Fixing Cracks in the Concrete: Random Oracles with Auxiliary Input, Revisited

Yevgeniy Dodis[1⋆], Siyao Guo[2⋆⋆], and Jonathan Katz[3⋆⋆⋆]

[1] New York University, USA
dodis@cs.nyu.edu
[2] Simons Institute, UC Berkeley
siyao.guo@berkeley.edu
[3] University of Maryland, USA
jkatz@cs.umd.edu

Abstract. We revisit the security of cryptographic primitives in the random-oracle model against attackers having a bounded amount of *auxiliary information* about the random oracle. This situation arises most naturally when an attacker carries out offline preprocessing to generate state (namely, auxiliary information) that is later used as part of an on-line attack, with perhaps the best-known example being the use of rainbow tables for function inversion. The resulting model is also critical to obtain accurate bounds against *non-uniform* attackers when the random oracle is instantiated by a concrete hash function.

Unruh (Crypto 2007) introduced a generic technique (called pre-sampling) for analyzing security in this model: a random oracle for which S bits of arbitrary auxiliary information can be replaced by a random oracle whose value is fixed in some way on P points; the two are distinguishable with probability at most $O(\sqrt{ST/P})$ by attackers making at most T oracle queries. Unruh conjectured that the distinguishing advantage could be made negligible for a sufficiently large polynomial P. We show that Unruh's conjecture is *false* by proving that the distinguishing probability is at least $\Omega(ST/P)$.

Faced with this negative general result, we establish new security bounds, — which are nearly optimal and beat pre-sampling bounds, — for specific applications of random oracles, including one-way functions, pseudorandom functions/generators, and message authentication codes. We also explore the effectiveness of *salting* as a mechanism to defend against offline preprocessing, and give quantitative bounds demonstrating that salting provably helps in the context of one-wayness, collision-resistance, pseudorandom generators/functions, and message authentication codes. In each case, using (at most) n bits of salt, where n is the length of the secret key, we get the same security $O(T/2^n)$ in the random oracle model with auxiliary input as we get without auxiliary input.

At the heart of our results is the compression technique of Gennaro and Trevisan, and its extensions by De, Trevisan and Tulsiani.

⋆ Work done while visiting the University of Maryland. Partially supported by gifts from VMware Labs and Google, and NSF grants 1619158, 1319051, 1314568.
⋆⋆ Work done while at NYU and visiting the University of Maryland.
⋆⋆⋆ Work supported in part by NSF award #1223623.

1 Introduction

The random-oracle model [4] often provides a simple and elegant way of analyzing the concrete security of cryptographic schemes based on hash functions. To take a canonical example, consider (naïve) password hashing where a password pw is stored as $H(pw)$, for H a cryptographic hash function, and we are interested in the difficulty of recovering pw from $H(pw)$ (i.e., we are interested in understanding the *one-wayness* of H). It seems difficult to formalize a concrete assumption about H that would imply the difficulty of recovering pw for *all* high-entropy distributions on pw; it would be harder still to come up with a natural assumption implying that for all distributions on pw with min-entropy k, recovering pw requires $O(2^k)$ work. If we model H as a random oracle, however, then both these statements can be proven easily—and this matches the best known attacks for many cryptographic hash functions.

Importantly, the above discussion assumes that no preprocessing is done. That is, we imagine an attacker who does no work prior to being given $H(pw)$ or, more formally, we imagine that the attacker is fixed before the random oracle H is chosen. In that case, the only way an attacker can learn information about H is by making explicit queries to an oracle for H, and the above-mentioned bounds hold. In practice, however, H is typically a standardized hash function that is known in advance, and *offline preprocessing attacks*—during which the attacker can query and store arbitrary information about H—can be a significant threat.

Concretely, let $H : [N] \to [N]$ and assume that pw is uniform in $[N]$. The obvious attack to recover pw from $H(pw)$ is an exhaustive-search attack which uses time $T = N$ in the online phase (equating time with the number of queries to H) to recover pw. But an attacker could also generate the entire function table for H during an offline preprocessing phase; then, given $H(pw)$ in the on-line phase, the attacker can recover pw in $O(1)$ time using a table lookup. The data structure generated during the offline phase requires $S = O(N)$ space (ignoring $\log N$ factors), but Hellman [12] showed a more clever construction of a data structure which, in particular, gives an attack using $S = T = O(N^{2/3})$ (see [13, Section 5.4.3] for a self-contained description). *Rainbow tables* implementing this approach along with later improvements (most notably by Oechslin [15]), are widely used in practice, and must be taken into account in any practical analysis of password security. Further work has explored improving these result and proving rigorous versions of them, as well as showing bounds on how well such attacks can perform [19, 9, 15, 10, 2, 7].

The above discussion in the context of function inversion gives a practical example of where *auxiliary information* about a random oracle (in this case, in the form of rainbow tables generated using the random oracle) can quantitatively change the security of a given application that uses the random oracle. For a more dramatic (but less practical) example, consider the case of collision finding. Given a random function $H : [N] \to [N]$, one can show that $O(\sqrt{N})$ queries are needed in order to find a collision in H (i.e., distinct points x, x' with $H(x) = H(x')$). But clearly we can find a collision in H during an offline pre-processing phase and store that collision using $O(1)$ space, after which it is trivial to output that

collision in an online phase in $O(1)$ time. The conclusion is that in settings where offline preprocessing is a possibility, security proofs in the random-oracle model must be interpreted carefully. (We refer the reader to [17, 5], as well as many of the references below, for further discussion.)

From a different viewpoint, another motivation for studying auxiliary information comes from the desire for obtaining accurate security bounds against *non-uniform attackers* when instantiating random oracle by a concrete hash function. Indeed, non-uniform attackers are allowed to have some arbitrary 'advice' before attacking the system. Translated to the random oracle model, this would require the attacker to be able to compute some arbitrary function of the *entire random oracle*, which cannot be done using only bounded number T of oracle queries. This mismatch already led to considerable confusion among both theoreticians and practitioners. We refer to [16, 6] for some in-depth discussion, here only mentioning two most well-known examples. (1) In the standard (non-uniform) model, no single function can be collision-resistant, while a single random oracle is trivially collision-resistant (without preprocessing); this is why in the standard model one considers a *family* of CRHFs, whose public key (which we call *salt*) is chosen *after* the attacker gets his non-uniform advice. To the best of our knowledge, prior to our work no meaningful CRHF bound was given for *salted* random oracle if (salt-independent) preprocessing was allowed. (2) In the standard (non-uniform) model, it is well known [1, 8, 6] that no pseudorandom generator (PRG) $H(x)$ can have security better than $2^{-n/2}$ even against linear-time attackers, where n is the seed-length of x. In contrast, an expanding random oracle can be trivially shown to be $(T/2^n)$-secure PRG in the traditional random oracle model, easily surpassing the $2^{-n/2}$ barrier in the standard model (even for huge T up to $2^{n/2}$, let alone polynomial T).

Random Oracle with Auxiliary Input. While somewhat different, the two motivating applications above effectively reduce to the following identical extension of the traditional random oracle model (ROM). A (computationally unbounded) attacker A can compute arbitrary S bits of information $z = z(\mathcal{O})$ about the random oracle $\mathcal{O}$ *before* attacking the system, and then use additional T *oracle* queries to $\mathcal{O}$ *during* the attack. Following Unruh [17], we call this the *Random Oracle Model with Auxiliary Input* (ROM-AI), and this is the model we thoroughly study in this work. As we mentioned, while the traditional ROM only uses one parameter T, the ROM-AI is parameterized by two parameters, S and T which roughly correspond to space (during off-line pre-processing) and time (during on-line attack). For the application to non-uniform security, one can also use the ROM-AI to get good estimates for non-uniform security against (non-uniform) circuits of size C by setting $S = T = C$.[1]

[1] Since circuit of size C can encode up to $S = \Omega(C)$ bits of information about a given hash function H, as well as evaluate it close to $T = \Omega(C)$ times, assuming H is efficient.

1.1 Handling Random Oracles with Auxiliary Input

Broadly speaking, there are three ways one can address the issue of preprocessing/auxiliary input in the random-oracle model: (1) by using a generic approach to analyze existing or proposed schemes, (2) by using an application-specific approach to analyze an existing or proposed scheme, or (3) by modifying existing schemes in an attempt to defeat preprocessing/non-uniform attacks. We discuss limited prior work on these three approaches below, before stating our results.

A generic approach. Unruh [17] was the first to propose a generic approach for dealing with auxiliary input in the random-oracle model. We give an informal overview of his results (a formal statement is given in Section 2). Say we wish to bound the success probability ϵ (in some experiment) of an online attacker making T random-oracle queries, and relying on S bits of (arbitrary) auxiliary information about the random oracle. Unruh showed that it suffices to analyze the success probability $\epsilon'(P)$ of the attack in the presence of a "pre-sampled" random oracle that is chosen uniformly subject to its values being fixed in some adversarial way on P adversarial points (where P is a parameter), and no other auxiliary information is given; ϵ is then bounded by $\epsilon'(P) + O(\sqrt{ST/P})$, while P is then chosen optimally as to balance out the resulting two terms (see an example below).

This is an impressive result, but it falls short of what one might hope for. In particular, P must be super-polynomial in order to make the "security loss" $O(\sqrt{ST/P})$ negligible, but in many applications if P is too large then the bound $\epsilon'(P)$ one can prove on an attacker's success probability in the presence of a "pre-sampled" random oracle with P fixed points becomes too high. Unruh conjectured that his bound was not tight, and that it might be possible to bound the "security loss" by a negligible quantity for P a sufficiently large polynomial.

An application-specific approach. Given that the generic approach might lead to very sub-optimal bounds, one might hope to develop a much tighter application-specific approach to get concrete bounds. To the best of our knowledge, no such work was done for the random oracle model with preprocessing. Indirectly, however, De et al. [7] adapted the beautiful compression "compression paradigm" introduced by Gennaro and Trevisan [11, 10] to show nearly tight security bounds for inverting inverting one-way *permutations* as well as *specific* PRGs (based on one-way permutations and hardcore bits). This was done not for the sake of analyzing security of these constructions,[2] but rather to show limitations of generic inversion/distinguishing attacks *all* one-way functions or PRGs. Still, this elegant theoretical approach suggests that application-specific techniques, such as the compression paradigm, might be useful in the analysis of schemes based on real-world hash functions, such as SHA.

"Salting." Even with optimal application-specific techniques, we have already discussed how preprocessing attacks can be effective for tasks like function in-

[2] For which we currently have no real-world candidates, since we do not have any candidates for efficient *uninvertible* "random permutations".

version and collision finding, as well as non-trivial distinguishing attacks against pseudorandom generators/functions.

A natural defense against preprocessing attacks, which has been explicitly suggested [14] and is widely used to defeat such attacks in the context of password hashing, is to use *salting*. Roughly, this involves choosing a random but public value a and including it in the input to the hash function. Thus, in the context of password hashing we would choose a uniform salt a and store $(a, H(a, pw))$; in the context of collision-resistant hashing we would choose and publish a and then look at the hardness of finding collisions in the function $H(a, \cdot)$; and in the context of pseudorandom generators we would choose a and then look at the pseudorandomness of $H(a, x)$ (for uniform x) given a.

De et al. [7] briefly study the effect of salting for inverting one-way *permutations* as well as *specific* PRGs (based on one-way permutations and hardcore bits), but beyond that we are aware of no analysis of the effectiveness of salting for defeating preprocessing in any other contexts, including the use of hash functions which are not permutations.[3] We highlight that although it may appear "obvious" that salting defeats, say, rainbow tables, it is not at all clear what is the *quantitative* security benefit of salting, and it is not clear whether rainbow tables can be adapted to give a (possibly different) online/offline tradeoff when salting is used.

1.2 Our Results

We address all three approaches outlined in the previous section. First, we investigate the generic approach to proving security in the random-oracle model with auxiliary input, and specifically explore the extent to which Unruh's presampling technique can be improved. Here, our result is largely negative: disproving Unruh's conjecture, we show that there is an attack for which the "security loss" stemming from Unruh's approach is at least $\Omega(ST/P)$. Although there remains a gap between our lower bound and Unruh's upper bound that will be interested to close, as we discuss next the upshot is that Unruh's technique is not sufficient (in general) for proving strong concrete-security bounds in the random-oracle model when preprocessing is a possibility.

Consider, e.g., the case of function inversion. One can show that the probability of inverting a random oracle $H : [N] \to [N]$ for which P points have been "pre-sampled" is $O(P/N + T/N)$. Combined with the security loss of $O(\sqrt{ST/2P})$ resulting from Unruh's technique and plugging in the optimal value of P, we obtain a security bound of $O((ST/N)^{1/3} + T/N)$ for algorithms making T oracle queries and using S bits of auxiliary input about H. And our negative result shows that the best bound one could hope to achieve by using Unruh's approach is $O((ST/N)^{1/2} + T/N)$. Both bounds fall short of the best known attacks, which succeed with probability $\Omega\left(\min\left\{\frac{T}{N}, (\frac{S^2T}{N^2})^{1/3}\right\} + \frac{T}{N}\right)$. Similar gaps exist for other cryptographic primitives.

[3] Bellare et al. [3] study security of salting for the purposes of multi-instance security, but they do not address the issue of preprocessing.

Faced with this, we turn to studying a more direct approach for proving tighter bounds for specific important applications of hash functions, such as their use as one-way functions, pseudorandom generators/functions (PRGs/PRFs) or message authentication codes (MACs).[4] Here we show much tighter, and in many cases optimal bounds for all of these primitives, which always beat the provable version of Unruh's pre-sampling (see Table 1 with value $K = 1$). Not surprisingly, our bounds are not as good as what is possible to show without pre-processing, since those bounds are no longer true once pre-processing is allowed. In particular, setting $S = T = C$ we now get meaningful non-uniform security bounds against circuits of size C for all of the above primitives, which often match the existing limitations known for non-uniform attacks. (For example, when $C = S = T$ is polynomial in n, we get that the optimal non-uniform PRG/PRF security is lower bounded by $2^{-n/2}$, matching existing attacks.)

Given these inherent limitation as compared to the traditional ROM without preprocessing, we formally examine the effects of "salting" as a way of mitigating or even defeating the effects of pre-processing/non-uniformity. As before, we look at the natural, "salted" constructions of one-way functions, PRGs, PRFs and MACs, but now can also examine collision-resistant hash functions (CRHFs), which can be potentially secure against pre-processing, once the salt is long-enough. In all these case we analyze the security of these constructions in the presence of auxiliary information about the random oracle. In fact, the "un-salted" results for one-way functions, PRGs, PRFs and MACs mentioned above are simply special cases of salted result with the cardinality K of the salting space is $K = 1$.

Our results are summarized in Table 1, where they are compared to the best known attacks using preprocessing. Our bounds for inverting one-way functions and distinguishing PRGs matches the bounds De et al. [7] for inverting one-way permutations and distinguishing PRGs based on one-way permutations and hardcore bits, but apply to real-world candidates for these primitives based on existing hash functions. In the case of CRHFs, our bound is tight and matches the best known attack of storing explicit collisions for roughly S distinct salts. In the remaining cases, although our bounds are not tight (but close), it is interesting to note that, assuming $N \geq T \geq S$, our results show that setting the length of the salt equal to the length of the secret (i.e., setting $K = N$) yields the same security bound $O(T/N)$ that is achieved for constructions in the standard random-oracle model *without* preprocessing. Summarizing a bit informally: using an n-bit salt and an n-bit secret gives n-bit security *even in the presence of preprocessing*. Namely, salts provably defeats pre-processing in these settings.

All our new bounds are proven using the "compression paradigm" introduced by Gennaro and Trevisan [11, 10]. The main idea is to argue that if some attacker succeeds with "high" probability, then that attacker can be used to reversibly encode (i.e., compress) a random oracle beyond what is possible from

[4] As we mentioned, collision-resistance is impossible without salting, which we discuss shortly.

	Security bounds (here)	Best known attacks
OWFs	$\frac{ST}{KN} + \frac{T}{N}$	$\min\left\{\frac{ST}{KN}, (\frac{S^2T}{K^2N^2})^{1/3}\right\} + \frac{T}{N}$
CRHFs	$\frac{S}{K} + \frac{T^2}{M}$	$\frac{S}{K} + \frac{T^2}{M}$
PRGs	$(\frac{ST}{KN})^{1/2} + \frac{T}{N}$	$(\frac{S}{KN})^{1/2} + \frac{T}{N}$
PRFs	$(\frac{ST}{KN})^{1/2} + \frac{T}{N}$	$(\frac{S}{KN})^{1/2} + \frac{T}{N}$
MACs	$\frac{ST}{KN} + \frac{T}{N} + \frac{T}{M}$	$\min\left\{\frac{ST}{KN}, (\frac{S^2T}{K^2N^2})^{1/3}\right\} + \frac{T}{N} + \frac{1}{M}$

Table 1. Security bounds and best known attacks using space S and time T for "salted" constructions of primitives based on a random oracle. The first three (unkeyed) primitives are constructed from a random oracle $\mathcal{O} : [K] \times [N] \to [M]$, where $[K]$ is the domain of the salt and $[N]$ is the domain of the secret; the final two (keyed) primitives are constructed from a random oracle $\mathcal{O} : [K] \times [N] \times [L] \to [M]$, where $[L]$ is the domain of the input. For simplicity, logarithmic factors and constant terms are omitted.

an information-theoretic point of view. Since we are considering attackers who perform preprocessing, our encoding must include the S-bit auxiliary information produced by the attacker. Thus, the main technical challenge we face is to ensure that our encoding compresses by (significantly) more than S bits.

Outlook. In this work we thoroughly revisited the ROM with auxiliary input, as we believe it has not gotten enough attention from the cryptographic community, despite being simultaneously important for the variety of reasons detailed above, and also much more interesting than the traditional ROM from a technical point in view. Indeed, even the most trivial one-line proof in the traditional ROM is either completely false once preprocessing is allowed (e.g., CRHFs), or becomes an interesting technical challenge (OWFs, PRGs, MACs) that requires new techniques, and usually teaches us something new about the primitive in question in relation to pre-processing.

Of course, given an abundance of works using random oracle, we hope our work will generate a lot of follow-up research analyzing the effects of pre-processing and non-uniformity for many other important uses of hash functions, as well as other idealized primitives (e.g., ideal ciphers).

2 Limits On the Power of Preprocessing

For two distributions D_1, D_2 over universe Ω, we use $\Delta(D_1, D_2)$ to denote their statistical distance $\frac{1}{2} \cdot \sum_{y \in \Omega} |\Pr[D_1 = y] - \Pr[D_2 = y]|$.

In this section, we revisit the result of Unruh [17] that allows one to replace arbitrary (bounded-length) auxiliary information about a random oracle $\mathcal{O}$ with a (bounded-size) set fixing the value of the random oracle on some fraction of points. For a set of tuples $Z = \{(x_1, y_1), \ldots\}$, we let $\mathcal{O}'[Z]$ denote a random oracle chosen uniformly subject to the constraints $\mathcal{O}'(x_i) = y_i$.

Theorem 1 ([17]). *Let $P, S, T \geq 1$ be integers, and let A_0 be an oracle algorithm that outputs state of length at most S bits. Then there is an oracle*

algorithm Pre *outputting a set containing at most* P *tuples such that for any oracle algorithm* A_1 *that makes at most* T *oracle queries,*

$$\Delta(A_1^{\mathcal{O}}(A_0^{\mathcal{O}}),\ A_1^{\mathcal{O}'[\mathsf{Pre}^{\mathcal{O}}]}(A_0^{\mathcal{O}})) \leq \sqrt{\frac{ST}{2P}}\,.$$

This theorem enables proving various results in the random-oracle model even in the presence of auxiliary input by first replacing the auxiliary input with a fixed set of input/output pairs and then using standard lazy-sampling techniques for the value of the random oracle at other points. However, applying this theorem incurs a cost of $\sqrt{ST/2P}$, and so super-polynomial P is required in order to obtain negligible advantage overall. It is open whether one can improve the bound in Theorem 1; Unruh conjectures [17, Conjecture 14] that for all polynomials S, T there is a polynomial P such that the statistical difference above is negligible. We disprove this conjecture by showing that the bound in the theorem cannot be improved (in general) below $O(ST/P)$. That is,

Theorem 2. *Consider random oracles* $\mathcal{O} : [N] \to \{0,1\}$*, and let* $S, T, P \geq 1$ *be integers with* $4P^2/ST + ST \leq N$*. Then there is an oracle algorithm* A_0 *that outputs* S*-bit state and an oracle algorithm* A_1 *that makes* T *oracle queries such that for any oracle algorithm* Pre *outputting a set containing at most* P *tuples,*

$$\Delta(A_1^{\mathcal{O}}(A_0^{\mathcal{O}}),\ A_1^{\mathcal{O}'[\mathsf{Pre}^{\mathcal{O}}]}(A_0^{\mathcal{O}})) \geq \frac{ST}{24P}\,.$$

Proof. Pick S disjoint sets $X_1, \ldots, X_S \subset [N]$, where each set is of size $t = T \cdot (4(P/ST)^2 + 1)$. Partition each set X_i into $t/T = 4(P/ST)^2 + 1$ disjoint blocks $X_{i,1}, \ldots, X_{i,t/T}$, each of size T. Algorithm $A_1^{\mathcal{O}}$ outputs an S-bit state where the ith bit is equal to $\mathrm{maj}(\oplus_{x\in X_{i,1}}\mathcal{O}(x), \ldots, \oplus_{x\in X_{i,t/T}}\mathcal{O}(x))$ where maj is the majority function. Algorithm $A_1^{\mathcal{O}}(b_1, \ldots, b_S)$ chooses a uniform block $X_{i,j}$ and outputs 1 iff $\oplus_{x\in X_{i,j}}\mathcal{O}(x) = b_i$.

We have

$$\begin{aligned}
&\Pr[A_1^{\mathcal{O}}(A_0^{\mathcal{O}}) = 1] \\
&= \Pr_{\mathcal{O},i,j}\left[\mathrm{maj}(\oplus_{x\in X_{i,1}}\mathcal{O}(x), \ldots, \oplus_{x\in X_{i,t/T}}\mathcal{O}(x)) = \oplus_{x\in X_{i,j}}\mathcal{O}(x)\right] \\
&= \Pr_{z_1,\ldots,z_{t/T}\leftarrow\{0,1\},j\leftarrow[t/T]}\left[\mathrm{maj}(z_1, \ldots, z_{t/T}) = z_j\right] \\
&= \mathop{\mathbb{E}}_{j}\left[\Pr\left[\sum_{i\neq j} z_i = \frac{t/T-1}{2}\right] + \frac{1}{2}\cdot\Pr\left[\sum_{i\neq j} z_i \neq \frac{t/T-1}{2}\right]\right] \\
&= \frac{1}{2} + \frac{1}{2}\cdot\Pr\left[\sum_{i>1} z_i = \frac{t/T-1}{2}\right] \\
&= \frac{1}{2} + \binom{t/T-1}{\frac{t/T-1}{2}}\cdot 2^{-t/T} \\
&\geq \frac{1}{2} + \frac{1}{3\sqrt{t/T-1}} \;=\; \frac{1}{2} + \frac{ST}{6P},
\end{aligned}$$

where the inequality uses $\sqrt{2\pi n}\,(n/e)^n \le n! \le e\sqrt{n}\,(n/e)^n$ so that

$$\binom{n}{n/2} \ge \frac{\sqrt{2\pi n}\,(n/e)^n}{(e\sqrt{n/2}\,(n/2e)^{n/2})^2} = \frac{2\sqrt{2\pi}}{e^2\sqrt{n}} \cdot 2^n \ge \frac{2}{3} \cdot \frac{2^n}{\sqrt{n}}\,.$$

On the other hand, for any algorithm Pre we have

$$\begin{aligned}
&\Pr[A_1^{\mathcal{O}'[\mathsf{Pre}^{\mathcal{O}}]}(A_0^{\mathcal{O}}) = 1] \\
&= \Pr_{i,j,\mathcal{O},\mathcal{O}'}[\mathrm{maj}(\oplus_{x\in X_{i,1}}\mathcal{O}(x),\ldots,\oplus_{x\in X_{i,t/T}}\mathcal{O}(x)) = \oplus_{x\in X_{i,j}}\mathcal{O}'(x)] \\
&\le \frac{P/T}{St/T} + \frac{1}{2}\cdot\left(1 - \frac{P/T}{St/T}\right) \\
&= \frac{1}{2} + \frac{P}{2St} \;\le\; \frac{1}{2} + \frac{ST}{8P}.
\end{aligned}$$

The first inequality above holds since, for any fixed $i, j, \mathcal{O}$,

$$\Pr_{\mathcal{O}'}\left[\mathrm{maj}(\oplus_{x\in X_{i,1}}\mathcal{O}(x),\ldots,\oplus_{x\in X_{i,t/T}}\mathcal{O}(x)) = \oplus_{x\in X_{i,j}}\mathcal{O}'(x)\right] = 1/2$$

unless the value of $\mathcal{O}'$ is fixed by $\mathsf{Pre}^{\mathcal{O}}$ at every point in $X_{i,j}$. But $\mathsf{Pre}^{\mathcal{O}}$ can ensure that the value of $\mathcal{O}'$ is fixed in that way for at most P/T out of the St/T blocks defined by i, j. This concludes the proof. ■

3 Function Inversion

For natural number n, we define $[n] = \{1,\ldots,n\}$. In this section, we prove bounds on the hardness of inverting "salted" random oracles in the presence of preprocessing. That is, consider choosing a random function $\mathcal{O} : [K]\times[N] \to [M]$ and then allowing an attacker A_0 (with oracle access to $\mathcal{O}$) to perform arbitrary preprocessing to generate an S-bit state st. We then look at the hardness of inverting $\mathcal{O}(a,x)$, given st and a, for algorithms A_1 making up to T oracle queries, where $a \in [K]$ and $x \in [N]$ are uniform. We consider two notions of inversion: computing x itself, or the weaker goal of finding any x' such that $\mathcal{O}(a,x') = \mathcal{O}(a,x)$. Assuming $N = M$ for simplicity in the present discussion, we show that in either case the probability of successful inversion is $O(\frac{ST}{KN} + \frac{T\log N}{N})$. We remark that the best bound one could hope to prove via a generic approach (i.e., using Theorem 1 with best-possible bound $O(ST/P)$) is[5] $O(\sqrt{ST/KN} + T/N)$.

By way of comparison, rainbow tables [12, 9, 15, 2, 7] address the case $K = 0$ (i.e., no salt), and give success probability $O(\min\{ST/N, (S^2T/N^2)^{1/3}\} + T/N)$. One natural way to adapt rainbow tables to handle salt is to compute K independent rainbow tables, each using space S/K, for the K reduced functions $\mathcal{O}(a,\cdot)$.

[5] Any such bound would take the form $O(ST/P + P/KN + T/N)$, where the first term is from application of the theorem, the second is the probability that the input to $\mathcal{A}_1$ is from the set of fixed points, and the third is the success probability of a trivial brute-force search. Setting $P = \sqrt{ST/KN}$ optimizes this bound.

Using this approach gives success probability $O(\min\{ST/KN, (S^2T/K^2N^2)^{1/3}\} + T/N)$. This shows that our bound is tight when $ST^2 < KN$.

We begin with some preliminary lemmas that we will rely on in this and the following sections.

Lemma 1. *Say there exist encoding and decoding procedures* $(\mathsf{Enc}, \mathsf{Dec})$ *such that for all* $m \in M$ *we have* $\mathsf{Dec}(\mathsf{Enc}(m)) = m$. *Then* $\mathbb{E}_m[\,|\mathsf{Enc}(m)|\,] \geq \log |M|$.

Proof. For $m \in M$, let $s_m = |\mathsf{Enc}(m)|$. Define $C = \sum_m 2^{-s_m}$, and for $m \in M$ let $q_m = 2^{-s_m}/C$. Then $\mathbb{E}_m[\,|\mathsf{Enc}(m)|\,] = -\mathbb{E}_m[\log q_m] - \log C$. By Jensen's inequality, $\mathbb{E}_m[\log q_m] \leq \log \mathbb{E}_m[q_m] = -\log |M|$, and by Kraft's inequality $C \leq 1$. The lemma follows. ■

Following De et al. [7], we also consider randomized encodings $(\mathsf{Enc}, \mathsf{Dec})$ for a set M. We say that an encoding has *recovery probability* δ if for all $m \in M$,

$$\Pr_r[\mathsf{Dec}(\mathsf{Enc}(m, r), r) = m] \geq \delta.$$

(Note that Dec is given the randomness used by Enc.) The *encoding length* of $(\mathsf{Enc}, \mathsf{Dec})$ is defined to be $\max_{m,r}\{\,|\mathsf{Enc}(m, r)|\}$.

Lemma 2 ([7]). *Suppose there exist randomized encoding and decoding procedures* $(\mathsf{Enc}, \mathsf{Dec})$ *for a set* M *with recovery probability* δ. *Then the encoding length of* $(\mathsf{Enc}, \mathsf{Dec})$ *is at least* $\log |M| - \log 1/\delta$.

Proof. By a standard averaging argument, there exists an r and a set $M' \subseteq M$ with $|M'| \geq \delta \cdot |M|$ such that $\mathsf{Dec}(\mathsf{Enc}(m, r), r) = m$ for all $m \in M'$. Let $\mathsf{Enc}', \mathsf{Dec}'$ be the deterministic algorithms obtained by fixing the randomness to r. By Lemma 1, $\mathbb{E}_{m'}[\,|\mathsf{Enc}'(m')|\,] \geq |M'| \geq |M| - \log 1/\delta$, and hence there exists an m' with $|\mathsf{Enc}'(m')| \geq |M| - \log 1/\delta$. ■

We now state and prove the main results of this section. Let $\mathsf{Func}(A, B)$ denote the set of all functions from A to B.

Theorem 3. *Consider random oracles* $\mathcal{O} \in \mathsf{Func}([K] \times [N],\ [M])$. *For any oracle algorithms* (A_0, A_1) *such that* A_0 *outputs* S*-bit state and* A_1 *makes at most* T *oracle queries,*

$$\Pr_{\mathcal{O},a,x}[A_1^{\mathcal{O}}(A_0^{\mathcal{O}}, a, \mathcal{O}(a, x)) = x] = O\left(\frac{ST}{KN} + \frac{T \log N}{N}\right).$$

Theorem 4. *Consider random oracles* $\mathcal{O} \in \mathsf{Func}([K] \times [N],\ [M])$. *For any oracle algorithms* (A_0, A_1) *such that* A_0 *outputs* S*-bit state and* A_1 *makes at most* T *oracle queries,*

$$\Pr_{\mathcal{O},a,x}[A_1^{\mathcal{O}}(A_0^{\mathcal{O}}, a, \mathcal{O}(a, x)) = x' : \mathcal{O}(a, x) = \mathcal{O}(a, x')] = \varepsilon\,,$$

if $\varepsilon = \Omega(\log MN/N)$, *then*

$$\varepsilon = O\left(\frac{ST}{K \cdot \alpha} + \frac{T \log N}{\alpha}\right)$$

where $\alpha = \min\{N/\log M, M\}$

To prove Theorem 3, we first prove the following lemma:

Lemma 3. *Consider random oracles $\mathcal{O} \in \mathsf{Func}([K] \times [N],\ [M])$. Assume there exist oracle algorithms (A_0, A_1) such that A_0 outputs S-bit state and A_1 makes at most T oracle queries, and such that*

$$\Pr_{\mathcal{O},a,x}[A_1^{\mathcal{O}}(A_0^{\mathcal{O}}, a, \mathcal{O}(a,x)) = x] = \epsilon.$$

Then there exists a randomized encoding for a set $\mathcal{F} \subseteq \mathsf{Func}([K] \times [N],\ [M])$ of size at least $\frac{\varepsilon}{2} \cdot M^{KN}$, with recovery probability at least 0.9 and encoding length (in bits) at most

$$KN \log M + S + K \log N - \frac{\varepsilon KN}{100T} \log\left(\frac{\varepsilon N}{100eT}\right).$$

Proof. By an averaging argument, there is a set $\mathcal{F} \subseteq \mathsf{Func}([K] \times [N],\ [M])$ of size at least $\varepsilon/2 \cdot |\mathsf{Func}([K] \times [N],\ [M])| = \frac{\varepsilon}{2} \cdot M^{KN}$ such that for all $\mathcal{O} \in \mathbb{F}$

$$\Pr_{a,x}[A_1^{\mathcal{O}}(A_0^{\mathcal{O}}, a, \mathcal{O}(a,x)) = x] \geq \epsilon/2.$$

Fix arbitrary $\mathcal{O} \in \mathcal{F}$. We encode $\mathcal{O}$ as follows. Let $\mathsf{st}_{\mathcal{O}}$ be the output of $A_0^{\mathcal{O}}$ and, for $a \in [K]$, let $U_a \subseteq [N]$ be the points x on which $A_1^{\mathcal{O}}(\mathsf{st}_{\mathcal{O}}, a, \mathcal{O}(a,x)) = x$. The high-level idea is that rather than encode the mapping $\{(x, \mathcal{O}(a,x))\}_{x \in U_a}$ explicitly, we will encode the set of points $\{\mathcal{O}(a,x)\}_{x \in U_a}$ and then use A_1 to recover the mapping. If we attempt this in the straightforward way, however, then it may happen that A_1 queries its oracle on a point for which the mapping is not yet known. To get around this issue, we instead use this approach for a random subset of U_a so that this only happens with small probability.

Specifically, the encoder uses randomness r to pick a set $R \subseteq [K] \times [N]$, where each $(a,x) \in [K] \times [N]$ is included in R with probability $1/10T$. For $a \in [K]$, let $G_a \subseteq R$ be the set of $(a,x) \in R$ such that $A_1^{\mathcal{O}}(\mathsf{st}_{\mathcal{O}}, a, \mathcal{O}(a,x)) = x$ and moreover A_1 does not query $\mathcal{O}$ on any $(a',x') \in R$ (except possibly (a,x) itself). Let $G = \bigcup_a G_a$. Define $V_a = \{\mathcal{O}(a,x)\}_{x \in G_a}$, and note that $|V_a| = |G_a|$.

As in De et al. [7], with probability at least 0.9 the size of G is at least $\varepsilon KN/100T$. To see this, note that by a Chernoff bound, R has at least $\varepsilon KN/40T$ points with probability at least 0.95. The expected number of points $(a,x) \in R$ for which $A_1^{\mathcal{O}}(\mathsf{st}_{\mathcal{O}}, a, \mathcal{O}(a,x)) = x$ but A_1 queries $\mathcal{O}$ on some point $(a',x') \in R$ (besides (a,x) itself) is at most $\frac{\varepsilon KN}{2} \cdot \frac{1}{10T} \cdot \left(1 - (1 - 1/10T)^T\right) \leq \frac{\varepsilon KN}{2000T}$. By Markov's inequality, with probability at least 0.95 the number of such points is at most $\frac{\varepsilon KN}{100T}$. So with probability at least 0.9, we have $|G| \geq \frac{3\varepsilon KN}{200T} \geq \frac{\varepsilon KN}{100T}$.

Assuming $|G| \geq \varepsilon KN/100T$, we encode $\mathcal{O}$ as follows:

1. Include $\mathsf{st}_{\mathcal{O}}$ and, for each $a \in [K]$, include $|V_a|$ and a description of V_a. This uses a total of $S + K \log N + \sum_{a \in [K]} \log \binom{M}{|G_a|}$ bits.
2. For each a and $y \in V_a$ (in lexicographic order), run $A_1^{\mathcal{O}}(\mathsf{st}_{\mathcal{O}}, a, y)$ and include in the encoding the answers to all the oracle queries made by A_1 that have not been included in the encoding so far, except for any queries in R. (By definition of G_a, there will be at most one such query and, if so, it will be the query (a,x) such that $\mathcal{O}(a,x) = y$.)

3. For each $(a, x) \in ([K] \times [N]) \setminus G$ (in lexicographic order) for which $\mathcal{O}(a, x)$ has not been included in the encoding so far, add $\mathcal{O}(a, x)$ to the encoding.

Steps 2 and 3 explicitly include in the encoding the value of $\mathcal{O}(a, x)$ for each $(a, x) \in ([K] \times [N]) \setminus G$. Thus, the total number of bits added to the encoding by those steps is $(KN - \sum_a |G_a|) \log M$.

To decode, the decoder first uses r to recover the set R defined above. Then it does the following:

1. Recover $\mathsf{st}_{\mathcal{O}}$, $\{|V_a|\}_{a \in K}$, and $\{V_a\}_{a \in K}$.
2. For each a and $y \in V_a$ (in lexicographic order), run $A_1(\mathsf{st}_{\mathcal{O}}, a, y)$ while answering the oracle queries of A_1 using the values stored in the encoding. The only exception is if A_1 ever makes a query $(a, x) \in R$, in which case y itself is returned as the answer. The output x of A_1 will be such that $\mathcal{O}(a, x) = y$.
3. For each $(a, x) \in [K] \times [N]$ (in lexicographic order) for which $\mathcal{O}(a, x)$ is not yet defined, recover the value of $\mathcal{O}(a, x)$ from the remainder of the encoding.

Assuming $|G| \geq \varepsilon KN/100T$, the encoding is not empty and the decoding procedure recovers $\mathcal{O}$. The encoding length is

$$S + K \log N + \sum_{a \in [K]} \log \binom{M}{|G_a|} + \left(KN - \sum_{a \in K} |G_a| \right) \log M.$$

Because $\binom{M}{|G_a|} \leq \left(\frac{eM}{|G_a|} \right)^{|G_a|}$, the encoding length is bounded by

$$\begin{aligned}
&S + K \log N + KN \log M - \sum_a |G_a| \log \left(\frac{|G_a|}{e} \right) \\
&\leq S + K \log N + KN \log M - |G| \log \left(\frac{|G|}{eK} \right) \\
&\leq S + K \log N + KN \log M - \frac{\varepsilon KN}{100T} \log \left(\frac{\varepsilon N}{100eT} \right),
\end{aligned}$$

where the second line uses concavity of the function $f(y) = -y \log(y/e)$, and the last line is because $|G| \geq \frac{\varepsilon KN}{100T}$. ∎

Lemma 3 gives an encoding for a set of size $\frac{\varepsilon}{2} \cdot M^{KN}$ with recovery probability 0.9, and encoding length at most $NK \log M + S + K \log N - \frac{\varepsilon KN}{100T} \log \left(\frac{\varepsilon N}{100eT} \right)$ bits. But Lemma 2 shows that any such encoding must have encoding length at least $NK \log M - \log \frac{2}{\varepsilon} - \log \frac{10}{9}$ bits. We thus conclude that

$$S + K \log N + \log \frac{20}{9\varepsilon} \geq \frac{\varepsilon KN}{100T} \log \left(\frac{\varepsilon N}{100eT} \right).$$

This implies Theorem 3 since either $\varepsilon < \frac{200eT}{N}$, or else it must be the case that $\varepsilon \leq (\frac{100T}{KN}) \cdot (S + K \log N + \log N)$.

We now prove Theorem 4. For fixed $\mathcal{O}$ and $a \in [K]$, let $Y_{\mathcal{O},a} \subseteq [M]$ be the set of points A_1 successfully inverts, i.e.,

$$Y_{\mathcal{O},a} = \{y \ : \ A_1^{\mathcal{O}}(A_0^{\mathcal{O}}, a, y) = x' : \mathcal{O}(a, x') = y\}.$$

Let $X_{\mathcal{O},a} \subseteq [N]$ be the pre-images of the points in $Y_{\mathcal{O},a}$. That is,

$$X_{\mathcal{O},a} = \{x \ : \ \mathcal{O}(a, x) \in Y_{\mathcal{O},a}\}.$$

We show a deterministic encoding for $\mathsf{Func}([K] \times [N], [M])$. Given a function $\mathcal{O}$, we encode it by including for each $a \in [K]$ the following information:

1. The set $X_{\mathcal{O},a}$ (along with its size), using $\log N + \binom{N}{|X_{\mathcal{O},a}|}$ bits.
2. The set $Y_{\mathcal{O},a}$ (along with its size), using $\log M + \binom{M}{|Y_{\mathcal{O},a}|}$ bits.
3. For each $x \in X_{\mathcal{O},a}$, the value $\mathcal{O}(a, x) \in Y_{\mathcal{O},a}$ encoded using $\log |Y_{\mathcal{O},a}|$ bits.
4. For each $x \notin X_{\mathcal{O},a}$, the value $\mathcal{O}(a, x)$ encoded using $\log M$ bits.

Decoding is done in the obvious way. The encoding length of $\mathcal{O}$ (in bits) is

$$K \log N + K \log M + \sum_{a \in [K]} \log \binom{N}{|X_{\mathcal{O},a}|} + \log \binom{M}{|Y_{\mathcal{O},a}|} + |X_{\mathcal{O},a}| \cdot \log |Y_{\mathcal{O},a}| + (N - |X_{\mathcal{O},a}|) \cdot \log M.$$

Using the inequality $\log \binom{A}{B} \leq B \cdot \log \frac{eA}{B}$ and the log-sum[6] inequality, the encoding length of $\mathcal{O}$ (in bits) is at most

$$K \log N + K \log M + \left(\sum_{a \in [K]} |X_{\mathcal{O},a}| \right) \cdot \log \frac{eN \sum_{a \in [K]} |Y_{\mathcal{O},a}|}{M \sum_{a \in [K]} |X_{\mathcal{O},a}|} + \left(\sum_{a \in [K]} |Y_{\mathcal{O},a}| \right) \cdot \log \frac{eKM}{\sum_{a \in [K]} |Y_{\mathcal{O},a}|} + KN \log M. \quad (1)$$

Let $\epsilon' \stackrel{\text{def}}{=} \Pr_{\mathcal{O},a,x}[A_1^{\mathcal{O}}(A_0^{\mathcal{O}}, a, \mathcal{O}(a, x)) = x]$, and note that $\mathbb{E}_{\mathcal{O}}[\sum_a |X_{\mathcal{O},a}|] = \epsilon NK$ and $\mathbb{E}_{\mathcal{O}}[\sum_a |Y_{\mathcal{O},a}|] = \epsilon' NK$. By averaging over $\mathcal{O}$ and log-sum inequality, the average encoding length of $\mathcal{O}$ is upper bounded by replacing $\sum_{a \in K} X_{\mathcal{O},a}$ by $\mathbb{E}_{\mathcal{O}}[\sum_{a \in K} |X_{\mathcal{O},a}|]$ and $\sum_{a \in K} Y_{\mathcal{O},a}$ by $\mathbb{E}_{\mathcal{O}}[\sum_{a \in K} |Y_{\mathcal{O},a}|]$ in (1), namely

$$K \log N + K \log M + (\varepsilon NK) \cdot \log \frac{eN\varepsilon' NK}{M\varepsilon NK} + (\varepsilon' NK) \cdot \log \frac{eKM}{\varepsilon' NK} + KN \log M.$$

Using the fact that (by Lemma 1) the encoding length must be at least $KN \log M$ bits and rearranging the inequality, we obtain

$$\frac{\log N + \log M}{N} + \varepsilon' \cdot \log \frac{eM}{\varepsilon' N} \geq \varepsilon \cdot \log \frac{M\varepsilon}{eN\varepsilon'}.$$

[6] The log-sum inequality states that for nonnegative $t_1, \ldots, t_n$ and $w_1, \ldots, w_n$, it holds that $\sum_{i=1}^n t_i \log(w_i/t_i) \leq \left(\sum_{i=1}^n t_i\right) \cdot \log(\sum_{i=1}^n w_i / \sum_{i=1}^n t_i)$. It also implies the average of $t_1 \log(w_1/t_1), \ldots, t_n \log(w_n/t_n)$ is less that $\bar{t} \log(\overline{w}/\bar{t})$ where $\bar{t}$ is the average of $t_1, \ldots, t_n$ and $\overline{w}$ is the average of $w_1, \ldots, w_n$.

If $\varepsilon = \Omega((\log MN)/N)$, then there exists a sufficiently large constant C such that $\varepsilon N \geq (\log MN)/C$. If $M\varepsilon/(eN\varepsilon') \leq 2^{C+1}$, then $\varepsilon = O(\varepsilon' N/M)$. Otherwise, $(M\varepsilon)/(eN\varepsilon') \geq 2^{C+1}$, then

$$\varepsilon' \log \frac{eM}{\varepsilon' N} \geq \varepsilon(C+1) - (\log MN)/N \geq \varepsilon,$$

which implies $\varepsilon = O(\varepsilon' \log M)$ (here we assume $\varepsilon' N \geq 1$). Overall we get $\varepsilon = O(\varepsilon' \max(\log M, N/M))$. By the bound on ε' from Theorem 3, we obtain the desired bound on ε.

4 Collision-Resistant Hash Functions

In this section, we prove the following theorem.

Theorem 5. *Consider random oracles $\mathcal{O} \in \mathsf{Func}([K] \times [N],\ [M])$. For any oracle algorithms (A_0, A_1) such that A_0 outputs S-bit state and A_1 makes at most T oracle queries,*

$$\Pr_{\mathcal{O},a}[(x,x') := A_1^{\mathcal{O}}(A_0^{\mathcal{O}}, a) : x \neq x' \wedge \mathcal{O}(a,x) = \mathcal{O}(a,x')] = O\left(\frac{S + \log K}{K} + \frac{T^2}{M}\right).$$

The bound in the above theorem matches (up to the $K^{-1} \log K$ term) the parameters achieved by the following: A_0 outputs collisions in $\mathcal{O}(a_i, \cdot)$ for each of $a_1, \ldots, a_S \in [K]$. Then A_1 outputs the appropriate collision if $a = a_i$, and otherwise performs a birthday attack in an attempt to find a collision.

To prove Theorem 5, we first prove the following lemma:

Lemma 4. *Consider random oracles $\mathcal{O} \in \mathsf{Func}([K] \times [N],\ [M])$. Assume there exist oracle algorithms (A_0, A_1) such that A_0 outputs S-bit state and A_1 makes at most T oracle queries, and such that*

$$\Pr_{\mathcal{O},a}[(x,x') := A_1^{\mathcal{O}}(A_0^{\mathcal{O}}, a) : x \neq x' \wedge \mathcal{O}(a,x) = \mathcal{O}(a,x')] = \epsilon.$$

Then there exists a deterministic encoding for the set $\mathsf{Func}([K] \times [N],\ [M])$ with expected encoding length (in bits) at most

$$S + KN \log M + \log K - \frac{\varepsilon K}{2} \log\left(\frac{\varepsilon M}{8eT^2}\right).$$

Proof. Fix $\mathcal{O}\colon [K] \times [N] \to [M]$, and let $\mathsf{st}_{\mathcal{O}} = A_0^{\mathcal{O}}$. Let $G_{\mathcal{O}}$ be the set of $a \in [K]$ such that $A_1^{\mathcal{O}}(\mathsf{st}_{\mathcal{O}}, a)$ outputs a collision in $\mathcal{O}(a, \cdot)$. We assume, without loss of generality, that if $A_1^{\mathcal{O}}(\mathsf{st}_{\mathcal{O}}, a)$ outputs x, x', then it must have queried $\mathcal{O}(a,x)$ and $\mathcal{O}(a,x')$ at some point in its execution. The basic observation is that we can use this to compress $\mathcal{O}(a, \cdot)$ for $a \in G_{\mathcal{O}}$. Specifically, rather than store both $\mathcal{O}(a,x)$ and $\mathcal{O}(a,x')$ (using $2 \log M$ bits), where x, x' is the collision in $\mathcal{O}(a, \cdot)$ output by A_1, we instead store the value $\mathcal{O}(a,x) = \mathcal{O}(a,x')$ *once*, along with the

indices i, j of the oracle queries $\mathcal{O}(a, x)$ and $\mathcal{O}(a, x')$ made by A_1 (using a total of $\log M + 2\log T$ bits). This is a net savings if $2\log T < \log M$. Details follow.

A simple case. To illustrate the main idea, we first consider a simple case where $A_1^{\mathcal{O}}(\mathsf{st}_{\mathcal{O}}, a)$ never makes oracle queries $\mathcal{O}(a', x)$ with $a' \neq a$. Under this assumption, we encode $\mathcal{O}$ as follows:

1. Encode $\mathsf{st}_{\mathcal{O}}$, $|G_{\mathcal{O}}|$, and $G_{\mathcal{O}}$. This requires $S + \log K + \log \binom{K}{|G_{\mathcal{O}}|}$ bits.
2. For each $a \in G_{\mathcal{O}}$ (in lexicographic order), run $A_1^{\mathcal{O}}(\mathsf{st}_{\mathcal{O}}, a)$ and let the second components of the oracle queries of A_1 be $x_1, \ldots, x_T$. (We assume without loss of generality these are all distinct.) If x, x' are the output of A_1, let $i < j$ be such that $\{x, x'\} = \{x_i, x_j\}$. Encode i and j, along with the answers to each of A_1's oracle queries (in order) except for the jth. Furthermore, encode $\mathcal{O}(a, x)$ for all $x \in [N] \setminus \{x_1, \ldots, x_T\}$ (in lexicographic order). This requires $(N-1) \cdot \log M + 2\log T$ bits for each $a \in G_{\mathcal{O}}$.
3. For each $a \notin G_{\mathcal{O}}$ and $x \in [N]$ (in lexicographic order), store $\mathcal{O}(a, x)$. This uses $N \log M$ bits for each $a \notin G_{\mathcal{O}}$.

Decoding is done in the obvious way.

The encoding length of $\mathcal{O}$ (in bits) is

$$S + \log K + \log \binom{K}{|G_{\mathcal{O}}|} + KN \log M - |G_{\mathcal{O}}| \cdot (\log M - 2\log T).$$

Using the inequality $\binom{K}{|G_f|} \leq (\frac{eK}{|G_f|})^{|G_f|}$, the expected encoding length (in bits) is thus

$$\begin{aligned}
&S + \log K + \mathbb{E}_{\mathcal{O}}\left[|G_{\mathcal{O}}| \cdot \log \frac{eK}{|G_{\mathcal{O}}|}\right] \\
&\quad + KN \log M - \mathbb{E}_{\mathcal{O}}[|G_{\mathcal{O}}|] \cdot (\log M - 2\log T) \\
&\leq S + \log K + \mathbb{E}_{\mathcal{O}}[|G_{\mathcal{O}}|] \cdot \log \frac{eK}{\mathbb{E}_{\mathcal{O}}[|G_{\mathcal{O}}|]} \\
&\quad + KN \log M - \mathbb{E}_{\mathcal{O}}[|G_{\mathcal{O}}|] \cdot (\log M - 2\log T) \\
&= S + \log K + KN \log M - \varepsilon K \log\left(\frac{\varepsilon M}{eT^2}\right),
\end{aligned}$$

where the inequality uses concavity of the function $y \cdot \log 1/y$, and the third line uses $\mathbb{E}_{\mathcal{O}}[|G_{\mathcal{O}}|] = \varepsilon K$.

The general case. In the general case, we need to take into account the fact that A_1 may make arbitrary queries to $\mathcal{O}$. This affects the previous approach because $A_1(\mathsf{st}_{\mathcal{O}}, a)$ may query $\mathcal{O}(a', x)$ for a value x that is output as part of a collision by $A_1(\mathsf{st}_{\mathcal{O}}, a')$.

To deal with this, consider running $A_1^{\mathcal{O}}(\mathsf{st}_{\mathcal{O}}, a)$ for all $a \in G_{\mathcal{O}}$. There are at most $T \cdot |G_{\mathcal{O}}|$ distinct oracle queries made overall. Although several of them may share the same prefix $a \in [K]$, there are at most $|G_{\mathcal{O}}|/2$ values of a that are used as a prefix in more than $2T$ queries. In other words, there is a set $G'_{\mathcal{O}} \subseteq G_{\mathcal{O}}$ of

size at least $|G_{\mathcal{O}}|/2$ such that each $a \in G'_{\mathcal{O}}$ is used in at most $2T$ queries when running $A_1^{\mathcal{O}}(\mathsf{st}_{\mathcal{O}}, a)$ for all $a \in G'_{\mathcal{O}}$.

To encode $\mathcal{O}$ we now proceed in a manner similar to before, but using $G'_{\mathcal{O}}$ in place of $G_{\mathcal{O}}$. Moreover, we run $A_1^{\mathcal{O}}(\mathsf{st}_{\mathcal{O}}, a)$ for all $a \in G'_{\mathcal{O}}$ (in lexicographic order) and consider all the distinct oracle queries made. For each $a \in G'_{\mathcal{O}}$, let $i_a < j_a \leq 2T$ be such that the i_ath and j_ath oracle queries that use prefix a are distinct but yield the same output. (There must exist such indices by assumption on A_1.) We encode (i_a, j_a) for all $a \in G'_{\mathcal{O}}$, along with the answers to all the (distinct) oracle queries made with the exception of the j_ath oracle query made using prefix a for all $a \in G'_{\mathcal{O}}$. The remainder of $\mathcal{O}(\cdot,\cdot)$ is then encoded in the trivial way as before. Decoding is done in the natural way.

Arguing as before, but with ϵK replaced by $\epsilon K/2$ and T replaced by $2T$, we see that the expected encoding length (in bits) is now at most

$$S + \log K + KN \log M - \frac{\varepsilon K}{2} \log\left(\frac{\varepsilon M}{8eT^2}\right),$$

as claimed. ■

Lemma 4 gives an encoding for $\mathsf{Func}([K] \times [N],\ [M])$ with expected length at most

$$S + \log K + KN \log M - \frac{\varepsilon K}{2} \log\left(\frac{\varepsilon M}{8eT^2}\right)$$

bits. But Lemma 1 shows that any such encoding must have expected length at least $NK \log M$ bits. We thus conclude that

$$S + \log K \geq \frac{\varepsilon K}{2} \log\left(\frac{\varepsilon M}{8eT^2}\right).$$

This implies Theorem 5 since either $\varepsilon \leq \frac{16eT^2}{M}$ or else $\varepsilon \leq \frac{2S+2\log K}{K}$.

5 Pseudorandom Generators and Functions

In this section, we prove the following theorems.

Theorem 6. *Consider random oracles $\mathcal{O} \in \mathsf{Func}([K] \times [N],\ [M])$ where it holds that $M > N$. For any oracle algorithms (A_0, A_1) such that A_0 outputs S-bit state and A_1 makes at most T oracle queries,*

$$\left|\Pr_{\mathcal{O},a,x}[A_1^{\mathcal{O}}(A_0^{\mathcal{O}}, a, \mathcal{O}(a,x)) = 1] - \Pr_{\mathcal{O},a,y}[A_1^{\mathcal{O}}(A_0^{\mathcal{O}}, a, y) = 1]\right| = O\left(\log M \cdot \left(\sqrt{\frac{ST}{KN}} + \frac{T \log N}{N}\right)\right).$$

Theorem 7. *Consider random oracles $\mathcal{O} \in \mathsf{Func}([K] \times [N] \times [L],\ \{0,1\})$. For any oracle algorithms (A_0, A_1) such that A_0 outputs S-bit state and A_1 makes at most T oracle queries to $\mathcal{O}$ and at most q queries to its other oracle,*

$$\left| \Pr_{\mathcal{O},a,k}[A_1^{\mathcal{O},\mathcal{O}(a,k,\cdot)}(A_0^{\mathcal{O}}, a) = 1] - \Pr_{\mathcal{O},a,f}[A_1^{\mathcal{O},f}(A_0^{\mathcal{O}}, a) = 1] \right| = O\left(q \cdot \left(\sqrt{\frac{ST}{KN}} + \frac{T \log N}{N} \right) \right),$$

where f is uniform in $\mathsf{Func}([L], \{0,1\})$.

Note that in both cases, an exhaustive-search attack (with $S = 0$) achieves distinguishing advantage $\Theta(T/N)$. With regard to pseudorandom generators (Theorem 6), De et al. [7] show an attack with $T = 0$ that achieves distinguishing advantage $\Omega(\sqrt{\frac{S}{KN}})$. Their attack can be extended to the case of pseudorandom functions (assuming $q > \log KN$) to obtain distinguishing advantage $\Omega(\sqrt{\frac{S}{KN}})$ in that case as well.

In proving the above, we rely on the following [7, Lemma 8.4]:

Lemma 5. *Fix a parameter ϵ, and oracle algorithms (A_0, A_1) such that A_0 outputs S-bit state and A_1 makes at most T queries to $\mathcal{O}$ but may not query its input. Let $\mathcal{F} \subseteq \mathsf{Func}([K] \times [N],\ \{0,1\})$ be such that if $\mathcal{O} \in \mathcal{F}$ then*

$$\Pr_{a,x}[A_1^{\mathcal{O}}(A_0^{\mathcal{O}}, a, x) = \mathcal{O}(a, x)] \geq \frac{1}{2} + \varepsilon.$$

Then there is a randomized encoding for $\mathcal{F}$ with recovery probability $\Omega(\varepsilon/T)$ and encoding length (in bits) at most $KN + S - \Omega\left(\frac{\varepsilon^2 NK}{T}\right) + O(1)$.

We now prove Theorem 6.

Proof. Let

$$\epsilon = \left| \Pr_{\mathcal{O},a,x}[A_1^{\mathcal{O}}(A_0^{\mathcal{O}}, a, \mathcal{O}(a,x)) = 1] - \Pr_{\mathcal{O},a,y}[A_1^{\mathcal{O}}(A_0^{\mathcal{O}}, a, y) = 1] \right|.$$

We assume for simplicity that M is a power of 2. By Yao's equivalence of distinguishability and predictability [18], there exist $i \in [\log M]$ and oracle algorithms (B_0, B_1) such that B_0 outputs at most $S+1$ bits and B_1 makes at most T oracle queries, and such that

$$\Pr_{\mathcal{O},a,x}[B_1^{\mathcal{O}}(B_0^{\mathcal{O}}, a, \mathcal{O}_1(a,x), \ldots, \mathcal{O}_{i-1}(a,x)) = \mathcal{O}_i(a,x)] \geq 1/2 + \varepsilon/\log M,$$

where $\mathcal{O}_i(a,x)$ denotes the ith bit of $\mathcal{O}(a,x)$. If B_1 queries (a,x) with probability at least $\varepsilon/2\log M$, we can turn B_1 into an algorithm that inverts $\mathcal{O}(a,x)$ with at least that probability; Theorem 3 then implies

$$\varepsilon = O\left(\log M \cdot \left(\frac{ST}{KN} + \frac{T \log N}{N} \right) \right). \tag{2}$$

Otherwise, we may construct algorithms (C_0, C_1) such that

- C_1 makes at most T oracle queries, and never queries its own input;
- C_0 runs B_0 and also outputs as part of its state the truth table of a function mapping $[K] \times [N]$ to outputs of length at most $(\log M - 1)$ bits;

and such that

$$\Pr_{\mathcal{O}_i,a,x}[C_1^{\mathcal{O}_i}(C_0^{\mathcal{O}_i}, a, x) = \mathcal{O}_i(a, x)] \geq 1/2 + \varepsilon/2 \log M.$$

This means that for at least an $(\varepsilon/4 \log M)$-fraction of $\mathsf{Func}([K] \times [N], \{0,1\})$ it holds that

$$\Pr_{a,x}[C_1^{\mathcal{O}_i}(C_0^{\mathcal{O}_i}, a, x) = \mathcal{O}_i(a, x)] \geq 1/2 + \varepsilon/4 \log M.$$

Lemma 5 thus implies that we can encode that set of functions using at most $KN + KN \cdot (\log M - 1) + S - \Omega\left(\frac{(\varepsilon/\log M)^2 KN}{T}\right) + O(1)$ bits. By Lemma 2, this means we must have

$$\Omega\left(\frac{(\varepsilon/\log M)^2 KN}{T}\right) - \log\left(\frac{\varepsilon}{T}\right) - \log\left(\frac{\varepsilon}{4 \log M}\right) \leq S + O(1),$$

which in turn implies $\varepsilon = O\left(\log M \cdot \sqrt{\frac{ST}{KN}}\right)$. This, combined with (2), implies the theorem. ■

As intuition for the proof of Theorem 7, note that we may view a pseudorandom function as a pseudorandom generator mapping a key to the truth table for a function, with the main difference being that the distinguisher is not given the entire truth table as input but instead may only access parts of the truth table via queries it makes. We may thus apply the same idea as in the proof of Theorem 6, with the output length (i.e., $\log M$) replaced by the number of queries the distinguisher makes. However in this case, Lemma 5 cannot be directly applied and a slightly more involved compression argument is required.

With this in mind, we turn to the proof of Theorem 7:

Proof. Let

$$\epsilon = \left| \Pr_{\mathcal{O},a,k}[A_1^{\mathcal{O},\mathcal{O}(a,k,\cdot)}(A_0^{\mathcal{O}}, a) = 1] - \Pr_{\mathcal{O},a,f}[A_1^{\mathcal{O},f}(A_0^{\mathcal{O}}, a) = 1] \right|.$$

By Yao's equivalence of distinguishability and predictability [18], there exist $i \in [q]$ and oracle algorithms (B_0, B_1) such that B_0 outputs at most $S + 1$ bits and B_1 makes at most T oracle queries to $\mathcal{O}$ and $i \leq q$ distinct queries to the second oracle, such that

$$\Pr_{\mathcal{O},a,k}[B_1^{\mathcal{O},\mathcal{O}(a,k,\cdot)}(B_0^{\mathcal{O}}, a) \text{ outputs } (x, b), \text{ s.t. } \mathcal{O}(a, k, x) = b] \geq \frac{1}{2} + \varepsilon/q,$$

where it is required that B_1 not query x to its second oracle. If B_1 queries $\mathcal{O}$ on any query with prefix (a, k), with probability at least $\varepsilon/2q$, we can turn B_1 into an algorithm that inverts random oracle $\mathcal{O}'$ from $[K] \times [N]$ to $\{0,1\}^L$ with

that probability where the output of $\mathcal{O}'(a,k)$ is the truth table of $\mathcal{O}(a,k,\cdot)$. Theorem 3 then implies

$$\varepsilon = O\left(q \cdot \left(\frac{ST}{KN} + \frac{T \log N}{N}\right)\right). \tag{3}$$

Otherwise, we may construct algorithms C_1 which behaves as B_1 except when B_1 queries $\mathcal{O}$ on any query with prefix (a,k), C_1 outputs a random guess. C_1 satisfies that

$$\Pr_{\mathcal{O},a,k}[C_1^{\mathcal{O},\mathcal{O}(a,k,\cdot)}(B_0^{\mathcal{O}},a) \text{ outputs } (x,b) \text{ s.t. } \mathcal{O}(a,k,x) = b] \geq 1/2 + \varepsilon/2q.$$

This means that for at least an $(\varepsilon/4q)$-fraction of $\mathsf{Func}\left([K] \times [N], \{0,1\}^{[L]}\right)$, it holds that

$$\Pr_{\mathcal{O},a,k}[C_1^{\mathcal{O},\mathcal{O}(a,k,\cdot)}(B_0^{\mathcal{O}},a) \text{ outputs } (x,b) \text{ s.t. } \mathcal{O}(a,k,x) = b] \geq 1/2 + \varepsilon/4q.$$

We can encode the set of functions using randomized encoding. Specifically, the encoder uses randomness r to pick a set $R \subseteq [K] \times [N]$, where each $(a,k) \in [K] \times [N]$ is included in R with probability $1/10T$. For $a \in [K]$, let $G \subseteq R$ be the set of $(a,k) \in R$ such that $C_1^{\mathcal{O},\mathcal{O}(a,k,\cdot)}(B_0^{\mathcal{O}},a)$ does not query $\mathcal{O}$ on any point with prefix $(a',k') \in G$. Let G_0 be the subset of G such that the output of C_1 is correct and $G_1 = G \setminus G_0$.

As in De et al. [7], with probability at least $\varepsilon/160qT$, $|G_0| - |G_1| \geq \frac{\varepsilon KN}{80qT}$ and $|G| = \Omega(\frac{KN}{T})$ hold. To see this, note by a Chernoff bound, G has $\Omega(\frac{KN}{T})$ points with probabiliyt at least $1 - e^{-\frac{2KN}{T}}$. The expected difference between $|G_0|$ and $|G_1|$ is at least $\frac{\varepsilon KN}{40qT}$. By averaging argument, with probability at least $\frac{\varepsilon}{80qT}$ their difference is at least $\frac{\varepsilon KN}{80qT}$. So with probability at $\frac{\varepsilon}{80qT} - e^{-\frac{2KN}{T}} \geq \frac{\varepsilon}{160qT}$, both events happen. Conditioned on that, we encode $\mathcal{O}$ as follows (otherwise we output empty string):

1. Include $B_0^{\mathcal{O}}$. This uses at most $S+1$ bits.
2. For each $(a,k) \in ([K] \times [N]) \setminus R$ (in lexicographic order), include the truth table of $\mathcal{O}(a,k,\cdot)$. Then for each $(a,k) \in R \setminus G$ (in lexicographic order), include the truth table of $\mathcal{O}(a,k,\cdot)$. This uses a total of $(KN - |G|) \cdot L$ bits.
3. Include a description of G_0. This uses $\log \binom{|G|}{|G_0|}$ bits.
4. For each $(a,k) \in G$ (in lexicographic order), include in the encoding the answers to all the oracle queries made by C_1 to the second oracle $\mathcal{O}(a,k,\cdot)$, and for every x such that (a,k,x) is not queried by C_1 to $\mathcal{O}(a,k,\cdot)$ and x is not the output of C_1, add $\mathcal{O}(a,k,x)$ to the encoding. This uses a total of $|G|\,(L-1)$ bits.

To decode, the decoder first uses r to recover the set R defined above. Then it does the following:

1. Recover $B_0^{\mathcal{O}}$.

2. For each $(a,k) \in ([K] \times [N]) \setminus R$, recover the truth table of $\mathcal{O}(a,k,\cdot)$. Identify set G by running C_1 with $B_0^{\mathcal{O}}$ on $(a,k) \in R$ because if C_1 on (a,k) only makes query outside R, then $(a,k) \in G$. Go over $(a,k) \in R \setminus G$, and recover the truth table of $\mathcal{O}(a,k,\cdot)$.
3. Recover G_0.
4. For each $(a,k) \in G$, run $C_1(B_0^{\mathcal{O}}, a)$ while answering the oracle queries to the first oracle using recovered values and to the second oracle using the values stored in the encoding. Suppose C_1 outputs x, b, if $(a,k) \in G_0$, recover $\mathcal{O}(a,k,x) = b$ otherwise $\mathcal{O}(a,k,x) = 1-b$. After that for which $\mathcal{O}(a,k,x)$ is not yet defined, recover the value of $\mathcal{O}(a,k,x)$ from the remainder of the encoding.

Because we condition on $|G| \leq KN/T$ and $|G_0| - |G_1| \geq \varepsilon KN/80qT$ which implies $\log \binom{|G|}{|G_0|} \leq |G|\, H(1/2 + \varepsilon KN/80T\,|G|) \leq |G| - \Omega((\varepsilon/q)^2 KN/T)$, where H is the binary entropy function. The maximal length is at most

$$KNL + S + 1 + \log \binom{|G|}{|G_0|} - |G| \leq KNL + S + O(1) - \Omega((\varepsilon/q)^2 KN/T).$$

By Lemma 2, we have

$$S \geq \Omega((\varepsilon/q)^2 KN/T) - \log \Omega(\frac{\varepsilon}{160qT}) - \log(\frac{\varepsilon}{4q}).$$

which implies $\varepsilon \leq O(q \cdot \sqrt{\frac{ST}{KN}})$. Overall we obtain $\varepsilon \leq O(q \cdot (\sqrt{\frac{ST}{KN}} + \frac{T}{N} \cdot \log N))$. ■

6 Message Authentication Codes (MACs)

In this section, we prove the following theorem.

Theorem 8. *Consider random oracles $\mathcal{O} \in \mathsf{Func}([K] \times [N] \times [L],\ [M])$. For any oracle algorithms (A_0, A_1) such that A_0 outputs S-bit state and A_1 makes at most T queries to $\mathcal{O}$,*

$$\Pr_{\mathcal{O},a,k}\left[(m,t) := A_1^{\mathcal{O},\mathcal{O}(a,k,\cdot)}(A_0^{\mathcal{O}}, a)\ :\ \mathcal{O}(a,k,m) = t\right] = O\left(\frac{ST}{KN} + \frac{T}{M} + \frac{T \log N}{N}\right),$$

where it is required that A_1 not query m to its second oracle.

Note that any generic inversion attack can be used to attack the above construction of a MAC by fixing some $m \in [L]$ and then inverting the function $\mathcal{O}(a,\cdot,m)$ given a; in this sense, it is perhaps not surprising that the bound above contains terms $\mathcal{O}\left(\frac{ST}{KN} + \frac{T \log N}{N}\right)$ as in Theorem 3. There is, of course, also a trivial guessing attack that achieves advantage $1/M$.

Proof. If A_1 queries $\mathcal{O}$ on any query with prefix (a,k), with probability at least $\varepsilon/2$, we can turn A_1 into an algorithm that inverts random oracle $\mathcal{O}'$ from $[K]\times[N]$ to $[M^L]$ with that probability where the output of $\mathcal{O}'(a,k)$ is the truth table of $\mathcal{O}(a,k,\cdot)$. Then by Theorem 3, we obtain $\varepsilon \le O(\frac{ST}{KN}+\frac{T\log N}{N})$. Otherwise, we may construct algorithms B_1 which behaves as A_1 except when B_1 queries $\mathcal{O}$ on any query with prefix (a,k), B_1 outputs a random guess. B_1 satisfies that

$$\Pr_{\mathcal{O},a,k}[B_1^{\mathcal{O},\mathcal{O}(a,k,\cdot)}(A_0^{\mathcal{O}},a) \text{ outputs } (m,t) \text{ s.t. } \mathcal{O}(a,k,m)=t] \ge \varepsilon/2.$$

where it is required that B_1 not query m to its second oracle.

Fix $\mathcal{O}\colon [K]\times[N]\times[L]\to[M]$. Let $U_{\mathcal{O}}$ be the set of (a,k) such that B_1 succeeds on (a,k). Let $G_{\mathcal{O}}$ be the subset of $U_{\mathcal{O}}$ such that for every $(a,k)\in G_{\mathcal{O}}$, $B_1^{\mathcal{O},\mathcal{O}(a,k,\cdot)}$ does not query its first oracle with any query with prefix $(a',k')\in G_{\mathcal{O}}$. Because B_1 makes at most T queries, there exists $G_{\mathcal{O}}$ with size at least $|U_{\mathcal{O}}|/(T+1)$.

We can encode $\mathcal{O}$ as follows.

1. Include $A_0^{\mathcal{O}}$, $|G_{\mathcal{O}}|$ and a description of $G_{\mathcal{O}}$. This uses a total of $S+\log KN+\log\binom{NK}{|G_{\mathcal{O}}|}$ bits.
2. For each $(a,k)\in([K]\times[N])\setminus G_{\mathcal{O}}$ (in lexicographic order), include the truth table of $\mathcal{O}(a,k,\cdot)$. This uses a total of $(KN-|G_{\mathcal{O}}|)\cdot L\log M$ bits.
3. For each $(a,k)\in G_{\mathcal{O}}$ (in lexicographic order), include in the encoding the answers to all the oracle queries made by B_1 to the second oracle $\mathcal{O}(a,k,\cdot)$, and then for every m such that (a,k,m) is not queried by C_1 to $\mathcal{O}(a,k,\cdot)$ and m is not the output of C_1, add $\mathcal{O}(a,k,m)$ to the encoding. This uses a total of $|G|\,(L-1)\log M$ bits.

Decoding is done in the obvious way. The encoding length is at most

$$KNL\log M+S+\log KN+\log\binom{KN}{|G_{\mathcal{O}}|}-|G_{\mathcal{O}}|\log M$$

By $\log\binom{KN}{|G_{\mathcal{O}}|}\le|G_{\mathcal{O}}|\log\frac{eKN}{|G_{\mathcal{O}}|}$ and log-sum inequality, the average length over all possible $\mathcal{O}$ is at most

$$KNL\log M+S+\log KN+\mathbb{E}[|G_{\mathcal{O}}|]\log\frac{eKN}{M\cdot\mathbb{E}_f[|G_{\mathcal{O}}|]}.$$

But Lemma 1 shows that any such encoding must have expected length at least $KNL\log M$ bits. We thus conclude that

$$S+\log KN\ge\mathbb{E}[|G_{\mathcal{O}}|]\log\frac{M\,\mathbb{E}[|G_{\mathcal{O}}|]}{eKN}\ge\frac{\varepsilon NK}{2(T+1)}\log\frac{M\varepsilon}{2e(T+1)}.$$

where the second inequality is due to the monotonicity of $y\log y$ for $y\ge1$ and $\mathbb{E}[|G_{\mathcal{O}}|]\ge\mathbb{E}[\frac{|U_{\mathcal{O}}|}{T+1}]\ge\frac{\varepsilon NK}{2(T+1)}$. This implies Theorem 8 since either $\varepsilon\le\frac{4e(T+1)}{M}$ or else $\varepsilon\le\frac{2(S+\log KN)(T+1)}{NK}=O(\frac{ST}{NK}+\frac{T\log N}{N})$. ∎

Acknowledgments

Jonathan Katz thanks Christine Evangelista, Aaron Lowe, Jordan Schneider, Lynesia Taylor, Aishwarya Thiruvengadam, and Ellen Vitercik, who explored problems related to salting and rainbow tables as part of an NSF-REU program in the summer of 2014.

References

1. Noga Alon, Oded Goldreich, Johan Håstad, and Rene Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures and Algorithms*, 3(3):289–304, 1992.
2. Elad Barkan, Eli Biham, and Adi Shamir. Rigorous bounds on cryptanalytic time/memory tradeoffs. In *Advances in Cryptology—Crypto 2006*, volume 4117 of *LNCS*, pages 1–21. Springer, 2006.
3. Mihir Bellare, Thomas Ristenpart, and Stefano Tessaro. Multi-instance security and its application to password-based cryptography. In *Advances in Cryptology—Crypto 2012*, volume 7417 of *LNCS*, pages 312–329. Springer, 2012.
4. Mihir Bellare and Phil Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conf. on Computer and Communications Security*, pages 62–73. ACM Press, 1993.
5. Daniel J. Bernstein and Tanja Lange. Non-uniform cracks in the concrete: The power of free precomputation. In *Advances in Cryptology—Asiacrypt 2013, Part II*, volume 8270 of *LNCS*, pages 321–340. Springer, 2013.
6. Daniel J. Bernstein and Tanja Lange. Non-uniform cracks in the concrete: The power of free precomputation. In *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, pages 321–340, 2013.
7. Anindya De, Luca Trevisan, and Madhur Tulsiani. Time space tradeoffs for attacks against one-way functions and PRGs. In *Advances in Cryptology—Crypto 2010*, volume 6223 of *LNCS*, pages 649–665. Springer, 2010. Version dated February 17, 2010 available at `http://ttic.uchicago.edu/~madhurt/Papers/dtt-new.pdf`.
8. Yevgeniy Dodis and John P. Steinberger. Message authentication codes from unpredictable block ciphers. In *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, pages 267–285, 2009.
9. Amos Fiat and Moni Naor. Rigorous time/space trade-offs for inverting functions. *SIAM Journal on Computing*, 29(3):790–803, 1999.
10. Rosario Gennaro, Yael Gertner, Jonathan Katz, and Luca Trevisan. Bounds on the efficiency of generic cryptographic constructions. *SIAM J. Computing*, 35(1):217–246, 2005.
11. Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 305–313. IEEE, 2000.
12. Martin Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Information Theory*, 26(4):401–406, 1980.
13. Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, 2nd edition*. Chapman & Hall/CRC Press, 2014.

14. Robert Morris and Ken Thompson. Password security: A case history. *Communications of the ACM*, 22(11):594–597, 1979.
15. Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology—Crypto 2003*, volume 2729 of *LNCS*, pages 617–630. Springer, 2003.
16. Phillip Rogaway. Formalizing human ignorance. In *Progressin Cryptology - VIETCRYPT 2006, First International Conferenceon Cryptology in Vietnam, Hanoi, Vietnam, September 25-28, 2006, Revised Selected Papers*, pages 211–228, 2006.
17. Dominique Unruh. Random oracles and auxiliary input. In *Advances in Cryptology—Crypto 2007*, volume 4622 of *LNCS*, pages 205–223. Springer, 2007.
18. Andrew C. Yao. Theory and applications of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 80–91. IEEE, 1982.
19. Andrew C.-C. Yao. Coherent functions and program checkers. In *22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 84–94. ACM Press, 1990.