CMC}(\mathcal{A}') = \mathrm{CMC}(\mathcal{A}) + m' \cdot \mathrm{dCMC}_{q,1}(\mathsf{C}^{\mathsf{H}})$. We assume, without loss of generality, that m' is less than the number of guesses made by $\mathcal{A}$ - as otherwise it would be impossible for $\mathcal{A}$ to ever win, no matter how low entropy the password distribution, therefore $\mathrm{CMC}(\mathcal{A}') = \mathcal{O}(\mathrm{CMC}(\mathcal{A}))$, where m' is the number of passwords recovered.

to bad_1 in G_1, carried through to G_7). 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_2 of G_3. 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_4). 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_3 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 $\mathcal{B}$. 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

$$\begin{aligned}
\Pr[\mathrm{G}_0] &= \Pr[\mathrm{G}_7] + \sum_{i=0}^{6} (\Pr[\mathrm{G}_i] - \Pr[\mathrm{G}_{i+1}]) \\
&\leq \mathbf{Adv}^{\text{SA-GUESS}}_{\mathcal{P}^{\mathsf{H}}_m,\ell,\mathsf{Gen}}(P, c) + \frac{2q^2}{N} + 2^{-\gamma} + \frac{m\ell q}{N} + \frac{(q_{\mathcal{P}} + nm\ell + 2q)^2}{N} + \frac{q_{\mathcal{P}} m\ell}{K} + \frac{q_{\mathcal{P}} m\ell}{N} \\
&\leq \mathbf{Adv}^{\text{SA-GUESS}}_{\mathcal{P}^{\mathsf{H}}_m,\ell,\mathsf{Gen}}(P, c) + 2^{-\gamma} + \frac{(2q_{\mathcal{P}} + nm\ell + 4q)^2}{N} + \frac{q_{\mathcal{P}} m\ell}{K} \ ,
\end{aligned}$$

where the last inequality is as a result of

$$(2q_{\mathcal{P}} + nm\ell + 4q)^2 \geq 2q^2 + m\ell q + q_{\mathcal{P}} m\ell + (q_{\mathcal{P}} + nm\ell + 2q)^2 \ ,$$

and $\gamma := ((P+1) \cdot \mathrm{CC}(\mathcal{G})[\log N - \log((q + q_{\mathcal{P}})m)] - (\mathrm{CMC}(\mathcal{A}) + \Delta))/t'$, and $\Delta = \mathrm{CMC}(\mathcal{P}^{\mathsf{H}}_m) + mt \log N$, and $t' = t + q_{\mathcal{P}}$. □

5.2.1 Interpretation

To aid the interpretation, consider a setting without corruptions, i.e., $c = 0$. Then, the dominating term in the bound is $\mathbf{Adv}^{\text{SA-GUESS}}_{\mathcal{P}^{\mathsf{H}}_m,\ell,\mathsf{Gen}}(P)$. As above, we set $P = (\text{CMC}(\mathcal{A}) + \Delta)/\text{CC}(\mathcal{G})(\log N/(q + q_{\mathcal{P}})m)$, which gives $2^{-\gamma} = ((q + q_{\mathcal{P}})m/N)^{\text{CC}(\mathcal{G})/t'}$. Assuming a uniform salt generator, and applying Lemma 3, we can bound the dominating term by $\mathbf{Adv}^{\text{GUESS}}_{\mathcal{P}^{\mathsf{H}}_m}(P) + m^2\ell^2/K$. Applying Lemma 2 gives $\binom{P}{m}\mathbf{Adv}^{\text{GUESS}}_{\mathcal{P}^{\mathsf{H}}_m}(m)$. For a password sampler $\mathcal{P}^{\mathsf{H}}$ 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

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

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.

We now consider this bound in terms of graph-based iMHFs from the literature, specifically d-iterate hashing, Argon2i-B [13], Catena [32], and general Balloon hashing [20]. For each construction, we consider the multi-instance security case where m passwords are sampled uniformly from a dictionary of size D, and the adversary is required to recover all m passwords.

We start with d-iterate hashing, i.e. the iMHF defined from the path graph of length d (denoted P_d). Whilst it is clear that d-iterate hashing is not memory-hard, there are some notable iteration-based constructions, e.g., PBKDF2 [50] and T/Key [39]. Noting that iteration has been shown to be amplifying for KDF security [10] (where the advantage of an attack decreases linearly with the number of iterations), we now confirm this for

unrecoverability. Note that $\mathrm{CC}(P_d) = d$, hence:

$$\mathbf{Adv}^{\mathrm{UR}}_{\mathcal{P}^{\mathsf{H}},\ell,\mathsf{Gen},P_d}(\mathcal{A}) \leq \tilde{\mathcal{O}}\left(\left(\frac{\mathrm{CMC}(\mathcal{A})}{dmD}\right)^m\right) .$$

Argon2 is a family of MHFs that won the Password Hashing Competition [48], we consider Argon2i, the graph-based iMHF version of Argon2. The tightest bounds on the cumulative complexity of the underlying graph are that of [19], giving lower and upper bounds of $\tilde{\Omega}(n^{1.75})$ and $\mathcal{O}(n^{1.767})$, where n is the number of nodes in the underlying graph. Catena, whilst not winning the Password Hashing Competition, did still receive a special mention. The graphs for Catena come in two flavors: butterfly and dragonfly. In [3], the cumulative complexity of these graphs has been bounded from below as $\tilde{\Omega}(n^{1.5})$ (for both) and from above as $\tilde{\mathcal{O}}(n^{1.625})$ (for Dragonfly) and $o(n^{1.625})$ (for Butterfly). Balloon Hashing is an iMHF based on random sandwich graphs. In [3] the cumulative complexity of these graphs has been bounded from below as $\tilde{\Omega}(n^{1.5})$ and above as $\tilde{\mathcal{O}}(n^{1.625})$.

In Table 5.1 we present upper and lower bounds based on the cumulative complexity of the graphs for each of the iMHFs, where for each construction we consider $\mathbf{Adv}^{\mathrm{UR}}_{\mathcal{P}^{\mathsf{H}},\ell,\mathsf{Gen},\mathsf{C}^{\mathsf{H}}}(\mathcal{A})$.

Numerical interpretation. One can see from the above table that a successful adversary needs to spend cumulative memory complexity equivalent to the construction's dCMC, multiplied by the size of the dictionary.

Let us assume a dictionary of ten billion (or approximately 2^{33}) passwords. This is roughly the size of modern leaked password banks [54].

Meta's quarterly reports [43] show roughly three billion users. Assuming a large scale attack on Facebook users, the bounds show that the adversary would need roughly 2^{60} times the CMC of a single computation to have a good advantage in the unrecoverability game.

Note that if the MHF is Argon2i, the OWASP Application Security Verification Standard (ASVS) [47] recommends a single round and a single degree of parallelism, with the fifty thousand nodes (though higher is better). Assuming that the application uses two rounds of one-hundred thousand nodes instead, (as this would mean the memory would be overwritten), we can substitute this value into the lower bound, to obtain roughly $n^{1.75} \approx 2^{30}$.

These rough number help to intuit the scale of the required attack, but the important result is asymptotic, i.e., that we can reduce the advantage in the UR game to password guessing advantage even when using MHFs in an adaptive setting.

iMHF	Lower bound	Upper bound
Argon2i	$\tilde{\Omega}\left(\left(\frac{\mathrm{CMC}(\mathcal{A})}{n^{1.767}mD}\right)^m\right)$	$\tilde{\mathcal{O}}\left(\left(\frac{\mathrm{CMC}(\mathcal{A})}{n^{1.75}mD}\right)^m\right)$
Catena - butterfly	$\tilde{\Omega}\left(\left(\frac{\mathrm{CMC}(\mathcal{A})}{n^{1.625}mD}\right)^m\right)$	$\tilde{\mathcal{O}}\left(\left(\frac{\mathrm{CMC}(\mathcal{A})}{n^{1.5}mD}\right)^m\right)$
Catena - dragonfly	$\tilde{\omega}\left(\left(\frac{\mathrm{CMC}(\mathcal{A})}{n^{1.625}mD}\right)^m\right)$	$\tilde{\mathcal{O}}\left(\left(\frac{\mathrm{CMC}(\mathcal{A})}{n^{1.5}mD}\right)^m\right)$
Balloon	$\tilde{\Omega}\left(\left(\frac{\mathrm{CMC}(\mathcal{A})}{n^{1.625}mD}\right)^m\right)$	$\tilde{\mathcal{O}}\left(\left(\frac{\mathrm{CMC}(\mathcal{A})}{n^{1.5}mD}\right)^m\right)$

Table 5.1: Bounds on the unrecoverability advantage for iMHF constructions

Game $\mathrm{CR}^{\mathcal{A}}_{\mathsf{C}^{\mathsf{H}}}$
$\mathsf{H} \twoheadleftarrow \mathrm{Fun}(N, M)$
$(x, y) \twoheadleftarrow \mathcal{A}^{\mathsf{H}}(\ )$
return $(\mathsf{C}^{\mathsf{H}}(x) = \mathsf{C}^{\mathsf{H}}(y)) \wedge (x \neq y)$

Figure 5.1: The CR game with respect to a construction C^{H}.

5.3 One-wayness

Password storage security can also be analyzed with respect to standard one-wayness security, where the adversary's goal is to recover *any* input as long as it produces the same output when evaluated by the construction as that of the actual password. In a single instance setting, moving between unrecoverability and one-wayness can be achieved in a straightforward manner up to the collision resistance of the construction. This is sufficient for practical security bounds since passwords are typically low entropy and the extra collision probability is dominated by the unguessability of the passwords. Moreover, passwords tend to come from small dictionaries that effectively render the hash function injective.

In this section, we give a corollary bounding the multi-instance one-wayness security of the constructions considered in the thesis. We do this by bounding the probability that an adversary can win on *even one* instance without recovering the passwords. To do this, we will make use of the following collision resistance game.

5.3.1 Collision resistance

Below we define the collision resistance (CR) game, which we make use of in Corollary 3. In the CR game, with respect to a construction C^{H}, an adversary $\mathcal{A}$ with oracle access to H must output a pair of distinct points (x, y) such that $\mathsf{C}^{\mathsf{H}}(x) = \mathsf{C}^{\mathsf{H}}(y)$. See Fig. 5.1 for the pseudo code. Formally, for a q-query adversary $\mathcal{A}$, and a hash based construction C^{H}, we define

$$\mathbf{Adv}^{\mathrm{CR}}_{\mathsf{C}^{\mathsf{H}}}(\mathcal{A}) := \Pr\left[\mathrm{CR}^{\mathcal{A}}_{\mathsf{C}^{\mathsf{H}}}\right] .$$

We now upper bound the advantage of a q-query adversary in the CR game w.r.t. a construction C^{H} from Definition 5, using a simple collision argument.

Lemma 7 (Collision resistance of graph-based iMHFs)**.** *Let q, n, and δ be integers, such that $q > n$, let $\mathcal{G}$ be a graph on n vertices with maximum in-degree δ, and C^{H} be the iMHF based on the graph $\mathcal{G}$ in the $(nN^{\delta}K, N)$-RO model as defined in Definition 5. Then for any q-query adversary $\mathcal{A}$ in the* CR *game with respect to C^{H}*

$$\mathbf{Adv}^{\mathrm{CR}}_{\mathsf{C}^{\mathsf{H}}}(\mathcal{A}) \leq \frac{(q+2n)^2}{N} .$$

Proof. A collision in the output of the construction means there must exist a collision in the underlying primitive. Therefore, if $\mathcal{A}$ wins there must exist two distinct points a_1 and a_2 in the table of the random oracle H such that $\mathsf{H}(a_1) = \mathsf{H}(a_2)$. Moreover, for some node v in $\mathcal{G}$, both a_1 and a_2 must begin with the index of v. Algorithm $\mathcal{A}$ makes at most q queries to H, fixing at most q points in the table. Recall that $\mathcal{A}$ outputs a final pair of inputs (x, y), possibly independent of its queries. The final pair (x, y) defines two labelings of $\mathcal{G}$, the (H, x)-labeling and (H, y)-labeling respectively. Together these labelings fix at most another $2n$ points in the table of H during the final verification step in the CR game when $\mathsf{C}^{\mathsf{H}}(x)$ and $\mathsf{C}^{\mathsf{H}}(y)$ are computed.

For each node v of the graph $\mathcal{G}$, let q_v be the number of queries made to H which begin with index v (including queries made by algorithm $\mathcal{A}$ and those in the final verification step of the game). Then the probability that $\mathcal{A}$ finds a collision in the labels of a node in the graph is at most

$$\frac{1}{N}\sum_{i=1}^{n} q_i^2 \leq \frac{1}{N}\left(\sum_{i=1}^{n} q_i\right)^2 \leq \frac{(q+2n)^2}{N} .$$

□

Note that a comparable bound is given for the Catena MHF in [32].

With this bound in hand, we now state and prove our corollary for OW security.

Corollary 3 (OW security of C^{H})**.** *Let all parameters be set as in Theorem 4. Let $q_{\mathcal{P}} + (m\ell + 2)n \leq q$. Then, for any q-query parallel* OW *adversary $\mathcal{A}$, there is a q-query parallel* UR *adversary $\mathcal{B}$ with a similar CMC s.t.*

$$\mathbf{Adv}^{\mathrm{OW}}_{\mathcal{P}^{\mathsf{H}}_m,\ell,\mathsf{Gen},\mathsf{C}^{\mathsf{H}}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{UR}}_{\mathcal{P}^{\mathsf{H}}_m,\ell,\mathsf{Gen},\mathsf{C}^{\mathsf{H}}}(\mathcal{B}) + \frac{4q^2}{N} \ .$$

Informally, we transition from the OW game to the UR game by setting a bad flag if the adversary wins *without* recovering even one of the passwords. We bound the probability bad via a reduction to a $(q + q_{\mathcal{P}} + m\ell n)$-query adversary in the collision resistance (CR) game (formally defined in Section 5.3.1). The reduction keeps track of the queries used to simulate $\mathcal{P}^{\mathsf{H}}_m$ and generate a challenge, and the analysis keeps track of the collisions in the $m\ell$ instances.

Proof. Let G_0 be the OW game with respect to the parameters in the theorem statement, and let $\mathcal{A}$ be an adversary in G_0. Let G_1 be a game where a bad flag is set when $\mathcal{A}$ wins, but there is at least one password which isn't recovered. That is, for all pairs $(i, j) \in [m] \times [\ell]$, it holds that $\mathsf{C}^{\mathsf{H}}(\mathbf{pw}[i], \mathbf{sa}[i, j]) = \mathsf{C}^{\mathsf{H}}(\bar{\mathbf{pw}}[i], \mathbf{sa}[i, j])$, but there is some $k \in [m]$ s.t. $\mathbf{pw}[k] \neq \bar{\mathbf{pw}}[k]$. Then $\Pr[\mathrm{G}_0] = \Pr[\mathrm{G}_1]$. Let G_2 be a modified game s.t. if bad is set, $\mathcal{A}$ loses and the game returns 0. Therefore G_1 and G_2 are identical until bad and $\Pr[\mathrm{G}_2] - \Pr[\mathrm{G}_1] \leq \Pr[\mathsf{bad} \text{ is set in } \mathrm{G}_2]$.

To bound the probability bad is set, we give a reduction from G_2 to the collision resistance game CR with respect to the same construction C^{H}. Let $\mathcal{A}$ be an adversary in G_2, we build an algorithm $\mathcal{C}$ in CR. Algorithm $\mathcal{C}$ samples vectors $\mathbf{pw}$ and $\mathbf{sa}$ according to the password sampler $\mathcal{P}^{\mathsf{H}}_m$ and Gen in G_2, which requires at most $q_{\mathcal{P}}$ queries to H. Algorithm $\mathcal{C}$ computes the challenge vector $\mathbf{y} = \mathsf{C}^{\mathsf{H}}(\mathbf{pw}, \mathbf{sa})$, with at most $m\ell n$ queries to H. Algorithm $\mathcal{C}$ then runs $\mathcal{A}$ on input $(\mathbf{y}, \mathbf{sa}, z)$, relaying $\mathcal{A}$'s q oracle queries to H. (Here $\mathcal{C}$ receives the queries as batches, but relays them sequentially, storing responses to return as a batch.) When $\mathcal{A}$ returns $\bar{\mathbf{pw}}$ algorithm $\mathcal{C}$ compares $\mathbf{pw}$ and $\bar{\mathbf{pw}}$ for any pair $(k, l) \in [m] \times [\ell]$ such that $\mathbf{pw}[k] \neq \bar{\mathbf{pw}}[k]$, to return $((\mathbf{pw}[k], \mathbf{sa}[k, l]), (\bar{\mathbf{pw}}[k], \mathbf{sa}[k, l]))$ as a candidate collision. If there is no such (k, l) then $\mathcal{C}$ aborts. Whenever bad is set in G_2, algorithm $\mathcal{C}$ makes at most $(q + q_{\mathcal{P}} + m\ell n)$ queries, and wins the CR game. By Lemma 7 in Section 5.3.1 we have:

$$\mathbf{Adv}^{\mathrm{CR}}_{\mathsf{C}^{\mathsf{H}}}(\mathcal{C}) \leq \frac{(q + q_{\mathcal{P}} + (m\ell + 2)n)^2}{N} \leq \frac{4q^2}{N} \ .$$

□

Previous collision resistance results. Previous work has argued for collision resistance of individual graph-based memory hard functions. The creators of Argon2id [14] argued that the construction is collision resistant based on some assumptions on the underlying compression function. Using a pseudorandom function P, they build the following compression function

$$G(X, Y) := P(Z) \oplus Z, \text{ where } Z = X \oplus Y.$$

For X and Y which are 1024-byte blocks. They argue that if $P(Z) \oplus Z$ is collision resistant, and 4-generalised collision resistant, (i.e., it is hard to find distinct a, b, c, and d such that $a \oplus b \oplus c \oplus d = P(a) \oplus P(b) \oplus P(c) \oplus P(d)$), then G is collision resistant. However they do not derive a bound in the collision resistant game.

The creators of Catena [33] also argue for collision resistance. Their results are limited to specific graph families used for the Catena framework, but they are asymptotically the same as the bounds derived above.

Finally, Boneh et. al. argue that their Balloon hash [20] construction can be made collision resistant by composing it with a collision resistant hash function. They do not derive a bound.

The result in this thesis contributes by covering a wider class of graphs that contains the above graph families, and by producing a bound via a cryptographic proof. Additionally, due to the modular approach, the bound can be lifted to a one-wayness result using the game hopping methodology in Corollary 3.

5.4 Conclusion

In this chapter, we derived security bounds for the UR game. Importantly, the technique enables the bounds to be given in terms of the CC of the underlying graph, the cumulative memory complexity of the adversary, and if necessary, the unguessability of the underlying password distribution. Along the way, we make the important contribution of accounting for passwords which depend on the hash. This is only achievable using the

results from Chapters 3 and 4, which provide the games and techniques for reductions to sa-Guess and sMH respectively.

In doing this we verify that the non-amortizability of graph-based memory hard functions not only translates to more adaptive adversaries on the sMH game, but to the advantage in the UR game too. That is, the adversarial effort scales with the number of instances and the CC of the underlying graph as desired.

While it seems intuitive that the security in the OW game should be the measure of security for password storage models, it is not necessarily the case. Although finding a collision for a password is in theory sufficient to gain unauthorized access, it is debatable to what extent this should count as cracking a password. A large economic motivator for password cracking is their sale on the black market [15]. The value of these passwords comes, in large part, from their reuse, and so the value of a collision on a password is not so clear. Additionally, the input field for the password-based log in system needs to accept the collision as a valid format. In the final part of this chapter, we resolve the issue with a proof in the up-to-date security models. We formalize the idea that the OW security is only a negligible collision bound away from the security in the UR game.

6 Multi-stage Indifferentiability

6.1 Background

We have provided bounds for the UR game that captures password storage, but memory-hard functions (MHFs) have broader applications. Wherever we want to prevent large-scale guessing attacks, we may want to use an MHF in place of the random oracle. As mentioned in the introduction, ideally we don't want to reprove security for every new application. Instead, we would like to derive indifferentiability bounds that guarantee security for an entire *class* of games. However, many applications are inherently multi-stage. As we have argued, any application that uses passwords should be modelled with a hash-dependent password sampler. This requirement alone means that correctly modelling password-based security necessitates multi-stage games that fall outside the scope of single-stage indifferentiability.

Furthermore, many large-scale attacks on passwords occur offline using techniques such as time-memory trade-offs [38, 46]. The UR game we define in the earlier sections does not account for these preprocessing attacks. The standard approach to model such attacks for random oracles is to use the auxiliary-input random oracle model (AI-ROM). In this model, an adversary is given offline access to the random oracle and can pass a preprocessing string to an online adversary. The security bounds one obtains are parameterised by both the online query complexity T, and the size of the auxiliary information S. We use the techniques of indifferentiability games to prove the suitability of a large class of graph-based functions, in a wide reaching class of games.

Related Work on Indifferentiability. The single-stage indifferentiability of iMHFs was established in [7], which introduced a composable definition for moderately hard functions. This work presented a framework that allows one to prove indifferentiability with an awareness of adversarial cumulative memory complexity (CMC). However, this framework was limited to single-stage games. Consequently, the unrecoverability bounds in the previous section do not simply follow from the indifferentiability result in [7].

Multi-stage indifferentiability has been explored in related contexts. In [45], Mittelbach examined multi-stage indifferentiability for a class of hash chain constructions, but this predated the Password Hashing Competition (PHC) and therefore did not account for the wide class of graph-based MHFs that concern us. Specifically, the hash-chain definition was restricted to graphs of indegree two, preventing direct application of the results to modern MHFs. Additionally, Mittelbach's multi-stage games did not account for preprocessing adversaries.

More recently, [23] studied indifferentiability in the presence of preprocessing, but this work restricted considerations to 'perfectly' indifferentiable constructions—those indifferentiable from a random oracle even to an unbounded adversary. While the authors showed this applied to the single-round sponge function, no such results exist for the graph-based MHF constructions we consider.

Overview. In this chapter, we study the indifferentiability of graph-based MHFs in a wide class of multi-stage games that account for both hash-dependent sampling and offline access to the random oracle. The class of MHFs we consider, based on the definitions established in [6], is not compatible with those studied in [44], and the class of games we consider is strictly wider than the single-stage games in [7]. Importantly, our results do not require the constructions to be perfectly indifferentiable, as in [23]. There are immediate definitional and technical challenges to proving security in such a general setting. While the result of this chapter will be to give a positive security result for large classes of graphs and games, it will also demonstrate some natural limitations of this approach.

6.2 Additional preliminaries

As the techniques and frameworks of this chapter are different from the others, there are some additional things to recall before moving onto the proofs.

6.2.1 Basic security definitions

Salted Construction. We consider hash-based constructions that take as input a tuple (sa, x), where sa is a salt and x is some value. We call these *salted constructions*. Typical salted constructions prepend the salt to the input of the first invocation of the underlying hash, but it may be used for every invocation or for a subset thereof.

Dealing with preprocessing. The auxiliary input random oracle model (AI-ROM) models preprocessing attacks with two stages. The offline phase grants a preprocessing adversary, which we denote $\mathcal{D}_0$, access to the full random oracle H drawn from $\mathrm{Fun}(N, M)$. The adversary $\mathcal{D}_0$ is computationally unbounded and can generate an S-bit preprocessing string z_0, which is passed to the online adversary $\mathcal{D}_1$. The online adversary $\mathcal{D}_1$ is computationally bounded and has oracle access to H.

Definition 4 (Adversary in the AI-ROM)**.** *Adversaries in the AI-ROM are two stage. An adversary $\mathcal{D} = (\mathcal{D}_0, \mathcal{D}_1)$ runs $\mathcal{D}_0$ and $\mathcal{D}_1$ in sequence.*

- *The offline adversary $\mathcal{D}_0$ is given* unbounded *offline access to the random oracle H, or equivalently, gets access to the full table of the random oracle H, which we denote $\mathcal{D}_0(\mathsf{H})$.*
- *The offline adversary $\mathcal{D}_0$ outputs an S-*bit *preprocessing string z_0, encoding information of its choice.*
- *The online adversary gets access z_0, and a* bounded *number of queries to the oracle H.*

Importantly, both the size of the preprocessing string, and the number of online queries are parameters in any bounds appearing in this model, where the security measures indistinguishability or unrecoverability of the random oracle.

The bit-fixing random oracle model (BF-ROM) models the preprocessing information as a list L of P pre-fixed points $(x, y) \in (N, M)$ which respect the definition of a function. In this model, rather than parties having access to the random oracle H given to $\mathcal{D}_0$, each party has access to a different random oracle H' conditioned on a list L, which we denote $\mathsf{H}'[L]$. The list L is extracted from $\mathcal{D}_0$'s output using a decompression algorithm Decomp. As the offline adversary $\mathcal{D}_0$ doesn't get access to H', the list L is independent of H'.

Now we recall a theorem from [26] which, roughly, bounds the difference between the AI-ROM and the BF-ROM. This means that one can bound advantage in the easier-to-analyse BF-ROM setting, and derive a bound in the AI-ROM with an additive factor to the bound.

Theorem 5 ([26])**.** *Let $N, M, S \in \mathbb{N}$, H be a random oracle drawn from $\mathrm{Fun}(N, M)$, and let $\mathcal{D}_0$ be an algorithm taking H as input and returning an S-bit string. Then there exists an algorithm $\mathsf{Decomp}_{\mathcal{D}_0}$ with the following*

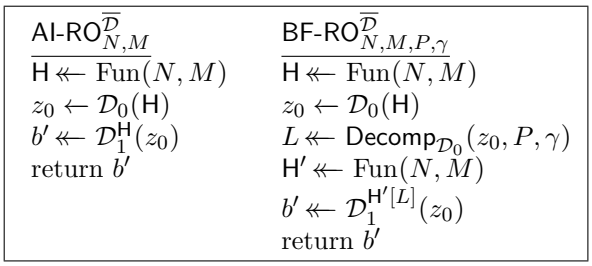

Figure 6.1: The auxiliary input random oracle game (left) and the bit fixing counterpart (right).

property: for all $P \in \mathbb{N}$, $\gamma > 0$ and for any distinguisher $\mathcal{D}_1$ making at most T queries to its oracle, we have

$$\left|\Pr\left[\mathsf{AI\text{-}RO}^{\overline{\mathcal{D}}}_{N,M}\right] - \Pr\left[\mathsf{BF\text{-}RO}^{\overline{\mathcal{D}}}_{N,M,P,\gamma}\right]\right| \leq \frac{2(S - \log \gamma^{-1})T}{P} + 2\gamma \ ,$$

where $\overline{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1)$, and the games $\mathsf{AI\text{-}RO}^{\overline{\mathcal{D}}}_{N,M}$ and $\mathsf{BF\text{-}RO}^{\overline{\mathcal{D}}}_{N,M,P,\gamma}$ are defined in figure Fig. 6.1.

6.2.2 The graph-based iMHFs in this chapter

Here, we must recall the graph-based MHF definitions from previous chapters, and make one alteration. Namely, we will consider graphs with a single source and a single sink. Thus the functions they define will be salted from a space of a suitable size to 'fill up' the rest of the input to H. This means, the size of the salt space is essentially set by the indegree of the graph. We will restate the definition below.

Let $\mathcal{G} = (V, E)$ be a directed acyclic graph (DAG), where the node set V is the set of integers 1 to n. 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 jth parent (resp. child) ordered by node index. Let δ_i be in-degree of the ith node, and let $\delta := \max_i\{\delta_i\}$. In this paper, we consider iMHFs which use the labeling scheme from AS15 [5]. We reproduce the definition of that labeling here, and extend it to allow for salting. While the initial label l in AS15 is drawn from a label set L, to allow salting for a salt set of size K, we extend the initial label l to be drawn from $[K]^+ \times L$, where $[K]^+ := [K] \cup \{\varepsilon\}$, i.e., l would take the form (sa, pw), with $sa = \varepsilon$ accounting for no salting used.

Definition 5 ((H, l)-labeling of graphs [5])**.** *Let $\mathcal{G} = (V, E)$ be a DAG with maximum in-degree δ, $[K]^+$ be a salt set, L be an arbitrary label set, and $\mathbb{H} := \mathrm{Fun}(V \times [K]^+ \times L^\delta, L)$. For a function $\mathsf{H} \in \mathbb{H}$ and an initial label $l \in [K]^+ \times L$, the (H, l)-labeling of $\mathcal{G}$ is a mapping $\mathbf{lab} : V \mapsto L$ defined recursively for all v as:*

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

We assume that $\varepsilon \in L$, and where there are fewer than $\delta + 2$ inputs to H, we pad the inputs with ε. For brevity, we use $\mathbf{pre\text{-}lab}(v)$ to refer to the H input which defines the label of node v, 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 a salt is only included when $\mathrm{indeg}(v) = 0$.

Using the labelling of a graph, we can describe graph-based iMHFs in the random oracle model. A construction C is formed from a graph $\mathcal{G}$ using the labelling from Definition 5, and when instantiated with the random oracle H, it forms a construction we denote as C^{H}. Computing C^{H} on $l = (sa, x)$, denoted as $\mathsf{C}^{\mathsf{H}}(l)$, will give an (H, l)-labelling of $\mathcal{G}$. We recall the formal definition below.

Definition 6 (Graph-based iMHF [5])**.** *Let $\mathcal{G} = (V, E)$ be a DAG with maximum in-degree δ and sink nodes $\{v_1, \ldots, v_z\}$ for some positive integer z. Let L be an arbitrary label set, $[K]^+$ be a salt set, and $\mathbb{H} = \mathrm{Fun}(V \times [K]^+ \times L^\delta, L)$. 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 $[K]^+ \times L$ to L^z. For input $l \in [K]^+ \times L$, the value of $\mathsf{C}^{\mathsf{H}} \in \mathsf{C}$ is defined as $\mathsf{C}^{\mathsf{H}}(l) := (\mathbf{lab}(v_1), \ldots, \mathbf{lab}(v_z))$, where v_i are arranged in lexicographic order and $\mathbf{lab}$ is the (H, l)-labeling of $\mathcal{G}$.*

In this thesis we are primarily considering the labelling set $L := [N]$. In this chapter, we limit our considerations to DAGs with a single sink *and* a single source.

6.3 Preprocessing and Indifferentiability

Indifferentiability. We start by recalling the standard indifferentiability notions, starting with the Real and Ideal games, presented in Fig. 6.2. In the Real-Indiff game, first the random oracle H is sampled, and then the distinguisher $\mathcal{D}$, with oracle access to the construction C^{H} and primitive H,

$\underline{\textsc{Real-Indiff}^{\mathcal{D}}_{\mathsf{C}^{\mathsf{H}}}}$	$\underline{\textsc{Ideal-Indiff}^{\mathcal{D},\mathcal{S}}_{\mathsf{RO}}}$
$\mathsf{H} \twoheadleftarrow \mathrm{Fun}(N, M)$	$\mathsf{RO} \twoheadleftarrow \mathrm{Fun}(N', M')$
$b \twoheadleftarrow \mathcal{D}^{\mathrm{Const},\mathrm{Prim}}()$	$b \twoheadleftarrow \mathcal{D}^{\mathrm{Const},\mathrm{Prim}}()$
$\underline{\mathrm{Const}(x)}$	$\underline{\mathrm{Const}(x)}$
return $\mathsf{C}^{\mathsf{H}}(x)$	return $\mathsf{RO}(x)$
$\underline{\mathrm{Prim}(x)}$	$\underline{\mathrm{Prim}(x)}$
return $\mathsf{H}(x)$	return $\mathcal{S}^{\mathsf{RO}}(x)$

Figure 6.2: The Real-Indiff and Ideal-Indiff games with respect to a distinguisher $\mathcal{D}$, a simulator $\mathcal{S}$, and a construction C^{H}.

returns a decision bit b. In the Ideal-Indiff game, the random oracle RO is sampled, and the distinguisher $\mathcal{D}$'s construction oracle is replaced with RO, and the primitive oracle is simulated by a simulator $\mathcal{S}$, which itself has oracle access to RO.

Note that sampling H from $\mathrm{Fun}(N, M)$ in Real-Indiff will induce the input and output sizes of N' and M' for the construction C^{H}, and hence, RO is drawn from $\mathrm{Fun}(N', M')$ in Ideal-Indiff. In this thesis, we will limit considerations to salted constructions with the same domain and range and the underlying hash.

For a distinguisher $\mathcal{D}$, a construction C^{H}, and a simulator $\mathcal{S}$, we define the indifferentiability advantage in this game as the distinguishing advantage between the real and ideal worlds. In the real world the distinguisher has oracle access to the construction C^{H} and the underlying primitive H. In the ideal world, the construction is replaced with a random oracle RO and the primitive is simulated by the simulator $\mathcal{S}$ which itself has access to the random oracle. Below we define what is meant for a construction to be indifferentiable from a random oracle.

Definition 7 (Indifferentiability)**.** *A construction C^{H}, based on primitive H, is* indifferentiable *from a random oracle RO if there exists a simulator $\mathcal{S}$ such that for any distinguisher $\mathcal{D}$, the following indifferentiability advantage is small.*

$$\mathbf{Adv}^{\mathsf{indiff}}_{\mathsf{C}^{\mathsf{H}},\mathcal{D},\mathcal{S}} := \left|\Pr\left[\textsc{Real-Indiff}^{\mathcal{D}}_{\mathsf{C}^{\mathsf{H}}}\right] - \Pr\left[\textsc{Ideal-Indiff}^{\mathcal{D},\mathcal{S}}_{\mathsf{RO}}\right]\right| .$$

Order of the quantifiers. While this notion requires that there exists a simulator $\mathcal{S}$ which simulates for every distinguisher $\mathcal{D}$, one can consider weaker notions which swap the order of the quantifiers, i.e., for every $\mathcal{D}$

$\textsc{Real}\text{-sa-ai-Indiff}^{\overline{\mathcal{D}}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen}}$	$\textsc{Ideal}\text{-sa-ai-Indiff}^{\mathcal{D}_1,\mathcal{D}_2,\overline{\mathcal{S}}}_{\mathsf{RO},\mathsf{Gen}}$
$\mathsf{H} \twoheadleftarrow \mathrm{Fun}(N, M)$	$\mathsf{RO} \twoheadleftarrow \mathrm{Fun}(N', M')$
$z_0 \leftarrow \mathcal{D}_0(\mathsf{H})$	$z_0 \leftarrow \mathcal{S}_0(\mathsf{RO})$
$(st, z_1) \twoheadleftarrow \mathcal{D}_1^{\mathrm{Prim}_1}()$	$(st, z_1) \twoheadleftarrow \mathcal{D}_1^{\mathrm{Prim}_1}()$
for $i \in [m]$:	for $i \in [m]$:
$\quad \mathbf{sa}[i] \twoheadleftarrow \mathsf{Gen}()$	$\quad \mathbf{sa}[i] \twoheadleftarrow \mathsf{Gen}()$
$b \twoheadleftarrow \mathcal{D}_2^{\mathrm{Const},\mathrm{Prim}_2}(st, z_0, z_1, \mathbf{sa})$	$b \twoheadleftarrow \mathcal{D}_2^{\mathrm{Const},\mathrm{Prim}_2}(st, z_0, z_1, \mathbf{sa})$
Proc. $\mathrm{Prim}_1(x)$	Proc. $\mathrm{Prim}_1(x)$
return $\mathsf{H}(x)$	return $\mathcal{S}_1^{\mathsf{RO}}(x)$
Proc. $\mathrm{Prim}_2(x)$	Proc. $\mathrm{Prim}_2(x)$
return $\mathsf{H}(x)$	return $\mathcal{S}_2^{\mathsf{RO}}(x; z_0, \mathbf{sa})$
Proc. $\mathrm{Const}(sa, x)$	Proc. $\mathrm{Const}(sa, x)$
if $sa \notin \mathbf{sa}$: return $\perp$	if $sa \notin \mathbf{sa}$: return $\perp$
return $\mathsf{C}^{\mathsf{H}}(sa, x)$	return $\mathsf{RO}(sa, x)$

Figure 6.3: The sa-ai-Indiff notion with respect to $\overline{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2)$ and $\overline{\mathcal{S}} = (\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2)$.

there exists a simulator $\mathcal{S}$. In this weaker notion, the simulator $\mathcal{S}$ can depend on the distinguisher $\mathcal{D}$. Indeed, for the multi-stage notions we consider in this paper, one could construct different notions by swapping quantifiers for the simulators and distinguishers at each stage.

Next, we introduce the *salted auxiliary-input indifferentiability* notion we call sa-ai-Indiff.

Description of the game. Both the Real-sa-ai-Indiff and Ideal-sa-ai-Indiff games have three stages. The Real-sa-ai-Indiff game begins with sampling the random oracle H from the space $\mathrm{Fun}(N, M)$. The preprocessing adversary $\mathcal{D}_0$ has unbounded access to H, and outputs an advice string z_0[1]. The first stage distinguisher $\mathcal{D}_1$, gets oracle access to H and outputs a state st along with a string z_1. Note that $\mathcal{D}_1$ does *not* receive z_0 as input. The salt generator Gen then samples a vector $\mathbf{sa}$ of m salts, before $\mathcal{D}_2$ is run with the state st, the preprocessing information z_0 from $\mathcal{D}_0$, the string z_1 from $\mathcal{D}_1$, and the salts $\mathbf{sa}$. The distinguisher $\mathcal{D}_2$ gets oracle access to both the primitive H, and a *salted construction* oracle, that only accepts queries of the form (sa, x) where sa is a salt in

[1]Note that the preprocessing string is generated deterministically, following the restrictions described in [26].

the vector **sa**.

The IDEAL-sa-ai-Indiff begins by sampling the random oracle RO from the space $\mathrm{Fun}(N', M')$. The preprocessing stage is simulated by the simulator $\mathcal{S}_0$ which has unbounded access to the random oracle RO, and outputs the preprocessing information z_0. The distinguishers $\mathcal{D}_1$ and $\mathcal{D}_2$ are run identically to the real game, but the oracles they are granted are changed. The primitive oracles Prim_1 and Prim_2 instead run simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively, each with oracle access to RO. The salted construction oracle replaces the construction C^{H} with the random oracle RO. The pseudocode for both games is presented in Fig. 6.3. Note that the second-stage simulator is not given access to z_1, but does have access to z_0 and the public salt. For a distinguisher $\overline{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2)$, such that $\mathcal{D}_0$ outputs S-bits of preprocessing information, $\mathcal{D}_1$ makes at most T_1 queries and $\mathcal{D}_2$ makes at most T_2 queries to the primitive and T_{Const} construction queries is a $(S, T_1, T_2, T_{\mathrm{Const}})$-distinguisher.

Salted and unsalted constructions. Ours is not the only way to ensure that all queries to the construction are salted. One option is to generate a fresh salt for each construction query; the appeal is there is no need to fix the number of salts generated. While this may seem to widen the reach of possible composition, unfortunately, it is actually too restrictive. For any useful definition we must allow $\mathcal{D}_2$ to query different points on the same salt, and to choose for which salt it query any given point. Instead, we generate the salts before $\mathcal{D}_2$ runs, so the *choice* of salt is available for each query. This notion is both simpler to analyze and still permits a composition theorem wide enough to capture useful games (see Figs. 6.6 and 6.7). Note that in Definition 8, where the salt generator is empty, we recover a simpler notion for unsalted constructions.

When we force the construction oracle to use a salt, we write (sa, x) to denote concatenation of the salt sa with the input x.

Definition 8 (Salted AI Indifferentiability)**.** *A construction* C^{H} *based on primitive* H *is salted AI indifferentiable from a random oracle* RO *with respect to a salt generator* Gen *if there exist simulators* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *such that for all distinguishers* $\mathcal{D}_1$ *and* $\mathcal{D}_2$*, and for every preprocessing distinguisher* $\mathcal{D}_0$*, there exists a simulator* $\mathcal{S}_0$ *such that the advantage below is small.*

$$\mathbf{Adv}^{\mathsf{sa\text{-}ai\text{-}indiff}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{D}},\overline{\mathcal{S}}} := \left| \Pr\left[\textsc{Real}\text{-sa-ai-Indiff}^{\overline{\mathcal{D}}}_{\mathsf{Gen},\mathsf{C}^{\mathsf{H}}}\right] - \Pr\left[\textsc{Ideal}\text{-sa-ai-Indiff}^{\mathcal{D}_1,\mathcal{D}_2,\overline{\mathcal{S}}}_{\mathsf{Gen},\mathsf{RO}}\right] \right| .$$

Where $\overline{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2)$ *and* $\overline{\mathcal{S}} = (\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2)$.

$\textsc{Real-SSU}^{\overline{\mathcal{A}}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{G}}}$	$\textsc{Ideal-SSU}^{\overline{\mathcal{B}}}_{\mathsf{RO},\mathsf{Gen},\overline{\mathcal{G}}}$
$\mathsf{H} \twoheadleftarrow \mathrm{Fun}(N, M)$	$\mathsf{RO} \twoheadleftarrow \mathrm{Fun}(N', M')$
$z \leftarrow \mathcal{A}_0(\mathsf{H})$	$z \leftarrow \mathcal{B}_0(\mathsf{RO})$
$st_1 \twoheadleftarrow \mathcal{G}_1^{\mathcal{A}_1^{\mathsf{H}}}()$	$st_1 \twoheadleftarrow \mathcal{G}_1^{\mathcal{B}_1^{\mathsf{RO}}}()$
for $i \in [m]$:	for $i \in [m]$:
$\quad \mathbf{sa}[i] \twoheadleftarrow \mathsf{Gen}()$	$\quad \mathbf{sa}[i] \twoheadleftarrow \mathsf{Gen}()$
$b \twoheadleftarrow \mathcal{G}_2^{\mathrm{Const},\mathcal{A}_2^{\mathsf{H}}(z,st_a)}(st_1, \mathbf{sa})$	$b \twoheadleftarrow \mathcal{G}_2^{\mathrm{Const},\mathcal{B}_2^{\mathsf{RO}}(z,st_a)}(st_1, \mathbf{sa})$
Proc. $\mathrm{Const}(sa, x)$	Proc. $\mathrm{Const}(sa, x)$
if $sa \notin \mathbf{sa}$: return $\perp$	if $sa \notin \mathbf{sa}$: return $\perp$
else: return $\mathsf{C}^{\mathsf{H}}(sa, x)$	else: return $\mathsf{RO}(sa, x)$

Figure 6.4: The pseudocode for the sample, salt, and use games, where $\overline{\mathcal{A}} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, $\overline{\mathcal{B}} = (\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2)$, and $\overline{\mathcal{G}} = (\mathcal{G}_1, \mathcal{G}_2)$.

6.3.1 Composition theorems

Using this definition, we now state and prove a composition theorem for a large class of games. These can be thought of as *sample, salt, and use (SSU)* games. Intuitively, this notion captures games which sample inputs $\mathbf{x}$ to a construction C^{H}, sample a salt vector $\mathbf{sa}$, and give the adversary access to the outputs $\mathsf{C}^{\mathsf{H}}(sa, x)$ for any of the values $x \in \mathbf{x}$ and $sa \in \mathbf{sa}$. The pseudocode to describe the structure of these games is given in Fig. 6.4. Note that the game $\mathcal{G}_1$, may alter the output of the first-stage adversary in generating the adversaries state st_a.

In Theorem 6 we show salted AI indifferentiability implies that for any adversary $\overline{\mathcal{A}}$ in the left-hand side class of Real-SSU games, i.e., $\mathrm{SSU}^{\overline{\mathcal{A}}}_{\mathsf{C}^{\mathsf{H}},\overline{\mathcal{G}},\mathsf{Gen}}$, there is an adversary $\overline{\mathcal{B}}$ in the right-hand side Ideal-SSU games, i.e., $\mathrm{SSU}^{\overline{\mathcal{B}}}_{\mathsf{RO},\overline{\mathcal{G}},\mathsf{Gen}}$, with a similar advantage.

Theorem 6 (Composition for salted AI indifferentiability)**.** *Let $m \in \mathbb{N}$ and C^{H} be a salted construction which is salted AI indifferentiable from a random oracle RO with respect to a salt generator Gen which outputs m salts. Then for any multi-stage game $\overline{\mathcal{G}} = (\mathcal{G}_1, \mathcal{G}_2)$ and any multi-stage preprocessing adversary $\overline{\mathcal{A}} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, there exists a multi-stage preprocessing adversary $\overline{\mathcal{B}} = (\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2)$ and a distinguisher $\overline{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2)$ such that*

$$\Pr\left[\mathrm{SSU}^{\overline{\mathcal{A}}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{G}}}\right] \leq \Pr\left[\mathrm{SSU}^{\overline{\mathcal{B}}}_{\mathsf{RO},\mathsf{Gen},\overline{\mathcal{G}}}\right] + \mathbf{Adv}^{\mathsf{sa\text{-}ai\text{-}indiff}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{D}},\overline{\mathcal{S}}} .$$

Intuitively, given an $\overline{\mathcal{A}}$ and $\overline{\mathcal{G}}$ as defined above, we have to construct a $\overline{\mathcal{D}}$ and a $\overline{\mathcal{B}}$ which satisfy the inequality. We do this by defining a distinguisher from $\overline{\mathcal{A}}$ and $\overline{\mathcal{G}}$, and using the corresponding simulator $\overline{\mathcal{S}}$ to build the new adversary $\overline{\mathcal{B}}$.

Proof. Let $\overline{\mathcal{A}} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ be an adversary in the multi-stage game $\overline{\mathcal{G}} = (\mathcal{G}_1, \mathcal{G}_2)$, then we show that there exists an adversary $\overline{\mathcal{B}} = (\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2)$ that satisfies the bound in the theorem statement.

First, we use $\overline{\mathcal{A}}$ to define the sa-ai-Indiff distinguisher $\overline{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2)$. Let the preprocessing distinguisher $\mathcal{D}_0$ be the adversary $\mathcal{A}_0$ which, given full access to the primitive, prepares the S-bit advice string to pass to the second-stage distinguisher $\mathcal{D}_2$. Let $\mathcal{D}_1$ be the combined system of $\mathcal{G}_1$ running adversary $\mathcal{A}_1$, such that $\mathcal{D}_1$ forwards all of $\mathcal{A}_1$'s primitive queries to its own primitive oracle, relaying the responses. Then the state st_1 that $\mathcal{G}_1$ passes to $\mathcal{G}_2$, is passed by $\mathcal{D}_1$ to $\mathcal{D}_2$. Let the second stage distinguisher $\mathcal{D}_2$ be the combined system of game $\mathcal{G}_2$ running with $\mathcal{A}_2$. Specifically, $\mathcal{D}_2$ receives the preprocessing string, the state st_1 from $\mathcal{G}_1$, and the public salt and uses it as input to $\mathcal{G}_2$. The distinguisher $\mathcal{D}_2$ then forwards all $\mathcal{A}_2$'s primitive queries and $\mathcal{G}_2$'s construction queries to its primitive and construction oracles respectively, relaying the responses.

The construction C^{H} is indifferentiable as defined in Definition 8, and so there exists a pair of simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ for every $\mathcal{D}_1$ and $\mathcal{D}_2$, and for each $\mathcal{D}_0$ there exists some $\mathcal{S}_0$ such that the distinguishing advantage is small. Let $\overline{\mathcal{S}} = (\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2)$ be the simulator (guaranteed by Definition 8) for $\overline{\mathcal{D}}$ in the sa-ai-Indiff game with respect to the same salt generator Gen and construction C^{H} from the statement. We use this simulator $\overline{\mathcal{S}}$, along with adversary $\overline{\mathcal{A}}$ to construct adversary $\overline{\mathcal{B}}$. Set $\mathcal{B}_0$ to be the preprocessing simulator $\mathcal{S}_0$, which prepares and forwards the S-bit preprocessing string. Set $\mathcal{B}_1$ to be the joint system ${\mathcal{A}_1}^{\mathcal{S}_1}$, and $\mathcal{B}_2$ to be ${\mathcal{A}_2}^{\mathcal{S}_2}$. It is important to note that $\mathcal{S}_2$ does not get direct access to z_1, and so the adversary $\mathcal{B}_2$ does not access any more information than $\mathcal{A}_2$. Then the difference in the advantages of $\overline{\mathcal{B}}$ and $\overline{\mathcal{A}}$ is exactly the distinguishing advantage between the real and ideal worlds, and by rearranging, we obtain the bound from the statement.

□

Proposition 7. *The* SSU *and* sa-ai-Indiff *notions are equivalent.*

In Theorem 6 we show that the sa-ai-Indiff security notion implies security in the SSU games. To show equivalence, we need to show the implication in the opposite direction, i.e., given a distinguisher $\overline{\mathcal{D}}$ in the sa-

ai-Indiff game, we build a game-adversary pair $\overline{\mathcal{G}}$ and $\overline{\mathcal{A}}$ in the real Real-SSU game.

Proof. First, given a distinguisher $\overline{\mathcal{D}}$ in the sa-ai-Indiff game, we build a game-adversary pair $\overline{\mathcal{G}}$ and $\overline{\mathcal{A}}$ in the real Real-SSU game. Let $\overline{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2)$ be a distinguisher in the sa-ai-Indiff game. To build the preprocessing adversary, we set $\mathcal{A}_0$ to be $\mathcal{D}_0$. Given the first-stage distinguisher $\mathcal{D}_1$, which outputs (st, z_1), we set $\mathcal{A}_1$ to be a dummy adversary, and $\mathcal{G}_1$ to be $\mathcal{D}_1$. Let $\mathcal{A}_2$ be a dummy adversary that immediately forwards its input to $\mathcal{G}_2$, and let $\mathcal{G}_2$ be $\mathcal{D}_2$. Then $\mathcal{A}_0$ outputs the preprocessing $z_0 = z$, and passes it straight to $\mathcal{A}_2$. The first stage game $\mathcal{G}_1^{\mathcal{A}_1}$ passes the state $st_1 = (st, z_1)$ to $\mathcal{G}_2$, which upon initializing $\mathcal{A}_2$ with an empty adversarial state st_a, immediately receives z from $\mathcal{A}_2$. Then $\mathcal{G}_2$ can run $\mathcal{D}_2$, on the input $(st, z_0, z_1, \mathbf{sa})$.

Let $\overline{\mathcal{S}}$ be the simulator for the distinguisher $\overline{\mathcal{D}}$. Then to define each stage of the adversary $\overline{\mathcal{B}} := (\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2)$, we compose the adversary $\mathcal{A}_i$ with $\mathcal{S}_i$, i.e., for each i in the set $\{0, 1, 2\}$ we set $\mathcal{B}_i := \mathcal{A}_i^{\mathcal{S}_i}$. By Definition 8, the SSU advantage is small. □

Comparable attacks. Intuitively, the construction C^{H} is a good replacement for a random oracle RO in any application if for any attacker $\mathcal{A}$ against the game where C^{H} is used, there is a similar attacker $\mathcal{B}$ with a similar advantage when the random oracle is used instead.

However, in some settings, the similarity of the adversary has to go beyond query complexity if we wish to obtain a meaningful bound. As an illustration, consider the case of hash-dependent password samplers in the unrecoverability game. Here, the password sampler is modelled by adversary $\mathcal{A}_1$ in the first stage. Suppose that $\mathcal{A}_1$ is a q-query hash-dependent sampler that outputs a uniform random password vector $\mathbf{pw}$, and $\mathcal{B}_1$ is a q-query adversary that outputs a low-entropy password vector $\mathbf{pw}'$. Then even if the advantages for $\overline{\mathcal{A}}$ and $\overline{\mathcal{B}}$ are close in these games, this tells us little about how closely C^{H} behaves to a random oracle. The difference in the password distribution confounds any meaningful comparison. In our proofs, we want to guarantee that the adversary $\mathcal{B}_1$ will satisfy a stronger notion of similarity.

We want the output z_1 of both $\mathcal{A}_1$ and $\mathcal{B}_1$ to have the same unguessability from the point of view of any algorithm with access to the construction. This would guarantee similar unguessability from the point of view of $\mathcal{A}_2$ and $\mathcal{B}_2$. This property is in fact implied by the single-stage indifferentiability of the construction, which we show next.

6.3.2 Comparability of attacks

Here we show that if the construction C^{H} is indifferentiable from a random oracle with the simulator $\mathcal{S}$, then the attack $\overline{\mathcal{B}}$ in the IDEAL-sa-Indiff game, that we build by the composition theorem, will be comparable to the attack $\overline{\mathcal{A}}$ in the REAL-sa-Indiff game. That is, from the point of view of $\mathcal{B}_2$, the output of $\mathcal{B}_1$ will have the same unpredictability as the output of $\mathcal{A}_1$. Note that in the application to the unrecoverability described above, if an adversary can guess the passwords, then they can predict the output. Recall that if C^{H} is indifferentiable from a random oracle, as in Definition 7, we have that

$$\mathbf{Adv}^{\mathsf{indiff}}_{\mathsf{C}^{\mathsf{H}},\mathcal{D},\mathcal{S}} := \left|\Pr\left[\text{REAL-Indiff}^{\mathcal{D}}_{\mathsf{C}^{\mathsf{H}}}\right] - \Pr\left[\text{IDEAL-Indiff}^{\mathcal{D},\mathcal{S}}_{\mathsf{RO}}\right]\right| ,$$

is small. By the composition theorem for ordinary single-stage indifferentiability proven in [42], for any game $\mathcal{G}$ and attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}$ such that

$$\left|\Pr\left[\mathcal{G}^{\mathcal{A}^{\mathsf{H}},\mathsf{C}^{\mathsf{H}}}\right] - \Pr\left[\mathcal{G}^{\mathcal{B}^{\mathsf{RO}},\mathsf{RO}}\right]\right| \leq \epsilon ,$$

where ϵ is small. In particular, this applies to the prediction game presented in Fig. 6.5. This game captures the unpredictability of the first stage adversary's output z_1. The REAL-Pred game is initialized by sampling a random function H. Then a predictor algorithm $\mathcal{P}r$ is given oracle access to the construction, and outputs a prediction $\overline{z_1}$. Next, adversary $\mathcal{A}$ is run with oracle access to H, and wins the game if it outputs $z_1 \neq \overline{z_1}$. The IDEAL-Pred game is syntactically identical, except for the random function RO which is sampled from the space of random function from N' to M'. The function RO replaces both the construction and the primitive oracles.

For adversaries $\mathcal{A}$ and $\mathcal{B}$ in the REAL-Pred and IDEAL-Pred games respectively, we say they are comparably unpredictable if for any prediction algorithm $\mathcal{P}r$, the advantage

$$\mathbf{Adv}^{\mathsf{pred}}_{\mathsf{C}^{\mathsf{H}},\mathcal{D},\mathcal{S}} := \left|\Pr\left[\text{REAL-Pred}^{\mathcal{A}}_{\mathsf{C}^{\mathsf{H}},\mathcal{P}r}\right] - \Pr\left[\text{IDEAL-Pred}^{\mathcal{B}}_{\mathsf{RO},\mathcal{P}r}\right]\right| ,$$

is small.

Claim 9 (Indiff $\implies$ Pred)**.** *Let C^{H} be indifferentiable from a random oracle, as in Definition 7. Then for every predictor algorithm $\mathcal{P}r$, and every adversary $\mathcal{A}$ in the* REAL-Pred *game, there exists an adversary $\mathcal{B}$ in the* IDEAL-Pred *game such that $\mathcal{A}$ and $\mathcal{B}$ are comparably unpredictable. is small.*

$\textsc{Real-Pred}^{\mathcal{A}}_{\mathsf{C}^{\mathsf{H}},\mathcal{P}r}$	$\textsc{Ideal-Pred}^{\mathcal{B}}_{\mathsf{RO},\mathcal{P}r}$
$\mathsf{H} \twoheadleftarrow \mathrm{Fun}(N, M)$	$\mathsf{RO} \twoheadleftarrow \mathrm{Fun}(N', M')$
$\overline{z_1} \twoheadleftarrow \mathcal{P}r^{\mathrm{Const}}$	$\overline{z_1} \twoheadleftarrow \mathcal{P}r^{\mathrm{Const}}$
$z_1 \twoheadleftarrow \mathcal{A}^{\mathrm{Prim}}$	$z_1 \twoheadleftarrow \mathcal{B}^{\mathrm{Prim}}$
return $z_1 \neq \overline{z_1}$	return $z_1 \neq \overline{z_1}$
$\underline{\mathrm{Const}(x)}$	$\underline{\mathrm{Const}(x)}$
return $\mathsf{C}^{\mathsf{H}}(x)$	return $\mathsf{RO}(x)$
$\underline{\mathrm{Prim}(x)}$	$\underline{\mathrm{Prim}(x)}$
return $\mathsf{H}(x)$	return $\mathsf{RO}(x)$

Figure 6.5: The pseudocode for a prediction game with adversaries $\mathcal{A}$, $\mathcal{B}$, and the prediction algorithm $\mathcal{P}r$, with respect to the construction C^{H}. Note that the Prediction algorithm $\mathcal{P}r$ is the same in each game.

Proof. This follows immediately from the indifferentiability of the construction, as the Pred game is in the range of the single-stage composition theorem. Essentially, let $\mathcal{D}$ forward queries from $\mathcal{P}r$, and $\mathcal{A}$ to their respective oracles, and then output $\bar{z} = z$. The indifferentiability result guarantees the existence of a simulator $\mathcal{S}$, such that $\mathcal{A}$ and $\mathcal{B} := \mathcal{A}^{\mathcal{S}}$ have similar advantage. □

Note that if $\mathcal{A}$ runs $\mathcal{A}_1$ from the sa-Indiff game, then $\mathcal{A}^{\mathcal{S}} = \mathcal{B}_1$ is comparably unpredictable. As the prediction game is in the range of the single-stage composition theorem, the simulator must be such that the unpredictability is conserved. In particular, this implies the unguessability it conserved. It doesn't mean the probability of predicting z_1 is small, but rather that the difference in probabilities is small. Lastly, note that in the Ideal world, both the $\mathcal{P}r$ algorithm and adversary $\mathcal{B}$ have oracle access to RO.

Note on public state. In defining a multi-stage indifferentiability notion, one has to decide to what extent the states passed between stages are public, i.e., available to the simulator. One may define the game such that there is a public state passed between adversaries, or passed between the games. In our definition we do not make such a specification.

The reason we do not specify a public state passed between the adversary is because, as we show in the composition theorem, the simulator is composed with the adversary and will already receive the adversarial state

as input. We do not specify that the state of the game is public because this would weaken the definition too much for our applications. Consider the unrecoverability game, the state passed between the first stage and second stage games contains the password vector **pw** sampled by the password sampler. For the above notion to be composable with the unrecoverability game, it is crucial these passwords are not available to the second-stage adversary, who in the unrecoverability game is only given the salts and the hashed passwords.

We instead define a setting where the adversarial stages can only communicate via the games. In particular, the game $\mathcal{G}_1$ runs $\mathcal{A}_1$, uses the output of $\mathcal{A}_1$ to generate the state st_1, and passes it to $\mathcal{G}_2$. Next, the game $\mathcal{G}_2$ generates the input st_a for the second-stage adversary $\mathcal{A}_2$ by using st_1, and salted construction queries.

Example SSU **games.** The first example of an SSU game is the auxiliary-input unrecoverability game AI-UR. This is an auxiliary input variant of the UR game in Fig. 2.3. There is an extra offline adversary $\mathcal{A}_0$ that passes preprocessing information to the online adversary $\mathcal{A}_1$

$\underline{\text{AI-UR}^{\mathcal{A}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\mathcal{P}}}$
$\mathsf{H} \twoheadleftarrow \text{Fun}([N],[M])$
$z_0 \leftarrow \mathcal{A}_0(\mathsf{H})$
$(\mathbf{pw}, z) \twoheadleftarrow \mathcal{P}_m^{\mathsf{H}}()$
for $(i,j) \in [m] \times [\ell]$:
  $\mathbf{sa}[i,j] \twoheadleftarrow \mathsf{Gen}()$
  $\mathbf{y}[i,j] \twoheadleftarrow \mathsf{C}^{\mathsf{H}}(\mathbf{sa}[i,j], \mathbf{pw}[i])$
$(\bar{\mathbf{pw}}) \twoheadleftarrow \mathcal{A}_1^{\mathsf{H},\text{COR}}(z_0, \mathbf{y}, \mathbf{sa}, z)$
return $\bigwedge_{i=1}^{m}(\mathbf{pw}[i] = \bar{\mathbf{pw}}[i])$

$\underline{\text{Proc. COR}(i)}$
return $\mathbf{pw}[i]$

Figure 6.6: The pseudocode for the AI-UR game.

Claim 10. *The* AI-UR *game is an* SSU *game.*

Proof. We identify the following SSU games $\overline{\mathcal{G}} = (\mathcal{G}_1, \mathcal{G}_2)$ and adversaries $\overline{\mathcal{A}} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, in the AI-UR game. The pseudocode for AI-UR if given in Fig. 6.6 for reference. Let $\mathcal{A}_0$ be as in the AI-UR game, and $\mathcal{G}_1^{\mathcal{A}_1} := \mathcal{P}_m$. Then $\mathcal{G}_1^{\mathcal{A}_1}$ forwards the pair $(\mathbf{pw}, z)$ to $\mathcal{G}_2$. Let $\mathcal{G}_2$ use **pw** and its Const

oracle to generate the challenge vector $\mathbf{y}$, as in the AI-UR game, and use $\mathbf{pw}$ to simulate the interface of a corruption oracle Cor. Then $\mathcal{A}'_2$ runs the adversary $\mathcal{A}_1$ from the AI-UR game, where $\mathcal{G}_2$ simulates the interface for Cor. The input to $\mathcal{A}'_2$ is the preprocessing z (directly from $\mathcal{A}'_0$), the public salt vector $\mathbf{sa}$, and both the challenge vector $\mathbf{y}$ and leakage z in the state st_a from $\mathcal{G}_2$. □

$\underline{\text{Real-AI-KDF}^{\mathcal{A}_0,\mathcal{P},\mathcal{D}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen}}}$	$\underline{\text{Ideal-AI-KDF}^{\mathcal{P},\mathcal{D},\overline{\mathcal{S}}}_{\mathsf{RO},\mathsf{Gen}}}$
$\mathsf{H} \twoheadleftarrow \text{Fun}(N, M)$	$\mathsf{RO} \twoheadleftarrow \text{Fun}(N', M')$
$z_0 \leftarrow \mathcal{A}_0(\mathsf{H})$	$z_0 \leftarrow \mathcal{S}_0(\mathsf{RO})$
$(\mathbf{pw}, z) \twoheadleftarrow \mathcal{P}^{\text{Prim}_1}()$	$(\mathbf{pw}, z) \twoheadleftarrow \mathcal{P}^{\text{Prim}_1}()$
for $(i,j) \in [m] \times [\ell]$:	for $(i,j) \in [m] \times [\ell]$:
$\quad \mathbf{sa}[i,j] \twoheadleftarrow \mathsf{Gen}()$	$\quad \mathbf{sa}[i,j] \twoheadleftarrow \mathsf{Gen}()$
$\quad \mathbf{y}[i,j] \twoheadleftarrow \mathsf{C}^{\mathsf{H}}(\mathbf{sa}[i,j], \mathbf{pw}[i])$	$\quad \mathbf{y}[i,j] \twoheadleftarrow \mathsf{C}^{\mathsf{H}}(\mathbf{sa}[i,j], \mathbf{pw}[i])$
$b \twoheadleftarrow \mathcal{D}^{\text{Prim}_2}(\mathbf{pw}, \mathbf{sa}, \mathbf{y}, z_0)$	$b \twoheadleftarrow \mathcal{D}^{\text{Prim}_2}(\mathbf{pw}, \mathbf{sa}, \mathbf{y}, z_0)$
$\underline{\text{Proc. Prim}_1(x)}$	$\underline{\text{Proc. Prim}_1(x)}$
return $\mathsf{H}(x)$	return $\mathcal{S}_1^{\mathsf{RO}}(x)$
$\underline{\text{Proc. Prim}_2(x)}$	$\underline{\text{Proc. Prim}_2(x)}$
return $\mathsf{H}(x)$	return $\mathcal{S}_2^{\mathsf{RO}}(x; z_0)$

Figure 6.7: The pseudocode for the AI-KDF games: Real (left) and Ideal (right).

Similarly, the AI-KDF game in Fig. 6.7 trivially falls under the reach of the SSU games. Note that this version of the KD game is adapted from the work of [30]. There are two important differences. Firstly, in their paper, Farshim and Tessaro only considered the case of hash-independent samplers, here the sampler is given oracle access to the hash in line with the SSU structure. Secondly, in [30], the simulator has access to a Test oracle rather than the RO. We instead present a version of the games which capture an idealised key-derivation functionality, but which are not 'super-ideal', in that there is no Test oracle. Note that the simulator $\mathcal{S}_2$ can use the RO to simulate a Test oracle.

Salt propagation game. In Fig. 6.8, we describe a game to capture the following intuitive property of a construction C^{H}: given an unpredictable

salt sa as part of an input, the queries used to evaluate the construction are also unpredictable. In other words, the construction propagates the unpredictability of the salt. We call the game that captures this property the salt propagation game.

The game is initialized by sampling the random oracle H. The first-stage adversary $\mathcal{A}_1$ has oracle access to H, makes $q_{\mathcal{A}}$ queries, and outputs some advice z_1. Note that $\mathcal{A}_1$'s oracle access is via the Prim_1 oracle, which collects queries in the set $Q_{\mathcal{A}}$. Next, a salt generator Gen samples a vector $\mathbf{sa}$ of m salts, before $\mathcal{A}_2$ is run on the input $(z_1, \mathbf{sa})$. The second-stage adversary $\mathcal{A}_2$ has access oracle access to the primitive H via Prim_2, and a salted construction oracle via Const. For each query evaluated by Const, the necessary queries to the underlying primitive are collected in the set Q_{C}. The adversary wins if $\mathcal{A}_1$ was able to predict any of the queries used to evaluate the constructions. This captures the fact that even with access to H, before the salts are sampled, the adversary has a low probability of guessing any of the queries used in evaluating the salted construction.[2]

For an adversary $\overline{\mathcal{A}} = (\mathcal{A}_1, \mathcal{A}_2)$, a salted construction C^{H} and salt generator Gen, we define the advantage in this game as

$$\mathbf{Adv}^{\mathsf{saprop}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{A}}} = \Pr\left[\text{SaProp}^{\overline{\mathcal{A}}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen}}\right] .$$

For adversaries in the SaProp game, we say that if $\mathcal{A}_1$ makes at most q queries to Prim_1 and $\mathcal{A}_2$ makes at most q_{C} queries to Const, then $\overline{\mathcal{A}} = (\mathcal{A}_1, \mathcal{A}_2)$ is a (q, q_{C})-query adversary.

Given an unpredictable salt generator Gen that produces m salts in a space of size K, we can construct an example of a trivially salt-propagating construction. For any hash-based construction $\mathsf{C}^{\mathsf{H}}(sa, x)$, let $\tilde{\mathsf{C}}^{\mathsf{H}}(sa, x) := \mathsf{C}^{\mathsf{H}(sa,\cdot)}(x)$. That is, the salt is used in *every* invocation to the underlying hash function. Then $\tilde{\mathsf{C}}^{\mathsf{H}}(sa, x)$ is trivially salt-propagating, as predicting any query in Q_{C} clearly involves predicting the salt.

Next, we use this definition to prove that a large class of graph-based memory-hard functions, as presented in Definition 6, are salt-propagating.

Proposition 8. *Let $K, m, q, q_{\mathsf{C}} \in \mathbb{N}$ and C^{H} be a graph-based memory-hard construction based on primitive H as defined in Definition 6, let Gen*

[2] In fact, the game captures a slightly stronger notion. One could consider a single-stage version of the SaProp game where an adversary $\mathcal{A}^{\mathsf{H}}$ must output a vector of inputs to a salted construction C^{H}, and collect queries similarly. Here, $\mathcal{A}_2$ can choose its queries to Const adaptively, which turns out to be technically necessary for our reduction in Claim 11.

$\underline{\mathrm{SaProp}^{\mathcal{A}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen}}}$
$\mathsf{H} \twoheadleftarrow \mathrm{Fun}(N, M)$
$z_1 \twoheadleftarrow \mathcal{A}_1^{\mathrm{Prim}_1}()$
for $i \in [m]$:
  $\mathbf{sa}[i] \twoheadleftarrow \mathsf{Gen}()$
$z_2 \twoheadleftarrow \mathcal{A}_2^{\mathrm{Prim}_2,\mathrm{Const}}(z_1, \mathbf{sa})$
return $(Q_{\mathcal{A}} \cap Q_{\mathsf{C}} \neq \emptyset)$

$\underline{\text{Proc. } \mathrm{Prim}_1(x)}$
$Q_{\mathcal{A}} \leftarrow Q_{\mathcal{A}} \cup \{x\}$
return $\mathsf{H}(x)$

$\underline{\text{Proc. } \mathrm{Prim}_2(x)}$
return $\mathsf{H}(x)$

$\underline{\text{Proc. } \mathrm{Const}(sa, x)}$
if $sa \notin \mathbf{sa}$: return $\perp$
return $\mathsf{C}^{\mathrm{Prim}_3}(sa, x)$

$\underline{\text{Proc. } \mathrm{Prim}_3(x)}$
$Q_{\mathsf{C}} \leftarrow Q_{\mathsf{C}} \cup \{x\}$
return $\mathsf{H}(x)$

Figure 6.8: The salt propagation game with respect to the construction C^{H} and salt generator Gen.

be a uniform salt generator sampling m salts from a space of size K, and $\overline{\mathcal{A}} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary in the salt-propagation game, with respect to C^{H} and Gen. Then,

$$\mathbf{Adv}^{\mathsf{saprop}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\mathcal{A}} \leq \frac{mq}{K} + \frac{mq_{\mathsf{C}}q}{M} \ .$$

Where $\mathcal{A}_1$ makes q primitive queries, and $\mathcal{A}_2$ makes q_{C} construction queries.

Proof. For the sets $Q_{\mathcal{A}}$ and Q_{C} to share a query, it must be of one of two forms: either it is at the start of a computation, i.e. of the form $(1, sa, x)$ or it is midway through the computation i.e. (i, x). Let bad_1 be the event that at least one query of the first form is contained in $Q_{\mathcal{A}} \cap Q_{\mathsf{C}}$. Let bad_2 be the event that at least one query of the second form is contained in $Q_{\mathcal{A}} \cap Q_{\mathsf{C}}$. Then we have $\Pr[\mathcal{A} \text{ wins}] \leq \Pr[\mathsf{bad}_1] + \Pr[\mathsf{bad}_2 | \neg \mathsf{bad}_1]$.

First we bound the probability of the event bad_1. As the salts are generated *after* $\mathcal{A}_1$ produces the advice z_1, and the salts are sampled uniformly at random, the probability of any of $\mathcal{A}_1$'s $|Q_{\mathcal{A}}|$ queries containing one of the m salts that Gen samples is at most $m|Q_{\mathcal{A}}|/K \leq mq/K$.

Next, we bound the probability of bad_2 given that bad_1 does not occur. Let q_i denote the number of unique queries $\mathcal{A}_1$ makes to the Prim_1 oracle, indexed with node index $i \in [n]$. For any $i > 1$, the event that $\mathcal{A}_1$ makes a query of the form $\mathrm{Prim}(i, x)$, which correctly labels the i-th node on the graph (with respect to an input chosen by $\mathcal{A}_2$), *without* honestly computing the construction up to that node's parents, requires guessing the output of H on a fresh query. The probability of this event is at most q_i/M. Taking the union bound over all q_{C} inputs, all m salts, and all n indices of the graph, we get an upper bound of $\Pr[\mathsf{bad}_2 | \neg \mathsf{bad}_1] \leq$

$\text{REAL-sa-Indiff}^{\overline{\mathcal{D}}}_{\mathsf{Gen},\mathsf{C}^{\mathsf{H}}}$	$\text{IDEAL-sa-Indiff}^{\overline{\mathcal{D}},\overline{\mathcal{S}}}_{\mathsf{Gen},\mathsf{RO}}$
$\mathsf{H} \twoheadleftarrow \text{Fun}(N, M)$	$\mathsf{RO} \twoheadleftarrow \text{Fun}(N', M')$
$(st, z_1) \twoheadleftarrow \mathcal{D}_1^{\text{Prim}_1}()$	$(st, z_1) \twoheadleftarrow \mathcal{D}_1^{\text{Prim}_1}()$
for $i \in [m]$:	for $i \in [m]$:
$\quad \mathbf{sa}[i] \twoheadleftarrow \mathsf{Gen}()$	$\quad \mathbf{sa}[i] \twoheadleftarrow \mathsf{Gen}()$
$b \twoheadleftarrow \mathcal{D}_2^{\text{Const},\text{Prim}_2}(st, z_1, \mathbf{sa})$	$b \twoheadleftarrow \mathcal{D}_2^{\text{Const},\text{Prim}_2}(st, z_1, \mathbf{sa})$
Proc. $\text{Prim}_1(x)$	Proc. $\text{Prim}_1(x)$
return $\mathsf{H}(x)$	return $\mathcal{S}_1^{\mathsf{RO}}(x)$
Proc. $\text{Prim}_2(x)$	Proc. $\text{Prim}_2(x)$
return $\mathsf{H}(x)$	return $\mathcal{S}_2^{\mathsf{RO}}(x; \mathbf{sa})$
Proc. $\text{Const}(sa, x)$	Proc. $\text{Const}(sa, x)$
if $sa \notin \mathbf{sa}$: return $\perp$	if $sa \notin \mathbf{sa}$: return $\perp$
else: return $\mathsf{C}^{\mathsf{H}}(sa, x)$	else: return $\mathsf{RO}(sa, x)$

Figure 6.9: The salted two-stage indifferentiability notion with respect to $\overline{\mathcal{D}} = (\mathcal{D}_1, \mathcal{D}_2)$ and $\overline{\mathcal{S}} = (\mathcal{S}_1, \mathcal{S}_2)$.

$\sum_{i=1}^{n} \sum_{k=1}^{m} \sum_{j=1}^{q_{\mathsf{C}}} q_i/M \leq \sum_{i=1}^{n} m q_{\mathsf{C}} q_i/M \leq m q_{\mathsf{C}} q/M$. Where the final inequality follows from the fact that $\sum_{i=1}^{n} q_i \leq q$.

□

Sa-indifferentiability. Below we introduce a multi-stage indifferentiability notion that we call sa-indifferentiability. Intuitively, this is the sa-ai-Indiff setting without the preprocessing distinguisher. In fact, from the point of view of $\mathcal{D}_2$, the game is identical to the sa-ai-Indiff game where the preprocessing distinguisher outputs no preprocessing information. The pseudocode for the game is presented in Fig. 6.9. We introduce this here to prove a separate result. We will show that, under certain conditions, an indifferentiability result can be lifted to a sa-Indiff setting.

Definition 9 (Sa-indifferentiability)**.** *A construction* C^{H} *based on primitive* H *is sa-indifferentiable from a random oracle* RO*, with respect to a salt generator* Gen*, if there exists a simulator* $\overline{\mathcal{S}}$ *such that for any distinguisher* $\overline{\mathcal{D}}$*, the following advantage is small.*

$$\mathbf{Adv}^{\mathsf{sa\text{-}indiff}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{D}},\mathcal{S}} := |\Pr\left[\text{REAL-sa-Indiff}^{\overline{\mathcal{D}}}_{\mathsf{Gen},\mathsf{C}^{\mathsf{H}}}\right] - \Pr\left[\text{IDEAL-sa-Indiff}^{\overline{\mathcal{D}},\overline{\mathcal{S}}}_{\mathsf{Gen},\mathsf{RO}}\right]| .$$

Where $\overline{\mathcal{D}} = (\mathcal{D}_1, \mathcal{D}_2)$ *and* $\overline{\mathcal{S}} = (\mathcal{S}_1, \mathcal{S}_2)$.

In the following theorem, we show that any salt-propagating construction which is indifferentiable is also sa-indifferentiable. As part of the proof, we give a reduction to the PRF distinguishing game, which we present in pseudocode after the proof.

Theorem 9 (Indiff $\implies$ sa-Indiff)**.** *Let $m, q, K, N, M, T_1, T_2, T_{Gen} \in \mathbb{N}$. Let* Gen *be a uniform salt generator that samples salts from a space of size K,* C^{H} *be a salt-propagating construction, and* PRF *be a secure PRF. Then if* C^{H} *is indifferentiable from a random oracle, it is also sa-indifferentiable from a random oracle with respect to* Gen*.*

That is, let $\mathcal{S}^*$ be a q-query simulator such that for any distinguisher $\mathcal{D}^*$ the advantage in the indifferentiability game is small, then for any two-stage distinguisher $\overline{\mathcal{D}} = (\mathcal{D}_1, \mathcal{D}_2)$ there exists a simulator $\overline{\mathcal{S}} = (\mathcal{S}_1, \mathcal{S}_2)$, an adversary $\overline{\mathcal{A}} = (\mathcal{A}_1, \mathcal{A}_2)$ in the salt propagation game, and an adversary $\mathcal{B}$ in the PRF distinguishing game such that

$$\mathbf{Adv}^{\mathsf{sa\text{-}indiff}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{D}},\overline{\mathcal{S}}} \leq \mathbf{Adv}^{\mathsf{indiff}}_{\mathsf{C}^{\mathsf{H}},\mathcal{D}^*,\mathcal{S}^*} + \mathbf{Adv}^{\mathsf{saprop}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{A}}} + \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{PRF},\mathcal{B}} \ .$$

Where T_{Gen} is the total number of queries to evaluate the construction with the honest algorithm, $\overline{\mathcal{D}} = (\mathcal{D}_1, \mathcal{D}_2)$ is a (T_1, T_2)-query distinguisher, $(\mathcal{S}_1, \mathcal{S}_2)$ is a $(q + T_1 + T_2 + 1, q + T_1 + T_2 + 1)$-query simulator, $\mathcal{B}$ is a $(T_1 + T_2 + T_{Gen})$-query adversary, and $\overline{\mathcal{A}}$ is a (T_1, T_2)-query adversary.

Proof. Intuitively, we need to prove the following two claims: that any multi-stage distinguisher $(\mathcal{D}_1, \mathcal{D}_2)$ can distinguish in the single-stage setting, *and* that we can use a single-stage simulator $\mathcal{S}$ to build a multi-stage simulator $(\mathcal{S}_1, \mathcal{S}_2)$.

To prove the first claim, let $(\mathcal{D}_1, \mathcal{D}_2)$ be a multi-stage distinguisher, and let $\mathcal{D}^*$ be a single-stage distinguisher that runs $\mathcal{D}_1$ and $\mathcal{D}_2$ in sequence, passing the state between them as necessary. More precisely, $\mathcal{D}$ runs $\mathcal{D}_1$, forwarding all queries to its own oracle, and relaying the responses. Using the salt generator Gen it generates the salt vector $\mathbf{sa}$, and runs $\mathcal{D}_2$ on input $(st, z_1, \mathbf{sa})$. The distinguisher $\mathcal{D}^*$ will forward all primitive queries to its own primitive oracle and relay the responses. For all construction queries, $\mathcal{D}^*$ will check they are prepended with a salt in $\mathbf{sa}$ before relaying them, and respond to $\mathcal{D}_2^*$ with $\perp$ otherwise. Finally, given a single-stage simulator $\mathcal{S}^*$, we build a multi-stage simulator $(\mathcal{S}_1, \mathcal{S}_2)$.

We start in the single-stage game, with an adversary $\mathcal{D}^*$ running a two-stage adversary $(\mathcal{D}_1, \mathcal{D}_2)$ in sequence, generating the necessary salt, and passing the necessary state. Intuitively, we make a sequence of game hops to reach a situation where we can "pull apart" both stages of the

simulator. That is, we want to transition to a setting in which the simulator maintains consistency *without* use of a shared state. To get there, we first rule out the event that $\mathcal{D}_1$'s primitive queries collide with any of the points sampled by the random oracle to evaluate the construction. This ensures that the responses sampled by the simulator in response to $\mathcal{D}_1$ are independent of $\mathcal{D}_2$'s queries to the construction oracle. Since the responses to $\mathcal{D}_2$'s primitive and construction queries are consistent by assumption, it remains to show that the responses to primitive queries of $\mathcal{D}_1$ and $\mathcal{D}_2$ are consistent between stages. To maintain consistency, $\mathcal{S}_1$ and $\mathcal{S}_2$ generate the responses to queries with a secure pseudorandom function PRF, using a shared seed sd which is generated by each simulator separately, and is information-theoretically hidden from the distinguishers. In the game hops that follow, if a game aborts, it returns $\perp$ from then on.

G_0: This game is the Ideal-Indiff game in Fig. 6.2, where RO is lazily sampled. As both stages of the simulator share a state, this game is identical to the single-stage indifferentiability game, where the distinguisher $\mathcal{D}^*$ runs $\mathcal{D}_1$, Gen, and $\mathcal{D}_2$ in sequence, passing the state as necessary. As C^{H} is indifferentiable from a random oracle, and both stages of the simulator run $\mathcal{S}^*$, again passing state as necessary, the advantage in this game is $\mathbf{Adv}^{\mathsf{indiff}}_{\mathsf{C}^{\mathsf{H}},\mathcal{D}^*,\mathcal{S}^*}$.

G_1: In this game, if $\mathcal{D}_1$ queries some x to Prim, and $\mathcal{D}_2$ later queries some x' to the construction oracle Const such that the evaluation of $\mathsf{C}^{\mathsf{H}}(sa, x')$ also queries x to H, we set bad_1 and abort. Ruling out this event ensures that all points sampled during the evaluation of the construction queries are independent of the queries made by $\mathcal{D}_1$ (and therefore fresh). Thus, $\Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1] \leq \Pr[\mathrm{G}_1 \text{ sets } \mathsf{bad}_1]$. In the salt unpredictability claim below, by a reduction to the salt propagation game in Fig. 6.8, we show that there exists a (T_1, T_2)-query adversary $\overline{\mathcal{A}} = (\mathcal{A}_1, \mathcal{A}_2)$ such that $\Pr[\mathrm{G}_1 \text{ sets } \mathsf{bad}_1] \leq \mathbf{Adv}^{\mathsf{saprop}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{A}}}$.

G_2: In this game we restrict the simulator $\mathcal{S}^*$ to have no shared state between the stages. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be the resulting simulators for the first and second stages respectively. For each stage of the simulator $\mathcal{S}_1$ and $\mathcal{S}_2$, we use a de-randomized version of $\mathcal{S}^*$. Let $q_{\mathcal{S}} := T_1 + T_2 + T_{Gen}$, then each stage of the simulator uses $q_{\mathcal{S}} + 1$ queries to generate the seed $sd := \bigoplus_{i=1}^{q_{\mathcal{S}}+1} \mathsf{RO}(i)$. This provides both $\mathcal{S}_1$ and $\mathcal{S}_2$ with shared randomness outside of either distinguisher's view. Using the shared seed sd, both simulators generate fresh responses consistently with a secure PRF. The identical random values mean

that both stages of the simulator remain consistent with each other, and as C^{H} is single-stage indifferentiable, $\mathcal{S}_2 = \mathcal{S}^*$ remains consistent with the construction queries. This game is the IDEAL-sa-ai-Indiff game. Let $\mathcal{B}$ be a $(T_1 + T_2 + T_{Gen})$-query adversary in the PRF distinguishing game. Then in the PRF advantage claim below, we show that the difference in the games $\Pr[\mathrm{G}_2] - \Pr[\mathrm{G}_1] \leq \mathbf{Adv}^{\mathsf{prf}}_{\mathcal{B},\mathsf{PRF}}$, where the key for PRF is the seed *sd*.

□

Claim 11 (Salt propagation)**.** *Let* C^{H} *be the salt propagating function in* G_1*, let* Gen *be the salt generator, and let* $\mathcal{D}_1$ *and* $\mathcal{D}_2$ *be the distinguishers from* G_1*, then there exists an adversary* $\overline{\mathcal{A}} = (\mathcal{A}_1, \mathcal{A}_2)$ *such that*

$$\Pr[\mathrm{G}_1 \textit{ sets } \mathsf{bad}_1] \leq \mathbf{Adv}^{\mathsf{saprop}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{A}}} .$$

Proof. Let $\overline{\mathcal{A}}$ be an adversary in the salt propagation game with respect to the salt generator Gen and construction C^{H}. Let $\mathcal{A}_1$ run $\mathcal{D}_1$, forwarding all $\mathcal{D}_1$'s queries to its own Prim_1 oracle and relaying the responses. Next, $\mathcal{A}_2$ receives the salt vector $\mathbf{sa}$, and advice (st, z_1) from $\mathcal{A}_1$. Then, $\mathcal{A}_2$ runs $\mathcal{D}_2$, on the input $(st, z_1, \mathbf{sa})$ forwarding all primitive queries to its Prim_2 oracle, and construction queries to its Const oracle. Note that $\mathcal{A}_1$ and $\mathcal{A}_2$ make at most T_1 queries to Prim_1 and at most T_2 queries to Const. If bad_1 is set, then the set of H queries made by $\mathcal{D}_1$ and the set of H queries made to evaluate the construction have at least one query in common. Therefore, $Q_{\mathcal{A}} \cap Q_{\mathsf{C}} \neq \emptyset$, and $\overline{\mathcal{A}}$ wins the salt propagation game.

□

Claim 12 (PRF advantage)**.** *Let* C^{H} *be the salt propagating function in* G_1*, let* Gen *be the salt generator, and let* $\overline{\mathcal{D}} = (\mathcal{D}_1, \mathcal{D}_2)$ *be the distinguisher, then there exists an adversary* $\mathcal{B}$

$$\Pr[\mathrm{G}_2] - \Pr[\mathrm{G}_1] \leq \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{PRF},\mathcal{B}} .$$

Where adversary $\mathcal{B}$ *is a* $(T_1 + T_2 + T_{Gen})$*-query adversary, and* $\mathcal{K}$ *samples a uniform random key from a space of size* M*.*

Proof. Intuitively, we argue that G_1 is equivalent to the 0-PRF distinguishing game in which the responses are generated with a random oracle. The second game, G_2, is the 1-PRF game where responses are generated by the PRF with respect to the key *sd*.

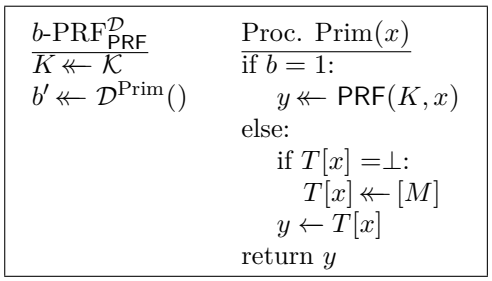

Figure 6.10: The pseudocode for the PRF distinguishing game.

Let $\mathcal{B}$ be an adversary in the game b'-PRF$^{\mathcal{B}}_{\mathsf{PRF}}$, where the key K is the seed sd generated by the simulators in G_2. Let $\mathcal{B}$ run the two-stage distinguisher $(\mathcal{D}_1, \mathcal{D}_2)$ in sequence, passing the state between the stages and generating the salt vector using Gen. To handle the primitive queries $\mathcal{B}$ will forward each one, from either $\mathcal{D}_1$ or $\mathcal{D}_2$, to $\mathcal{B}$'s own oracle. Adversary $\mathcal{B}$ evaluates all construction queries via its primitive oracle, using the honest canonical algorithm for construction C^{H}, and relays the output as its response. When $(\mathcal{D}_1, \mathcal{D}_2)$ ends, $\mathcal{B}$ returns the bit b it receives from $\mathcal{D}_2$. Note, the total number of queries made by $\mathcal{B}$ is the sum of the primitive queries of both $\mathcal{D}_1$ and $\mathcal{D}_2$ and the primitive queries required to evaluate the construction for each of $\mathcal{D}_2$'s construction queries. This is $T_1 + T_2 + T_{Gen}$ queries in total. When $b = 0$, the adversary $\mathcal{B}$ simulates G_1 for distinguisher $\overline{\mathcal{D}}$, and when $b' = 1$, $\mathcal{B}$ simulates G_2. Therefore, the difference $\Pr[\mathrm{G}_2] - \Pr[\mathrm{G}_1]$ is at most $\mathbf{Adv}^{\mathsf{prf}}_{\mathcal{B},\mathsf{PRF}}$. □

As G_2 is the IDEAL-sa-Indiff game, by collecting terms, we obtain the stated result. □

PRF **games.** Here we describe the pseudocode for the PRF game from the reduction in Claim 12 – which we present in Fig. 6.10. The game begins by sampling a key K, and then a distinguisher $\mathcal{D}$ is given oracle access to either a lazily-sampled random oracle, or the pseudo-random function PRF, both with the same range M. The distinguisher wins the game if it can detect whether it is interacting with the PRF or the lazily-sampled random oracle with the same range.

The advantage in this game is given by

$$\mathbf{Adv}^{\mathsf{prf}}_{\mathsf{PRF},\mathcal{D}} := \left| \Pr\left[0\text{-PRF}^{\mathcal{D}}_{\mathsf{PRF}}\right] - \Pr\left[1\text{-PRF}^{\mathcal{D}}_{\mathsf{PRF}}\right] \right| .$$

$\textsc{Real}$-sa-bf-Indiff$^{\overline{\mathcal{D}}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen}}$	$\textsc{Ideal}$-sa-bf-Indiff$^{\mathcal{D}_1,\mathcal{D}_2,\overline{\mathcal{S}}}_{\mathsf{RO},\mathsf{Gen}}$
$(L, z_0) \leftarrow \mathcal{D}_0()$	$(L', z_0) \leftarrow \mathcal{S}_0()$
$\mathsf{H} \twoheadleftarrow \mathrm{Fun}(N, M)$	$\mathsf{RO} \twoheadleftarrow \mathrm{Fun}(N', M')$
$(st, z_1) \twoheadleftarrow \mathcal{D}_1^{\mathrm{Prim}_1}()$	$(st, z_1) \twoheadleftarrow \mathcal{D}_1^{\mathrm{Prim}_1}()$
for $i \in [m]$:	for $i \in [m]$:
$\quad \mathbf{sa}[i] \twoheadleftarrow \mathsf{Gen}()$	$\quad \mathbf{sa}[i] \twoheadleftarrow \mathsf{Gen}()$
$b \twoheadleftarrow \mathcal{D}_2^{\mathrm{Prim}_2,\mathrm{Const}}(st, z_0, z_1, \mathbf{sa})$	$b \twoheadleftarrow \mathcal{D}_2^{\mathrm{Prim}_2,\mathrm{Const}}(st, z_0, z_1, \mathbf{sa})$
Proc. $\mathrm{Prim}_1(x)$	Proc. $\mathrm{Prim}_1(x)$
return $\mathsf{H}[L](x)$	return $\mathcal{S}_1^{\mathsf{RO}[L']}(x)$
Proc. $\mathrm{Prim}_2(x)$	Proc. $\mathrm{Prim}_2(x)$
return $\mathsf{H}[L](x)$	return $\mathcal{S}_2^{\mathsf{RO}[L']}(x; z_0, \mathbf{sa})$
Proc. $\mathrm{Const}(sa, x)$	Proc. $\mathrm{Const}(sa, x)$
if $sa \notin \mathbf{sa}$: return $\perp$	if $sa \notin \mathbf{sa}$: return $\perp$
else: return $\mathsf{C}^{\mathsf{H}}(sa, x)$	else: return $\mathsf{RO}[L'](sa, x)$

Figure 6.11: The salted two-stage indifferentiability with bit fixing notion with respect to $\overline{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2)$ and $\overline{\mathcal{S}} = (\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2)$.

Remark. Since we are proving a multi-stage indifferentiability notion, as commented on in Section 6.3.1, we must consider the comparability of the attack. But, as this result lifts the indifferentiability of a salt-propagating construction to a multi-stage setting, by the argument in Section 6.3.1, the comparability of the attack is preserved. In particular, when considering the unrecoverability game Fig. 2.3 (where there is no preprocessing information), the password sampler produced by the combined system of adversary and simulator retains a comparable unguessability.

Remark. Note that the security of the final bound depends on the advantage of an adversary in the PRF game, with respect to PRF, and so our bound relies on a computational assumption rather than any information-theoretic guarantees.

6.3.3 Bit-fixing indifferentiability

Here, in order to use the framework of [26], we introduce an auxiliary game in the BF-ROM. This is essentially the sa-ai-Indiff notion, where the preprocessing is modelled as a list of prefixed points, on which the random oracles are conditioned. The pseudocode for the games is presented in Fig. 6.11.

In the following, for a distinguisher $\overline{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2)$, such that $\mathcal{D}_0$ outputs a list of size P, $\mathcal{D}_1$ makes at most T_1 queries and $\mathcal{D}_2$ makes at most T_2 queries, we write that $\overline{\mathcal{D}}$ is a $(P, T_1, T_2, T_{\text{Const}})$-distinguisher.

Definition 10 (Salted BF indifferentiability)**.** *A construction* C^{H} *based on primitive* H *is salted BF indifferentiable from a random oracle* RO*, with respect to a salt generator* Gen*, if there exist simulators* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *such that for all distinguishes* $\mathcal{D}_1$ *and* $\mathcal{D}_2$*, and for every preprocessing distinguisher* $\mathcal{D}_0$*, there exists a simulator* $\mathcal{S}_0$ *such that the advantage*

$$\mathbf{Adv}^{\mathsf{sa\text{-}bf\text{-}indiff}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{D}},\overline{\mathcal{S}}} := \left| \Pr\left[\textsc{Real}\text{-sa-bf-Indiff}^{\overline{\mathcal{D}}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen}}\right] - \Pr\left[\textsc{Ideal}\text{-sa-bf-Indiff}^{\mathcal{D}_1,\mathcal{D}_2,\overline{\mathcal{S}}}_{\mathsf{RO},\mathsf{Gen}}\right] \right|$$

is small. Where $\overline{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2)$ *and* $\overline{\mathcal{S}} = (\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2)$*,* $\overline{\mathcal{D}}$ *is a* $(P, T_1, T_2, T_{\text{Const}})$*-distinguisher and* $\overline{\mathcal{S}}$ *is a* (P, T_1, T_2)*-simulator.*

Description of the game. Naturally, these games closely follow the structure of the sa-ai-Indiff games; indeed, the Real games are syntactically identical. As such, both the Real-sa-bf-Indiff and Ideal-sa-bf-Indiff games have three stages. In line with the bit-fixing random oracle model, the preprocessing is generated independently of the random oracle, as it is resampled according to the list L.

Why is bit-fixing introduced? It is reasonable to ask why an additional notion needs to be introduced, and whether it is useful in its own right, e.g. as a model for a particular kind of adversary, or whether it is only a tool in the proof. The answer is that it is introduced only as a tool to prove sa-ai-Indiff bounds. Indeed, the bit-fixing model is essentially a reduction from a world where the adversary can encode any information in z_0 (i.e. the sa-ai-Indiff world), to one where this encoding amounts to a list of fixed input-output pairs L.

Another natural question is how the bit-fixing and auxiliary-input notions compare. From the answer to the previous questions it can be seen that the auxiliary input notion is in fact the stronger notion; one has to restrict the adversaries capabilities to get from the one to the other.

The FindChain **function.** For the main result of the chapter we analyse the sa-bf-Indiff security of a large class of graph-based constructions described by the labelling scheme in Definition 5. We observe that such a labelling scheme allows one to define a function, for each underlying graph, that traverses the labels backwards, i.e., from sink to source, intuitively

by reversing the directions on the underlying DAG. Observing that such a function exists for each underlying DAG, we call it the FindChain function, and use it, in a black-box manner, for the proof of the main result.

Next, we prove the most important step in this chapter's main result. Informally, we prove that any salt-propagating, indexed, single sink, graph-based construction, built from a random oracle H, where there exists a FindChain function, is indifferentiable in the sa-bf-Indiff game. In particular, all constructions that meet these requirements are secure in SSU games.

As part of the proof we give a reduction to an augmented version of the SaProp game, that allows us to account for the prefixed points in L. We give a full description of the game, which we call bf-SaProp, in Section 6.3.3, with the pseudocode presented in Fig. 6.13. Intuitively, this game is the same as the SaProp game, but accounts for the prefixed points in the adversary's win condition, i.e., if any of $\mathcal{A}_1$'s primitive queries *or* any of the P prefixed points overlap with the primitive queries used to evaluate the construction. Advantage in this game is denoted by $\mathbf{Adv}^{\mathsf{bf\text{-}saprop}}$. Informally, we can view a (P, T_1, T_2)-adversary in the bf-SaProp game as essentially equivalent to a $(P + T_1, T_2)$-adversary in the SaProp game.

A key technical challenge in the proof is that for $\mathcal{D}_0$, the list L fixes input-output pairs in the underlying hash, but in the ideal world, the construction oracle is answered by $\mathsf{RO}[L]$, and so L fixes input-output pairs for the construction. Additionally, while $\mathcal{D}_1$ is unaware of the preprocessing, any simulated responses to its queries still need to be consistent with L. To see why, note that any information gathered by $\mathcal{D}_1$'s queries can be forwarded to $\mathcal{D}_2$, which does see the preprocessing. Unfortunately, while the $\mathcal{S}_2$ simulator has access to the preprocessing z_0, it cannot necessarily recover L as the Decomp algorithm requires a potentially unbounded computation. The answer we provide is a rather simple simulator which effectively swaps the role of the random oracle RO to simulate the underlying primitive, except for at the start of salted chains. This is only possible because the salts are unpredictable, in particular, they are unpredictable to the preprocessing adversary.

Theorem 10 (sa-bf-Indiff)**.** *Let $K, N, M, T_1, T_2, T_{\text{Const}}, T_{Gen}, m, P \in \mathbb{N}$, Gen be an unpredictable salt generator sampling m salts from a space of size K, let H be a random oracle in the (N, M)-random oracle model, and let C^{H} be an indexed single-sink salt-propagating construction with domain N' and range M'. Then C^{H} is salted BF indifferentiable from a random*

oracle RO *in the* (N, M) *random oracle model, with respect to* Gen.

That is, there exist simulators $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *such that for all distinguishers* $\mathcal{D}_1$ *and* $\mathcal{D}_2$*, and for every preprocessing distinguisher* $\mathcal{D}_0$*, there exists a simulator* $\mathcal{S}_0$ *such that*

$$\mathbf{Adv}^{\mathsf{sa\text{-}bf\text{-}indiff}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{D}},\overline{\mathcal{S}}} \leq \mathbf{Adv}^{\mathsf{bf\text{-}saprop}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{A}}} + \frac{P \cdot T_1}{M} + \frac{(T_1 + T_2 + T_{Gen})^2}{N} + \frac{T_{Gen} T_2}{N} \ ,$$

where $\overline{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2)$*,* $\overline{\mathcal{S}} = (\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2)$*, and* $\overline{\mathcal{A}} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$*. Here,* $\overline{\mathcal{D}}$ *is a* $(P, T_1, T_2, T_{\mathrm{Const}})$*-distinguisher,* $\overline{\mathcal{S}}$ *is a* (P, T_1, T_2)*-simulator,* $\overline{\mathcal{A}}$ *is a* (P, T_1, T_2)*-adversary and* T_{Gen} *is the total number of primitive queries made by the construction oracle.*

We present the simulators $\overline{\mathcal{S}} = (\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2)$ in Fig. 6.12.

Proof. G_0: This game is the Real-sa-bf-Indiff game in Fig. 6.11, in which H is lazily sampled.

G_1: This game sets bad_1 and aborts when distinguisher $\mathcal{D}_1$ queries any point fixed in L. That is, for each query x, if there is a point $(x, y) \in L$, the game sets bad_1 and aborts. Ruling out this event ensures that all $\mathcal{D}_1$ queries and their responses are independent of the points in L. The games are identical until bad_1 and so, $\Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1] \leq \Pr[\mathrm{G}_1 \text{ sets } \mathsf{bad}_1]$. Since $\mathcal{D}_1$ makes at most T_1 queries to the primitive oracle, and there are P pre-fixed points in L, we have $\Pr[\mathrm{G}_1 \text{ sets } \mathsf{bad}_1] \leq \frac{P \cdot T_1}{M}$.

G_2: In this game, if either $\mathcal{D}_0$ fixes some point x, or $\mathcal{D}_1$ queries some x to Prim, and $\mathcal{D}_2$ later queries some x' to the construction oracle Const such that the evaluation of $\mathsf{C}^{\mathsf{H}}(sa, x')$ also queries x to H, we set bad_2 and abort. Ruling out this event ensures that all points sampled during the evaluation of the construction queries are independent of both the prefixed points in L and the queries made by $\mathcal{D}_1$ (and therefore fresh). Thus, $\Pr[\mathrm{G}_2] - \Pr[\mathrm{G}_1] \leq \Pr[\mathrm{G}_2 \text{ sets } \mathsf{bad}_2]$. We bound the probability of bad_2 via a reduction to a salt propagation game. In order to account for the prefixed points, we augment the SaProp game to allow for an offline adversary. We leave the proof to Claim 14 where we introduce the augmented game (in Fig. 6.13) and by a reduction, we show that there exists a (P, T_1, T_2)-query adversary $\overline{\mathcal{A}} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ such that $\Pr[\mathrm{G}_2 \text{ sets } \mathsf{bad}_2] \leq \mathbf{Adv}^{\mathsf{bf\text{-}saprop}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{A}}}$.

G_3: This game sets a bad flag bad_3 if the *response* to any query indexed with $i < n$ has already been used as part of an *input* to a query indexed with any $j \in \mathbf{Ch}(i)$, or if there is a collision in the hash outputs. By ruling out this event, we ensure that no two computations of the construction on distinct inputs collide. Note that this includes both the computations evaluated for the construction oracle queries, *and* those evaluated with the primitive oracle. Games G_2 and G_3 are identical until bad_2. In the game there are a total of at most $T_1 + T_2 + T_{Gen}$ queries to the primitive, and so we have $\Pr[G_2] - \Pr[G_3] \leq \Pr[G_3 \text{ sets } \mathsf{bad}_3] \leq \frac{(T_1+T_2+T_{Gen})^2}{N}$.

G_4: This game sets bad_4 when $\mathcal{D}_2$ makes a primitive query containing a label generated during by the construction oracle, without first completing the computation from the corresponding input, and then aborts after bad_4 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 $\mathbf{lab}(\mathbf{Pa}_i(v))$. By ruling out the bad_3 and bad_2 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 input-salt pair (sa, x) queries by $\mathcal{D}_2$ to the construction oracle, the queries made by the game to evaluate $\mathsf{C}^{\mathsf{H}}(sa, x)$ assign to each node v a label l. If $\mathcal{D}_2$ makes a query containing the label l, without fully computing the $\mathsf{C}^{\mathsf{H}}(sa, x)$ up to the corresponding node v, then bad_4 is set. We show in the construction point prediction claim below that $\Pr[G_4] - \Pr[G_3] \leq \Pr[G_4 \text{ sets } \mathsf{bad}_4] \leq T_{Gen}T_2/N$.

G_5: In this game, the response to any $\mathcal{D}_2$ primitive query that completes a full computation, is generated by querying the construction oracle. Whenever $\mathcal{D}_2$ makes a primitive a query indexed with n, i.e., potentially labeling the final node in the construction, the game runs

the FindChain function. This function checks previous primitive queries made by $\mathcal{D}_2$, to check if the query in question completes a full instance of the construction from a *salted* input. In the case where the query completes a chain, beginning with a salt $sa \in \mathbf{sa}$, the game makes a construction query for the input found by FindChain and forwards the response. That is, where FindChain finds a completed computation from an input (sa, x), for $sa \in \mathbf{sa}$, the game responds with $\mathsf{C}^{\mathsf{H}[L]}(sa, x)$. Note that as C^{H} has a single sink, the check only needs to be made at the end of any potential computation. The games are identical and so $\Pr[\mathrm{G}_5] = \Pr[\mathrm{G}_4]$.

G_6: In this game, we no longer store the intermediate points generated in the challenge phase, and if $\mathcal{D}_2$ uses the primitive oracle to evaluate the construction on the same input it queries to the construction oracle directly, the points are freshly sampled. Until a query from $\mathcal{D}_2$ resamples these intermediate points, the values are information theoretically hidden from the distinguisher. Thus: $\Pr[\mathrm{G}_6] = \Pr[\mathrm{G}_5]$.

G_7: In this game, if $\mathcal{D}_2$ queries the primitive oracle with a salted input, i.e. the start of a potential chain, the response is sampled independently of the prefixed list L. For all other intermediate points the game generates responses with $\mathsf{H}[L](x)$. As neither the prefixed points in L, nor the queries of $\mathcal{D}_1$ contain any salt $sa \in \mathbf{sa}$, the responses are consistent with both L and $\mathcal{D}_1$. Due to the FindChain function introduced in G_5, the responses remain consistent with $\mathcal{D}_2$ construction queries and so $\Pr[\mathrm{G}_7] = \Pr[\mathrm{G}_6]$.

G_8: In this game, we make three conservative changes. Firstly, we swap the random function from H, to a random oracle RO sampled from $\mathrm{Fun}(N', M')$. The preprocessing advice is then generated by a simulator $\mathcal{S}_0$ which has access to the whole of RO. To generate the advice, $\mathcal{S}_0$ runs $\mathcal{D}_0$, and to generate the list L it uses the Decomp algorithm from Theorem 5. The next change is to the way the primitive queries are sampled. For queries from $\mathcal{D}_1$ and $\mathcal{D}_2$ to the oracle $\mathsf{H}[L]$, responses are now sampled according to $\mathsf{RO}[L]$, Note that as C^{H} is a single sink graph-based construction, the ranges of the random oracles are the same, i.e., $M = M'$. The final change is that the responses to $\mathcal{D}_2$'s construction queries are now sampled according to $\mathsf{RO}[L]$.

As this is the Ideal-sa-bf-Indiff game with simulators described in Fig. 6.12, by collecting terms, we obtain the bound in the theorem statement.

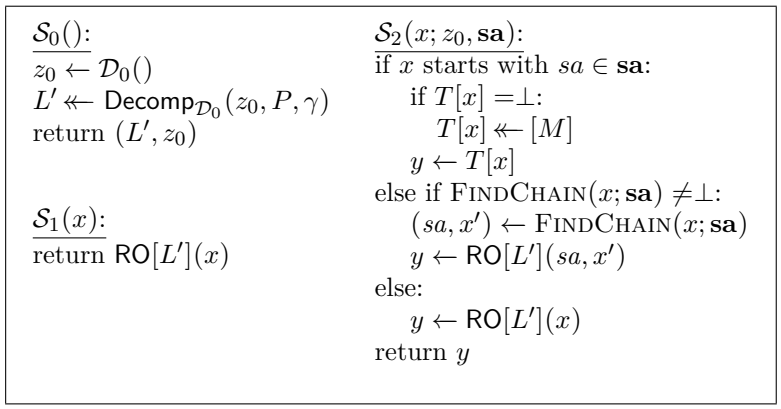

Figure 6.12: pseudocode for the simulators $\overline{\mathcal{S}} = (\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2)$, in the proof of Theorem 10

□

Claim 13 (Construction point prediction)**.** *For the parameters that we have above, we have that,*

$$\Pr[\mathsf{bad}_3 \text{ is set in } \mathrm{G}_4] \leq \frac{T_2 T_{Gen}}{N} \ .$$

Proof. For each node $i \in [1, n]$, the queries to the construction oracle create T_{Gen} labels. As there are no collisions in the queries, each label is unique and sampled uniformly at random. Consider the labels for the node with index $n-1$. Suppose $\mathcal{D}_2$ does not compute the construction from an input to node $n-1$, but still makes a query containing the label for this node with respect to one of the T_{Gen} inputs. As there are no collisions in $\mathcal{D}_2$'s primitive queries, $\mathcal{D}_2$ either obtained the label by making a query with the labels for the *parents* of node $n-1$, with respect to the same input, or guessed the output of an unqueried point. The probability $\mathcal{D}_2$ can guess the label for an unqueried point indexed by i (with respect to any of the T_{Gen} inputs) is at most $q_i T_{Gen}/N$. By induction over the node parents, we have $\Pr[\mathsf{bad} \text{ is set in } \mathrm{G}_4] \leq \sum_{i=1}^{n} q_i T_{Gen}/N \leq \frac{T_2 T_{Gen}}{N}$. □

Remark. For this result, we can guarantee the comparability of the attack given by the composition theorem. Note that as the simulator $\mathcal{S}_1$ perfectly simulates H for $\mathcal{D}_1$, the combined system of the adversary $\mathcal{A}_1$ and simulator $\mathcal{S}_1$, which defines $\mathcal{B}_1$, will sample points with exactly the same unpredictability.

$\underline{\text{bf-SaProp}^{\mathcal{A}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen}}}$
$\mathsf{H} \twoheadleftarrow \text{Fun}(N, M)$
$(L, z_0) \leftarrow \mathcal{A}_0(\mathsf{H})$
$z_1 \twoheadleftarrow \mathcal{A}_1^{\text{Prim}_1}()$
for $i \in [m]$:
  $\mathbf{sa}[i] \twoheadleftarrow \mathsf{Gen}()$
$z_2 \twoheadleftarrow \mathcal{A}_2^{\text{Prim}_2,\text{Const}}(z_0, z_1, \mathbf{sa})$
return $(Q_{\mathcal{A}} \cap Q_{\mathsf{C}} \neq \emptyset) \vee (\{L\} \cap Q_{\mathsf{C}} \neq \emptyset)$

$\underline{\text{Proc. Prim}_1(x)}$
$Q_{\mathcal{A}} \leftarrow Q_{\mathcal{A}} \cup \{x\}$
return $\mathsf{H}(x)$

$\underline{\text{Proc. Prim}_2(x)}$
return $\mathsf{H}(x)$

$\underline{\text{Proc. Const}(sa, x)}$
if $sa \notin \mathbf{sa}$: return $\perp$
return $\mathsf{C}^{\text{Prim}_3}(sa, x)$

$\underline{\text{Proc. Prim}_3(x)}$
$Q_{\mathsf{C}} \leftarrow Q_{\mathsf{C}} \cup \{x\}$
return $\mathsf{H}(x)$

Figure 6.13: The salt propagation game with respect to the construction C^{H} and salt generator Gen, in the presence of prefixed points.

Salt propagation with preprocessing

Here we describe the bit-fixing salt propagation game bf-SaProp, which we put to use as part of the proof for Theorem 10. The pseudocode is presented in Fig. 6.13. Essentially this game is identical to the SaProp game from Fig. 6.8, except for an offline adversary which fixes a list of L points. For an adversary $\overline{\mathcal{A}} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, a salted construction C^{H} and salt generator Gen, we define the advantage in this game as

$$\mathbf{Adv}^{\mathsf{bf\text{-}saprop}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{A}}} = \Pr\left[\text{bf-SaProp}^{\overline{\mathcal{A}}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen}}\right] .$$

For adversaries in the bf-SaProp game, we say that if $\mathcal{A}_0$ prefixes a list of size P, $\mathcal{A}_1$ makes at most q queries to Prim_1 and $\mathcal{A}_2$ makes at most q_{C} queries to Const, then $\overline{\mathcal{A}} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ is a (P, q, q_{C})-query adversary.

Below, we give a reduction from the game G_2 in Theorem 10 to the bf-SaProp game to bound the probability of bad_2 event occurring in G_1.

Claim 14 (Bit-fixing salt propagation)**.** *Let C^{H} be the salt propagating function G_2, let Gen be the salt generator, and let $\mathcal{D}_1$ and $\mathcal{D}_2$ be the distinguishers from G_2 in Theorem 10, then there exists an adversary $\overline{\mathcal{A}} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ such that*

$$\Pr[\text{G}_2 \textit{ sets } \mathsf{bad}_2] \leq \mathbf{Adv}^{\mathsf{bf\text{-}saprop}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{A}}} .$$

Intuitively, we adapt the proof of Claim 11 to account for the offline adversary and the list of prefixed points.

Proof. Let $\overline{\mathcal{A}}$ be an adversary in the salt propagation game with respect to the salt generator Gen and construction C^{H}. Let $\mathcal{A}_0$ run $\mathcal{D}_0$ to generate

the preprocessing information z_0 and the list L. Let $\mathcal{A}_1$ run $\mathcal{D}_1$, forwarding all $\mathcal{D}_1$'s queries to its own Prim_1 oracle and relaying the responses. Next, $\mathcal{A}_2$ receives the salt vector $\mathbf{sa}$, and advice (st, z_1) from $\mathcal{A}_1$. Then, $\mathcal{A}_2$ runs $\mathcal{D}_2$ on the input $(st, z_0, z_1, \mathbf{sa})$, forwarding all primitive queries to its Prim_2 oracle, and construction queries to its Const oracle. Note that $\mathcal{A}_0$ fixes at most P points in H, the online adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ makes T_1 queries to Prim_1 and at most T_2 queries to Const. If bad_2 is set, then the set of H queries made to evaluate the construction and either the set of prefixed points $\{L\}$, or the H queries made by $\mathcal{D}_1$, have at least one query in common. Therefore, either $Q_{\mathcal{A}} \cap Q_{\mathsf{C}} \neq \emptyset$, or $\{L\} \cap Q_{\mathsf{C}} \neq \emptyset$ and $\overline{\mathcal{A}}$ wins the salt propagation game.

□

Step up to general auxiliary input. With the bit-fixing bounds in hand, we can finally apply the Theorem 5 bound from [25]. This takes our bound in sa-bf-Indiff and gives a bound for general auxiliary input in sa-ai-Indiff. Note the $\mathbf{Adv}^{\mathsf{bf\text{-}saprop}}$ term still appears in this bound, and we address this in our application to MHFs – which appears in the next section.

Theorem 11 (sa-bf-Indiff $\implies$ sa-ai-Indiff)**.** *Let N, M, S, T_1, T_2, and P be natural numbers and γ be a real number, let C^{H} be a construction based on random oracle H, and let Gen be a salt generator. Then for any* sa-ai-Indiff *distinguisher $\mathcal{D} = (\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2)$ where $\mathcal{D}_0$ outputs S-bits of auxiliary information, and $\mathcal{D}_1$ and $\mathcal{D}_2$ make at most T_1 and T_2 queries to their oracles respectively, then there exists a* sa-bf-Indiff *distinguisher $(\tilde{\mathcal{D}}_0, \tilde{\mathcal{D}}_1, \tilde{\mathcal{D}}_2)$ where $\tilde{\mathcal{D}}_0$ outputs at most S-bits of auxiliary information and a list of size at most P, such that for any* sa-bf-Indiff *simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2)$ we have*

$$\mathbf{Adv}^{\mathsf{sa\text{-}ai\text{-}indiff}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\mathcal{D},\mathcal{S}} \leq \mathbf{Adv}^{\mathsf{sa\text{-}bf\text{-}indiff}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\tilde{\mathcal{D}},\mathcal{S}} + \frac{(S + \log \gamma^{-1})(T_1 + T_2 + T_{Gen})}{P} + 2\gamma \ .$$

Proof. Let $(\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2)$ be a sa-bf-Indiff distinguisher. Then we apply Theorem 5 to the Real-sa-ai-Indiff game to obtain a distinguisher $(\tilde{\mathcal{D}}_0, \tilde{\mathcal{D}}_1, \tilde{\mathcal{D}}_2)$ in the sa-bf-Indiff game. There are a total of $T_1 + T_2 + T_{Gen}$ queries to H in the real game. Let $\mathcal{S}$ be a simulator for $(\tilde{\mathcal{D}}_0, \tilde{\mathcal{D}}_1, \tilde{\mathcal{D}}_2)$, then $\mathcal{S}$ is also a simulator in the sa-ai-Indifff game, as they are syntactically identical. □

6.3.4 Application to graph-based iMHFs

In order to apply this result to graph-based iMHFs as defined in Definition 5, we first derive an upper-bound for a (P, T_1, T_2)-adversary in the bf-SaProp game. That is, we need to bound the additional probability that the set of salted construction queries Q_{C} intersects with the set of P pre-fixed points in L. Using the argument from Proposition 8, and the fact that the salts are unpredictable, it is clear from inspection that we need only bring in two more terms: one to bound the probability the P points intersect with the m salts, and one to bound the probability they intersect with the queries in T_{Gen}. This gives the bound

$$\mathbf{Adv}^{\mathsf{bf\text{-}saprop}}_{\mathrm{Const}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{A}}} \leq \frac{T_{Gen}P}{M} + \frac{mP}{K} + \frac{mT_1}{K} + \frac{mT_{Gen}T_1}{M} \ .$$

With this bound in hand, we can combine Theorem 10 and Theorem 11, to get a final bound of the form

$$\mathbf{Adv}^{\mathsf{sa\text{-}ai\text{-}indiff}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\mathcal{D},\mathcal{S}} \leq \frac{P(T_{Gen}+T_1)}{M} + \frac{mP}{K} + \frac{(S+\log\gamma^{-1})(T_1+T_2+T_{Gen})}{P} + \dots \ ,$$

where we collect terms with P. Let $S' := (S + \log\gamma^{-1})$ and $T' := (T_1 + T_2 + T_{Gen})$, and simplify to get the following terms involving P

$$\frac{P(KT_{Gen}+mM+KT_1)}{KM} + \frac{S'T'}{P} \ .$$

Let $\alpha := (KT_{Gen} + mM + KT_1)$, and then set the two terms involving P to be equal. This gives,

$$P = \sqrt{\frac{S'T'MK}{\alpha}} \ .$$

By substituting this into the bound in Theorem 11, we obtain

$$\mathbf{Adv}^{\mathsf{sa\text{-}ai\text{-}indiff}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\mathcal{D},\mathcal{S}} \leq \sqrt{\frac{\alpha S'T'}{KM}} + \frac{mT_1}{K} + \frac{mT_{Gen}T_1}{M} + \frac{(T_1+T_2+T_{Gen})^2}{N} + \frac{T_{Gen}T_2}{N} + 2\gamma \ .$$

Remark. Note, the term mT_1/K, which is the only one to have the denominator K, is not quadratic in adversarial queries. If this weren't the case, we would require salt sizes too big.

6.4 C-amplification and limitations

By slightly restricting the class of games we consider, we can extend Theorem 9 to attain c-amplification for a weak-indifferentiability simulator. To do so, we consider a version of the game where the first-stage simulator $\mathcal{S}_1$ can privately share an extra bit of information directly with the second-stage simulator $\mathcal{S}_2$. Clearly, this changes the range of the composition theorem. The following result will only apply to games $\overline{\mathcal{G}}$ where the first-stage game $\mathcal{G}_1$ preserves the integrity of the advice from A_1. That is, while $\mathcal{G}_1$ may add to the advice, the second stage adversary $\mathcal{A}_2$ must be able to obtain $\mathcal{A}_1$'s output from the advice z_1.

Thus, for any such *advice-preserving* multi-stage game $\overline{\mathcal{G}}$, and any adversary $\overline{\mathcal{A}} = (\mathcal{A}_1, \mathcal{A}_2)$ in the real world, we can use such a simulator to build an adversary $\overline{\mathcal{B}} = (\mathcal{B}_1, \mathcal{B}_2)$ where $\mathcal{B}_i := \mathcal{A}_i^{\mathcal{S}_i}$ for $i \in \{1, 2\}$. Here, $\mathcal{B}_1$ will return the advice strings of $\mathcal{A}_1$ and $\mathcal{S}_1$, where the advice of $\mathcal{S}_1$ is passed without change.

As the simulator $\mathcal{S}_1$ can share some additional advice bits with simulator $\mathcal{S}_2$ via the state of game $\mathcal{G}_1$. These restrictions on composition require some minor adjustments to the syntax of the sa-Indiff game. Next we introduce a new game with these tweaks.

Auxiliary sa-indifferentiability. To accommodate the necessary changes we define the sa-Indiff-aux game. This game is essentially the sa-Indiff game with the following alterations: After $\mathcal{D}_1$ has run, $\mathcal{S}_1$ outputs its simulator-specific information, which is passed directly to $\mathcal{S}_2$. We use s to denote the simulator-specific information shared between stages, The pseudocode for the game is presented in Fig. 6.14.

We then define the auxiliary sa-indifferentiability advantage in the following way.

Definition 11 (Auxiliary sa-indifferentiability)**.** *A construction* C^{H} *based on primitive* H *is auxiliary sa-indifferentiable from a random oracle* RO*, with respect to a salt generator* Gen*, if there exists a simulator* $\overline{\mathcal{S}}$ *such that for any distinguisher* $\overline{\mathcal{D}}$*, the following advantage is small.*

$$\mathbf{Adv}^{\mathsf{sa\text{-}indiff\text{-}aux}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{D}},\overline{\mathcal{S}}} := |\Pr\left[\textsc{Real}\text{-sa-Indiff-aux}^{\overline{\mathcal{D}}}_{\mathsf{Gen},\mathsf{C}^{\mathsf{H}}}\right] - \Pr\left[\textsc{Ideal}\text{-sa-Indiff-aux}^{\overline{\mathcal{D}},\overline{\mathcal{S}}}_{\mathsf{Gen},\mathsf{RO}}\right]| \,.$$

Where $\overline{\mathcal{D}} = (\mathcal{D}_1, \mathcal{D}_2)$ *and* $\overline{\mathcal{S}} = (\mathcal{S}_1, \mathcal{S}_2)$*.*

As in Definitions 8 and 10 we can reverse the order of quantifiers to

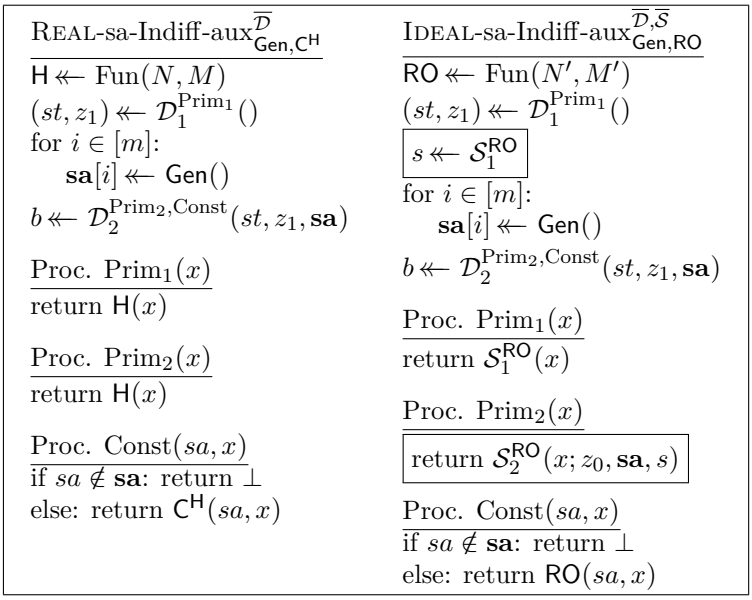

Figure 6.14: The salted two-stage indifferentiability notion, with private communication between the simulators, and with respect to $\overline{\mathcal{D}} = (\mathcal{D}_1, \mathcal{D}_2)$ and $\overline{\mathcal{S}} = (\mathcal{S}_1, \mathcal{S}_2)$. The alterations to the standard sa-Indiff game are boxed.

achieve a slightly weaker notion. That is, for any distinguisher $\overline{\mathcal{D}}$, there exists a simulator $\overline{\mathcal{S}}$ such that $\mathbf{Adv}^{\mathsf{sa\text{-}indiff\text{-}aux}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{D}},\overline{\mathcal{S}}}$ is small.

Next we formally define a class of games for which this indifferentiability notion achieves composition. As can be expected, this is a sub-class of the SSU games from Fig. 6.4. We present the pseudocode for these games in Fig. 6.15.

Informally, these games preserve some part of the state passed from $\mathcal{A}_1$ to $\mathcal{A}_2$. We label this part of the state s. Note that the second-stage adversary gets access to s directly, and it is out of the view of $\mathcal{G}_2$. The games are described in Fig. 6.15 where the alterations made to the SSU games are displayed in boxes.

Updates to the composition. Having made alterations to the class of games we consider, we again have to prove a composition theorem. While the theorem goes through essentially unchanged, it is presented in Theorem 12. We show auxiliary sa-indifferentiability implies that for any adversary $\overline{\mathcal{A}}$ in the REAL-SSU-aux games, there is an adversary $\overline{\mathcal{B}}$ in the IDEAL-SSU-aux games with a similar advantage.

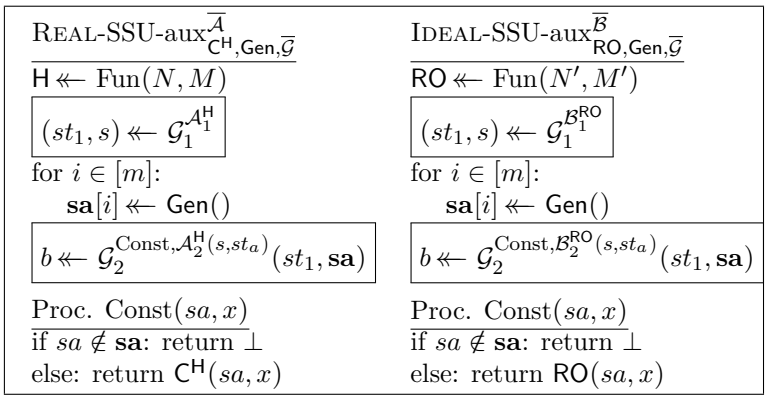

Figure 6.15: The pseudocode for the sample, salt, and use games, where $\overline{\mathcal{A}} = (\mathcal{A}_1, \mathcal{A}_2)$, $\overline{\mathcal{B}} = (B_1, \mathcal{B}_2)$, and $\overline{\mathcal{G}} = (\mathcal{G}_1, \mathcal{G}_2)$.

Theorem 12 (Composition for auxiliary sa-indifferentiability)**.** *Let $m \in \mathbb{N}$ and C^{H} be a salted construction which is auxiliary sa-indifferentiable from a random oracle RO with respect to a salt generator Gen which outputs m salts. Then for any multi-stage game $\overline{\mathcal{G}} = (\mathcal{G}_1, \mathcal{G}_2)$ and any multi-stage preprocessing adversary $\overline{\mathcal{A}} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a multi-stage preprocessing adversary $\overline{\mathcal{B}} = (\mathcal{B}_1, \mathcal{B}_2)$ and a distinguisher $\overline{\mathcal{D}} = (\mathcal{D}_1, \mathcal{D}_2)$ such that*

$$\Pr\left[\text{SSU-aux}^{\overline{\mathcal{A}}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{G}}}\right] \leq \Pr\left[\text{SSU-aux}^{\overline{\mathcal{B}}}_{\mathsf{RO},\mathsf{Gen},\overline{\mathcal{G}}}\right] + \mathbf{Adv}^{\mathsf{sa\text{-}indiff\text{-}aux}}_{\mathsf{C}^{\mathsf{H}},\mathsf{Gen},\overline{\mathcal{D}},\overline{\mathcal{S}}} .$$

Intuitively, given an $\overline{\mathcal{A}}$ and $\overline{\mathcal{G}}$ as defined above, we have to construct a $\overline{\mathcal{D}}$ and a $\overline{\mathcal{B}}$ which satisfy the inequality. We do this by defining a distinguisher from $\overline{\mathcal{A}}$ and $\overline{\mathcal{G}}$, and using the corresponding simulator $\overline{\mathcal{S}}$ to build the new adversary $\overline{\mathcal{B}}$.

Proof. Let $\overline{\mathcal{A}} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary in the multi-stage game $\overline{\mathcal{G}} = (\mathcal{G}_1, \mathcal{G}_2)$, then we show that there exists an adversary $\overline{\mathcal{B}} = (\mathcal{B}_1, \mathcal{B}_2)$ that satisfies the bound in the theorem statement.

First, we use $\overline{\mathcal{A}}$ to define the sa-Indiff-aux distinguisher $\overline{\mathcal{D}} = (\mathcal{D}_1, \mathcal{D}_2)$. Let $\mathcal{D}_1$ be the combined system of $\mathcal{G}_1$ running adversary $\mathcal{A}_1$, such that $\mathcal{D}_1$ forwards all of $\mathcal{A}_1$'s primitive queries to its own primitive oracle, relaying the responses. Then the state st_1 that $\mathcal{G}_1$ passes to $\mathcal{G}_2$, is passed by $\mathcal{D}_1$ to $\mathcal{D}_2$. Let the second stage distinguisher $\mathcal{D}_2$ be the combined system of the game $\mathcal{G}_2$ running with $\mathcal{A}_2$. Specifically, $\mathcal{D}_2$ receives the states s and st_1 from $\mathcal{G}_1$, and the public salt and uses it as input to $\mathcal{G}_2$. The distinguisher

$\mathcal{D}_2$ then forwards all $\mathcal{A}_2$'s primitive queries and $\mathcal{G}_2$'s construction queries to its primitive and construction oracles respectively, relaying the responses.

The construction C^{H} is indifferentiable as defined in Definition 11, and so there exists a pair of simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ for every $\mathcal{D}_1$ and $\mathcal{D}_2$ such that the distinguishing advantage is small. Let $\overline{\mathcal{S}} = (\mathcal{S}_1, \mathcal{S}_2)$ be the simulator (guaranteed by Definition 11) for $\overline{\mathcal{D}}$ in the sa-Indiff-aux game with respect to the same salt generator Gen and construction C^{H} from the statement. We use this simulator $\overline{\mathcal{S}}$, along with adversary $\overline{\mathcal{A}}$ to construct adversary $\overline{\mathcal{B}}$. Set $\mathcal{B}_1$ to be the joint system ${\mathcal{A}_1}^{\mathcal{S}_1}$, and $\mathcal{B}_2$ to be ${\mathcal{A}_2}^{\mathcal{S}_2}$. Then the difference in the advantages of $\overline{\mathcal{B}}$ and $\overline{\mathcal{A}}$ is exactly the distinguishing advantage between the real and ideal worlds, and by rearranging, we obtain the bound from the statement. □

Password leakage and applications.

While the restrictions in the sa-Indiff-aux game are unusual, they are justified through the applications. Consider a password sampler $\mathcal{P}^{\mathsf{H}}$ that samples a vector of passwords $\mathbf{pw}$ and leakage z. In both the UR and KD security games, the second stage adversary is given access to password leakage z. Indeed, the leakage is public as it is given to the second-stage adversary without alteration, even when (as in the UR game) the password vector $\mathbf{pw}$ is hidden. Applying Theorem 12 requires some interpretation. For any password sampler $\mathcal{P}^{\mathsf{H}}$, that produces a password vector $\mathbf{pw}$ and leakage z, and any adversary $\mathcal{A}^{\mathsf{H}}$ in the UR game, there exists a sampler $\mathcal{Q}^{\mathsf{RO}}$ which produces a password vector $\mathbf{pw}'$ and leakage z', where z' is z, and an adversary $\mathcal{B}^{\mathsf{RO}}$ which can obtain at least 1-bit of private leakage, i.e., leakage outside of the view of the games $\mathcal{G}_1$ and $\mathcal{G}_2$. Any SSU games where the sampler models some values drawn from sources or distributions for which an adversary may feasibly derive some side-information. One example is the multi-instance version of the hiding security game for commitment schemes, which one may use to secure a digital auction. Here, the sampler can model the bids each of the parties commit to, and there are many ways to derive contextual side information on the distribution of the bidding values, e.g., expecting higher bids from parties with greater need for an item.

Patterns which do *not* fit this restricted class are those where any side information is unhelpful or impossible. For example, the binding security game for commitment schemes using these MHFs. Here, the adversary does not benefit from side information on the distribution of bids, as their win condition relies on finding a collision in the construction.

Derandomisation with c-amplification.

By limiting the scope of the composition theorem, we are able to derandomise the simulators without 'blowing up' the simulator query complexity. Essentially, by putting to work the channel of communication guaranteed to the simulators, we can use a recent technique developed [23] to derandomise without excessive queries. The technique allows one to lift an indifferentiability result from a setting where the simulators have shared access to randomness, to one in which they can share a single bit between stages. With this, the result is also weakened to weak-indifferentiability, where the simulator provided depends on the distinguisher.

Note that in the proof of Theorem 9, we provide a simulator which makes at least $q + 1$ queries to the random oracle (to generate some shared randomness), and then an additional query for each completed chain. We want to improve this so that the simulator only needs to make a single query per completed chain, as this would give c-amplification.

Shared randomness model. We first consider a model where the simulators have access to enough shared randomness to lazily sample a random oracle. Since the randomness is outside the view of the distinguisher, it will allow the simulation of primitive responses to be consistent across rounds. We will then adapt and apply the derandomisation technique of [23] to remove this requirement at the cost of a single bit of private information shared between the stages of the simulators. While this technique was introduced for a preprocessing indifferentiability setting, the design of our SSU-aux games allows us to apply it here.

Query-constrained derandomisation. Given a multi-stage simulator $(\mathcal{S}_1^*, \mathcal{S}_2^*)$ such that $\mathcal{S}_1^*$ and $\mathcal{S}_2^*$ both share access to the same randomness, we build simulators $(\mathcal{S}_1, \mathcal{S}_2)$ which don't share randomness. We adapt Lemma 2 from [23], to get the following statement.

Lemma 8. *Suppose* C^{H} *is c-amplifying sa-indifferentiable from a random oracle* RO *in the shared randomness model, with respect to a salt generator* Gen*. Then the construction* C^{H} *is weakly c-amplifying auxiliary sa-indifferentiable from a random oracle without shared randomness.*

The proof follows that of Lemma 2 in [23]. The difference lies in the changes to the games we consider, i.e., our auxiliary SSU notion, in which neither simulator is a preprocessing simulator. However, the games still

involve a private bit of information passed between the simulators, and so we can apply the technique of [23].

Proof. By assumption, for any distinguisher $\overline{\mathcal{D}} = (\mathcal{D}_1, \mathcal{D}_2)$ in the sa-indifferentiability game, with respect to salt generator Gen and construction C^{H}, there exists a c-amplifying simulator $\overline{\mathcal{S}}^* = (\mathcal{S}_1^*, \mathcal{S}_2^*)$ in the shared randomness model, such that the distinguishing advantage is negligible. Let the distinguishing advantage be ϵ, and let p be the probability that $\overline{\mathcal{D}}$ outputs 1 in the ideal world. We will build a c-amplifying simulator $\overline{\mathcal{S}}$ in the auxiliary sa-indifferentiability game, without shared randomness, such that $\overline{\mathcal{D}}$ will output 1 with probability p. We denote the shared randomness, hard-coded into both $\mathcal{S}_1^*$ and $\mathcal{S}_2^*$, by SR.

Then there are two cases. In the first case, for any $SR \in \{0,1\}^*$ the probability $\overline{\mathcal{D}}$ returns 1 is p. In the second case, there are two values SR_0 and SR_1 such that the probability $\overline{\mathcal{D}}$ returns 1, in either setting is p_0 and p_1 respectively, and $p_0 < p < p_1$.

In the first case, we can pick any value for the shared randomness SR, and set $\overline{\mathcal{S}} = \overline{\mathcal{S}}^*[SR]$. In the second case, we can hard-code both SR_0 and SR_1 into each stage of $\overline{\mathcal{S}}$. Then before $\mathcal{S}_1$ runs, it will sample a bit b such that,

$$\Pr_{\overline{\mathcal{S}}}[b = 1] = \frac{p - p_0}{p_1 - p_0} \ .$$

Then given the bit b, the simulator $\mathcal{S}_1$ will run $\mathcal{S}_1^*[SR_s]$, and set the private bit $s = b$, so that the second stage simulator can run $\mathcal{S}_2^*[SR_s]$. Note that this doesn't increase the query complexity of the simulator and so the c-amplification is preserved. Then the probability that the distinguisher $\overline{\mathcal{D}}$, interacting with simulator $\overline{\mathcal{S}}$, returns 1 is

$$p_0 \cdot \Pr_{\overline{\mathcal{S}}}[b = 0] + p_1 \cdot \Pr_{\overline{\mathcal{S}}}[b = 1] = p \ .$$

□

The shared randomness hard-coded in should be used to simulate responses with a secure PRF. As we don't consider adversaries with unbounded computation, we don't need to share enough to sample the whole random oracle. Importantly, we can observe that the $q + 1$ extra simulator queries in Theorem 9 are used to construct some shared randomness (outside of the view of the distinguisher). The above technique allows us to remove these queries, at the cost of two things: 1) a single bit of private information shared between the two online simulators, and 2) a simulator which depends on the distinguisher.

6.4.1 Bit-fixing and c-amplification.

Note that the simulator we provide for the sa-bf-Indiff game is not c-amplifying. Below we outline a simple attack on the c-amplification of a bit-fixing simulator in the sa-bf-Indiff game, and show that any successful simulator in the sa-bf-Indiff game *cannot* be c-amplifying.

Attack on c-amplification in the sa-bf-Indiff **game.** Let $\mathcal{D} = (\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2)$ be a distinguisher in the sa-bf-Indiff game. Then $\mathcal{D}_0$ fixes P points in the list L, and encodes them in z_0 for $\mathcal{D}_2$. The simulator $\mathcal{S}_2$ must remain consistent with each point $(x, y) \in L$, as $\mathcal{D}_2$ can query points in L. If $\mathcal{S}_2$ is inconsistent with the list L, the distinguisher $\mathcal{D}_2$ can detect this by consulting the preprocessing. Without access to L, for any unsalted query x', the simulator $\mathcal{S}_2$ cannot tell whether $x' \in L$ or not. As the simulator $\mathcal{S}_2$ does not have direct access to the list L, and only sees the encoded z_0, for each query x such that $(x, y) \in L$, the simulator must query $\mathsf{H}[L](x)$ to find y. There is no other way for $\mathcal{S}_2$ to efficiently recover L because the Decomp algorithm is potentially unbounded in its time complexity, and so running Decomp on the preprocessing z_0 is out of the question for meaningful security bounds. Therefore $\mathcal{S}_2$ must forward all such queries to the RO, which results in a RO query for every query it receives, and at least n queries for each instance $\overline{\mathcal{D}}$ computes.

6.5 Conclusion

This chapter is somewhat of a departure from the earlier ones, and although it continues the study of iMHFs, the techniques it uses are substantially different. We asked the question: in what scenarios can we securely swap out the random oracle for an iMHF?

By introducing a salt propagation game and a suitable indifferentiability notion, we were able to extend the research of [45] to show that salt-propagating indifferentiable constructions, are secure in the sa-Indiff game with respect to an indifferentiable salt generator. This offers a more modular approach to proving multi-stage indifferentiability, as it doesn't rely on the definitional overhead of the [45] framework. Moreover, due to the single-stage indifferentiability results of [7], our result guarantees security for secrets sampled *hash-dependently*, when stored with salted graph-based iMHFs – a class of function not covered by [45]. In this way, it extends both previous works.

By introducing the SSU framework, we were able to prove security bounds in a wide class of games when instantiated with graph-based iMHFs. Firstly, by proving composition in all SSU games, we showed that security in the sa-ai-Indiff game implies one can securely swap-out the RO for an iMHF in any auxiliary-input sample, salt, & use game. In particular, we showed that this includes the AI-UR game – which models password storage security against offline time-memory trade-offs. We also gave a definition of AI-KDF games, which fall under this category, and while this definition is different from the super-ideal definition of [30], it still captures an idealised security setting for key-derivation.

To prove these bounds we used the bit-fixing technique of [25]. This necessitated introducing the bit-fixing version of the auxiliary-input games, and using the game-hopping techniques to prove security bounds there. Using the main result of [25], we lifted this bound from the bit-fixing setting to the auxiliary-input setting.

Finally, we made some progress in optimising the simulators. The simulators we produced in Theorems 9 and 10 were not c-amplifying, i.e., they made many more queries to the random oracle than the number of full computations completed by the adversary.

In Theorem 9 the simulator was derandomised by generating a shared key. To do so, each stage of the simulator made more queries to the random oracle than the total number of adversarial queries. If the initial single-stage simulator was c-amplifying, then the multi-stage simulator generated in Theorem 9 will not preserve this. In Theorem 10, the simulator roughly has one RO query for every primitive query of the adversary, which is clearly not c-amplifying.

As the motivation behind MHFs is to impede the adversary from computing too many instances, c-amplification is clearly important. Intuitively, the indifferentiability result formalises the idea that for every attack on the construction, there is a similar attack on the monolithic random oracle. If the simulator is not c-amplifying, then we say that for an attacker against the construction, there is an attacker against the random oracle with similar advantage, but that makes many more full computations of the RO; we have a weaker statement.

To bridge this gap, we define an indifferentiability notion in which the simulators can communicate an extra bit of information privately. This allows us to adapt a derandomisation technique from the recent work of Carolan, Poremba, and Zhandry [23]. Adapting the argument, we are able to show that if a construction can be shown to be sa-indifferentiable with shared randomness, then it is weakly sa-indifferentiable without

shared randomness, when the simulators can communicate an extra bit of information. By giving a composition theorem, we show that this captures a class of games which communicate some information between the adversaries untouched; in fact, we motivate this with the example of an extra bit of private password leakage.

This means, in the multi-stage setting, we can retain simulator efficiency, and thus c-amplification, if we accept a weak indifferentiability simulator, and a slightly restricted composition theorem. Any extra queries needed for derandomisation are no longer needed. We concluded with an attack to show that, at least via the bit-fixing technique, one cannot achieve c-amplification for the sa-ai-Indiff simulators.

7 Conclusion

In conclusion, we have progressed towards answering two important questions in the security of password storage and memory hard functions. Firstly, we wanted to study the unrecoverability advantage for hash-dependent passwords stored under graph-based memory hard functions. In particular we wanted to relate the unrecoverability of the passwords to the CMC of the attacking adversary, rather than mere query complexity. This required extensions to both FT21 and AS15. Both extensions were in some sense hash-dependent, but each in a different way. To the FT21 framework, we introduced the hash-dependent password sampler, and provided reduction proofs to verify it as a sound addition, i.e., even with a hash-dependent sampler one can move from unrecoverability to unguessability. This addition was to give a treatment of hash-dependent password samplers. To AS15, we introduced hash-dependent adversaries, and provided a proof to guarantee that the graph-based iMHFs from the AS15 labelling, remain memory hard in this setting. This was not to introduce hash-dependent samplers directly, but to more closely model adversaries computing the iMHF in all security games – they may pick inputs adaptively. In particular it allows us to give a reduction from the unrecoverability setting, and bound the total number of computations an adversary can make of the iMHF.

The consequence of this was a combined framework that allowed us to prove unrecoverability bounds in terms of the CMC available to an adversary. We were able to conclude that, even for adaptive adversaries, and even for hash-dependent passwords, the required work scales roughly linearly in the number of instances recovered. This shows that graph-based iMHFs are a secure method of password storage.

Next we extended the analysis to an indifferentiability setting. We defined a multi-stage preprocessing indifferentiability game under which we could analyse the security of graph-based iMHFs. The notion is defined such that it captures a class of games we call SSU, which contains a KDF game as well as a preprocessing unrecoverability game.

We introduced the notion of salt propagation in the SaProp game to capture a sufficient condition for lifting single-stage indifferentiability

results to our multi-stage setting (without preprocessing). We then showed that graph-based iMHFs (with a single source and single sink) are multi-stage indifferentiable with auxiliary input. Observing that a c-amplifying simulator is out of the question in the presence of preprocessing, we gave a restricted SSU definition to explore under what conditions a c-amplifying simulator is achievable. This requires a single bit of private information shared between the indifferentiability simulators. We argue that this gives composition in games which preserve, and ignore, at least one bit of adversarial information. For example, in the UR game, the leakage on the password distribution is essentially ignored by the game.

Future work. There are a number of interesting future directions. Firstly, it is equally important to derive concrete unrecoverability bounds (in a multi-instance setting) for other classes of memory-hard functions. In particular, Scrypt is a key example of data-dependent memory hard functions, for which recent work proved tight lower bounds on required CMC. Both extending these results (to the stronger hash-dependent notion), and deriving the unrecoverability bounds are interesting and important open problems.

Other questions are raised by the second set of contributions. One important direction would be to consider graph-based iMHFs instantiated with different underlying primitives, not just permutations, or compression functions, but other constructions, i.e., Merkle Dagård-like iteration. Especially useful would be a modular way to show that some construction makes a good replacement for a random oracle in the construction of an iMHF.

Another interesting angle would be to extend the indifferentiability analyses in this thesis to dMHFs like Scrypt.

The possibility of a c-amplifying simulator for any class of MHFs in the auxiliary-input setting ought to be explored. While this is ruled out with the bit-fixing techniques, other proof techniques may shed light on the issue.

Bibliography

[1] B Allbery. '`pwgen`: random but pronounceable password generator'. In: *USENET posting in comp. sources. misc* (1988).

[2] Joël Alwen and Jeremiah Blocki. 'Efficiently Computing Data-Independent Memory-Hard Functions'. In: *Advances in Cryptology – CRYPTO 2016, Part II*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9815. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2016, pp. 241–271. DOI: 10.1007/978-3-662-53008-5_9.

[3] Joël Alwen, Jeremiah Blocki and Krzysztof Pietrzak. 'Depth-Robust Graphs and Their Cumulative Memory Complexity'. In: *Advances in Cryptology – EUROCRYPT 2017, Part III*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10212. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, Apr. 2017, pp. 3–32. DOI: 10.1007/978-3-319-56617-7_1.

[4] Joël Alwen, Binyi Chen, Krzysztof Pietrzak, Leonid Reyzin and Stefano Tessaro. 'Scrypt Is Maximally Memory-Hard'. In: *Advances in Cryptology – EUROCRYPT 2017, Part III*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10212. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, Apr. 2017, pp. 33–62. DOI: 10.1007/978-3-319-56617-7_2.

[5] Joël Alwen and Vladimir Serbinenko. 'High Parallel Complexity Graphs and Memory-Hard Functions'. In: *47th Annual ACM Symposium on Theory of Computing*. Ed. by Rocco A. Servedio and Ronitt Rubinfeld. Portland, OR, USA: ACM Press, June 2015, pp. 595–603. DOI: 10.1145/2746539.2746622.

[6] Joël Alwen and Vladimir Serbinenko. 'High parallel complexity graphs and memory-hard functions'. In: *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. 2015, pp. 595–603.

[7] Joël Alwen and Björn Tackmann. 'Moderately Hard Functions: Definition, Instantiations, and Applications'. In: *TCC 2017: 15th Theory of Cryptography Conference, Part I*. Ed. by Yael Kalai and Leonid Reyzin. Vol. 10677. Lecture Notes in Computer Science. Baltimore, MD, USA: Springer, Heidelberg, Germany, Nov. 2017, pp. 493–526. DOI: 10.1007/978-3-319-70500-2_17.

[8] Daniel V. Bailey, Markus Dürmuth and Christof Paar. 'Statistics on Password Re-use and Adaptive Strength for Financial Accounts'. In: *Security and Cryptography for Networks*. Ed. by Michel Abdalla and Roberto De Prisco. Cham: Springer International Publishing, 2014, pp. 218–235. ISBN: 978-3-319-10879-7.

[9] Mihir Bellare, David Pointcheval and Phillip Rogaway. 'Authenticated Key Exchange Secure against Dictionary Attacks'. In: *Advances in Cryptology – EUROCRYPT 2000*. Ed. by Bart Preneel. Vol. 1807. Lecture Notes in Computer Science. Bruges, Belgium: Springer, Heidelberg, Germany, May 2000, pp. 139–155. DOI: 10.1007/3-540-45539-6_11.

[10] Mihir Bellare, Thomas Ristenpart and Stefano Tessaro. 'Multi-instance Security and Its Application to Password-Based Cryptography'. In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2012, pp. 312–329. DOI: 10.1007/978-3-642-32009-5_19.

[11] Mihir Bellare, Thomas Ristenpart and Stefano Tessaro. 'Multi-instance Security and Its Application to Password-Based Cryptography'. In: *Advances in Cryptology - Crypto 2012*. Vol. 7417. Lecture Notes in Computer Science. Springer, 2012, pp. 312–329. DOI: 10.1007/978-3-642-32009-5_19. URL: https://www.iacr.org/archive/crypto2012/74170310/74170310.pdf.

[12] Mihir Bellare and Phillip Rogaway. 'Random Oracles are Practical: A Paradigm for Designing Efficient Protocols'. In: *ACM CCS 93: 1st Conference on Computer and Communications Security*. Ed. by Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu and Victoria Ashby. Fairfax, Virginia, USA: ACM Press, Nov. 1993, pp. 62–73. DOI: 10.1145/168588.168596.

[13] Alex Biryukov, Daniel Dinu and Dmitry Khovratovich. *Fast and Tradeoff-Resilient Memory-Hard Functions for Cryptocurrencies and Password Hashing*. Cryptology ePrint Archive, Report 2015/430. https://eprint.iacr.org/2015/430. 2015.

[14] Alex Biryukov, Daniel Dinu and Dmitry Khovratovich. 'Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications'. In: *IEEE European Symposium on Security and Privacy (EuroS&P)*. 2016, pp. 292–302. DOI: 10.1109/EuroSP.2016.31.

[15] Jeremiah Blocki, Ben Harsha and Samson Zhou. *On the Economics of Offline Password Cracking*. 2020. arXiv: 2006.05023 [cs.CR]. URL: https://arxiv.org/abs/2006.05023.

[16] Jeremiah Blocki, Benjamin Harsha, Siteng Kang, Seunghoon Lee, Lu Xing and Samson Zhou. 'Data-Independent Memory Hard Functions: New Attacks and Stronger Constructions'. In: *Advances in Cryptology – CRYPTO 2019, Part II*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11693. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2019, pp. 573–607. DOI: 10.1007/978-3-030-26951-7_20.

[17] Jeremiah Blocki, Blake Holman and Seunghoon Lee. 'The Parallel Reversible Pebbling Game: Analyzing the Post-quantum Security of iMHFs'. In: *TCC 2022: 20th Theory of Cryptography Conference, Part I*. Ed. by Eike Kiltz and Vinod Vaikuntanathan. Vol. 13747. Lecture Notes in Computer Science. Chicago, IL, USA: Springer, Heidelberg, Germany, Nov. 2022, pp. 52–79. DOI: 10.1007/978-3-031-22318-1_3.

[18] Jeremiah Blocki, Blake Holman and Seunghoon Lee. *The Impact of Reversibility on Parallel Pebbling*. Cryptology ePrint Archive, Paper 2024/334. https://eprint.iacr.org/2024/334. 2024. URL: https://eprint.iacr.org/2024/334.

[19] Jeremiah Blocki and Samson Zhou. 'On the Depth-Robustness and Cumulative Pebbling Cost of Argon2i'. In: *TCC 2017: 15th Theory of Cryptography Conference, Part I*. Ed. by Yael Kalai and Leonid Reyzin. Vol. 10677. Lecture Notes in Computer Science. Baltimore, MD, USA: Springer, Heidelberg, Germany, Nov. 2017, pp. 445–465. DOI: 10.1007/978-3-319-70500-2_15.

[20] Dan Boneh, Henry Corrigan-Gibbs and Stuart E. Schechter. 'Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks'. In: *Advances in Cryptology – ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. Lecture Notes in Computer Science. Hanoi, Vietnam: Springer, Heidelberg, Germany, Dec. 2016, pp. 220–248. DOI: 10.1007/978-3-662-53887-6_8.

[21] Joseph Bonneau and Ilya Mironov. 'Cache-Collision Timing Attacks Against AES'. In: *Cryptographic Hardware and Embedded Systems – CHES 2006*. Ed. by Louis Goubin and Mitsuru Matsui. Vol. 4249. Lecture Notes in Computer Science. Yokohama, Japan: Springer, Heidelberg, Germany, Oct. 2006, pp. 201–215. DOI: 10.1007/11894063_16.

[22] Carlo Brunetta, Hans Heum and Martijn Stam. 'Multi-instance Secure Public-Key Encryption'. In: *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part II*. Ed. by Alexandra Boldyreva and Vladimir Kolesnikov. Vol. 13941. Lecture Notes in Computer Science. Atlanta, GA, USA: Springer, Heidelberg, Germany, May 2023, pp. 336–367. DOI: 10.1007/978-3-031-31371-4_12.

[23] Joseph Carolan, Alexander Poremba and Mark Zhandry. *(Quantum) Indifferentiability and Pre-Computation*. Cryptology ePrint Archive, Paper 2024/1727. 2024. URL: https://eprint.iacr.org/2024/1727.

[24] Binyi Chen and Stefano Tessaro. 'Memory-Hard Functions from Cryptographic Primitives'. In: *Advances in Cryptology – CRYPTO 2019, Part II*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11693. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2019, pp. 543–572. DOI: 10.1007/978-3-030-26951-7_19.

[25] Sandro Coretti, Yevgeniy Dodis, Siyao Guo and John P. Steinberger. 'Random Oracles and Non-uniformity'. In: *EUROCRYPT (1)*. Springer, 2018, pp. 227–258. DOI: 10.1007/978-3-319-78381-9_9.

[26] Sandro Coretti, Yevgeniy Dodis, Siyao Guo and John P. Steinberger. 'Random Oracles and Non-uniformity'. In: *Advances in Cryptology – EUROCRYPT 2018, Part I*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10820. Lecture Notes in Computer Science. Tel Aviv, Israel: Springer, Heidelberg, Germany, Apr. 2018, pp. 227–258. DOI: 10.1007/978-3-319-78381-9_9.

[27] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov and XiaoFeng Wang. 'The tangled web of password reuse.' In: *NDSS*. Vol. 14. 2014. 2014, pp. 23–26.

[28] Charles Dodd, Pooya Farshim, Siamak F. Shahandashti and Karl Southern. *Multi-Instance Unrecoverability of iMHF-Based Password Hashing*. Cryptology ePrint Archive, Paper 2026/018. 2026. URL: https://eprint.iacr.org/2026/018.

[29] Yevgeniy Dodis, Thomas Ristenpart and Thomas Shrimpton. 'Salvaging Merkle-Damgård for Practical Applications'. In: *Advances in Cryptology – EUROCRYPT 2009*. Ed. by Antoine Joux. Vol. 5479. Lecture Notes in Computer Science. Cologne, Germany: Springer, Heidelberg, Germany, Apr. 2009, pp. 371–388. DOI: 10.1007/978-3-642-01001-9_22.

[30] Pooya Farshim and Stefano Tessaro. 'Password Hashing and Preprocessing'. In: Springer-Verlag, 2021. DOI: 10.1007/978-3-030-77886-6_3.

[31] Pooya Farshim and Stefano Tessaro. 'Password Hashing and Preprocessing'. In: *Advances in Cryptology – EUROCRYPT 2021, Part II*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12697. Lecture Notes in Computer Science. Zagreb, Croatia: Springer, Heidelberg, Germany, Oct. 2021, pp. 64–91. DOI: 10.1007/978-3-030-77886-6_3.

[32] Christian Forler, Stefan Lucks and Jakob Wenzel. *Catena: A Memory-Consuming Password Scrambler*. Cryptology ePrint Archive, Report 2013/525. https://eprint.iacr.org/2013/525. 2013.

[33] Christian Forler, Stefan Lucks and Jakob Wenzel. *Catena: A Memory-Consuming Password-Scrambling Framework*. Cryptology ePrint Archive, Paper 2013/525. 2013. URL: https://eprint.iacr.org/2013/525.

[34] Christian Forler, Stefan Lucks and Jakob Wenzel. 'Memory-Demanding Password Scrambling'. In: *Advances in Cryptology – ASIACRYPT 2014, Part II*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8874. Lecture Notes in Computer Science. Kaoshiung, Taiwan, R.O.C.: Springer, Heidelberg, Germany, Dec. 2014, pp. 289–305. DOI: 10.1007/978-3-662-45608-8_16.

[35] Eran Gabber, Phillip B. Gibbons, Yossi Matias and Alain J. Mayer. 'How to Make Personalized Web Browising Simple, Secure, and Anonymous'. In: *FC'97: 1st International Conference on Financial Cryptography*. Ed. by Rafael Hirschfeld. Vol. 1318. Lecture Notes in Computer Science. Anguilla, British West Indies: Springer, Heidelberg, Germany, Feb. 1997, pp. 17–32.

[36] Ashrujit Ghoshal and Stefano Tessaro. 'The Query-Complexity of Preprocessing Attacks'. In: *Advances in Cryptology – CRYPTO 2023, Part II*. Ed. by Helena Handschuh and Anna Lysyanskaya. Vol. 14082. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2023, pp. 482–513. DOI: 10.1007/978-3-031-38545-2_16.

[37] J. Alex Halderman, Brent Waters and Edward W. Felten. 'A convenient method for securely managing passwords'. In: *WWW*. ACM, 2005, pp. 471–479.

[38] Martin Hellman. 'A cryptanalytic time-memory trade-off'. In: *IEEE transactions on Information Theory* 26.4 (1980), pp. 401–406.

[39] Dmitry Kogan, Nathan Manohar and Dan Boneh. 'T/Key: Second-Factor Authentication From Secure Hash Chains'. In: *ACM CCS 2017: 24th Conference on Computer and Communications Security*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin and Dongyan Xu. Dallas, TX, USA: ACM Press, Oct. 2017, pp. 983–999. DOI: 10.1145/3133956.3133989.

[40] Hugo Krawczyk. 'Cryptographic Extraction and Key Derivation: The HKDF Scheme'. In: *Advances in Cryptology – CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2010, pp. 631–648. DOI: 10.1007/978-3-642-14623-7_34.

[41] Atul Luykx, Elena Andreeva, Bart Mennink and Bart Preneel. *Impossibility Results for Indifferentiability with Resets*. Cryptology ePrint Archive, Paper 2012/644. 2012. URL: https://eprint.iacr.org/2012/644.

[42] Ueli M. Maurer, Renato Renner and Clemens Holenstein. 'Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology'. In: *TCC 2004: 1st Theory of Cryptography Conference*. Ed. by Moni Naor. Vol. 2951. Lecture Notes in Computer Science. Cambridge, MA, USA: Springer, Heidelberg, Germany, Feb. 2004, pp. 21–39. DOI: 10.1007/978-3-540-24638-1_2.

[43] Meta Platforms, Inc. *Meta Reports First Quarter 2026 Results*. Meta Investor Relations. Accessed: June 1, 2026. Apr. 2026. URL: https://investor.atmeta.com/investor-news/press-release-details/2026/Meta-Reports-First-Quarter-2026-Results/default.aspx.

[44] Arno Mittelbach. 'Cryptophia's Short Combiner for Collision-Resistant Hash Functions'. In: *ACNS 13: 11th International Conference on Applied Cryptography and Network Security*. Ed. by Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel and Reihaneh Safavi-Naini. Vol. 7954. Lecture Notes in Computer Science. Banff, AB, Canada: Springer, Heidelberg, Germany, June 2013, pp. 136–153. DOI: 10.1007/978-3-642-38980-1_9.

[45] Arno Mittelbach. 'Salvaging Indifferentiability in a Multi-stage Setting'. In: *Advances in Cryptology – EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. Lecture Notes in Computer Science. Copenhagen, Denmark: Springer, Heidelberg, Germany, May 2014, pp. 603–621. DOI: 10.1007/978-3-642-55220-5_33.

[46] Philippe Oechslin. 'Making a Faster Cryptanalytic Time-Memory Trade-Off'. In: *Advances in Cryptology – CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2003, pp. 617–630. DOI: 10.1007/978-3-540-45146-4_36.

[47] OWASP Foundation. *Application Security Verification Standard (ASVS) Version 5.0.0*. Standard. Available at https://github.com/OWASP/ASVS. Open Worldwide Application Security Project, May 2025. URL: https://github.com/OWASP/ASVS.

[48] *Password Hashing Competition*. www.password-hashing.net. URL: https://www.password-hashing.net/.

[49] Colin Percival and Simon Josefsson. *The scrypt Password-Based Key Derivation Function*. RFC 7914, IETF. 2016. DOI: 10.17487/RFC7914. URL: https://www.rfc-editor.org/info/rfc7914.

[50] *PKCS #5: Password-based cryptography specification*. RSA Data Security, Inc. Version 2.0. Sept. 2000.

[51] Niels Provos and David Mazières. 'A Future-Adaptable Password Scheme'. In: *Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference, June 6-11, 1999, Monterey, California, USA*. USENIX, 1999, pp. 81–91. URL: http://www.usenix.org/events/usenix99/provos.html.

[52] Thomas Ristenpart, Hovav Shacham and Thomas Shrimpton. *Careful with Composition: Limitations of Indifferentiability and Universal Composability*. Cryptology ePrint Archive, Paper 2011/339. 2011. URL: https://eprint.iacr.org/2011/339.

[53] Thomas Ristenpart, Hovav Shacham and Thomas Shrimpton. *Careful with Composition: Limitations of Indifferentiability and Universal Composability*. Cryptology ePrint Archive, Report 2011/339. https://eprint.iacr.org/2011/339. 2011.

[54] Gabriel Arquelau Pimenta Rodrigues, Pedro Augusto Giacomelli Fernandes, André Luiz Marques Serrano, Geraldo Pereira Rocha Filho, Guilherme Fay Vergara, Guilherme Dantas Bispo, Robson de Oliveira Albuquerque and Vinícius Pereira Gonçalves. 'From RockYou to RockYou2024: Analyzing Password Patterns Across Generations, Their Use in Industrial Systems and Vulnerability to Password Guessing Attacks'. In: *Journal of Internet Services and Applications* 16.1 (2025), pp. 69–86. URL: https://journals-sol.sbc.org.br/index.php/jisa/article/view/5034.

[55] Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh and John C. Mitchell. 'Stronger Password Authentication Using Browser Extensions'. In: *USENIX Security 2005: 14th USENIX Security Symposium*. Ed. by Patrick D. McDaniel. Baltimore, MD, USA: USENIX Association, July 2005.

[56] Maliheh Shirvanian, Christopher Robert Price, Mohammed Jubur, Nitesh Saxena, Stanislaw Jarecki and Hugo Krawczyk. 'A hidden-password online password manager'. In: *SAC*. ACM, 2021, pp. 1683–1686.

[57] Tom Van Vleck. *`gpw-js` Password Generator*. https://www.multicians.org/thvv/gpw-js.html.

[58] Zeljko Vrba. *`secpwgen` Linux manual page*. https://linux.die.net/man/1/secpwgen. 2005.

[59] Simon S. Woo. 'How Do We Create a Fantabulous Password?' In: *WWW*. ACM / IW3C2, 2020, pp. 1491–1501.

[60] Ka-Ping Yee and Kragen Sitaker. 'Passpet: convenient password management and phishing protection'. In: *SOUPS*. Vol. 149. ACM International Conference Proceeding Series. ACM, 2006, pp. 32–43.