How to Simulate Random Oracles with Auxiliary Input

Yevgeniy Dodis [1] Aayush Jain [2] Huijia Lin [3]
罗辑 (Ji Luo) [3] Daniel Wichs [4,5]

[1] New York University
dodis@cs.nyu.edu

[2] Carnegie Mellon University
aayushja@andrew.cmu.edu

[3] Paul G. Allen School of Computer Science & Engineering,
University of Washington, Seattle
{rachel,luoji}@cs.washington.edu

[4] Northeastern University
wichs@ccs.neu.edu

[5] NTT Research

1 May 2026

Abstract

The *random oracle model* (ROM) allows us to optimistically reason about security properties of cryptographic hash functions, and has been hugely influential in designing practical cryptosystems. But it is overly optimistic against non-uniform adversaries, and often suggests security properties and security levels unachievable by any real hash function. To reconcile with this discrepancy, Unruh [CRYPTO '07] proposed the *auxiliary-input random oracle model* (AI-ROM), where a non-uniform attacker additionally gets a bounded amount of advice about the random oracle.

Proving security in the AI-ROM is often much more difficult, but a series of works starting with Unruh provided useful technical tools to do so. Although these tools lead to good results in the information-theoretic setting, they are unsatisfactory in the computational setting, where the random oracle is used alongside other computational hardness assumptions. At the most basic level, we did not even know whether it is possible to efficiently simulate random oracle queries given auxiliary input, which has remained as an explicit open problem since the work of Unruh.

In this work, we resolve the above open problem and show how to efficiently simulate auxiliary-input random oracles. Moreover, the simulation has low concrete overhead, leading to small losses in exact security. We use it to prove the security of a broad class of computational schemes in the AI-ROM, including the first non-interactive zero-knowledge (NIZK) scheme in the AI-ROM. As a tool of independent interest, we develop a new notion of ultra-secure pseudorandom functions with fast RAM evaluation, which can achieve 2^λ security while having sublinear $o(\lambda)$ evaluation time.

This is the full version of a paper with the same title (DOI 10.1109/FOCS61266.2024.00080) in the proceedings of FOCS 2024 published by IEEE.

Contents

1 Introduction **1**
- 1.1 Presampling, Limitations, and Need for Better Simulation 3
- 1.2 Our Results in a Nutshell 5
- 1.3 Our Results in Detail and Techniques 7
- 1.4 How to Construct Fast Pseudorandom Functions 12

2 Preliminaries **15**
- 2.1 Auxiliary-Input Random Oracle Model 16
- 2.2 Leakage Simulation 18
- 2.3 Exponentially Secure Pseudorandom Functions 19

3 Simulating AI-ROM Using PRF **20**
- 3.1 Comparison with Presampling 21
- 3.2 Proofs 22

4 PRF with Fast RAM Evaluation **24**
- 4.1 Construction from Goldreich's Function 24
- 4.2 How to Sample and Access the Expander 27
- 4.3 Discussion on the Assumption 31

5 Direct Applications **32**
- 5.1 Resolving Unruh's Conjecture 32
- 5.2 Tighter Reductions from Standard-Model Assumptions 32

6 Combining Presampling and Our Simulation **35**
- 6.1 Salting Defeats Auxiliary Information 36
- 6.2 Non-Interactive Zero-Knowledge in AI-ROM 38

References **40**

Appendix

A Leakage Simulation Lemma **45**

1 Introduction

Random Oracles. The random oracle model (ROM) [BR93] is an idealization of cryptographic hash functions into public random functions $\mathcal{O}$ that can be queried on arbitrary inputs. This optimistic modeling attempts to capture the intuition that hash functions can satisfy a wide range of security properties beyond standard ones such as one-wayness or collision resistance. The ROM has been hugely successful and influential in allowing us to easily design and analyze cryptographic schemes that use hash functions. Essentially all cryptosystems used in practice, including Fiat–Shamir or full-domain-hash signatures (e.g., DSA, Schnorr, RSA), CCA encryption (e.g., RSA-OAEP), and Bitcoin, are only shown to be secure in the ROM. Unfortunately, we have no formal proof that security in the ROM implies security in the real world, when the oracle is replaced with a good hash function, and there are contrived counterexamples [CGH98,GK03] showing that this is not always the case. Nevertheless, security in the ROM is seen as providing very strong heuristic evidence of security in the real world. Moreover, the ROM is not only used to analyze security in the asymptotic sense, but is also used to analyze the *exact security* of a scheme, which is crucial for choosing concrete parameters.

Non-Uniformity. Unfortunately, the ROM does not adequately capture *non-uniform adversaries* (i.e., families of circuits or TM/RAM with advice), and security in the ROM cannot be used as heuristic evidence for real-world security in the non-uniform setting. For example, it is easy to construct seedless collision-resistant hash functions in the ROM, but they are known to be impossible against non-uniform adversaries in the real world. Indeed, a non-uniform attacker can simply hardwire a collision into the (short) advice string.

In a different vein, the ROM gives us incorrect predictions for the security levels that we can expect against non-uniform adversaries. For example, the seminal work of Hellman [Hel80] showed that a non-uniform attacker of size $2^{2\lambda/3}$ can invert a random length-preserving function on λ bits with constant probability, and Fiat and Naor [FN91] extended this to any function using circuits of size $2^{3\lambda/4}$. In contrast, in the ROM, the λ-bit random oracle itself can easily seen to be $Q/2^{\lambda}$ hard to invert against attackers making Q random oracle queries, which gives negligible inversion probability for $Q = 2^{3\lambda/4}$. Moreover, this gap is far from theoretical. The famous *rainbow tables* [Oec03] were developed based on these ideas, and are used on a massive scale to crack weak passwords in the real world. As another example of this kind, one can construct a simple pseudorandom generator (PRG) in the ROM with seed length λ, for which no adversary running in time Q can distinguish the output from uniform with advantage better than $Q/2^{\lambda}$. However, for every real PRG, the works of (e.g.) [AGHP90,DTT10] show that there is a simple non-uniform adversary running in time $O(\lambda)$ that has significantly higher distinguishing advantage $2^{-\lambda/2}$. Similar discrepancies between the security levels achievable in the ROM versus the real world in the non-uniform setting exist for many other primitives (see [Unr07a,DGK17, CDGS18]).

To summarize, while the ROM gives good heuristic predictions for the security properties and levels that cryptographic hash functions can achieve against uniform adversaries, it frequently gives wrong, overly optimistic predictions in the non-uniform setting.

Auxiliary-Input ROM. To bridge the above discrepancy, Unruh [Unr07a] defined the *auxiliary-input random oracle model* (AI-ROM) as a way of capturing non-uniform attacks in the ROM. In the AI-ROM, a random function $\mathcal{O}$ is chosen initially, but the adversary is then given some S bits of arbitrary auxiliary input $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O})$ about the function $\mathcal{O}$. The mapping ai can be an arbitrary function and may not be efficient or even computable. The adversary gets the auxiliary input z and oracle access to $\mathcal{O}$, after which it gets to break the cryptosystem by making up to Q additional queries to $\mathcal{O}$. The honest users only get oracle access to $\mathcal{O}$, as in the standard ROM. We can think of S and Q as an arbitrary polynomial when discussing polynomial security, or can put precise bounds when discussing exact security.[1]

Unlike the ROM, the AI-ROM appears to rule out many security properties that are "too good to be true" in the non-uniform setting. For example, one cannot construct seedless collision-resistant hash functions in the AI-ROM since a collision (of any compressing function determined by $\mathcal{O}$) can be provided as part of the auxiliary input z. Moreover, the generic non-uniform attacks against one-way functions, pseudorandom generators, and other primitives discussed above carry over to the AI-ROM, ensuring that for these primitives in the AI-ROM, one cannot achieve exact security levels known to be impossible in the plain model.

Proving Security in AI-ROM. While the AI-ROM prevents us from proving many unrealistic security properties, the flip side is that it makes it difficult to prove anything. Even simple properties that are obvious in the ROM become difficult to prove in the AI-ROM. This is because many useful proof techniques in the ROM do not seem to extend to AI-ROM, including "lazy sampling", (dynamic) programmability of the oracle, and the "unpredictability-to-randomness" conversion. For example, lazy sampling allows one to not commit to the entire truth table of the oracle upfront, and instead answer every fresh query with a fresh random value. Unfortunately, this cannot be done naïvely in the AI-ROM, where the auxiliary input z may be correlated with the entire oracle, and choosing a fresh random value will be easily detectable by the attacker. This leads to the main goal of this work.

> ***Main Goal:*** *Develop techniques for showing* tight security bounds *of cryptographic applications in the AI-ROM.*

As we will see below, this question has been relatively well understood for what we call *information-theoretic applications* of the AI-ROM. This means that the only computational component in the security of such application comes from the use of cryptographic hash function H. Once H has been abstracted as AI-ROM $\mathcal{O}$ with parameters S, Q, the attacker can be assumed to be otherwise computationally unbounded.[2] Examples of such applications include pseudorandomness, one-wayness, or ("salted") collision resistance of the hash function. While very useful in themselves, in most real-world applications — Fiat–Shamir signatures, full-domain hash, Bitcoin, RSA-OAEP among many many others — the hash function is only part of the larger system, and additional computational assumptions should be used. We will see that the existing state-of-the-art techniques give extremely poor exact security bounds

[1] Moreover, in the circuit model we can often view $S = Q$ as the size of the adversary's non-uniform circuit, while in the RAM model it is more convenient to separate the two.

[2] In the sense that placing additional computational limitations does not appear to help proving stronger results for such applications.

for such *computational* applications, despite the fact we believe all these applications are actually secure. The main result of this work will be a *new AI-ROM simulation technique* to give a *much tighter solution* to our main goal for *all* such computational applications.

Presampling Technique. The most general known solution to our main goal comes from the beautiful work of Unruh [Unr07a] (which also defined AI-ROM). The technique is called *presampling* and could be thought as a particular AI-ROM simulation method, which effectively reduces AI-ROM to a "friendlier" intermediate model termed *bit-fixing* ROM (BF-ROM) by [CDGS18]. Unlike (S,Q)-AI-ROM, which has two parameters S (size of auxiliary input) and Q (number of online queries), the P-BF-ROM has a single parameter P, allowing the attacker to fix the oracle $\mathcal{O}'$ arbitrarily at any set of P points in the domain. After this, the values of $\mathcal{O}'$ on all the remaining points are chosen randomly and independently, just as in the ROM. Intuitively, as long as a given application can avoid using the P "presampled" points for security, traditional ROM techniques (including lazy sampling and others) are back in business.

More formally, Unruh showed that any adversary who gets S-bit auxiliary input $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O})$ and makes Q oracle queries to $\mathcal{O}$ (call it an (S,Q)-attacker) cannot distinguish interacting with $\mathcal{O}$ from interacting with an oracle $\mathcal{O}'$ in the P-BF-ROM, where the P points of presampling and the oracle outputs are allowed to depend on the function ai and the value $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O})$. In terms of concrete bounds, the distinguishing advantage ε is upper-bounded by $\sqrt{SQ/P}$, and this was later improved by [CDGS18] to SQ/P. Moreover, the latter bound $\varepsilon \le SQ/P$ is known to be tight at this level of generality [CDGS18].

The work of [CDGS18] also showed that one can take fewer presampled points $P = \mathrm{O}(SQ)$ and still ensure small "multiplicative distinguishing advantage" — any event that happens with probability at most δ when the adversary has oracle access to $\mathcal{O}'$ can happen with probability at most 2δ when the adversary has oracle access to $\mathcal{O}$. This is useful for analyzing unpredictability (e.g., one-wayness) rather than indistinguishability (e.g., pseudorandomness) applications. We also note that several other works [GT00,DTT10,DGK17] explored other approaches to proving specific security properties in the AI-ROM, directly without going through presampling. However, all these techniques suffer even worse limitations than presampling, which we discuss next.

1.1 Presampling, Limitations, and Need for Better Simulation

Presampling is a very general and powerful technique. For a simple information-theoretic example, presampling allows us to easily prove that a length-doubling random oracle $\mathcal{O} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ is a good PRG in the AI-ROM. We first replace $\mathcal{O}$ by $\mathcal{O}'$ at security loss $\varepsilon = SQ/P$ and then replace $\mathcal{O}'(x)$ for a random seed $x \xleftarrow{\$} \{0,1\}^\lambda$ by a uniformly random output at security loss $(P+Q)/2^\lambda$, reflecting the probability that x falls into the set of the P presampled points or that the adversary queries the oracle on x. Since in this simple example, PRG is our final goal, we can optimize the value of P for the best-possible bound. Concretely, by setting $P = \sqrt{2^\lambda SQ}$, the total distinguishing advantage is $\mathrm{O}(\sqrt{SQ/2^\lambda} + Q/2^\lambda)$, negligible for polynomial S, Q. Moreover, for polynomial S, Q, it asymptotically matches the known linear-time $2^{-\lambda/2}$

attack on all non-uniform PRG.[3]

Computational Applications? Unfortunately, things start to go astray once we add computational assumptions to the picture. In fact, the difficulty already comes up when we try analyzing schemes that do not make any meaningful use of the AI-ROM! This question is interesting in it own right, as it directly compares the power of AI-ROM versus the standard non-uniform model for *standard-model problems*, which make sense in both models. It also must be settled before we go for the real question we care about — applications which use cryptographic hash functions *together with* additional computational assumptions, i.e., virtually all real-world applications.

For example, say we want to argue that the decisional Diffie–Hellman (DDH) problem is hard in the AI-ROM. At first, this may appear obvious — the DDH problem does not involve any oracles, so why would it be any easier in the AI-ROM than in the plain model?! But if we want to reduce breaking DDH in the plain model to breaking it in the AI-ROM, we need to *efficiently simulate AI-ROM queries*. In the standard ROM, this is trivial via lazy sampling, but as we already saw, this technique does not work in the AI-ROM. Again, on an intuitive level, such queries should be completely useless in breaking DDH. Formally, though, it is unclear how to do such simulation without the attacker noticing some inconsistency with its advice (and perhaps refusing to break DDH for this reason). Indeed, without the ability to simulate the oracle, we cannot formally argue that the AI-ROM does not give the adversary some unspecified computational power, which may perhaps even suffice to break the DDH assumption!

Unruh's Conjecture. The above issue was already noted by Unruh, who conjectured — but left it as one of the main open problems — that "common sense wins". Namely, efficient simulation of the AI-ROM should be possible [Unr07b; Conjecture 15]:

> *Consider an adversary $\mathcal{A}$ that gets some S-bit auxiliary input $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O})$ and makes oracle queries to $\mathcal{O}$. Is there an efficient non-uniform simulator $\mathcal{S}$ that simulates the view of $\mathcal{A}$ such that no efficient environment can tell apart whether it is interacting with $\mathcal{A}^{\mathcal{O}}(z)$ or $\mathcal{S}(w)$? What is the concrete efficiency of $\mathcal{S}$ compared to that of $\mathcal{A}$?*

The environment does not have access to the oracle. For example, we can think of the environment as the challenger for DDH. In that case, for any attacker $\mathcal{A}^{\mathcal{O}}(z)$ in the AI-ROM that breaks DDH, a similarly efficient non-uniform simulator $\mathcal{S}$ in the plain model should break DDH as well.

One could attempt to employ presampling to construct the simulator $\mathcal{S}$. The non-uniform simulator would have $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O})$ as advice along with the table of P presampled input/output pairs. It would then run $\mathcal{A}$, and answer any oracle query that falls into the table of presampled inputs with the corresponding presampled output and use lazy sampling on all the other inputs. Unfortunately, to achieve negligible distinguishing advantage ε for adversaries making Q oracle queries, the non-uniform advice that the simulator needs would be of size SQ/ε, which is super-polynomial! In terms of concrete efficiency, it looks even worse. For example, to argue that DDH with certain parameters has $\varepsilon = 2^{-\lambda}$ security against circuits of size 2^{λ} (bounding both S, Q by 2^{λ}) in the AI-ROM, we would need DDH with those parameters to have

[3] Interestingly, the bound is not tight for some larger super-polynomial ranges of S, Q, but this was subsequently settled by [GGKL21].

security $\varepsilon = 2^{-\lambda}$ against much larger circuits of size $2^{4\lambda}$ in the plain model, by setting $P = SQ/\varepsilon = 2^{3\lambda}$ and noting that the above described simulator $\mathcal{S}$ needs to scan through the table of size P on every oracle query made by $\mathcal{A}$.[4]

Perhaps one could hope to get better simulation by improving the analysis of presampling, to allow for fewer presampled values P. Unruh conjectured this to be possible [Unr07a; Section 6] with $P = \text{poly}(S, Q)$ and a negligible ε. Unfortunately, the work of [DGK17] showed that this conjecture is false and that $\varepsilon = \Theta(SQ/P)$ is optimal. Therefore, to get better simulation, an entirely new technique is necessary.

We also notice that the situation is also far from satisfactory for unpredictability applications, such as computational Diffie–Hellman problem (CDH), despite the more optimized presampling for such applications [CDGS18]. Indeed, even though the size of the presampled set is reduced to $P = O(SQ)$, it is still quadratic in S when $Q = \Omega(S)$. Therefore, to convert such $(S, \Omega(S))$-attacker in the AI-ROM to the standard non-uniform model, the non-uniform advice (which includes the presampled set) will have quadratic size $S' = \Omega(P) = \Omega(S^2)$. Moreover, the actual simulation (in the circuit model) might take even cubic time $\Omega(S^3)$, as each evaluation inside the presampled set will take time $\Omega(S') = \Omega(S^2)$.[5] Thus, even though this resolves a variant of Unruh's conjecture for unpredictability up to polynomial factors, it does not match our intuition expecting $S' = O(S)$ to hold.

Need for Better Simulation. To summarize, while presampling works in converting AI-ROM attackers to standard non-uniform ones, the reduction is not efficient in the asymptotic sense (polynomial/negligible) nor in terms of concrete security. In particular, it gives too pessimistic bounds in the non-uniform setting.[6] Put differently, for computational applications of hash functions, the route through the intermediate BF-ROM appears to be too wasteful, and a better simulation technique is needed. In this work, we present such a technique and then use it to prove various computational security notions in the AI-ROM.

1.2 Our Results in a Nutshell

Key Idea: Use a PRF. The main idea of our new technique is to use a pseudorandom function (PRF) to simulate AI-ROM in the standard non-uniform model. The new non-uniform advice z' will be a key k for the PRF, together with the S-bit output of the AI-ROM leakage function ai applied to the *entire truth table V of the PRF*. As such, the adversary $\mathcal{A}$ would not be able to find any inconsistencies of the simulated leakage not matching the evaluation. Of course, this (necessary) step is still insufficient, as our goal will be to make the PRF key k as short as possible (say, $O(S)$ bits long) and the (computationally unbounded) leakage function ai might be able to tell the truth table V of a PRF from an exponentially long truly random string. Here we will rely on a beautiful sequence of works [JP14,CCL18], which shows that any "short enough" (in our case, S-bit long) leakage $\mathsf{ai}(\mathcal{O})$ could be replaced by a different leakage function $h(\mathcal{O})$, where h is "computationally efficient". Once this is done, we will be able to

[4] This last 2^{λ} factor can be removed if we consider non-uniform simulation in the RAM model instead of the circuit model.

[5] In the RAM model, we still get a quadratic slow-down.

[6] This is the opposite problem to the one we had when adapting standard ROM bounds to the non-uniform setting, where the bounds were unrealistically good.

formally use the PRF security to argue that the actual (efficient) non-uniform attacker should not be able to detect the fact that we replaced $\mathsf{ai}(\mathcal{O})$ by $h(V)$.

Super-Fast PRF. Ignoring some subtleties of parameter choices, one issue remains — efficiency of the PRF. Namely, in the AI-ROM, the cost of Q queries is abstracted out as equal to Q. With our PRF simulation, however, each such call will require evaluating the PRF on some input, and the key k for this PRF will be at least S bits long (otherwise, we cannot apply the leakage simulation technique of [JP14,CCL18]). Hence, if done naïvely, we will spend at least time $\Omega(SQ)$ to simulate the Q random oracle queries from the attacker. This is still interesting and will compare favorably with presampling, but we will try to do better. Namely, as a contribution of independent interest, we define and construct a novel kind of PRF with λ bits of security and key length (think $\lambda \sim S$), where individual PRF evaluations take time polylogarithmic in λ (and polynomial in the input length). In particular, each individual PRF evaluation only reads a small subset of the key bits.

Applications. Taken together, we will achieve a novel kind of AI-ROM simulation in the standard non-uniform model, where $S' \sim S$ and $Q' \sim Q$ for S' being the size of the new advice and Q' being the time to simulate the oracle queries. This immediately *settles Unruh's conjecture in the affirmative*, with essentially tight simulation, confirming our intuition that AI-ROM should not help with standard-model problems.

Unfortunately, our new simulation by itself is not sufficient for achieving our main goal — security of computational applications in the AI-ROM. This is because once we simulate AI-ROM by a PRF, we lose all the nice (usually information-theoretic) properties of the random oracle, which are needed to analyze the hash-function part of the applications. For example, if we analyze hashed ElGamal encryption, our simulation will allow us to use the DDH assumption to replace the Diffie–Hellman group element by a random group element, but now it is unclear how to argue that applying the hash function effectively acts as a PRG (from group to strings), as the hash function is already replaced by a PRF whose key is no longer secret.

We develop a relatively general template to this dilemma, by *combining* our new PRF-simulation with presampling. We describe this in more detail in Section 1.3, here only highlighting the main insight. In essence, our template allows us to combine the best features of both simulation methods, without inheriting any of the disadvantages. Concretely, we borrow a fresh-independent random oracle part from presampling, which allows us to use ROM techniques (such as lazy sampling) to tackle the hash-function part of the analysis, provided honest parties avoid evaluating the hash function on the presampled set P. On the other hand, we no longer store the (quite long) truth table of P as an advice, and instead use our more efficient PRF simulation for the behavior of the random oracle inside P (which will likely be used by the attacker). Some subtleties need to be addressed to make it work, but we mention the two main applications where we overcame these subtleties.

- Extending the previous information-theoretic proofs of [DGK17,CDGS18] for information-theoretic applications in the AI-ROM, we show that *salting generically defeats preprocessing* for all (even computational) AI-ROM applications, with a tighter reduction than before.

- We present the first known construction of *non-interactive zero-knowledge (NIZK)*

proofs in the AI-ROM. When the hash function is instantiated, this gives the first heuristic NIZK candidate which is secure against preprocessing attacks.

We now describe these results, including the two applications above, in more detail.

1.3 Our Results in Detail and Techniques

Consider a random oracle $\mathcal{O} : \{0,1\}^* \to \{0,1\}^\lambda$ mapping arbitrary input strings to λ-bit outputs. Sometimes, we also consider restricted domain $\{0,1\}^n$ with fixed input length n, for some polynomial n in λ.

As mentioned above, the goal is to simulate the random oracle $\mathcal{O}$ and advice $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O})$, using an efficient simulator $\mathcal{S}$ that receives some (non-uniform) advice w — meaning that $\mathcal{S}^{\mathcal{A}}(w)$ acts indistinguishably from $\mathcal{A}^{\mathcal{O}}(z)$ to a bigger environment (see Main Theorem). We want the computational overhead of the simulation to be as small as possible, in particular, paying close attention to the time overhead, namely $T_{\mathcal{S}^{\mathcal{A}}}$ compared to $T_{\mathcal{A}}, Q_{\mathcal{A}}$, as well as the additional amount of advice that $\mathcal{S}$ receives, i.e., $(|w| - |z|)$. For some applications, the memory overhead of $\mathcal{S}^{\mathcal{A}}$ is also important (e.g., for achieving memory-tight reduction or optimal time-space trade-off). Our simulation is also memory-efficient, though we do not go into details on this front in this work.

Our key lemma presents a simple and generic way of using PRF to simulate AI-ROM. A PRF consists of key distributions $k \xleftarrow{\$} D_{\mathrm{prf}}(1^{\lambda_{\mathrm{prf}}})$ for different security parameters λ_{prf}, together with an evaluation algorithm $F(k,x) = y$. We require the PRF to have $(2^{\lambda_{\mathrm{prf}}}, \varepsilon)$-security, meaning that for all $2^{\lambda_{\mathrm{prf}}}$-time distinguishers, oracle access to $F(k,\cdot)$ with a random k is indistinguishable to the random oracle $\mathcal{O}$. We show that the random oracle with S-bit advice $(\mathcal{O}(\cdot), z \xleftarrow{\$} \mathsf{ai}(\mathcal{O}))$ can be simulated by the PRF and some advice on its key $(F(k,\cdot), \widetilde{z} \xleftarrow{\$} \widetilde{\mathsf{ai}}(k))$, as long as the PRF is using sufficiently large security parameter $\lambda_{\mathrm{prf}} = S + \Theta(\lambda)$.

Main Lemma (simulating AI-ROM using PRF)**.** *All variables implicitly depend on the security parameter λ. Let $\mathsf{dom} \subseteq \{0,1\}^*$ and $\{0,1\}^\lambda$ be the domain/codomain of the random oracle. Fix an arbitrary $(2^{\lambda_{\mathrm{prf}}}, \varepsilon)$-secure PRF (D_{prf}, F) with the same domain/codomain and an arbitrary (randomized)[7] advice function ai with S-bit outputs for $S \le 2^\lambda$. There exists a (randomized) advice function $\widetilde{\mathsf{ai}}$ with S-bit outputs and a threshold $\lambda_{\mathrm{prf},0}$ such that*

- *the threshold is upper-bounded by $\lambda_{\mathrm{prf},0} \le S + \mathsf{O}(\lambda)$, and*
- *for every 2^λ-time adversary $\mathcal{A}$ and every PRF security parameter $\lambda_{\mathrm{prf}} > \lambda_{\mathrm{prf},0}$,*

$$\left|\Pr[\mathcal{A}^{\mathcal{O}(\cdot)}(z) \to 1] - \Pr[\mathcal{A}^{F(k,\cdot)}(\widetilde{z}) \to 1]\right| \le 2^{-\lambda} + \varepsilon,$$

where $\mathcal{O}$ is a random oracle, $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O})$ an oracle-dependent advice, $k \xleftarrow{\$} D_{\mathrm{prf}}(1^{\lambda_{\mathrm{prf}}})$ a randomly sampled PRF key, and $\widetilde{z} \xleftarrow{\$} \widetilde{\mathsf{ai}}(k)$ a key-dependent advice.

The PRF simulation is conceptually simple and its proof crucially relies on the leakage simulation lemma of [GW11,JP14,CCL18]. The idea is viewing $\mathsf{ai}(\mathcal{O})$ as a short leakage of $\mathcal{O}$, which by prior works can be simulated relatively efficiently as $h(\mathcal{O})$ in time

[7] Allowing randomized ai is not technically necessary, yet it makes the lemma easier to use.

$T_h = \mathrm{O}(2^S \cdot S \cdot 2^{2\lambda} \cdot T_{\mathcal{A}})$, at an advantage loss of $2^{-\lambda}$. Therefore, as long as PRF is ε-secure against this much time, we can switch $\mathcal{O}$ to $F(k,\cdot)$ and let $\widetilde{z} = \widetilde{\mathsf{ai}}(k) = h(F(k,\cdot))$ be the corresponding advice. See Section 3 for the formal proof.

Now to simulate AI-ROM for $\mathcal{A}$, the simulator $\mathcal{S}^{\mathcal{A}}$ should simply run $\mathcal{A}^{F(k,\cdot)}(\widetilde{z})$. It can do so if receiving $w = (k, \widetilde{z})$ as non-uniform advice. The simulation overhead is directly linked to PRF efficiency, in particular, the PRF key length $|k|$ corresponds to the additional amount of non-uniform advice and the PRF evaluation time determines the time overhead of simulation. Both quantities vary with the PRF security parameter λ_{prf}. More precisely...

Main Theorem (AI-ROM simulation)**.** *Continuing with the symbols defined in the Main Lemma, there exists a universal simulator $\mathcal{S}$ and a (randomized) non-uniform advice function* nu *satisfying the following.*

- *For every 2^λ-time adversary $\mathcal{A}$ and 2^λ-time machine $\mathcal{B}$ with binary output,*

$$\left|\Pr[\mathsf{out}_{\mathcal{B}}\langle \mathcal{B} \leftrightarrow \mathcal{A}^{\mathcal{O}}(z)\rangle = 1] - \Pr[\mathsf{out}_{\mathcal{B}}\langle \mathcal{B} \leftrightarrow \mathcal{S}^{\mathcal{A}}(w)\rangle = 1]\right| \le 2^{-\lambda} + \varepsilon,$$

  where $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O})$ is an advice dependent on random oracle $\mathcal{O}$ and w is a non-uniform advice from distribution $w \xleftarrow{\$} \mathsf{nu}()$.

- *The simulation overhead is related to the PRF efficiency as follows:*

$$\begin{aligned} &\text{(advice)} && |w| - |z| \le |k| + \mathrm{O}(\log(S + |k|)), \\ &\text{(time)} && T_{\mathcal{S}^{\mathcal{A}}} - T_{\mathcal{A}} \le T_{\mathcal{A},F}, \end{aligned}$$

  where $T_{\mathcal{A}}$ is an upper bound on the running time of $\mathcal{A}$, and $|k|$ is the PRF key length when using security parameter $\lambda_{\mathrm{prf}} = S + \mathrm{O}(\lambda)$, and $T_{\mathcal{A},F}$ is an upper bound on the total time for evaluating the PRF on all oracles calls made by $\mathcal{A}$.

Clearly, a more efficient PRF leads to a tighter AI-ROM simulation. By plugging in different PRF in the above theorem, we obtain different simulation efficiency as summarized in Table 1, which we explain in detail below.

Subexponential PRF and Resolving Unruh's Conjecture. When plugging in a vanilla subexponentially secure PRF, our theorem resolves Unruh's conjecture affirmatively. A subexponentially secure PRF for n-bit inputs is a $(2^{\lambda_{\mathrm{prf}}}, \varepsilon)$-secure PRF with key length $|k| = \mathrm{poly}(\lambda_{\mathrm{prf}}, n)$.[8] PRF evaluation takes polynomial time $\mathrm{poly}(|x|, |k|, \lambda)$ (recall that λ is the output length). Therefore, when $\lambda_{\mathrm{prf}} = S + \mathrm{O}(\lambda)$, the advice overhead of our simulation is $|k| = \mathrm{poly}(\lambda, S, n)$ and the time overhead per oracle query is $\mathrm{poly}(\lambda, S, n)$ (see row 3 in Table 1). This gives simulation with *polynomial* overhead, independent of the *negligible* distinguishing advantage $(\varepsilon + 2^{-\lambda})$. See Section 5 for more details.

Supporting Unrestricted Domain and Linear Dependency on S. Our simulation technique applies to AI-ROM with unrestricted domain $\{0,1\}^*$, using PRF with the same domain. Concretely, one can instantiate such PRF using the Goldreich–Goldwasser–Micali (GGM) PRF built from a length-doubling PRG [GGM86], where the PRF key is simply a PRG seed and evaluation time grows linearly with the input length $|x|$. Consider using different PRG...

[8] In the literature, subexponentially secure PRF often refers to one that achieves security against $2^{|\lambda'|^\beta}$-time adversaries when using security parameter λ'. This is equivalent to our formulation, by setting λ' to be $\lambda_{\mathrm{prf}}^{1/\beta}$, which causes the key length to grow polynomially with λ_{prf}.

Table 1. Comparison of methods for distinguishing. λ is a security parameter, as well as the output length. S denotes the length of advice adversary gets in AI-ROM, T the adversary runtime, Q the number of oracle queries. Rows 4–6 refer to [GGM86] tree PRF for unbounded length inputs, using length doubling PRG with subexponential (row 4) or exponential (rows 5, 6) security, and evaluation time depending polynomially (rows 4, 5) or quasilinearly (row 6) on the key length. In the last row, $0 < \gamma < 1$ is an arbitrary small constant. All parameters are for achieving a distinguishing advantage of ε or $(\varepsilon + 2^{-\lambda})$ between AI-ROM and simulation.

method	domain	extra advice	time overhead per query
presampling [CDGS18]	$\{0,1\}^n$	$\Theta(n\lambda SQ/\varepsilon)$	$\Theta(n\lambda SQ/\varepsilon)$
presampling [CDGS18]	$\{0,1\}^*$	$\Theta(T\lambda SQ/\varepsilon)$	$\Theta(T\lambda SQ/\varepsilon)$
generic subexp.-PRF (§ 3)	$\{0,1\}^n$	$\mathrm{poly}(\lambda, S, n)$	$\mathrm{poly}(\lambda, S, n)$
subexp.-OWF + GGM	$\{0,1\}^*$	$\mathrm{poly}(\lambda, S)$	$\lvert x\rvert\,\mathrm{poly}(\lambda, S)$
exp.-PRG + GGM	$\{0,1\}^*$	$\mathrm{O}(\lambda + S)$	$\lvert x\rvert\,\mathrm{poly}(\lambda, S)$
dbl.eff.-PRG + GGM	$\{0,1\}^*$	$\widetilde{\mathrm{O}}(\lambda + S)$	$\lvert x\rvert\,\widetilde{\mathrm{O}}(\lambda + S)$
(RAM) super-fast PRF (§ 4)	$\{0,1\}^n$	$(n + \log\lambda)^{1+\gamma} \cdot (\lambda + S)^{1+\gamma}$	$\lambda \cdot (n + \log\lambda)^{1+\gamma} \cdot \mathrm{poly}(\log(\lambda + S))$

- A subexponentially secure PRG has seed length $\mathrm{poly}(\lambda_{\mathrm{prf}})$ and evaluation time $\mathrm{poly}(\lambda_{\mathrm{prf}})$, which can be based on any subexponentially secure OWF. Using it yields simulation with advice overhead $\mathrm{poly}(\lambda, S)$ and time overhead $|x|\,\mathrm{poly}(\lambda, S)$ for an oracle query x.
- An exponentially secure PRG has seed length $\mathrm{O}(\lambda_{\mathrm{prf}})$ and evaluation time $\mathrm{poly}(\lambda_{\mathrm{prf}})$, which for example can be based on the exponential hardness of DDH. Plugging it reduces the advice overhead to linear $\mathrm{O}(\lambda + S)$.
- A "doubly efficient" PRG has both short seeds $\widetilde{\mathrm{O}}(\lambda_{\mathrm{prf}})$ and short evaluation time $\widetilde{\mathrm{O}}(\lambda_{\mathrm{prf}})$. Such a PRG can be constructed from, for example, the exponential hardness of ring learning with rounding (R-LWR) with polynomial modulus. The double efficiency translates to both small advice overhead $\widetilde{\mathrm{O}}(\lambda + S)$ and small time overhead $|x|\,\widetilde{\mathrm{O}}(\lambda + S)$.

We note that the last instantiation yields improvement over simulation via presampling in all aspects — smaller dependency on all parameters.

Even Tighter Simulation via Super-Fast PRF. So far, simulation time overhead grows with to the advice length S. This means the overall simulation time $T_{\mathcal{S}}$ is at least $ST_{\mathcal{A}}$. Can we achieve $T_{\mathcal{S}} \sim T_{\mathcal{A}}$ while keeping $S_{\mathcal{S}} \sim S$? Since the PRF security parameter needs to be large $\lambda_{\mathrm{prf}} = \Omega(S + \lambda)$, it is impossible to have PRF evaluation time below $|k| > \lambda_{\mathrm{prf}}$ if evaluation reads the entire key. Nevertheless, we show that in the RAM model, it is possible to construct a PRF whose evaluation reads only a small portion of the key. Our construction is based on a variant of Goldreich's PRG assumption [Gol00], and crucially using the polynomial preprocessing technique of [KU11] that has recently found application in building doubly efficient PIR [LMW23].

For n-bit inputs and *one-bit* outputs, our fast PRF has key length $(n\lambda_{\mathrm{prf}})^{1+\gamma}$ and evaluation time $n^{1+\gamma}\,\mathrm{poly}(\log\lambda_{\mathrm{prf}})$, where $0 < \gamma < 1$ can be set to an arbitrarily small constant. For λ-bit outputs, we use the simple domain extension. Such a PRF enables

AI-ROM simulation where the time overhead is $\lambda(n+\log\lambda)^{1+\gamma}\,\mathrm{poly}(\log(\lambda+S))$ (see last row of Table 1). The overall simulation achieves $T_{\mathcal{S}} \sim T_{\mathcal{A}}$ and $S_{\mathcal{S}} \sim S$, modulo $\mathrm{poly}(\lambda)$ and $o(S)$ factors.

To the best of our knowledge, our construction gives the first PRF where the evaluation time does not grow with the security parameter — efficiency does not "pay" for stronger security. We believe that such a PRF is of independent interests. See Section 1.4 for an overview of the fast PRF construction.

Applications. We apply our AI-ROM simulation technique in two applications: *i)* showing that salting generically defeats preprocessing, with a tighter reduction than prior works [CDGS18]; and *ii)* constructing the first NIZK (with polynomial time simulation and negligible distinguishing advantage) in the AI-ROM.

Salting. Salting is a technique that has been widely used in the practice for defeating preprocessing (going back to password hashing [MT79]). It uses a random oracle $\mathcal{O}$ (or a real-world hash function) to build another random oracle $\mathcal{O}_r$, by prepending every oracle call with a public string r (the salt) chosen randomly after preprocessing, i.e., $\mathcal{O}_r(x) = \mathcal{O}(r,x)$. One can generically upgrade a scheme Π secure in the ROM to a scheme $\widetilde{\Pi}$ secure in the AI-ROM, by using Π with the salted random oracle $\mathcal{O}_r$, i.e., $\widetilde{\Pi}^{\mathcal{O}} = \Pi^{\mathcal{O}_r}$. This is shown via a reduction $(\mathcal{R}, \mathsf{nu})$ satisfying that if an adversary $\mathcal{A}$ with advice $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O})$ can break the security of $\widetilde{\Pi}^{\mathcal{O}}$, the adversary $\mathcal{R}^{\mathcal{A}}$ with advice $w \xleftarrow{\$} \mathsf{nu}()$ can break $\Pi^{\mathcal{O}'}$ in the ROM. Previously, the presampling techniques were used to build the reduction [CDGS18], which internally simulates AI-ROM for $\mathcal{A}$.

As discussed earlier, reduction via presampling is wasteful. A better approach is to plug in our PRF-based simulation instead. We reduce the per-query overhead of $\mathcal{R}^{\mathcal{A}}$ over $\mathcal{A}$ from $\Theta(n\lambda SQ/\varepsilon)$ (using presampling) to $\mathrm{poly}(\lambda, S, n)$ using subexponentially secure PRF, or $n\,\widetilde{O}(\lambda+S)$ using doubly efficient PRG (which in turn can be based on R-LWR with polynomial modulus), or even $\lambda(n+\log\lambda)^{1+\gamma}\,\mathrm{poly}(\log(\lambda+S))$ using super-fast PRF (based on a variant of Goldreich's PRG). This establishes a reduction where $T_{\mathcal{R}^{\mathcal{A}}} \sim T_{\mathcal{A}}$ and $S_{\mathcal{R}^{\mathcal{A}}} \sim S_{\mathcal{A}}$. See more details in Section 6.1.

NIZK with Preprocessing. To showcase the benefits of our tight simulation, we build the first NIZK secure against preprocessing. Here, efficient simulation (of the cheating verifier's view) is the end goal. It is known that NIZK necessarily relies on some trusted set-up, such as the random oracle, common random/reference string, etc. Unconditionally secure NIZK in the ROM and computationally secure NIZK in the CRS model from various assumptions are known. However, it is unknown whether they remain secure when the adversaries possess some preprocessed information of the set-up.

Using salting/simulation technique, we show how to start with a NIZK scheme in the ROM with statistical soundness against time-unbounded cheating prover and upgrade its security to the AI-ROM. We generically "salt" the NIZK scheme, by making the honest prover $\mathcal{P}$ generate the proof using a salted random oracle $\mathcal{O}(r,\cdot)$, and send the tuple (r,π) as the new proof. (Correspondingly, the verifier verifies π with respect to $\mathcal{O}(r,\cdot)$.) Since soundness holds against time-unbounded adversaries, additional advice about $\mathcal{O}$ does not enhance the power of the cheating prover, and statistical soundness remains intact. For zero knowledge, the task is efficiently simulating the view of a cheating verifier $\mathcal{V}^*$, in a way that is negligibly indistinguishable. The

view includes the proof (r, π), the oracle $\mathcal{O}$, and the advice $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O})$. Using our salting/simulation technique, we can first show that the interaction between $\mathcal{P}^{\mathcal{O}(r,\cdot)}$ and $(\mathcal{V}^*)^{\mathcal{O}}(z)$ can be simulated by the interaction between $\mathcal{P}^{\mathcal{O}'}$ and $(\mathcal{V}')^{\mathcal{O}'}$ in the ROM. The attacker $\mathcal{V}'$ internally runs $\mathcal{V}^*$ by simulating oracle $\mathcal{O}$ as follows. For a call on input (r, x), it returns $\mathcal{O}'(x)$, and for a call on (r', x) with $r' \neq r$, it returns $F(k, (r', x))$. The attacker $\mathcal{V}'$ also feeds $\mathcal{V}^*$ the simulated advice $z' \xleftarrow{\$} \mathsf{ai}'(k)$. Now, the simulator $\mathcal{S}$ guaranteed in the ROM can be used to simulate the proof $\widetilde{\pi}$ and oracle $\mathcal{O}'$ to $\mathcal{V}'$. This gives rise to the final simulator in the AI-ROM.

Thanks to the efficiency of our AI-ROM simulation, the resulting simulator has polynomial overhead, while achieving negligible distinguishing advantage. Moreover, more efficient instantiation of the PRF leads to tighter simulation. See Section 6.2 for more details.

A Four-Step Generic Technique. Both our salting and NIZK applications use a new technique that combines presampling and our PRF simulation (Main Lemma), which allows us to enjoy the best of both techniques. Consider salting, where we aim show that if a scheme $\Pi^{\mathcal{O}}$ is secure against adversary $\mathcal{B}^{\mathcal{O}}$ in the ROM, then the salted scheme $\Pi^{\mathcal{O}(r,\cdot)}$ is secure against adversary $\mathcal{A}^{\mathcal{O}}(z)$ in the AI-ROM. Applying our PRF simulation lemma directly does not work. It proves that the interaction $\Pi^{\mathcal{O}(r,\cdot)} \leftrightarrow \mathcal{A}^{\mathcal{O}}(z)$ is indistinguishable to $\Pi^{F(k,(r,\cdot))} \leftrightarrow \mathcal{A}^{F(k,\cdot)}(\widetilde{z})$, where the AI-ROM $(\mathcal{O}, z)$ is simulated by the PRF $(F(k,\cdot), \widetilde{z})$. But this does not allow reducing to security in the ROM. To make things worse, once the oracle is replaced by PRF, we cannot apply the many useful tricks in the ROM, such as lazy sampling, programming, extraction, etc. Instead, we consider the following 4-step technique.

- Start from the a real-world game $\Pi^{\mathcal{O}(r,\cdot)} \leftrightarrow \mathcal{A}^{\mathcal{O}}(r, z(\mathcal{O}))$ in the AI-ROM.
- *Use presampling*. Go to the bit-fixing ROM with table $\mathcal{T}$,

$$\Pi^{\mathcal{O}[\mathcal{T}](r,\cdot)} \leftrightarrow \mathcal{A}^{\mathcal{O}[\mathcal{T}]}(r, z'(\mathcal{T})), \qquad \text{where } \mathcal{O}[\mathcal{T}](x) = \begin{cases} y, & \text{if } (x,y) \in \mathcal{T}; \\ \mathcal{O}(x), & \text{otherwise.} \end{cases}$$

  The advice z' contains only information of $\mathcal{T}$, independent of $\mathcal{O}$. (The size of $\mathcal{T}$ can be tuned and will bear no computational efficiency consequences.)

- *Re-route relevant queries*. Since the salt r is random, it is overwhelmingly likely that $\mathcal{T}$ contains no input starting with r. In such case, we can re-route salted queries $x = (r, x')$ to a new oracle $\mathcal{O}^*$ independent of $\mathcal{O}$ and $\mathcal{T}$. This brings us to the game

$$\Pi^{\mathcal{O}'(r,\cdot)} \leftrightarrow \mathcal{A}^{\mathcal{O}'}(r, z'(\mathcal{T})), \qquad \text{where } \mathcal{O}'(x) = \begin{cases} \mathcal{O}^*(x'), & \text{if } x = (r, x'); \\ \mathcal{O}[\mathcal{T}](x), & \text{otherwise.} \end{cases}$$

  In other words, all salted queries are "liberated" from $\mathcal{O}[\mathcal{T}]$.

- *Undo presampling*. Apply the presampling technique reversely to turn the BF-ROM $\mathcal{O}[\mathcal{T}]$ and advice $z'(\mathcal{T})$ back to the AI-ROM $(\mathcal{O}, z(\mathcal{O}))$, while keeping all salted queries answered by the independent $\mathcal{O}^*$.

$$\Pi^{\mathcal{O}^*} \leftrightarrow \mathcal{A}^{\mathcal{O}''}(r, z(\mathcal{O})), \qquad \text{where } \mathcal{O}''(x) = \begin{cases} \mathcal{O}^*(x'), & \text{if } x = (r, x'); \\ \mathcal{O}(x), & \text{otherwise.} \end{cases}$$

At this point, all Π-relevant queries are answered by $\mathcal{O}^*$, while $\mathcal{A}$ additionally has some "meaningless" interaction with $\mathcal{O}$ and advice z about it.

- *Apply PRF simulation*. By our main lemma, $(\mathcal{O}, z)$ can be simulated by PRF $(F(k,\cdot), \widetilde{z})$, allowing us to move to the game

$$\Pi^{\mathcal{O}^*} \leftrightarrow \mathcal{A}^{\mathcal{O}'''}(r, \widetilde{z}), \qquad \text{where } \mathcal{O}'''(x) = \begin{cases} \mathcal{O}^*(x'), & \text{if } x = (r, x'); \\ F(k, x), & \text{otherwise.} \end{cases}$$

  Here, the adversary $\mathcal{A}^{\mathcal{O}'''}(r, \widetilde{z})$ can be emulated by an adversary $\mathcal{B}^{\mathcal{O}^*}$ given $(k, \widetilde{z})$ as non-uniform advice. We can now rely on ROM security, and enjoy the fact that by our PRF simulation $T_{\mathcal{B}} \sim T_{\mathcal{A}}$ and $S_{\mathcal{B}} \sim S_{\mathcal{A}}$.

We remark that our 4-step technique is relatively generic. The key idea is re-routing all security-relevant queries to an independent oracle $\mathcal{O}^*$ for which no preprocessed information is available, and then simulate the remaining queries and advice using our PRF simulation. In our template, the former is achieved by salting and by doing and undoing presampling. During the former, all information-theoretic techniques can be applied without creating later burdens, because they are undone before computational techniques kick in. The latter enjoys the efficiency of our PRF simulation. We believe that this technique will find further applications.

1.4 How to Construct Fast Pseudorandom Functions

We now come to the question of designing fast pseudorandom functions. For this part, λ is the PRF security parameter, tunable in applications (i.e., λ_{prf}). We consider PRF family $F : \{0,1\}^{\ell} \times \{0,1\}^n \to \{0,1\}$ with key length $\ell = \ell(\lambda, n)$, where the input length is thought of as a varying parameter bounded in $\text{poly}(\lambda)$ and the key length is allowed to grow polynomially in both λ and n. We are interested in two properties.

- The PRF is secure against adversaries up to time 2^{λ}.
- Evaluating the PRF on any input $x \in \{0,1\}^n$ is fast in the RAM model. Namely, each query can be answered in time $n^{O(1)} \text{poly}(\log \lambda)$, i.e., the evaluation time effectively depends only on the input length but not the security level. Our PRF achieves evaluation time of $n^{1+\gamma} \text{poly}(\log \lambda)$ for arbitrarily small constant $\gamma > 0$.

We observe that none of the prior constructions of PRF are known to achieve security-independent evaluation time in the RAM model.

Leveraging Sparsity of Goldreich's Function. We are looking to build functions that read only a few bits of the key on every query, as constrained by the evaluation time. Therefore, it is imperative that we look for sparsity-related hard problems.

In fact, it was already observed in [AR16] that Goldreich's function [Gol00], when instantiated with super-constant locality, gives rise to extremely efficient PRF. We will discuss this in a bit but first take a slight detour to recap Goldreich's one-way function. A reader familiar with it can skip to the next heading.

Detour: Goldreich's One-Way Function/PRG. Goldreich's one-way function candidate $f_{P,G} : \{0,1\}^\ell \to \{0,1\}^m$ is indexed by a local Boolean predicate $P : \{0,1\}^L \to \{0,1\}$ and a hypergraph $G = (S_1, \ldots, S_m)$ with m hyperedges, where each hyperedge $S_i \subseteq [\ell]$ is of size L. The function is defined as $f_{P,G}(s) = (y_1, \ldots, y_m)$, where each y_i is computed by applying P to the bits selected by the set S_i (denoted by $y_i = P(s|_{S_i})$). The smaller the locality, the more efficiently computable the function is. In particular, if P is L-local, then given S_i, each output bit y_i can be computed in $\widetilde{O}(L)$ time. How small can L be? If one is interested in polynomial stretch, constant $L \geq 5$ suffices. However, such a function is trivially not pseudorandom when $m \geq \ell^L$, as the S_i's repeat. This means that with constant locality, we cannot hope for a stretch beyond a polynomial. For PRF, one might need to support super-polynomial or subexponential stretch. Therefore, one has to work with a locality L that is (sublinear) polynomial in the input length. This regime was used by [AR16] to build an efficient PRF.

Based on a long history [Gol00,CM01,Fei02,MST03,FKO06,CEMT09,BQ09,ABW10, ABR12,BQ12,App12,App13,OW14,AL16] of study culminating in [AL16], for appropriately chosen hypergraph G satisfying certain expansion criteria and predicate P, it is widely believed that such functions should be pseudorandom for $m = \ell^{\Omega(L)}$. If the predicate has large resiliency and large rational degree (see Section 4.3 for their definitions) and if the hypergraph is chosen so that it is $(t, 0.99)$-expanding for a large t (the union of hyperedges $\{S_i\}_{i \in I}$, for any $I \subseteq [m]$ of size up to t, has size at least $0.99 \cdot |I| \cdot L$), then the function $f_{P,G}$ can be conjectured to be pseudorandom against algorithms running in time $2^{\widetilde{O}(t)}$. The conjecture holds for arbitrary graphs with this expansion property, which is satisfied by a random hypergraph with high probability. This is backed up by provable lower bounds against various classes of attacks such as sum-of-squares and linear tests and polynomial calculus [AL16]. The properties regarding resiliency and rational degree are easily satisfied by the predicate $\mathsf{XorMaj}_{L'}$ that takes as input $L = 2 \cdot L'$ bits and computes

$$\mathsf{XorMaj}_{L'}(s_1, \ldots, s_{L'}, s_{L'+1}, \ldots, s_L) = s_1 \oplus \cdots \oplus s_{L'} \oplus \mathsf{majority}(s_{L'+1}, \ldots, s_L).$$

This predicate has resiliency $L/2$ and rational degree $L/4$. Our assumption hence can be stated as follows.

Assumption 1 (Goldreich's Assumption)**.** *Let $\ell \in \mathbb{N}$ be the input length, $L = 2L'$ the locality (up to a sublinear polynomial in seed length ℓ), P the predicate $\mathsf{XorMaj}_{L'}$, and G any L-uniform hypergraph with m hyperedges such that each G is $(t, 0.99)$-expanding, then*

$$\left\{ s \xleftarrow{\$} \{0,1\}^\ell \ : \ f_{P,G}(s) \right\} \approx \left\{ y \xleftarrow{\$} \{0,1\}^m \ : \ y \right\}$$

for adversaries up to time $2^{\widetilde{O}(t)}$.

Efficient Weak PRF. As shown in [AR16], one can build a highly efficient weak PRF from Goldreich's PRG (weak means that the security definition only allows uniformly random queries). The high-level idea is that the PRF key is a random seed $s \xleftarrow{\$} \{0,1\}^\ell$ for Goldreich's PRG. A random input x is interpreted as a set $S_x \subseteq [\ell]$ of size L. The evaluation simply outputs $y_x = P(s|_{S_x})$. Such a PRF can be evaluated in time $\widetilde{O}(L)$. The security holds because for appropriately chosen m, L, ℓ, the uniformly random queries $S_{x_1}, \ldots, S_{x_m}$ are likely to satisfy the expansion property needed for Assumption 1.

We now consider how small L can be (for faster evaluation) to support a large number of queries and maintain strong security. Remember that the goal is to support $m = 2^n$ queries and remain secure against 2^λ-time adversaries.

As mentioned previously, security is trivially broken once the number of samples exceed ℓ^L, so ℓ^L should be much greater than 2^n. This mandates that L be at least n up to polylogarithmic factors. We will set L to be a large constant multiple of n. For security against 2^λ-time adversaries making $m = 2^n$ queries, we need the hypergraph $(S_{x_1}, \ldots, S_{x_m})$ to be $(t, 0.99)$-expanding for some small polynomial t in λ. A routine calculation shows that this can be achieved by setting the key size ℓ to be a polynomial in λ slightly larger than t. For example, setting $\ell = n\lambda^2$ makes a random hypergraph $(\lambda^{1.5}, 0.99)$-expanding with overwhelming probability. This will yield a weak PRF that can be computed in $\widetilde{O}(n)$ time in the RAM model with key size $O(n\lambda^2)$ and that is secure against 2^λ-size adversaries based on Assumption 1.

Strong PRF via t-Wise Independence. While fast weak PRF can be constructed using Goldreich's PRG, it is unclear if strong PRF can be built this easily. Indeed, in the previous construction, if the adversary picks the L-uniform hyperedges S_x arbitrarily, the induced graph may not be an expander anymore. Weak PRF get around this issue by mandating that the edges be chosen at random. How can we have adversarially chosen hyperedges $\{S_i\}_{i\in[m]}$ form an expander? We must regain some control on its randomness.

Our first observation, which is folklore, is that to generate a $(t, 0.99)$-expander with high probability, the hyperedges do not have to be chosen fully at random. In fact, if $\{S_i\}_{i\in[m]}$ are t-wise independent random, then they form an expander with high probability. We will follow this approach. Note that a similar idea was used in [AR16], but their construction yielded PRF that are linear time in the key length. We show how to avoid the key-length dependency altogether.

How does one generate a function mapping an input $x \in [m]$ to a hyperedge S_x such that the hypergraph $\{S_x\}_{x\in[m]}$ follows a t-wise independent distribution? A list of L distinct elements from $[\ell]$ can be sampled by picking $v_1 \xleftarrow{\$} [\ell]$, then $v_2 \xleftarrow{\$} [\ell] \setminus \{v_1\}$, and so on until $v_L \xleftarrow{\$} [\ell] \setminus \{v_1, \ldots, v_{L-1}\}$. With some preprocessed information helping with arithmetics, this procedure can be done in time $L\,\mathrm{poly}(\log \ell)$ when the randomness is given as a natural number less than $N = \ell(\ell-1)\cdots(\ell-(L-1))$. It remains to generate this randomness using a t-wise independent hash function.

We use a degree-t polynomial f with random coefficients for hashing. Let p be the prime modulus. For each x, we interpret $f(x)$ as a natural number less than p, then reduce it modulo N to obtain the randomness $h(x)$. This incurs a statistical error of N/p for a single x, from which we determine how to choose p.

Fast Evaluation via Polynomial Preprocessing. We now analyze the running time of the PRF above. Given x, we first evaluate the polynomial f, then convert $f(x)$ to S_x, and lastly evaluate Goldreich's PRG using S_x. The last two steps are within the desired complexity. Evaluating $f(x)$ naïvely takes $\Omega(t \log p)$ time, which is prohibitive, as we need to set $t > \lambda$ for security.

We rely on a powerful tool to address this issue, developed in the beautiful work by Kedlaya and Umans [KU11]. Their result says that one could preprocess a degree-t univariate polynomial f over $\mathbb{Z}_p$ into a data structure using which, in the RAM model,

evaluating f at any point $x \in \mathbb{Z}_p$ can be done in time $\mathrm{poly}(\log t) \cdot \log^{1+\gamma'} p$, where γ' can be chosen as any (small) positive constant.

Lemma 6 (reinterpreted from Theorem 5.1 in [KU11]). *Let $\gamma' > 0$ be any constant. Let p be a prime and f a polynomial of degree t over $\mathbb{Z}_p$. In the RAM model, one can compute, in time $\mathrm{O}((t \log p)^{1+\gamma'})$, a data structure such that given any $\alpha \in \mathbb{Z}_p$, the value $f(\alpha)$ can be computed in time $\mathrm{poly}(\log t) \cdot \log^{1+\gamma'} p$.*

Our degree t is below $\lambda^{1+\gamma'}$ and we can choose p to be roughly $2^n \ell^L = 2^{n\,\mathrm{poly}(\log(\lambda+n))}$, so the (online) polynomial evaluation takes time $n^{1+\gamma}\,\mathrm{poly}(\log \lambda)$ (we tune γ to be slightly larger than γ'). Since $L = \mathrm{O}(n)$, our overall evaluation time is still just that. This concludes the claim that our PRF can be evaluated in time $n^{1+\gamma}\,\mathrm{poly}(\log \lambda)$ for arbitrarily small constant $\gamma > 0$. Recently, such techniques played an important role in constructing doubly efficient PIR schemes from ring LWE [LMW23].

Relation with [AR16]. A reader might recall that the work of [AR16] also constructed fast PRF. Their PRF does not yield efficiency to our desideratum. The goal of that work was to construct PRF with key size n and output length n (as opposed to one bit in our case) taking $\widetilde{\mathrm{O}}(n)$ time to evaluate. They start by constructing a weak PRF and convert it to a standard PRF similarly to us by instantiating a graph sampler using a independence generator that uses ring-based computation by Miles and Viola [MV15] (this offers efficiency advantages due to FFT over rings). However, their efficiency is only achieved in an amortized sense, i.e., when the PRG output length is long enough. The output length itself is linear in the key length, so each PRF output block could take time proportional to the key/output length. Since the key size is polynomial in the security parameter, the running time is strongly dependent on the security parameter. In contrast, our PRF truly achieves security-independent evaluation time in the RAM model.

Changes Since Conference Version. The exposition of super-fast PRF has undergone some notable updates after FOCS 2024. The evaluation time is clarified (Table 1 and Section 4). The choice of hash function modulus no longer relies on Cramér's conjecture. Certain details in how the hyperedges are sampled are clarified.

2 Preliminaries

The security parameter is λ. For $N \in \mathbb{N}$, we write $[N]$ for $\{1, \ldots, N\}$. For $n \in \mathbb{N}$, the set $\{0,1\}^{\le n}$ consists of all bit-strings of length at most n, and $\{0,1\}^*$ is the set of all bit-strings. Given $p \in \mathbb{N}_{\ge 2}$, we denote by $\mathbb{Z}_p$ the integers modulo p. Logarithms without a base are natural by default. A natural number $n \in \mathbb{N}$ has an easy-to-parse prefix-free encoding $1^{\lceil \log_2(n+1) \rceil} 0 b_{\lceil \log_2(n+1) \rceil - 1} \cdots b_1 b_0$, where $b_{\lceil \log_2(n+1) \rceil - 1} \cdots b_1 b_0$ is the binary form of n without a leading zero. The encoding length is $(2\lceil \log_2(n+1) \rceil + 1)$.

Let X be quantified objects. Every appearance of C_X is a new non-negative number (constant) that depends on X (but not any other object) and makes a proposition or a proof correct. For example, C_{ram} depends on the exact model of RAM programs, $C_{\mathrm{ckt,prf}}$ depends on the exact model of circuits (e.g., AND/OR/NOT or just NAND) and the PRF being used, and C without any subscript is an absolute constant.

Symbols. Table 2 explains select symbols used throughout the paper.

Table 2. Select symbols used in this work.

context	symbol(s)	meaning
generic in preliminaries	λ	security parameter
	C, C_X	absolute constant, X-dependent constant
	nu, w, M	standard-model non-uniformity, string, length
	ai, z, S	oracle-dependent auxiliary input, string, length
	M, T	(concrete) size, (concrete or asymptotic) time
	A, E, ε	algorithm, experiment, advantage
	n, m	input/output lengths of random oracle
	Q	number of queries
fast PRF from Goldreich's PRG	ℓ	length of PRG seed / PRF key
	m	length of PRG output / PRF truth table
	L, P	locality, predicate
	t, p	degree of independence, modulus
	$k, \overline{k}$	preprocessed key, length bound

2.1 Auxiliary-Input Random Oracle Model

We consider concrete and asymptotic security against random-access machines (RAM) and circuits, randomized and possibly connected to oracles. Throughout the paper, we fix a universal oracle-aided RAM, which interprets all RAM *programs*. For randomness, at each step, one random bit is provided, which can be saved and retrieved later using random access. Only oracles are put as superscripts and the notation $A(x)$ does not necessitate scanning x in full. We also assume that any input can be hardwired into programs with small overhead. Precisely speaking, there exists a (uniform) linear-time deterministic Turing machine that given a two-input program A and an input z, outputs a single-input program for $A_z(x) = A(z, x)$ that is no more than $(|A| + |z| + C_{\mathrm{ram}} \cdot \log(|A| + |z| + 2))$ bits long and runs in the time of A.[9] We also fix a model of circuits and use RAM programs for generation of circuits.

Definition 1 (advice, algorithm, adversary, experiment, indistinguishability)**.** Denote by X some object about which adversaries get advice. A *concrete X-advice* is a randomized procedure that outputs a bit-string of fixed length given X. An *asymptotic X-advice* is a sequence of concrete X-advices, or equivalently, a randomized procedure that outputs a bit-string of length determined by λ given (λ, X). An asymptotic $\perp$-advice captures standard-model non-uniformity and is denoted by nu. An $\mathcal{O}$-advice, with $\mathcal{O}$ being the random oracle (Definition 3), captures the so-called "auxiliary input" and is denoted by ai. The outputs of nu, ai are w, z. The length of (asymptotic) w is $M(\lambda)$, and that of z is S or $S(\lambda)$.

A *concrete algorithm* A is a distribution[10] over either RAM programs or circuits with

[9] Suppose programs are written in a real-world-like instruction set and A receives each input as a pair of registers (location and length), then A_z can be storing the location and the length of z in the two registers that A regards as z (i.e., initializing z), followed by the code of A, followed by z itself. If we include the time to initialize each input (e.g., x) in the total time, then the (internal) initialization of z in A_z replaces the (external) initialization of z in A, and the total time does not change.

[10] A concrete algorithm has *two* phases of randomness: first, a random RAM program or circuit is sampled from this distribution; then, the RAM program or circuit runs with random bits. The

uniformly bounded *size* M[11] and *time* T. An *asymptotic algorithm* A is either a RAM program whose time is bounded for each λ or an always-halting RAM that generates circuits.[12] For asymptotic circuit algorithms, its *time* $T(\lambda)$ is the maximum time to generate a circuit on λ; for asymptotic RAM algorithms, its *time* follows the usual definition.

The term *adversary* loosely means algorithms with or without either of nu, ai, and *experiment*, algorithms that interact with adversaries in black box. For advantage bound ε and adversary class $\mathcal{A}$, *indistinguishability* is denoted by "$\overset{\varepsilon}{\approx}_{\mathcal{A}}$". In *concrete indistinguishability*, $\varepsilon \in [0, +\infty)$ and the inequality must hold for all (concrete) adversaries in $\mathcal{A}$. In *asymptotic indistinguishability*, $\varepsilon : \mathbb{N} \to [0, +\infty)$ and for each (asymptotic) adversary in $\mathcal{A}$, the inequality must hold for all but finitely many λ's.

We talk about M (length of nu) of an asymptotic adversary, but never its size. Once upper bounds are put on M, T, there are only finitely many derandomized concrete algorithms — this is important when we invoke the leakage simulation lemma.

A security property often involves multiple parameters. It is convenient to unify them into a single one and talk about "bits of security". In this work, we use the following simple definition. Some works studying this notion are [MW18,WY21,WY23, MSW24,Lee24].

Definition 2 (strong bit level of security)**.** Let $\lambda' \in \mathbb{N}$. The indistinguishability of two concrete experiments is *strongly λ'-bit* if the advantage is bounded by $2^{-\lambda'}$ for all adversary whose complexities are bounded by $2^{\lambda'}$.

Let $\lambda' : \mathbb{N} \to \mathbb{N}$ be a function. Two asymptotic experiments has *strongly λ'-bit indistinguishability* if for all adversary whose complexities are bounded by $2^{\lambda'(\lambda)}$ for all but finitely many λ's, the advantage is bounded by $2^{-\lambda'(\lambda)}$ for all but finitely many λ's.

The notion of adversary's complexities is contextual. They can be size, time, number of queries, among others. We intentionally say "strong" in Definition 2 to differentiate it from the other versions. As an example of Definition 2, recall that the one-wayness advantage of a Q-query adversary against a random oracle $\{0,1\}^{2\lambda+2} \to \{0,1\}^{2\lambda+2}$ (without auxiliary input) is bounded by $(Q+2)/2^{2\lambda+2}$, so it has λ strong bits of security.

Definition 3 (AI-ROM)**.** Let dom be a function mapping λ to subsets of $\{0,1\}^*$ (subject to Definition 4) and $m : \mathbb{N} \to \mathbb{N}_+$ a polynomially bounded function. In the *auxiliary-input random oracle model* of type $\mathsf{dom} \to \{0,1\}^m$, on λ, every algorithm is given oracle access to a random function $\mathcal{O} : \mathsf{dom}(\lambda) \to \{0,1\}^{m(\lambda)}$ and adversaries get $z \xleftarrow{\$} \mathsf{ai}(\lambda, \mathcal{O})$ as input (Definition 1). One can also consider AI-ROM for a fixed λ with concrete adversaries and $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O})$.

---

first stage essentially permits randomized standard-model non-uniformity, necessary to formulate simulators reliant on randomness that is not efficiently sampleable. The second stage is preserved primarily for RAM, so that randomness (whose length could be as large as the time) does not "creep into" non-uniformity (its size could be much smaller than time) for accurate accounting.

[11]There is no notion of "amount" of standard-model non-uniformity (w) for a concrete algorithm, which only works for a specific value of λ. We regard it as being entirely non-uniform, hence the overloading of M.

[12]The machine generating circuits takes as input 1^λ and potentially some advice string. Instead of simply letting the circuits be given, by separating the advice and the efficient circuit generation procedure, our formulation enables accurate accounting of the amount of advice for an asymptotic circuit algorithm, which might be much smaller than its circuit size.

Definition 4 (possible domains of AI-ROM)**.** An AI-ROM must use one of the following kinds of domains:

- *fixed-length domain,* $\mathsf{dom}(\lambda) = \{0,1\}^{n(\lambda)}$ for a polynomially bounded $n : \mathbb{N} \to \mathbb{N}$;
- *(bounded-)variable-length domain,* $\mathsf{dom}(\lambda) = \{0,1\}^{\leq n(\lambda)}$ for some $n : \mathbb{N} \to \mathbb{N}$ such that $\log(n+1)$ is polynomially bounded;
- *full domain,* $\mathsf{dom}(\lambda) = \{0,1\}^*$.

A random oracle of type $\{0,1\}^n \to \{0,1\}^m$ can be implemented by (m calls to) one of type $\{0,1\}^{n+\lceil \log_2 m \rceil} \to \{0,1\}$.

2.2 Leakage Simulation

We rely on the leakage simulation lemma of [CCL18].

Lemma 1 (generalized from [CCL18; Theorem 2]; ¶)**.** *There exists a RAM program h subject to the following conditions. Given*

- *a non-empty set $\mathcal{X}$, advice length $S \in \mathbb{N}$, desired error $0 < \varepsilon \leq 1$,*
- *a set $\mathcal{F}$ of measurable functions $\mathcal{X} \times \{0,1\}^S \to \{0,1\}$ (distinguishers), and*
- *a distribution $\mathcal{D}$ over $\mathcal{X} \times \{0,1\}^S$,*

there exist

- *a list $\mathcal{L}$ of no more than $C \cdot S\varepsilon^{-2}$ functions from $\mathcal{F}$ and*
- *a string z' of no more than $C \cdot (S\varepsilon^{-2} + (S + \log \varepsilon^{-1})^2 \varepsilon^{-1})$ bits*[13]

such that

- *$(X, Z) \overset{\varepsilon}{\approx}_{\mathcal{F}} (X, \widetilde{Z})$, where $(X, Z) \overset{\$}{\leftarrow} \mathcal{D}$ and $\widetilde{Z} \overset{\$}{\leftarrow} h^{X,\mathcal{L}}(z')$ and*
- *$h^{x,\mathcal{L}}(z')$ is computed by calling $f(x, z)$ for each $f \in \mathcal{L}$ in its advice and each $z \in \{0,1\}^S$ plus some other processing in no more than $C_{\mathrm{ram}} \cdot 2^S(S+1)\varepsilon^{-2}$ RAM time.*

Alternatively, $h^{x,\mathcal{L}}(z')$ can be computed by $\mathcal{L}^x$-aided circuits, which make exactly $2^S|\mathcal{L}|$ oracle calls and can be generated in time $C_{\mathrm{ckt}} \cdot 2^S(S^2+1)\varepsilon^{-2}$.[14]

In our version, $\mathcal{X}$ can be a set of arbitrary cardinality and X is explicitly allowed to be provided as an oracle. It enables us to easily handle "monolithic" random oracles, e.g., $\{0,1\}^* \to \{0,1\}$, whose truth tables reside in a set of continuum cardinality. The proof in [CCL18] has some typos and is not presented for infinite $\mathcal{X}$ or $\mathcal{F}$. For completeness, we reprove Lemma 1 in Section A.

[13] The string z' consists of a lookup table for arithmetics as in [CCL18], plus the "negation bits" for functions in $\mathcal{L}$ in case $\mathcal{F}$ is not already closed under negation.

[14] The oracle-aided circuit can be generated by a fixed RAM program taking z' as input. The squaring of S does not make our version worse than [CCL18]. In the circuit model, functions in $\mathcal{F}$ are often also represented by circuits, so each is necessarily of size at least S since it has to read the entire z. Once we replace the oracle gates by their circuits, the oracle calls become the real bottleneck and the overall efficiency is the same as in [CCL18].

We remark that considering infinite $\mathcal{X}$ is a matter of presentation, not substance. For example, for monolithic random oracles, a time-bounded adversary cannot query them on an input longer than the time bound, hence we can truncate their truth tables to a suitable finite size and apply the version in [CCL18]. Nevertheless, our exposition removes the doubt whether the simulator's efficiency could degrade with the encoding length of elements in $\mathcal{X}$ — it cannot, inferred from the fact that $\mathcal{X}$ could be infinite!

2.3 Exponentially Secure Pseudorandom Functions

Exponentially secure PRF is a central tool used in this work.

Definition 5 (PRF)**.** Let InOut be a set whose elements are of the form $(\lambda, \mathsf{dom}, m)$, where $\lambda \in \mathbb{N}$, the domain dom is a subset of $\{0,1\}^*$, and $m \in \mathbb{N}_+$ is the output length. A *pseudorandom function for* InOut consists of two ingredients.

- The *key distribution* $D = \{D_{\lambda,\mathsf{dom},m}\}_{(\lambda,\mathsf{dom},m)\in\mathsf{InOut}}$ is a family (indexed by InOut) of distributions over $\{0,1\}^*$.
- The *evaluation algorithm* $\mathsf{Eval}(k, x)$ is deterministic and efficient. It takes as input k drawn from $D_{\lambda,\mathsf{dom},m}$ and $x \in \mathsf{dom}$, and outputs $y \in \{0,1\}^m$.

The key length must be polynomially bounded. There exists a polynomial $\overline{k}$ such that for all $(\lambda, \mathsf{dom}, m) \in \mathsf{InOut}$, it holds that

$$\Pr\left[\, k \xleftarrow{\$} D_{\lambda,\mathsf{dom},m} \;:\; |k| \le \overline{k}(\lambda, |\mathsf{desc}(\mathsf{dom})|, m) \,\right] = 1,$$

where $|\mathsf{desc}(\mathsf{dom})|$ is the encoding length of dom in Definition 6.

The key distribution need *not* be efficiently sampleable. Our PRF with fast evaluation (Eval running in time sublinear in λ) is primarily described to always pick an expander (inefficient D), but it can be modified to be efficient.

Definition 6 (possible domains of PRF)**.** A PRF must use one of the possible domains of AI-ROM (Definition 4) and they are encoded as follows:

- *fixed-length domain,* $\mathsf{dom} = \{0,1\}^n$, encoded as $\mathsf{desc}(\{0,1\}^n) = 1^n$;
- *(bounded-)variable-length domain,* $\mathsf{dom} = \{0,1\}^{\le n}$, encoded as $\mathsf{desc}(\mathsf{dom}) = n$ in binary with exactly one leading zero;
- *full domain,* $\mathsf{dom} = \{0,1\}^*$, encoded as a distinguished, fixed bit-string (e.g., $\mathsf{desc}(\{0,1\}^*) = 10$ to be different from the other kinds).

In our applications, we will tune the λ of PRF while keeping dom, m fixed, which are determined by the type of AI-ROM. The following definition captures this by allowing the adversary to choose dom, m, possibly independently of λ.

Definition 7 (exponential security of PRF)**.** Let $\varepsilon : \mathbb{N} \to [0, +\infty)$ be negligible. A PRF is $(2^\lambda, \varepsilon)$-*secure* (in a particular model, circuits or RAM) if for all λ,[15] it holds that $E_{\mathrm{prf},\lambda} \overset{\varepsilon(\lambda)}{\approx}_{\mathcal{A}_\lambda} E_{\$,\lambda}$, where $\mathcal{A}_\lambda$ is the set of concrete adversaries of size and time at most 2^λ, and $E_{\mathrm{prf},\lambda}$ and $E_{\$,\lambda}$ with $A \in \mathcal{A}_\lambda$ work as follows.

[15] This is not an asymptotic indistinguishability, as the inequality must hold even for small λ's.

- **Setup.** Launch $A()$[16] and receive from it $(\mathsf{dom}, 1^m)$ such that $(\lambda, \mathsf{dom}, m) \in \mathsf{InOut}$ with dom encoded as specified in Definition 6. In $E_{\mathrm{prf},\lambda}$, sample $k \xleftarrow{\$} D_{\lambda,\mathsf{dom},m}$. In $E_{\$,\lambda}$, sample a random function $\mathcal{O} : \mathsf{dom} \to \{0,1\}^m$.
- **Challenge.** Let A continue to run as either $A^{\mathsf{Eval}(k,\cdot)}$ or $A^{\mathcal{O}}$.
- **Guess.** A outputs a single bit, which is the output of the experiment.

We remark that our definition includes the usual version of subexponential security, e.g., security against 2^λ-size/time adversary with λ^2-bit key. In our formulation, λ is the desired security level (in terms of size and time, but not necessarily advantage) and the key length is chosen according to complexity leveraging.

On the other hand, as security is required against 2^λ-size/time adversaries, it can be shown that $|k|$ must be at least roughly λ. Suppose for simplicity that $|k|$ is not randomized, $|\mathsf{desc}(\mathsf{dom})|, m \le \lambda$, $|\mathsf{dom}| > |k|$, $\varepsilon < \frac{1}{2}$. Consider the simple attack of enumerating all strings of $|k|$ bits as the candidate PRF keys, evaluating the PRF with each key on $(|k|+1)$ fixed points, and comparing them with the oracle responses to see whether the oracle is possibly the PRF. Both size and time of the attack are bounded by $2^{|k|}(|k|+1)\,\mathrm{poly}(\lambda)$, and it has advantage at least $\frac{1}{2}$. Therefore, $2^{|k|}(|k|+1)\,\mathrm{poly}(\lambda) \ge 2^\lambda$, which implies $|k| \ge \lambda - C_{\mathrm{model,prf}} \cdot \log(\lambda+2)$. Similarly, any circuit of Eval that handles at least $(\lambda+1)$ inputs has to be of size $\Omega(\lambda/\log\lambda)$.

3 Simulating AI-ROM Using PRF

We present our theorem of simulating AI-ROM using any exponentially secure PRF.

Theorem 2 (concrete simulation; ¶)**.** *For each model (either RAM or circuits), there exists a function*

$$\lambda_{\mathrm{prf},0}(\mathsf{dom}, m, S, M, T, \varepsilon_{\mathrm{leak}}) \le S + C_{\mathrm{model}} \cdot \log \frac{|\mathsf{desc}(\mathsf{dom})| + m + S + M + T + 1}{\varepsilon_{\mathrm{leak}}}$$

such that for all PRF (D, Eval) with $(2^\lambda, \varepsilon_{\mathrm{prf}})$-security, AI-ROM of type $\mathsf{dom} \to \{0,1\}^m$, $\mathcal{O}$-advice ai of length S, upper bounds M, T of adversary size and time, leakage simulation error $0 < \varepsilon_{\mathrm{leak}} \le 1$, complexity-leveraged $\lambda_{\mathrm{prf}} \ge \lambda_{\mathrm{prf},0}(\mathsf{dom}, m, S, M, T, \varepsilon_{\mathrm{leak}})$ such that $(\lambda_{\mathrm{prf}}, \mathsf{dom}, m) \in \mathsf{InOut}$, there exists a k-advice $\widetilde{\mathsf{ai}}$ of length S so that $E_{\mathrm{airom}} \overset{\varepsilon}{\approx}_{M,T} E_{\mathrm{prf}}$ (concrete RAM or circuits) with $\varepsilon = \varepsilon_{\mathrm{leak}} + \varepsilon_{\mathrm{prf}}(\lambda_{\mathrm{prf}})$, where the experiments work as follows with A that outputs a single bit.

- *In E_{airom}, sample a random function $\mathcal{O} : \mathsf{dom} \to \{0,1\}^m$ then $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O})$. Run and output $A^{\mathcal{O}}(z)$.*
- *In E_{prf}, sample $k \xleftarrow{\$} D_{\lambda_{\mathrm{prf}},\mathsf{dom},m}$ then $z \xleftarrow{\$} \widetilde{\mathsf{ai}}(k)$. Run and output $A^{\mathsf{Eval}(k,\cdot)}(z)$.*

In the RAM model, with A regarded as an oracle, E_{prf} is a concrete algorithm with[17]

$$\begin{aligned} M_E &\le M_{\mathrm{ram,Eval}} + S + |k| + C_{\mathrm{ram}} \cdot \log(M_{\mathrm{ram,Eval}} + S + |k| + 2), \\ T_E &\le Q_A \cdot T_{\mathrm{ram,Eval}} \qquad + C_{\mathrm{ram}} \cdot \log(M_{\mathrm{ram,Eval}} + S + |k| + 2), \end{aligned}$$

[16] This is a slight abuse of notation. Here, A represents a sample from the distribution of circuits or RAM programs (also called A).

[17] Here, $|k|$ depends on the specific PRF being used and is polynomial in $\lambda_{\mathrm{prf}}, |\mathsf{desc}(\mathsf{dom})|, m$ (hence polynomial in $|\mathsf{desc}(\mathsf{dom})|, m, S$ and polylogarithmic in $M, T, \varepsilon_{\mathrm{leak}}$). The time of E only includes its initialization and responses to A's queries.

where Q_A is the maximum number of queries made by A. In the two models (RAM and circuits), $E_{\mathrm{prf}}(A)$ is a concrete adversary with

$$\begin{aligned}
\text{(RAM)}\quad & M' \le M_A + M_{\mathrm{ram,Eval}} + S + |k| + C_{\mathrm{ram}} \cdot \log(M_A + M_{\mathrm{ram,Eval}} + S + |k| + 2),\\
& T' \le T_A + Q_A \cdot T_{\mathrm{ram,Eval}} \quad + C_{\mathrm{ram}} \cdot \log(M_A + M_{\mathrm{ram,Eval}} + S + |k| + 2),\\
\text{(circuits)}\quad & M' = T' = M_A + Q_A \cdot M_{\mathrm{ckt,Eval}}.
\end{aligned}$$

Theorem 3 (asymptotic simulation; ¶)**.** *The asymptotic version $E_{\mathrm{airom}} \overset{\varepsilon}{\approx}_{M,T} E_{\mathrm{prf}}$ of Theorem 2 holds for some $\lambda_{\mathrm{prf},0}(S,\lambda) \le S(\lambda) + C_{\mathrm{model}} \cdot (\log(S(\lambda)+1) + \lambda + 1)$, any M, T such that $M(\lambda), T(\lambda) \le 2^\lambda$ for sufficiently large λ, and $\varepsilon_{\mathrm{leak}}(\lambda) = 2^{-\lambda}$, where $M(\lambda)$ is now the length of $w \xleftarrow{\$} \mathsf{nu}(\lambda)$ and the experiments also feed 1^λ and $w \xleftarrow{\$} \mathsf{nu}(\lambda)$ into A. In particular, $E_{\mathrm{airom}} \approx E_{\mathrm{prf}}$. In the RAM model, with $A(1^\lambda, w, \cdot)$ regarded as an oracle, E_{prf} is an asymptotic algorithm with*

$$\begin{aligned}
M_E &= S + |k| + 2\lceil \log_2(S+1) \rceil + 1,\\
T_E &\le Q_A \cdot T_{\mathrm{ram,Eval}} + C_{\mathrm{ram}} \cdot \log(S + |k| + 2).
\end{aligned}$$

In both models (RAM and circuits), the sampling of (w, z, k) in E_{prf} can be combined into $\widetilde{\mathsf{nu}}(\lambda)$, and $E_{\mathrm{prf}}(A)$ is an asymptotic adversary with

$$\begin{aligned}
\text{(both)}\quad & M' = M_A + S + |k| + 2\lceil \log_2(M_A+1) \rceil + 2\lceil \log_2(S+1) \rceil + 2,\\
\text{(RAM)}\quad & T' \le T_A + Q_A \cdot T_{\mathrm{ram,Eval}} + C_{\mathrm{ram}} \cdot \log(M_A + S + |k| + 2),\\
\text{(circuits)}\quad & T' \le C_{\mathrm{ckt}} \cdot (M_A + S + |k| + T_A + Q_A \cdot T_{\mathrm{ckt,Eval}} + 1).
\end{aligned}$$

3.1 Comparison with Presampling

Before proceeding to the proofs, we compare our AI-ROM simulation theorems with presampling [Unr07a,CDGS18,GLLZ21]. Let's first recall the presampling theorem.

Lemma 4 (presampling with additive error [CDGS18; Lemma 1])**.** *For all AI-ROM type $\mathsf{dom} \to \{0,1\}^m$ such that $\mathsf{dom} \neq \{0,1\}^*$, $\mathcal{O}$-advice ai of length S, upper bound Q of number of queries, number $P \in \mathbb{N}_+$ of fixed coordinates, decomposition error $0 < \varepsilon_{\mathrm{decomp}} \le 1$, there exists a z-advice $\mathsf{ai}_{\mathrm{presamp}}$ whose output shall be interpreted as a set $\mathcal{T}$ of no more than P pairs of $(x,y) \in \mathsf{dom} \times \{0,1\}^m$ with distinct x's such that $E_{\mathrm{airom}} \overset{\varepsilon}{\approx}_{\mathcal{A}} E_{\mathrm{presamp}}$ for $\varepsilon = (S + \log_2(1/\varepsilon_{\mathrm{decomp}})) \cdot Q/P + \varepsilon_{\mathrm{decomp}}$, where $\mathcal{A}$ is the set of concrete adversaries making at most Q queries to the oracle and having single-bit output. The experiments are as follows.*

- *E_{airom} is the same as in Theorem 2.*
- *In E_{presamp}, sample two independent random functions $\mathcal{O}, \mathcal{O}_{\mathrm{presamp}} : \mathsf{dom} \to \{0,1\}^m$, then $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O}_{\mathrm{presamp}})$ and $\mathcal{T} \xleftarrow{\$} \mathsf{ai}_{\mathrm{presamp}}(z)$. Define $\mathcal{O}[\mathcal{T}](x)$ to be y if $(x,y) \in \mathcal{T}$ for some y and $\mathcal{O}(x)$ otherwise. Run and output $A^{\mathcal{O}[\mathcal{T}]}(z)$.*

AI-ROM is not efficiently implementable by standard-model algorithms. Both our theorem and presampling provide a way to simulate AI-ROM efficiently. Their major differences are in Table 3. We make the following remarks.

One major advantage of our method is that it achieves the standard definition of simulation (negligible error against polynomial-time distinguishers) in polynomial time, which resolves Unruh's conjecture (Corollary 8) that AI-ROM does not create

Table 3. Comparison between our theorems and presampling for $\{0,1\}^n \to \{0,1\}^m$.

aspect	our theorems	presampling
oracle	PRF instance	random except at P points
assumption	exponentially secure PRF	✓ none
adversary	computational	query-bounded
error	✓ could be exponential	inverse polynomial
extra advice	✓ $\mathrm{poly}(S,\lambda,n,m)$	$\Theta((n+m)SQ/\varepsilon)$
per-query size (circuit)	✓ $\mathrm{poly}(S,\lambda,n,m)$	$\Theta((n+m)SQ/\varepsilon)$
per-query time (RAM)	$\mathrm{poly}(S,\lambda,n,m)$	✓ $\widetilde{\mathrm{O}}(n+m)$ [with perfect hashing]

computational advantage. With presampling, the simulation efficiency grows inverse-polynomially with the error — this growth relation is tight [DGK17,CDGS18] — so it will *never* achieve negligible error in polynomial time.

Although our theorems only work with computationally bounded adversaries, the time bound is exponential, covering the practically relevant case. Our simulation efficiency does *not* depend on the number of queries, hence it could enjoy significant savings in some parameter regimes compared to presampling.

The table shows efficiency from a generic exponentially secure PRF (per Definition 5; more classically known as "subexponentially secure PRF"). By using a PRF with very fast evaluation (Section 4) in the RAM model, the per-query time of our simulator is $m(n+\log m)^{1+\gamma}\,\mathrm{poly}(\log(S+\lambda))$, where $\gamma > 0$ is an arbitrarily small positive constant. Despite still worse than presampling, its significant savings on extra advice come out on top. We depict practical attacks as having S much smaller than Q, so the independence with Q is meaningful. Concretely, our simulator only needs extra advice $((n+\log m)(S+\lambda))^{1+\gamma}$, hence is better as long as Q is not too small.

Unlike presampling, where the correlation is controlled by $\mathcal{T}$, we have little to say about the correlation between the PRF used as the oracle and the auxiliary input. This leads to great difficulty in proving security *reliant on* the random oracle, excluding the techniques of programming and extraction. Nevertheless, our simulation could be used to reduce assumptions *unrelated* to the random oracle more tightly to their standard-model versions. The two methods are not mutually exclusive. In applications, for each pair of neighboring hybrids, we can utilize the suitable method.

3.2 Proofs

We derive our concrete AI-ROM simulation in two steps. For simplicity, let's assume $|\mathsf{desc}(\mathsf{dom})|, m, S, M, T \le 2^\lambda$ and $\varepsilon_{\mathrm{leak}} \ge 2^{-\lambda}$ (the setting useful for asymptotics). The $\mathcal{O}$-advice ai we are given with could be arbitrarily complex — difficult to handle — so we first rely on the leakage simulation lemma to bring its complexity down to $2^{S+\mathrm{O}(\lambda)}$, while keeping $(\mathcal{O}, z)$ indistinguishable to size-M time-T adversaries. We then regard the leakage simulator together with the underlying adversary as an adversary against the PRF, whose overall complexity is dominated by that of the leakage simulator. Because the PRF is secure against exponential-time adversaries, we can adjust the security parameter to some $\lambda_{\mathrm{prf}} = S + \mathrm{O}(\lambda)$. Replacing the random oracle by PRF completes the proof.

Note that in the intermediate step where we invoke PRF security, its adversary runs in *excessive* time as it includes the leakage simulator — a merely polynomial

hardness assumption would not survive it (even with complexity leveraging, the adjusted security parameter could become super-polynomial). However, once we arrive at the final stage, the PRF key k and the leakage z can be regarded as (standard-model) advice, the leakage simulator is no longer part of the adversary, and the adversary, with oracle queries answered by PRF, is *again efficient*, only suffering a blow-up polynomial in S, λ.

We now present the formal proof.

Proof (Theorem 2). Consider the RAM version. Define

$$\begin{aligned}
\mathcal{P} &= \left\{ A \,\middle|\, A \text{ is a RAM program with single-bit output, } M_A \le M,\ T_A \le T \right\},\\
\mathcal{A} &= \left\{ A \,\middle|\, A \text{ is a distribution over } \mathcal{P} \right\},\\
\mathcal{F} &= \left\{ f_{A,r} : (\mathcal{O}, z) \mapsto A^{\mathcal{O}}(z; r) \,\middle|\, A \in \mathcal{P}, \text{ randomness } r \in \{0,1\}^T \right\}.
\end{aligned}$$

Let h be the RAM program in Lemma 1, then there exist a list $\mathcal{L}$ of functions in $\mathcal{F}$ and a bit-string z', of promised lengths, such that $(\mathcal{O}, \mathsf{ai}(\mathcal{O})) \overset{\varepsilon_{\text{leak}}}{\approx}_{\mathcal{F}} (\mathcal{O}, h^{\mathcal{O},\mathcal{L}}(z'))$, which implies $(\mathcal{O}, \mathsf{ai}(\mathcal{O})) \overset{\varepsilon_{\text{leak}}}{\approx}_{\mathcal{A}} (\mathcal{O}, h^{\mathcal{O},\mathcal{L}}(z'))$.

Encode $\mathcal{L}$ into a bit-string z'' by writing down first a prefix-free encoding of $|\mathcal{L}|$, next a prefix-free encoding of T, then for each $f_{A,r} \in \mathcal{L}$, a prefix-free encoding of M_A, next A itself, then r itself. By Lemma 1 and the definition of $\mathcal{F}$, the length of z'' is bounded by $C \cdot (S+1)\varepsilon_{\text{leak}}^{-2} \cdot (M+T+1)$. For each $A \in \mathcal{A}$, let $A^{\mathcal{O}}_{\text{prf}}(z', z'')$ first choose dom, m (from the random oracle model it targets) as the PRF challenge lengths, next parse z'' into $\mathcal{L}$, then run $z \xleftarrow{\$} h^{\mathcal{O},\mathcal{L}}(z')$, and lastly run and output $A^{\mathcal{O}}(z)$. We have

$$\begin{aligned}
M_{A_{\text{prf}}} &\le \overbrace{C_{\text{ram}}}^{\text{including parsing}} \cdot \Big(\overbrace{|\mathsf{desc}(\mathsf{dom})| + m}^{\mathsf{dom}, m} + \overbrace{1}^{\text{gluing}, h, A} + M_A + \overbrace{S\varepsilon_{\text{leak}}^{-2} + (S + \log \varepsilon_{\text{leak}}^{-1})^2 \varepsilon_{\text{leak}}^{-1} + |z''|}^{z'} \Big),\\
T_{A_{\text{prf}}} &\le C_{\text{ram}} \cdot \Big(\underbrace{(|\mathsf{desc}(\mathsf{dom})| + m + |z''|)^{C_{\text{ram}}}}_{\text{parsing, gluing}} + \underbrace{2^S \cdot S\varepsilon_{\text{leak}}^{-2} \cdot T}_{\mathcal{L}^{\mathcal{O}} \text{ in } h} + \underbrace{2^S (S+1)\varepsilon_{\text{leak}}^{-2}}_{\text{other in } h} + T_A \Big).
\end{aligned}$$

Both are bounded by $2^{\lambda_{\text{prf},0}}$ for some

$$\lambda_{\text{prf},0} \le S + C_{\text{ram}} \log((|\mathsf{desc}(\mathsf{dom})| + m + S + M + T + 1)/\varepsilon_{\text{leak}}).$$

For $\lambda_{\text{prf}} \ge \lambda_{\text{prf},0}$, it follows from PRF security that $\mathcal{O}$ (random) and $\mathsf{Eval}(k, \cdot)$ (with $k \xleftarrow{\$} D_{\lambda_{\text{prf}},\mathsf{dom},m}$) are $\varepsilon_{\text{prf}}(\lambda_{\text{prf}})$-indistinguishable to A_{prf}'s. By definition, this is exactly $(\mathcal{O}, h^{\mathcal{O},\mathcal{L}}(z')) \overset{\varepsilon_{\text{prf}}(\lambda_{\text{prf}})}{\approx}_{\mathcal{A}} (\mathsf{Eval}(k, \cdot), h^{\mathsf{Eval}(k,\cdot),\mathcal{L}}(z'))$.

Defining $\widetilde{\mathsf{ai}}(k)$ as $h^{\mathsf{Eval}(k,\cdot),\mathcal{L}}(z')$, we obtain $E_{\text{airom}} \overset{\varepsilon}{\approx}_{\mathcal{A}} E_{\text{prf}}$ by hybrid argument, where $\varepsilon = \varepsilon_{\text{leak}} + \varepsilon_{\text{prf}}(\lambda_{\text{prf}})$ and subscript "$\mathcal{A}$" means the same as "M, T (for RAM)". To construct the concrete algorithm E_{prf}, recall (Definition 1) that it is a *distribution* over RAM programs. We let the distribution sample $k \xleftarrow{\$} D_{\lambda_{\text{prf}},\mathsf{dom},m}$ and $z \xleftarrow{\$} \widetilde{\mathsf{ai}}(k)$, then output a RAM program with z, k hardwired that calls $A^{\mathsf{Eval}(k,\cdot)}(z)$. The bounds of M_E, T_E follow easily, where the model-dependent terms are due to hardwiring the input and combining Eval with gluing code. The derivation of $E_{\text{prf}}(A)$ (for RAM and circuits) is similar. We only note that for circuits, M_A and $M_{\text{ckt,Eval}}$ already subsume S and $|k|$, so the latter two terms do not appear in M'. □

Proof (Theorem 3). Let (nu, A) be an asymptotic RAM adversary such that for sufficiently large λ it holds that $M_A(\lambda), T_A(\lambda) \le 2^\lambda$. For each λ, let $w \xleftarrow{\$} \mathsf{nu}(\lambda)$ and $A'_\lambda(\cdot) = A(1^\lambda, w, \cdot)$, then A'_λ is a concrete RAM adversary with

$$M'_\lambda \le |A| + \lambda + M_A(\lambda) + C_{\mathrm{ram}} \cdot \log(|A| + \lambda + M_A(\lambda) + 2), \qquad T'_\lambda = T_A(\lambda).$$

Now, $|\mathsf{desc}(\mathsf{dom}(\lambda))| \le 2^\lambda$, $m(\lambda) \le 2^\lambda$, $M'_\lambda \le 2^{\lambda+1}$, and $T'_\lambda \le 2^\lambda$, all of which hold for sufficiently large λ, for which we apply Theorem 2 with $\mathsf{dom}(\lambda), m(\lambda), S(\lambda), M'_\lambda, T'_\lambda$ and $\varepsilon_{\mathrm{leak}} = 2^{-\lambda}$ and conclude

$$\begin{aligned}\lambda_{\mathrm{prf},0} &\le S(\lambda) + C_{\mathrm{ram}} \cdot \log \frac{2^\lambda + 2^\lambda + S(\lambda) + 2^{\lambda+1} + 2^\lambda + 1}{2^{-\lambda}} \\ &\le S(\lambda) + C_{\mathrm{ram}} \cdot (\log(S(\lambda) + 1) + \lambda + 1).\end{aligned}$$

The circuit version is similar. For polynomial indistinguishability, we further rely on $S(\lambda) \le \mathrm{poly}(\lambda) \le 2^\lambda$ for sufficiently large λ.

For the second part. Encode (z, k) by writing down a prefix-free encoding of $S = |z|$ then z, k themselves. This gives the bound of M_E. The algorithm $E_{\mathrm{prf}}^{A(w,\cdot)}(z, k)$ parses its input then calls $A^{\mathsf{Eval}(k,\cdot)}(w, z)$, giving the bound of T_E. The derivation of $E_{\mathrm{prf}}(A)$ is similar. □

4 PRF with Fast RAM Evaluation

In this section, we construct exponentially secure PRF (Definition 5) $\{0,1\}^n \to \{0,1\}$ with key size $C \cdot (n\lambda)^{1+\gamma}$ and evaluation time $n^{1+\gamma}\,\mathrm{poly}(\log \lambda)$ in the RAM model, where $\gamma > 0$ is an arbitrarily small constant. To obtain PRF of type $\{0,1\}^n \to \{0,1\}^m$, we use our construction of type $\{0,1\}^{n+\lceil \log_2 m \rceil} \to \{0,1\}$, whose key size becomes $C \cdot ((n + \lceil \log_2 m \rceil)\lambda)^{1+\gamma}$ and evaluation time, $Cm(n + \lceil \log_2 m \rceil)^{1+\gamma}\,\mathrm{poly}(\log \lambda)$.

In the subsections, we *mostly* do not talk about PRF with multi-bit output and will repurpose m for the output length of Goldreich's function, which corresponds to the PRF *truth table* length.

4.1 Construction from Goldreich's Function

Our candidate PRF with fast RAM evaluation is based on Goldreich's expander-based one-way function [Gol00] that has received years of study.

Goldreich's One-Way Function. The function $f_{P,G} : \{0,1\}^\ell \to \{0,1\}^m$ is specified by an L-local Boolean predicate $P : \{0,1\}^L \to \{0,1\}$ and a hypergraph $G = (S_1, \ldots, S_m)$ where each hyperedge is a list S_i of L distinct indices from $[\ell]$. On input $x \in \{0,1\}^\ell$, the function outputs $(y_1, \ldots, y_m)$ for $y_i = P(x|_{S_i})$. Here, $x|_{S_i}$ are the bits of x indexed by S_i, in the order as given in S_i.

We will use $f_{P,G}$ (not just as an OWF, but) as a PRF $[m] \to \{0,1\}$, where the PRF key is the input to $f_{P,G}$ and the PRF input indexes the output of $f_{P,G}$. The predicate and the (expander) graph must be chosen carefully based on insights developed in a long series of works [Gol00,CM01,Fei02,MST03,FKO06,CEMT09,BQ09,ABW10,ABR12,BQ12, App12,App13,OW14,AL16]. We discuss these aspects at a high level below. Further details can be found in Section 4.3.

Choosing Predicate P. Typically, the locality L is some constant for output length of f being a polynomial $m = \ell^{O(L)}$, i.e., the degree of output length as a polynomial is linearly related to the locality. Since we are interested in using f as a PRF, its output length (PRF truth table size) needs to be super-polynomial. We will therefore work with a super-constant locality $L = O(\ell^\delta)$ for some constant $0 < \delta < 1$. Such parameters have been considered in the past [AR16] in a similar context of constructing PRF.

Based on the extensive cryptanalysis culminating in [AL16], a good predicate should simultaneously optimize *resiliency* and *rational degree* (see Section 4.3). The gold standard for P is $\mathsf{XorMaj}_{L'}$ that, on input of $L = 2 \cdot L'$ bits, computes

$$\mathsf{XorMaj}_{L'}(x_1, \ldots, x_{L'}, x_{L'+1}, \ldots, x_L) = x_1 \oplus \cdots \oplus x_{L'} \oplus \mathsf{majority}(x_{L'+1}, \ldots, x_L).$$

Choosing (Expander) Graph G. For a uniformly random hypergraph $G = (S_1, \ldots, S_m)$, with probability $1/\ell^{O(L)}$, two edges might turn out to be the same. In this case, an image from $f_{P,G}$ is trivially distinguishable from random. More generally, one could find "cycles"/"even covers", which are $S_{i_1}, \ldots, S_{i_t}$ such that every variable (vertex) appears an even number of times in them. When $t > 0$ is small, this might cause $y_{i_1}, \ldots, y_{i_t}$ to be biased, leading to an attack in time $2^{\widetilde{O}(t)}$. Therefore, we must choose G so that all cycles have sufficiently large t. Small cycles can be ruled out if the graph is sufficiently expanding.

An L-uniform hypergraph $G = (S_1, \ldots, S_m)$ is said to be (t, α)-expanding, if for every non-empty set $T \subseteq [m]$ of size $r \le t$, it holds that

$$\left| \bigcup_{i \in T} S_i \right| > \alpha \cdot L \cdot r.$$

In the union operation, the S_i's are regarded as sets.

Observe that if $S|_T$ form a cycle for $T = \{i_1, \ldots, i_t\}$, then G is not $(t, 1/2)$-expanding — every vertex in the union of $S|_T$ appears at least 2 times, discounting the union size from $L \cdot t$ by a factor of $1/2$. As we will discuss in Section 4.3, lower bounds against certain classes of algorithms (attacks currently known to be the best), including polynomial calculus, light linear tests, sum-of-squares, etc., can be proven, provided that α is large enough. We work with a conservative $\alpha = 0.99$ for Assumption 1. Similar assumptions were also used in [AR16] to design of efficient PRF.

Assumption 1 (pseudorandomness of Goldreich's function)**.** Let $0 < \delta < 1$ be a constant, $\ell \in \mathbb{N}$ the input length, and $L = 2 \cdot L' = O(\ell^\delta)$ the locality. Let G be any L-uniform hypergraphs with m hyperedges such that G is $(t, 0.99)$-expanding. The assumption states that

$$\left\{ x \xleftarrow{\$} \{0,1\}^\ell \ : \ f_{\mathsf{XorMaj}_{L'},G}(x) \right\} \approx_{2^{\widetilde{O}(t)}} \left\{ y \xleftarrow{\$} \{0,1\}^m \ : \ y \right\},$$

where the subscript of $\approx$ means that the adversaries are concrete and of size $2^{\widetilde{O}(t)}$.

We note that based on our current understanding, we can strengthen Assumption 1 to $2^{-\widetilde{O}(t)}$ *also* in advantage. Moreover, the work of [AR16] showed that the above assumption follows from one-wayness with slightly worse parameters.

Our candidate PRF is simply Goldreich's PRG with appropriate L, ℓ dependent on the security parameter λ and the PRF input length n. The output length is set to $m = 2^n$. Before describing our construction, we need the following notion.

Definition 8 (graph sampler and accessor)**.** A *graph sampler and accessor* consists of two algorithms.

- $\mathsf{GraphSetup}(1^\ell, m, 1^L, 1^t)$ takes as input the number ℓ of vertices, the number m of edges, the locality L, the expansion bound t. It outputs pp representing a $(t, 0.99)$-expanding L-uniform hypergraph with m edges over ℓ vertices. This algorithm does not have to be efficient. The length of pp should be small.
- $\mathsf{GraphEdge}(\mathsf{pp}, i)$ is a deterministic efficient (RAM) algorithm. On input pp and index $i \in [m]$, it outputs the edge S_i, represented as a length-L list of distinct integers from $[\ell]$.

In Section 4.2, we will construct a graph sampler and accessor with *very fast* GraphEdge in the RAM model. Its precise time complexity is not simple to specify, and we defer this to Construction 4.

Ingredients and Parameters of Construction 1. We rely on Construction 4, a highly efficient graph sampler and accessor. Let γ' be a constant determined (specified later) by the tunable, arbitrarily small constant $0 < \gamma < 1$ of the target PRF efficiency, which is also related to the running time of GraphEdge.

Construction 1 (exponentially secure PRF with fast evaluation)**.** It works as follows (see Definition 5).

- The distribution $D_{n,1}$ (i.e., for $\{0,1\}^n \to \{0,1\}$) sets[18] and samples

$$\ell \leftarrow \lceil n \cdot \lambda^{1+\gamma'} \rceil, \qquad m \leftarrow 2^n, \qquad L = 2 \cdot L' \leftarrow 100n, \qquad t \leftarrow \lfloor \lambda^{1+\gamma'/2} \rfloor,$$
$$\mathsf{pp} \xleftarrow{\$} \mathsf{GraphSetup}(1^\ell, m, 1^L, 1^t), \qquad s \xleftarrow{\$} \{0,1\}^\ell,$$

  and outputs $k = (1, \mathsf{pp}, s)$.

  We note that for security, $D_{n,1}$ needs to set $\lambda = \Omega(n^{1/\delta})$ when an over-small λ is requested. This is a non-issue as in our applications, n is a scheme-prescribed value (say, λ *for the application*, fixed) and (PRF's) $\lambda = \lambda_{\mathrm{prf}}$ is leveraged beyond the amount S of auxiliary information (determined by adversary, growing).

- The distribution $D_{n,m'}$ is $D_{n+\lceil \log_2 m' \rceil, 1}$ modified to output $k = (\boxed{m'}, \mathsf{pp}, s)$.
- $\mathsf{Eval}(k, x)$ first parses k into (m', pp, s). For j from 1 to m', it writes j as a zero-padded $\lceil \log_2 m' \rceil$-bit number after x to form x_j, runs $S_{x,j} \leftarrow \mathsf{GraphEdge}(\mathsf{pp}, x_j)$, and computes $x'_j = \mathsf{XorMaj}_{L'}(s|_{S_{x,j}})$. The algorithm outputs $x' = x'_1 \cdots x'_{m'}$.

Theorem 5. *Under Assumption 1, Construction 1 is exponentially secure (Definition 7). When instantiated with Construction 4 and setting* $\gamma' = \sqrt{1+\gamma} - 1$*, when* $n, m', \lambda \geq 1$ *and* λ *is not adjusted up by* $D_{n,m'}$*, the scheme has key length* $k \leq C \cdot ((n + \lceil \log_2 m' \rceil)\lambda)^{1+\gamma}$ *and RAM evaluation time* $m'(n + \lceil \log_2 m' \rceil)^{1+\gamma} \operatorname{poly}(\log \lambda)$*. (The key size is dominated by* $|\mathsf{pp}|$ *and the evaluation time is dominated by that to run* GraphEdge*.)*

See Remark 2 for the derivation of efficiency parameters of Theorem 5.

[18] These parameters have a lot of room, need to be considered together with the underlying graph sampler and accessor, and are not intended to be optimal.

4.2 How to Sample and Access the Expander

We describe our graph sampler and accessor. The idea is that the hyperedges are computed by a t-wise independent hash function.

We will make use of the following powerful result.

Lemma 6 (corollary of [KU11; Theorem 5.1])**.** *Fix any constant $\gamma' > 0$. Let $f(x) \in \mathbb{Z}_p[x]$ be a polynomial of degree t over the integers modulo a prime p. In the RAM model, one can compute from t, p, and the coefficients of $f(x)$, in time $O((t \log p)^{1+\gamma'})$, a data structure of such size and with which, given any $\alpha \in \mathbb{Z}_p$, the value $f(\alpha)$ can be computed in time $\mathrm{poly}(\log t) \cdot \log^{1+\gamma'} p$.*

The take-home message of Lemma 6 is that after some preprocessing, a polynomial can be evaluated in time much shorter than that needed to specify it in the first place. It enables us to efficiently compute the hyperedges.

Recall that for Assumption 1 to hold, the hypergraph should be $(t, 0.99)$-expanding. The simplest construction is to sample the hyperedges uniformly at random, which would be too long to write down hence not sufficient for us. The proof that a random hypergraph is expanding only relies on t-wise independence among the hyperedges. We will generate such a distribution using a random degree-t polynomial with an appropriate prime modulus p. Fast GraphEdge relies on Lemma 6.

We also need a fast algorithm to convert a number to an ordered list of indices. This is done in two steps. *First,* a number is converted to its heterogeneous base representation, effectively[19] a bijection $[v_1 \cdots v_{L^+}] \to [v_1] \times \cdots \times [v_{L^+}]$...

Construction 2 (heterogeneous base conversion)**.** Let basepp be some efficient preprocessed information about (bases) $v_1, \ldots, v_{L^+} \in \mathbb{N}_{\geq 2}$ (with $L^+ \geq 1$) to be described later. The algorithm BaseConv(basepp, v) takes as input basepp and $u \in \mathbb{N}$ such that $v < v_1 \cdots v_{L^+}$. It computes the unique $u_1, \ldots, u_{L^+} \in \mathbb{N}$ such that $u_i < v_i$ for all $i \in [L^+]$ and

$$u = u_1 + v_1(u_2 + v_2(\cdots(u_{L^+-1} + v_{L^+-1}u_{L^+})\cdots)) = \sum_{i=1}^{L^+} u_i \prod_{j=1}^{i-1} v_j.$$

The algorithm works by divide-and-conquer — first perform division with remainder

$$u = u' + u'' \cdot v_1 \cdots v_{\lfloor L^+/2 \rfloor}, \qquad u', u'' \in \mathbb{N}, \qquad 0 \leq u' < v_1 \cdots v_{\lfloor L^+/2 \rfloor},$$

then recursively convert u' with respect to bases $v_1, \ldots, v_{\lfloor L^+/2 \rfloor}$ and u'' with respect to bases $v_{\lfloor L^+/2 \rfloor + 1}, \ldots, v_{L^+}$. The quotient u'' is computed by standard reduction to multiplication — to compute $\lfloor x/y \rfloor$ as a function of $x \in \mathbb{N}$ and $x \leq \overline{x}$ (with $\overline{x} \geq 1$) for a fixed $y \in \mathbb{N}_+$, store

$$y_1 = \lceil \log_2(\overline{x}y) \rceil, \qquad y_2 = \lceil 2^{y_1}/y \rceil$$

in preprocessed information and compute $\lfloor x \cdot y_2 \cdot 2^{-y_1} \rfloor$ (multiply by y_2, then discard the last y_1 bits). The remainder u' is then easily computed.

Binary integer multiplication is done using Fourier transform in quasilinear time. Let $\overline{v} = \max\{v_1, \ldots, v_{L^+}\}$, then by routine calculation, basepp and BaseConv can be computed in time $L^+ \, \mathrm{poly}(\log L^+, \log \overline{v})$.

[19] The actual function uses 0-based indices.

... *Second,* an element in $[\ell] \times [\ell-1] \times \cdots \times [\ell-(L-1)]$ is bijectively converted to an ordered list of L distinct indices from $[\ell]$, as follows.[20]

Construction 3 (sampling without replacement)**.** Suppose $\ell, L, u_1, \ldots, u_L \in \mathbb{N}$, $L \le \ell$, and $u_i < \ell - (i-1)$ for all $i \in [L]$. We define $\mathsf{SampNoReplace}(\ell, L, u_1, \ldots, u_L) = (v_1, \ldots, v_L)$, where v_i is the $(u_i+1)^{\text{st}}$ smallest element in $[\ell] \setminus \{v_1, \ldots, v_{i-1}\}$. Equivalently, we have $v_i = u_i + j_i$, where j_i is the unique $j \in [i]$ such that $\{v_1, \ldots, v_{i-1}\}$ has exactly $(j-1)$ elements less than or equal to $(u_i + j)$.

Assuming correct input format (no checks will be performed), we implement SampNoReplace as follows. Prepare an empty balanced search tree BST to hold at most L elements in $[\ell]$ in ascending order. For $i = 1, \ldots, L$, maintain the invariant that BST holds $\{v_1, \ldots, v_{i-1}\}$, which is initially true. Given such BST and any $j \in [i]$, whether $j < j_i$ or not can be decided in RAM time $\mathrm{poly}(\log(\ell+1))$.[21] The j_i can be found in $O(\log i)$ decisions by binary search over $[i]$. Store $v_i = u_i + j_i$ and also insert v_i into BST for the next iteration.

Overall, this procedure takes time $(L+1)\,\mathrm{poly}(\log(\ell+1))$.

It should be mentioned that Constructions 2 and 3 are nothing new and "are well-known to the competitive programming community".

With those, we are ready to present our graph sampler and accessor.

Construction 4 (graph sampler and accessor)**.** It works as follows.

- $\mathsf{GraphSetup}(1^\ell, m, 1^L, 1^t)$ computes $N = \ell!/(\ell-L)!$, finds a prime p between $2Nm$ and $4Nm$ and a degree-t polynomial $h(z) = a_0 + a_1 z + a_2 z^2 + \cdots + a_t z^t$ over $\mathbb{Z}_p$ such that the GraphEdge below defines a $(t, 0.99)$-expander hypergraph. (This is ***not*** efficiently checkable. See later for analysis.) The algorithm preprocesses $h(z)$ using Lemma 6 to obtain data structure ku, and preprocesses base conversion to

$$\ell,\ \ell-1,\ \ldots,\ \ell-(L-1),\quad \underbrace{2,\ \ldots,\ 2}_{\lceil \log_2(4m) \rceil \text{ of them (highest positions)}},$$

  into basepp (Construction 2). It outputs $\mathsf{pp} = (\mathsf{ku}, \mathsf{basepp}, \ell, L)$.

- $\mathsf{GraphEdge}(\mathsf{pp}, i)$ first parses $(\mathsf{ku}, \mathsf{basepp}, \ell, L) \leftarrow \mathsf{pp}$. It evaluates $h(z)$ at i using ku (Lemma 6), resulting in (by abuse of notation) an integer $0 \le h(i) < p$. The algorithm runs $\mathsf{BaseConv}(\mathsf{basepp}, h(i))$ of Construction 2 to obtain

$$u_1, \ldots, u_L, \quad u_{-1}, u_{-2}, \ldots \qquad \text{with } 0 \le u_j < \ell - (j-1) \text{ for all } j \in [L].$$

  Discarding the high bits $u_{-1}, u_{-2}, \ldots$, the algorithm runs $\mathsf{SampNoReplace}(\ell, L, u_1, \ldots, u_L)$ of Construction 3 to obtain $(v_1, \ldots, v_L)$, a list of L distinct elements in $[\ell]$. It outputs $S_i = (v_1, \ldots, v_L)$.

Theorem 7 (¶)**.** *In Construction 4, a desired $h(z)$ exists when*

$$0.99tL < \ell \qquad \text{and} \qquad \sum_{T=1}^{t} \left(\frac{m}{T}\right)^T \left(\frac{0.99 \cdot TL}{\ell}\right)^{0.01 \cdot TL} \exp\left(1.99 \cdot TL + T\right) \le \frac{1}{2}.$$

[20] The actual function uses 0-based indices in the input list and 1-based indices in the output list.

[21] Each comparison takes time $\mathrm{poly}(\log(\ell+1))$, the tree BST performs $\mathrm{poly}(\log(i+1))$ comparisons plus other basic operations, and $\ell \ge i$ absorbs i.

Fix the arbitrary constant $\gamma' > 0$ for Lemma 6, then $|\mathsf{pp}|$ *is*

$$\mathrm{O}\left(\log^{1+\gamma'}(\ell^L m)\cdot t^{1+\gamma'} + (L+\log m)\,\mathrm{poly}(\log L, \log\log m, \log \ell)\right)$$

and the running time of GraphEdge *(in the RAM model) is*

$$\mathrm{O}\left(\log^{1+\gamma'}(\ell^L m)\cdot \mathrm{poly}(\log t) + (L+\log m)\,\mathrm{poly}(\log L, \log\log m, \log \ell)\right).$$

Remark 1 (efficient version of GraphSetup). If an efficient GraphSetup is desired, we can relax the notion so that the pp produced by GraphSetup could be non-$(t, 0.99)$-expanding with probability less than $(\varepsilon_1 + \varepsilon_2)$ for some $\varepsilon_1, \varepsilon_2 > 0$. Construction 4 and Theorem 7, as stated, correspond to $\varepsilon_1 = \varepsilon_2 = \frac{1}{2}$. When generalized,

- Construction 4 is changed to sample p between Nm/ε_1 and $2Nm/\varepsilon_1$ and to use $\lceil \log_2(2m/\varepsilon_1)\rceil$ twos in the base conversion,
- Theorem 7 is changed to require

$$\sum_{T=1}^{t}\left(\frac{m}{T}\right)^T\left(\frac{0.99\cdot TL}{\ell}\right)^{0.01\cdot TL}\exp\left(1.99\cdot TL + T\right) \le \varepsilon_2,$$

- the size of pp and the time complexity of GraphEdge are changed to have every appearance of m replaced by m/ε_1, and
- the proof is changed to use $\Pr[V] < \varepsilon_1$ and $\Pr[Q \cap \neg V] \le \varepsilon_2$.

We note that when applying the efficient version to Theorem 5 (and other places in this work), one should ***not*** simply set $\varepsilon_1 = 2^{-\lambda}$ *at the current level of abstraction*. The λ here is λ_{prf} in higher-level applications. Using $\varepsilon_1 = 2^{-\lambda_{\mathrm{prf}}}$ is an overkill and ruins efficiency by introducing high dependency on S, the dominant term in λ_{prf}. Instead, the higher-level application needs to peek into the PRF abstraction and tune ε_1 to the desired/bearable *additive advantage loss*.

Remark 2 (parameters for Theorem 5). We analyze the case of $D_{n,1}$ (i.e., PRF output length is $m' = 1$). The case of general m' (by simple domain extension) follows easily. Construction 1 uses

$$\ell = \lceil n\cdot\lambda^{1+\gamma'}\rceil, \qquad m = 2^n, \qquad L = 100n, \qquad t = \lfloor \lambda^{1+\gamma'/2}\rfloor,$$

where Lemma 6 in Theorem 7 is tuned to use the same $\gamma' = \sqrt{1+\gamma} - 1$. The first condition $(0.99tL < \ell)$ is satisfied when $\lambda > 99^{2/\gamma'}$. For the second one, each term

$$\begin{aligned}\left(\frac{m}{T}\right)^T\left(\frac{0.99\cdot TL}{\ell}\right)^{0.01\cdot TL}\exp\left(1.99\cdot TL+T\right) &\le T^{-T}\cdot 2^{nT}\cdot\left(\frac{99T}{\lambda^{1+\gamma'}}\right)^{nT}\cdot\exp((199n+1)T)\\ (1\le T\le t\le \lambda^{1+\gamma'/2} \text{ and } n\ge 1) \qquad &\le 1\cdot 2^{nT}\cdot(99\lambda^{-\gamma'/2})^{nT}\cdot\exp(200nT)\\ &= (198\lambda^{-\gamma'/2}\exp(200))^{nT},\end{aligned}$$

and it is satisfied when $\lambda > (792\exp(200))^{2/\gamma'}$.[22] The size of $|\mathsf{pp}|$ is

$$n^{1+\gamma'}\lambda^{(1+\gamma')(1+\gamma'/2)}\,\mathrm{poly}(\log n, \log\lambda) < C\cdot(n\lambda)^{(1+\gamma')^2} = C\cdot(n\lambda)^{1+\gamma},$$

and the time of GraphEdge is

$$n^{1+\gamma'}\,\mathrm{poly}(\log n, \log\lambda) < n^{(1+\gamma')^2}\,\mathrm{poly}(\log\lambda) = n^{1+\gamma}\,\mathrm{poly}(\log\lambda).$$

[22] For the efficient version with $\varepsilon_2 \le \frac{1}{2}$, use $\lambda > (198(2/\varepsilon_2)^{1/n}\exp(200))^{2/\gamma'}$.

Proof (Theorem 7). The size of pp and the time complexity of GraphEdge are evident from Lemma 6 and Constructions 2 and 3.

We show that a desired $h(z)$ exists by the probabilistic method. Fix the prime p and let $a_0, \ldots, a_t \xleftarrow{\$} \mathbb{Z}_p$. The idea is that h is good enough as a t-wise independent hash function with codomain $\{0, 1, 2, \ldots, N\lfloor p/N \rfloor - 1\}$, which, with output taken remainder modulo N then converted to a hyperedge, yields a t-wise independent hash function with codomain being the set of hyperedges.

Let V_i be the (bad) event that $N\lfloor p/N \rfloor \le h(i) < p$, where $h(i)$ is interpreted as its smallest non-negative representative. Define $V = \bigcup_{i\in[m]} V_i$. Since $h(i)$ is uniformly random over $\mathbb{Z}_p$ for each i, by union bound, we have

$$\Pr[V] = \Pr\left[\bigcup_{i\in[m]} V_i\right] \le m \cdot \frac{p - N\lfloor p/N \rfloor}{p} < m \cdot \frac{N}{p} \le \frac{1}{2},$$

where the last inequality is by $p \ge 2Nm$.

Write $\alpha = 0.99$. For $T \in [t]$ and $1 \le i_1 < \cdots < i_T \le m$, let $Q_{i_1,\ldots,i_T}$ be the (bad) event that $S_{i_1}, \ldots, S_{i_T}$ fail to expand, namely, $|S_{i_1} \cup \cdots \cup S_{i_T}| < \alpha \cdot T \cdot L$. Conditioned on $\bigcap_{j\in[T]} \neg V_{i_j}$, the joint distribution of $(S_{i_1}, \ldots, S_{i_T})$ (regarded as a tuple of sets) is uniform over all T-tuples of size-L subsets of $[\ell]$. Therefore,

$$\begin{aligned}\Pr[Q_{i_1,\ldots,i_T} \cap \neg V] \le \Pr\left[Q_{i_1,\ldots,i_T} \cap \bigcap_{j\in[T]} \neg V_{i_j}\right] &\le \Pr\left[Q_{i_1,\ldots,i_T} \,\middle|\, \bigcap_{j\in[T]} \neg V_{i_j}\right] \\ &\le \binom{\ell}{\lfloor \alpha TL \rfloor} \cdot \binom{\lfloor \alpha TL \rfloor}{L}^T \Big/ \binom{\ell}{L}^T,\end{aligned}$$

where the right-hand side (over-)counts by choosing a superset for $S_{i_1} \cup \cdots \cup S_{i_T}$ then S_{i_j}'s from this superset. Let Q be the event that the graph is not $(t, 0.99)$-expanding, then Q is the union of all $Q_{i_1,\ldots,i_T}$'s. By union bound,

$$\Pr[Q \cap \neg V] \le \sum_{T=1}^{t} \sum_{1\le i_1<\cdots<i_T\le m} \Pr[Q_{i_1,\ldots,i_T} \cap \neg V] \le \sum_{T=1}^{t} \binom{m}{T} \cdot \binom{\ell}{\lfloor \alpha TL \rfloor} \cdot \binom{\lfloor \alpha TL \rfloor}{L}^T \Big/ \binom{\ell}{L}^T.$$

Applying $(a/b)^b \le \binom{a}{b} \le (a/b)^b \exp(b)$, we obtain

$$\begin{aligned}\Pr[Q \cap \neg V] &\le \sum_{T=1}^{t} \left(\frac{m}{T}\right)^T \left(\frac{\ell}{\lfloor \alpha TL \rfloor}\right)^{\lfloor \alpha TL \rfloor} \left(\frac{\lfloor \alpha TL \rfloor}{L}\right)^{L\cdot T} \left(\frac{\ell}{L}\right)^{(-L)\cdot T} \exp\left(T + \lfloor \alpha TL \rfloor + L \cdot T\right) \\ &= \sum_{T=1}^{t} \left(\frac{m}{T}\right)^T \left(\frac{\lfloor \alpha TL \rfloor}{\ell}\right)^{TL - \lfloor \alpha TL \rfloor} \exp\left(T + \lfloor \alpha TL \rfloor + TL\right) \\ &\le \sum_{T=1}^{t} \left(\frac{m}{T}\right)^T \left(\frac{\alpha TL}{\ell}\right)^{(1-\alpha)TL} \exp\left((1+\alpha)TL + T\right),\end{aligned}$$

where the last inequality uses $\frac{\alpha TL}{\ell} \le \frac{\alpha tL}{\ell} < 1$ to relax the exponent on $\frac{\alpha TL}{\ell}$. When the right-hand side is bounded by $\frac{1}{2}$ as required in the theorem statement, we have

$$\Pr[Q] \le \Pr[Q \cap \neg V] + \Pr[V] < 1,$$

i.e., not all choices of $h(z)$ are non-expanding. Therefore, a desired $h(z)$ exists. The case of Remark 1 follows similarly. □

4.3 Discussion on the Assumption

We now discuss cryptanalysis around Assumption 1. This is a highly simplified survey of a long line of works [Gol00,CM01,Fei02,MST03,FKO06,CEMT09,BQ09,ABW10,ABR12, BQ12,App12,App13,OW14] and in particular, the state-of-the-art work of [AL16]. The two main kinds of attacks on Goldreich's one-way function considered thereof are as follows.

Linear Attacks. Given a hypergraph $G = (S_1, \ldots, S_m)$ and a potential function output $(y_1, \ldots, y_m) \stackrel{?}{=} f_{P,G}(x)$, this attack considers $\mathbb{Z}_2$-linear functions, i.e., $y \mapsto \bigoplus_{i \in [T]} y_i$ for some set $T \subseteq [m]$. We say that $f_{P,G}$ is an ε-biased generator if

$$\left| \mathop{\mathbb{E}}_{x} \left[\bigoplus_{i \in T} y_i \right] - \frac{1}{2} \right| \leq \varepsilon \qquad \text{for all } \emptyset \neq T \subseteq [m].$$

As guided by prior works, one typically splits the analysis into two different cases.

The first case is *light tests*. These are for T of size at most the expansion threshold t (recall that G is $(t, 0.99)$-expanding). For such tests, if P has resiliency $L/2$ (the notion of *resiliency* is described later), it can be shown [AL16] that $\left| \mathbb{E}_x \left[\bigoplus_{i \in T} y_i \right] - \frac{1}{2} \right| = 0$.

For *heavy tests*, showing that ε is small for $|T| > t$ requires some work. It was proven in [AL16] that if the predicate has large *rational degree* (described later) $\Omega(L)$ and if G is $(t, 0.99)$-expanding for large enough t, then ε is subexponentially small. If L is constant, the work of [AL16] proves ε-biased property for the $\mathsf{XorMaj}_{L/2}$ predicate as long as the hypergraph is expanding. Unfortunately, the calculations break down for *heavy* tests when L is *super-constant* or even polynomial in the seed length ℓ. Nevertheless, security can still be conjectured against heavy tests. Such a conjecture was implicit in [AR16], which relied on super-constant locality to implement a PRF.

Polynomial Calculus. They are another important class of attacks, algebraic beyond linear. These include Gröbner-basis-style attacks and other models, which manipulate equations algebraically and refute/solve equations algebraically. The work of [AL16] modelled these equations in polynomial calculus proof system [CEI96] and showed lower bounds in this model. If the predicate P has a rational degree $L/4$ (satisfied by XorMaj) and if the hypergraph is $(t, 0.99)$-expanding, then Goldreich's PRG system cannot be refuted by polynomial calculus proof system of degree

$$D = t \cdot L \cdot \left(\frac{1}{16} - \frac{1}{100} \right) = \Omega(t \cdot L).$$

This roughly corresponds to $2^{\widetilde{O}(D)}$-time algebraic attacks.

Summary. All in all, our current understanding is that to design a good PRG, aside from the hypergraph G satisfying $(t, 0.99)$-expansion for large t, the L-local predicate P must satisfy the following properties.

- *Resilient*. We say that P has resiliency k if the XOR of P with any k of its variables is unbiased (e.g., $x \mapsto P(x) \oplus x_1$ should be unbiased). For $\mathsf{XorMaj}_{L/2}$, it has resiliency $L/2$.

- *High Rational Degree*. We say that P has rational degree e if e is the minimum such that $PQ = R$ (as functions) for two polynomials Q, R (not both zero) of degree at most e (i.e., P is equal to a fraction function whose numerator and denominator are both of degree at most e). The computation is done over $\mathbb{Z}_2[x_1, \ldots, x_L]$. For $\mathsf{XorMaj}_{L/2}$, it has rational degree $L/4$.

There are other attacks, too, such as sum-of-squares and convex-optimization-based methods. One can prove lower bounds for those algorithms (versions of $2^{\widetilde{O}(t)}$ time) as in [KMOW17,OW14,AOW15] based on resiliency and expansion condition alone. We refer the readers to [AL16] for a survey of other attacks.

5 Direct Applications

5.1 Resolving Unruh's Conjecture

Assuming the existence of any exponentially secure PRF, Theorem 3 resolves Unruh's conjecture that oracle-dependent auxiliary input does not give computational power. We state it in our notation.

Corollary 8 (Conjecture 15 of [Unr07b])**.** *Let the AI-ROM be of type* $\mathsf{dom} \to \{0,1\}^m$*. For all efficient adversary* A *and* $\mathcal{O}$*-advice* ai *of polynomial length, there exist an efficient simulator* $\mathcal{S}$ *and a* $\perp$*-advice* nu *of polynomial length such that for all efficient algorithm* B*, it holds that* $\mathbf{E}_{\mathrm{airom}} \approx \mathbf{E}_{\mathrm{sim}}$*. The experiments work as follows.*

- *In* $\mathbf{E}_{\mathrm{airom}}$*, sample a random* $\mathcal{O} : \mathsf{dom} \to \{0,1\}^m$ *then* $z \xleftarrow{\$} \mathsf{ai}(\lambda, \mathcal{O})$*. Let* $A^{\mathcal{O}}(1^\lambda, z)$ *interact with* $B(1^\lambda)$ *and output whatever* B *outputs.*
- *In* $\mathbf{E}_{\mathrm{sim}}$*, sample* $w \xleftarrow{\$} \mathsf{nu}(\lambda)$*. Let* $\mathcal{S}(1^\lambda, w)$ *interact with* $B(1^\lambda)$ *and output whatever* B *outputs.*

We note that Theorem 3 is stronger as the simulator has more structure than required by Unruh's conjecture. Namely, nu only depends on ai, and $\mathcal{S}$ is black-box in A.

5.2 Tighter Reductions from Standard-Model Assumptions

Consider a standard-model computational assumption such as factoring and DDH. Does the assumption also hold in AI-ROM? Auxiliary input provides at least the power of standard-model non-uniformity. Formally, it could provide more leverage since the adversary is given oracle access to the random function, correlated with the auxiliary input. Yet it is hard to conceive how such correlation can help with a task unrelated to the random function.

Presampling answers the above question in the positive for polynomial hardness, but its reduction loss is unsatisfactory. Notably, an auxiliary input of S bits could only be simulated by an advice of $\Omega(SQ)$ bits (the bit-fixing truth table), and consequently, in the circuit model, each query suffers a blow-up of $\Omega(SQ)$ in size.

Our methods provide tighter reductions, albeit under computational assumptions. We state them and compare them with presampling, first by computation model, then by type of task (searching or distinguishing). Note that for presampling, lazy sampling outside the bit-fixing table also incurs cost per query. We give it the benefit of discounting such cost — our method is still advantageous or comparable!

Assumptions as Indistinguishability. Let E be an experiment that is black-box and straight-line in the adversary. We require that the size/time of E itself be bounded by $2^{C_{\mathrm{model},E}(\lambda+1)}$. A search assumption can be stated as $E \approx 0$, and a decision assumption, $E \approx \frac{1}{2}$, where $0, \frac{1}{2}$ represent experiments simply outputting 1 with probability $0, \frac{1}{2}$. Let M', T', ε' be the parameters of indistinguishability.

We can lift E into E_{airom} that works with AI-ROM adversaries. Let the AI-ROM be of type $\mathsf{dom} \to \{0,1\}^m$. The experiment E_{airom} samples $w \xleftarrow{\$} \mathsf{nu}()$, a random $\mathcal{O} : \mathsf{dom} \to \{0,1\}^m$, then $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O})$, and runs and outputs $E(A^{\mathcal{O}}(w,z))$. Let the level of indistinguishability ($E_{\mathrm{airom}} \approx 0$ or $E_{\mathrm{airom}} \approx \frac{1}{2}$) be S, M, T, Q, ε.

We are interested in the relation between S, M, T, Q, ε and M', T', ε'. Throughout this subsection we assume $M', T', S, M, T, Q \le 2^\lambda$ and $\varepsilon', \varepsilon \ge 2^{-\lambda}$. We also assume a $(2^\lambda, 2^{-\lambda})$-secure PRF (D, Eval), i.e., $\varepsilon_{\mathrm{prf}}(\lambda) \le 2^{-\lambda}$.

Circuits. The bottleneck of reduction efficiency is Eval. Its circuit size is lower-bounded by roughly $\lambda_{\mathrm{prf}} = S + \Omega(\lambda)$ set in Theorem 3, which is the smallest reduction overhead we can hope for. The following notion is more relaxed, yet it is the most efficient one for which we know a candidate.

Definition 9 (quasilinear efficiency of PRF)**.** A PRF (Definition 5) is *quasilinearly efficient* if $|k| \le (\lambda + m) \log^{C_{\mathrm{ckt,prf}}}(\lambda + m + 2)$ and

- (for RAM) the RAM time of Eval,
- (for concrete circuits) the circuit size of Eval with k hardwired, or
- (for asymptotic circuits) the time to generate the circuit of Eval given k

is bounded by $(\lambda + m)(|x| + 1) \log^{C_{\mathrm{ckt,prf}}}(\lambda + m + |x| + 2)$, where $|x|$ is (for RAM) the input length or (for circuits) the maximum length of input that the (generated) circuit must handle.

A candidate of quasilinearly efficient and exponentially secure PRF is GGM tree PRF based on an exponentially secure PRG from ring LWE.

Corollary 9 (from standard-model to AI-ROM, circuit model)**.** *Assuming the existence of quasilinearly efficient and $(2^\lambda, 2^{-\lambda})$-secure PRF, suppose $E \overset{\varepsilon'}{\approx}_{M',T'} 0$ holds in the standard model, then $E_{\mathrm{airom}} \overset{\varepsilon}{\approx}_{S,M,T,Q} 0$ holds in AI-ROM of type $\mathsf{dom} \to \{0,1\}^m$ if $\varepsilon \ge \varepsilon' + 2^{-\lambda}$ and*

$$\begin{aligned}
\text{(concrete)} \quad & M' \ge M + Q \cdot (S + \lambda + m)(\min\{M, n\} + 1)(\lambda + 2)^{C_{\mathrm{ckt},E,\mathrm{airom,prf}}}, \\
\text{(asymptotic)} \quad & M' \ge M + (S + \lambda + m)(\lambda + 2)^{C_{\mathrm{ckt},E,\mathrm{airom,prf}}}, \\
& T' \ge C_{\mathrm{ckt},E,\mathrm{airom,prf}} \cdot (M + T), \\
& \qquad + (Q + 1) \cdot (S + \lambda + m)(\min\{M, n\} + 1)(\lambda + 2)^{C_{\mathrm{ckt},E,\mathrm{airom,prf}}}.
\end{aligned}$$

where n is consistent with $\mathsf{dom} = \{0,1\}^n$ or $\mathsf{dom} = \{0,1\}^{\le n}$ or $n = +\infty$ if $\mathsf{dom} = \{0,1\}^*$, and $M = T$ and $M' = T'$ for the concrete case. Suppose $E \approx 0$ has λ' strong bits of security (Definition 2) and $\lambda' \le \lambda$, then $E_{\mathrm{airom}} \approx 0$ has

- $(\lambda'/3 - C_{\mathrm{ckt},E,\mathrm{airom,prf}} \cdot \log(\lambda + 2))$ *strong bits of security regardless of n, and*
- $(\lambda'/2 - C_{\mathrm{ckt},E,\mathrm{airom,prf}} \cdot \log(\lambda + 2))$ *strong bits of security if $n \le \mathrm{poly}(\lambda)$.*

The same applies to lifting $E \approx \frac{1}{2}$ to $E_{\mathrm{airom}} \approx \frac{1}{2}$.

Corollary 9 follows from Theorems 2 and 3. Bounds $S, M, T, |x| \le 2^\lambda$ and $m \le \mathrm{poly}(\lambda)$ are applied to simplify some terms in M', T'.

Comparison with Presampling. We provide a back-of-envelope calculation of concrete reductions using presampling.[23] Refer to Theorems 1 (Lemma 4) and 2 in [CDGS18]. The number of presampled points is P. Each presampled point takes roughly $(\min\{M, n\} + m)$ bits to describe the input/output values.

For search assumption ($E \approx 0$), we set $P = (S + \lambda)Q$ for

$$\varepsilon = 2\varepsilon' + 2^{-\lambda}, \qquad M' \sim M + Q \cdot (S + \lambda)Q(\min\{M, n\} + m).$$

Suppose $E \approx 0$ has λ' strong bits of security, then presampling proves $E_{\text{airom}} \approx 0$ for

- roughly $\lambda'/4$ strong bits of security regardless of n, and
- roughly $\lambda'/3$ strong bits of security if $n \le \text{poly}(\lambda)$.

For decision assumption ($E \approx \frac{1}{2}$), we set $P = (S + \lambda)Q/\varepsilon''$ for

$$\varepsilon' = \varepsilon + 2^{-\lambda} + \varepsilon'', \qquad M' \sim M + Q \cdot (S + \lambda)Q(\min\{M, n\} + m)/\varepsilon''.$$

Suppose $E \approx \frac{1}{2}$ has λ' strong bits of security, then presampling proves $E_{\text{airom}} \approx \frac{1}{2}$ for

- roughly $\lambda'/5$ strong bits of security regardless of n, and
- roughly $\lambda'/4$ strong bits of security if $n \le \text{poly}(\lambda)$.

In summary, regardless of the nature of assumption or the domain of AI-ROM, our method saves a constant factor in the loss of bit level of security.

RAM. We rely on PRF with fast evaluation. It currently only works for small domain (input length is *a priori* polynomially bounded).

Corollary 10 (from standard-model to AI-ROM, RAM model)**.** *Assuming the existence of $(2^\lambda, 2^{-\lambda})$-secure PRF with fast evaluation and fixing a positive constant $\gamma > 0$, suppose $E \overset{\varepsilon'}{\approx}_{M',T'} 0$ holds in the standard model, then $E_{\text{airom}} \overset{\varepsilon}{\approx}_{S,M,T,Q} 0$ holds in AI-ROM of type either $\{0,1\}^n \to \{0,1\}^m$ or $\{0,1\}^{\le n} \to \{0,1\}^m$ for $n \le \text{poly}(\lambda)$, if $\varepsilon \ge \varepsilon' + 2^{-\lambda}$ and*[24]

$$M' \ge M + C_{\text{ram},E,\text{airom},\text{prf},\gamma} \cdot (n + 1 + \log m)^{1+\gamma}(S + \lambda + 1)^{1+\gamma},$$
$$T' \ge T + (Q + 1) \cdot m(n + 1 + \log m)^{1+\gamma}(\lambda + 2)^{C_{\text{ram},E,\text{airom},\text{prf},\gamma}}.$$

Suppose $E \approx 0$ has λ' strong bits of security (Definition 2) and $\lambda' \le \lambda$, then $E_{\text{airom}} \approx 0$ has $(\lambda/(1+\gamma) - C_{\text{ram},E,\text{airom},\text{prf},\gamma} \cdot \log(\lambda + 2))$ strong bits of security. The same applies to lifting $E \approx \frac{1}{2}$ to $E_{\text{airom}} \approx \frac{1}{2}$.

Comparison with Presampling. We provide a back-of-envelope calculation of concrete reductions using presampling. Refer to Theorems 1 (Lemma 4) and 2 in [CDGS18]. The number of presampled points is P.

In the RAM model, the query time can be reduced by storing a perfect hash table of the presampled points in the advice. The perfect hash table takes $C(P + 1)(n + m)$ bits, and hashing an input takes $(n + 2)^{C_{\text{ram}}}$ time.

[23] We do not claim that presampling works with infinite-domain oracles. Instead, we avoid this issue by truncating the oracle to the relevant portion (input length at most M).

[24] The bounds simplify to the same form in both concrete and asymptotic scenarios — constants depend on the scenario.

For search assumption ($E \approx 0$), we set $P = (S+\lambda)Q$ for

$$\varepsilon = 2\varepsilon' + 2^{-\lambda}, \qquad M' \sim M + (S+\lambda)Q(n+m), \qquad T' \sim T + Q \cdot (n+2)^{C_{\mathrm{ram}}}.$$

Suppose $E \approx 0$ has λ' strong bits of security, then presampling proves $E_{\mathrm{airom}} \approx 0$ for roughly $\lambda'/2$ strong bits of security.

For decision assumption ($E \approx \frac{1}{2}$), we set $P = (S+\lambda)Q/\varepsilon''$ for

$$\varepsilon' = \varepsilon + 2^{-\lambda} + \varepsilon'', \qquad M' \sim M + (S+\lambda)Q(n+m)/\varepsilon'', \qquad T' \sim T + Q \cdot (n+2)^{C_{\mathrm{ram}}}.$$

Suppose $E \approx \frac{1}{2}$ has λ' strong bits of security, then presampling proves $E_{\mathrm{airom}} \approx \frac{1}{2}$ for roughly $\lambda'/3$ strong bits of security.

In summary, for small-domain AI-ROM, regardless of the nature of assumption, our method can be made arbitrarily tight, while presampling always loses at least half of the bit level.

Degenerate Cases and Other Remarks. Under our formulation of bit level of security, when the truth table of AI-ROM is below roughly $2^{\lambda'/2}$ bits (e.g., $\{0,1\}^{\lambda'/2} \to \{0,1\}$), neither simulation by PRF nor presampling should be used for circuits. For RAM, the threshold size of truth table can be arbitrarily close to $2^{\lambda'}$ bits. In those cases, the brute-force reduction of hardwiring the entire truth table and the auxiliary information is better.

However, an oracle with such a small truth table often leads to even less bit level of security for primitives constructed from the oracle. In a cryptosystem built from multiple primitives, a mismatch among their security levels often indicates wasteful choices of security parameters. The situation is further complicated if those components suffer different security losses in reductions.

The restriction $\lambda' \le \lambda$ is only there because we have set the statistical error to $2^{-\lambda}$. It can be adjusted for $\lambda' > \lambda$.

6 Combining Presampling and Our Simulation

As discussed in Section 3.1, it is difficult to use our simulation to analyze security reliant on the AI-ROM, since the whole oracle is replaced by a PRF, which has little entropy. We present a simple trick for combining the powers of presampling and our simulation. Oftentimes, the security only depends on the behavior of the random oracle at points *unpredictable* to the adversary but known to the security experiment. We can use presampling to argue that the portion of the oracle relevant to security is essentially independent of the auxiliary input. Let E be the security experiment, A the adversary, $\mathcal{O}, \mathcal{O}_{\mathrm{presamp}}, \mathcal{O}_{\mathrm{ro}} : \mathsf{dom} \to \{0,1\}^m$ three independent random functions, ai the auxiliary input, and (D, Eval) the PRF. The hybrids typically go as follows.

1. E interacts with $A(z)$, where $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O})$ and E, A have access to $\mathcal{O}$.

2. E interacts with $A(z)$, where $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O}_{\mathrm{presamp}})$ and E, A have access to $\mathcal{O}[\mathcal{T}]$. Here, $\mathcal{T} \xleftarrow{\$} \mathsf{ai}_{\mathrm{presamp}}(\mathcal{O}_{\mathrm{presamp}})$ is the presampled bit-fixing table per Lemma 4 with an appropriate choice of size P.

3. E interacts with $A(z)$, $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O}_{\mathrm{presamp}})$ and E, A have access to

$$\mathcal{O}'(x) = \begin{cases} \mathcal{O}_{\mathrm{ro}}(x), & \text{if } x \text{ is "relevant";} \\ \mathcal{O}[\mathcal{T}](x), & \text{otherwise.} \end{cases}$$

   Again, $\mathcal{T} \xleftarrow{\$} \mathsf{ai}_{\mathrm{presamp}}(\mathcal{O}_{\mathrm{presamp}})$ is the presampling. The notion of "relevant" is specific to the application. In addition, whether a point is relevant must be decided by the first time it is queried without consulting $\mathcal{T}$. This transition holds as long as the relevant points hit $\mathcal{T}$ with only negligible probability.

4. E interacts with $A(z)$, where $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O})$ and E, A have access to

$$\mathcal{O}'(x) = \begin{cases} \mathcal{O}_{\mathrm{ro}}(x), & \text{if } x \text{ is "relevant";} \\ \mathcal{O}(x), & \text{otherwise.} \end{cases}$$

   This is just undoing presampling.

5. E interacts with $A(z)$, where $z \xleftarrow{\$} \widetilde{\mathsf{ai}}(k)$ for $k \xleftarrow{\$} D_{\mathsf{dom},m}$ and E, A have access to

$$\mathcal{O}'(x) = \begin{cases} \mathcal{O}_{\mathrm{ro}}(x), & \text{if } x \text{ is "relevant";} \\ \mathsf{Eval}(k, x), & \text{otherwise.} \end{cases}$$

   This is our AI-ROM simulation (Theorems 2 and 3).

In the rest of this section, we show how to encapsulate the presampling–rerouting–undoing trick into a notion of salting scheme and demonstrate its applications.

6.1 Salting Defeats Auxiliary Information

In [CDGS18], it is shown using presampling that salting generically defeats auxiliary information. However, their reduction is loose due to presampling with large table size P. The reduction can be decomposed into two phases — first an argument that the "salted portion" is independent of the auxiliary information, then a reduction to security in the ROM. The second phase is computational and suffers the tightness issue in simulating the "non-salted portion". Another downside is that due to salting, the oracle domain shrinks.

We encapsulate the first phase into a notion of *salting scheme*, whose security is roughly indifferentiability, possibly with domain extension. It sets a clear boundary between the information-theoretic technique of presampling and our computational simulation with PRF. We remark that *defining* indifferentiability for auxiliary-input models is challenging by itself [DFMT20]. Thanks to the presence of salting, our definition circumvents the difficulty and has provably secure instantiations.

Definition 10 (salting scheme)**.** Let $\mathsf{dom}(\lambda) \to \{0,1\}^{m(\lambda)}$ and $\mathsf{dom}_{\mathrm{ro}}(\lambda) \to \{0,1\}^{m_{\mathrm{ro}}(\lambda)}$ be the types of AI-ROM and ROM. A *salting scheme* consists of two efficient algorithms.

- $\mathsf{HashGen}(1^\lambda)$ outputs a hash key hk.
- $\mathsf{Hash}^{\mathcal{O}}(\mathsf{hk}, x_{\mathrm{ro}})$ is deterministic and given oracle access to the AI-ROM. It takes as input hk and $x_{\mathrm{ro}} \in \mathsf{dom}_{\mathrm{ro}}(\lambda)$, and outputs $y_{\mathrm{ro}} \in \{0,1\}^{m_{\mathrm{ro}}(\lambda)}$.

Definition 11 (security of salting scheme)**.** A salting scheme is *secure* if there exists an efficient stateful simulator $\mathcal{S}$ such that $E_{\text{real}} \approx E_{\text{sim}}$, where the experiments with $(\mathsf{nu}, \mathsf{ai}, A)$ work as follows.

- In both, sample $w \xleftarrow{\$} \mathsf{nu}(\lambda)$, random $\mathcal{O} : \mathsf{dom}(\lambda) \to \{0,1\}^{m(\lambda)}$, and $z \xleftarrow{\$} \mathsf{ai}(\lambda, \mathcal{O})$.
- In E_{real}, sample $\mathsf{hk} \xleftarrow{\$} \mathsf{HashGen}(1^\lambda)$. Run and output $A^{\mathcal{O},\mathsf{Hash}(\mathsf{hk},\cdot)}(1^\lambda, w, z, \mathsf{hk})$.
- In E_{sim}, sample random $\mathcal{O}_{\text{ro}} : \mathsf{dom}_{\text{ro}}(\lambda) \to \{0,1\}^{m_{\text{ro}}(\lambda)}$ and $\mathsf{hk} \xleftarrow{\$} \mathcal{S}(1^\lambda)$. Run and output $A^{\mathcal{S}^{\mathcal{O},\mathcal{O}_{\text{ro}}},\mathcal{O}_{\text{ro}}}(1^\lambda, w, z, \mathsf{hk})$.

Security Properties as Indistinguishability. An experiment that is black-box and straight-line in the adversary corresponds to "application" in [CDGS18; Definition 3], which captures either unpredictability or indistinguishability (together, a security property) of some primitive implicit in the description of the experiment — for example, an experiment in the ROM might play the IND-CPA security game of RSA-OAEP with the adversary, hence captures the IND-CPA security of RSA-OAEP. Recall that in Section 5.2, we explained how to formulate search/decision assumptions as indistinguishability. The same applies here — unpredictability is modeled as $E \approx 0$ and indistinguishability is modeled as $E \approx \frac{1}{2}$. We show that security properties in the ROM can be lifted to AI-ROM with the help of a salting scheme.

Consider ROM of type $\mathsf{dom}_{\text{ro}} \to \{0,1\}^{m_{\text{ro}}}$ and let $E^{\mathcal{O}_{\text{ro}}}$ be an experiment that is black-box and straight-line in the adversary, which is given the same oracle given to E. Let $M'_A, T'_A, Q'_A, \varepsilon'$ be the parameters of $E \approx 0$ or $E \approx \frac{1}{2}$, and T'_E, Q'_E the complexities of E itself (the experiment is assumed to be uniform).

Let $\mathsf{dom} \to \{0,1\}^m$ be the type of AI-ROM and $(\mathsf{HashGen}, \mathsf{Hash})$ a secure salting scheme. Define $E^{\mathcal{O}}_{\text{airom}}$ working with $(\mathsf{nu}, \mathsf{ai}, A^{\mathcal{O}})$ as follows.

- Sample $w \xleftarrow{\$} \mathsf{nu}()$ and $z \xleftarrow{\$} \mathsf{ai}(\mathcal{O})$. Run $\mathsf{hk} \xleftarrow{\$} \mathsf{HashGen}()$.
- Run and output $E^{\mathsf{Hash}(\mathsf{hk},\cdot)}(A^{\mathcal{O}}(w, z, \mathsf{hk}))$.

Let the parameters of $E_{\text{airom}} \approx 0$ or $E_{\text{airom}} \approx \frac{1}{2}$ be S, M, T, Q, ε.

Denote by $\mathcal{S}$ the simulator of the salting scheme and $S_{\text{salt}}, M_{\text{salt}}, T_{\text{salt}}, Q_{\text{salt}}, \varepsilon_{\text{salt}}$ the parameters of its security. Let (D, Eval) be a $(2^\lambda, \varepsilon_{\text{prf}})$-secure PRF with key length $|k|$. Let $T_{\mathcal{S}}$ be the per-query time of $\mathcal{S}^{\mathsf{Eval}(k,\cdot),\mathcal{O}_{\text{ro}}}$. As usual, all integer-valued parameters are assumed to be bounded by 2^λ.

Theorem 11 (¶)**.** *In the RAM model, the asymptotic versions of $E_{\text{airom}} \approx 0$ or $E_{\text{airom}} \approx \frac{1}{2}$ can be derived if $\varepsilon \geq \varepsilon_{\text{salt}} + 2^{-\lambda} + \varepsilon_{\text{prf}}(\lambda_{\text{prf}}) + \varepsilon'$, and (for salting),*

$$S_{\text{salt}} \geq S, \qquad M_{\text{salt}} \geq M, \qquad T_{\text{salt}} \geq T + T'_E, \qquad Q_{\text{salt}} \geq Q + Q'_E,$$

and (for simulation using PRF) $\lambda_{\text{prf}} \geq S + C_{\text{ram}}(\lambda + 1)$, and (for reduction to E),

$$M'_A \geq M + S + |k| + C(\lambda + 1), \qquad T'_A \geq T + Q \cdot T_{\mathcal{S}} + C_{\text{ram}}(\lambda + 1), \qquad Q'_A \geq Q.$$

Compared with [CDGS18], we incur less loss in reduction to the (computational) security in ROM, thanks to the efficient simulation of AI-ROM.

Proof (Theorem 11). Let $\mathcal{O} : \mathsf{dom} \to \{0,1\}^m$ and $\mathcal{O}_{\text{ro}} : \mathsf{dom}_{\text{ro}} \to \{0,1\}^{m_{\text{ro}}}$ be uniformly random. We go through the following hybrids.

- H_0 is E_{airom}, i.e., run and output

$$E^{\mathsf{Hash}(\mathsf{hk},\cdot)}(A^{\mathcal{O}}(w,z,\mathsf{hk})) \quad \text{with} \quad w \xleftarrow{\$} \mathsf{nu}(), \quad z \xleftarrow{\$} \mathsf{ai}(\mathcal{O}), \quad \mathsf{hk} \xleftarrow{\$} \mathsf{HashGen}().$$

- In H_1, we replace $\mathsf{Hash}(\mathsf{hk},\cdot)$ by $\mathcal{O}_{\text{ro}}$. Run and output

$$E^{\mathcal{O}_{\text{ro}}}(A^{\mathcal{S}^{\mathcal{O},\mathcal{O}_{\text{ro}}}}(w,z,\mathsf{hk})) \quad \text{with} \quad w \xleftarrow{\$} \mathsf{nu}(), \quad z \xleftarrow{\$} \mathsf{ai}(\mathcal{O}), \quad \mathsf{hk} \xleftarrow{\$} \mathcal{S}().$$

  $\mathsf{H}_0 \approx \mathsf{H}_1$ follows from the conditions for salting and incurs advantage error $\varepsilon_{\text{salt}}$.

- In H_2, we simulate $(\mathcal{O}, z)$ using PRF. Sample $k \xleftarrow{\$} D_{\lambda_{\text{prf}},\mathsf{dom},m}$ and run and output

$$E^{\mathcal{O}_{\text{ro}}}(A^{\mathcal{S}^{\mathsf{Eval}(k,\cdot),\mathcal{O}_{\text{ro}}}}(w,z,\mathsf{hk})) \quad \text{with} \quad w \xleftarrow{\$} \mathsf{nu}(), \quad z \xleftarrow{\$} \widetilde{\mathsf{ai}}(k), \quad \mathsf{hk} \xleftarrow{\$} \mathcal{S}().$$

  $\mathsf{H}_1 \approx \mathsf{H}_2$ follows from Theorem 3 and incurs advantage error $(2^{-\lambda} + \varepsilon_{\text{prf}}(\lambda_{\text{prf}}))$.

By our choice of parameters, the advantage of $\mathsf{H}_2 \approx 0$ or $\mathsf{H}_2 \approx \frac{1}{2}$ is bounded by ε'. The proof completes by hybrid argument. □

Example Instantiations. The simplest instantiation of a salting scheme in AI-ROM of type $\{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$ that creates a random oracle of type $\{0,1\}^{\lambda} \to \{0,1\}^{\lambda}$ is as follows.

- $\mathsf{HashGen}()$ samples and outputs $\mathsf{hk} \xleftarrow{\$} \{0,1\}^{\lambda}$.
- $\mathsf{Hash}^{\mathcal{O}}(\mathsf{hk}, x)$ outputs $\mathcal{O}(\mathsf{hk} \| x)$.

Its security proof follows the presampling–rerouting–undoing blueprint explained at the beginning of this section. The simulator simply chooses to direct query to $\mathcal{O}$ or $\mathcal{O}_{\text{ro}}$ depending on whether the prefix matches hk. Therefore, there is essentially no simulator overhead during reduction to ROM security, implying an arbitrarily tight reduction using PRF with fast evaluation.

Any salting scheme can be composed with an indifferentiable domain extension scheme as needed.

6.2 Non-Interactive Zero-Knowledge in AI-ROM

We show how to construct NIZK in AI-ROM starting from NIZK in ROM with statistical soundness and computational zero-knowledge against polynomial-time adversaries.

Definition 12 (NIZK in ROM or AI-ROM)**.** Let $L \subseteq \{0,1\}^*$ and $\mathcal{O}$ be either AI-ROM or ROM. A *non-interactive zero-knowledge* (NIZK) proof for L consists of two algorithms.

- $P^{\mathcal{O}}(1^{\lambda}, x)$ takes $x \in L$ as input and outputs a proof $\pi \in \{0,1\}^*$. The algorithm P does not have to be efficient, but π must be polynomially long in $|x|, \lambda$.
- $V^{\mathcal{O}}(1^{\lambda}, x, \pi)$ takes x, π as input and outputs a single bit. It is efficient.

The scheme must be *complete,* i.e., for all $\lambda \in \mathbb{N}$ and $x \in L$, it holds that

$$\Pr[V^{\mathcal{O}}(1^{\lambda}, x, P^{\mathcal{O}}(1^{\lambda}, x))] = 1.$$

The NIZK is *statistically sound* if for all $\lambda \in \mathbb{N}$ and $x \notin L$ and inefficient algorithm $\widetilde{P}^{\mathcal{O}}$, it holds that

$$\Pr[V^{\mathcal{O}}(1^{\lambda}, x, \widetilde{P}^{\mathcal{O}}(x)) = 1] \leq 2^{-\lambda}.$$

Definition 13 (zero-knowledge in ROM)**.** A NIZK for L in ROM is zero-knowledge if there exists an efficient stateful simulator $\mathcal{S}$ such that $E_{\text{real}} \approx E_{\text{sim}}$, where the experiments with adversary A work as follows.

- Sample $w \xleftarrow{\$} \mathsf{nu}(\lambda)$.
- In E_{real}, launch $A^{\mathcal{O}_{\text{ro}}}(w)$ and receive from it $x \in L$. Run $\pi \xleftarrow{\$} P^{\mathcal{O}_{\text{ro}}}(1^\lambda, x)$ and send π to A.
- In E_{sim}, launch $A^{\mathcal{S}}(w)$ and receive from it $x \in L$. Run $\pi \xleftarrow{\$} \mathcal{S}(x)$ and send π to A.
- A continues with its oracle and outputs some string, which is the output of the experiment.

$E_{\text{real}} \approx E_{\text{sim}}$ is a standard-model indistinguishability, which means the distinguisher taking the output of one of the experiments has no oracle access.

NIZK in ROM can be obtained by taking any NIZK in the common *random* string model, such as [FLS90,GOS06], and use the output of the random oracle at a few fixed points as the CRS.

Definition 14 (zero-knowledge in AI-ROM)**.** A NIZK for L in AI-ROM is zero-knowledge if for all $(\mathsf{nu}, \mathsf{ai}, A)$, there exists an efficient simulator $(\widetilde{\mathsf{nu}}, \mathcal{S})$ such that $E_{\text{real}} \approx E_{\text{sim}}$,[25] where the experiments work as follows.

- In E_{real}, sample $w \xleftarrow{\$} \mathsf{nu}(\lambda)$, random $\mathcal{O}$, $z \xleftarrow{\$} \mathsf{ai}(\lambda, \mathcal{O})$. Launch $A^{\mathcal{O}}(1^\lambda, w, z)$ and receive from it $x \in L$. Run $\pi \xleftarrow{\$} P^{\mathcal{O}}(1^\lambda, x)$ and send π to A. The adversary A continues to run with oracle access to $\mathcal{O}$, and eventually outputs some string, which is the output of the experiment.
- In E_{sim}, sample $\widetilde{w} \xleftarrow{\$} \widetilde{\mathsf{nu}}(\lambda)$. Run and output $\mathcal{S}(1^\lambda, \widetilde{w})$.

$E_{\text{real}} \approx E_{\text{sim}}$ is a standard-model indistinguishability, which means the distinguisher taking the output of one of the experiments has no oracle nor oracle-dependent auxiliary input.

We remark that Definition 14 insists that the simulator be standard-model efficient, without the help of AI-ROM.

NIZK in AI-ROM from NIZK in ROM. Let $(P_{\text{ro}}, V_{\text{ro}})$ a NIZK for L in ROM $\{0,1\}^\lambda \to \{0,1\}^\lambda$. The following is a NIZK for L in AI-ROM $\{0,1\}^{3\lambda} \to \{0,1\}^{2\lambda}$.

- $P^{\mathcal{O}}(1^\lambda, x)$ samples $\mathsf{hk} \xleftarrow{\$} \{0,1\}^\lambda$, runs $\pi_{\text{ro}} \xleftarrow{\$} P_{\text{ro}}^{\mathcal{O}(\mathsf{hk}\|\cdot)}(1^{2\lambda}, x)$, and outputs $(\mathsf{hk}, \pi_{\text{ro}})$.
- $V^{\mathcal{O}}(1^\lambda, x, \mathsf{hk}, \pi_{\text{ro}})$ runs $V^{\mathcal{O}(\mathsf{hk}\|\cdot)}(1^{2\lambda}, x, \pi_{\text{ro}})$.

Soundness follows by a union bound. The simulator works as follows.

- Let $\widetilde{\mathsf{nu}}$ be the standard-model advice promised by Theorem 3 and $\mathcal{S}_{\text{ro}}$ the ROM-NIZK simulator for $(P_{\text{ro}}, V_{\text{ro}})$.

[25] Recall that $\mathsf{nu}, \mathsf{ai}, \widetilde{\mathsf{nu}}$ can be computationally unbounded but their output sizes are polynomially bounded, and $A, \mathcal{S}$ are polynomial-time.

- On input $\widetilde{w} = (w, z, k)$, sample $\mathsf{hk} \xleftarrow{\$} \{0,1\}^\lambda$, launch $\mathcal{S}_{\mathrm{ro}}$, and run $A^{\mathcal{O}}(w, z)$ with

$$\mathcal{O}(\mathsf{hk}' \| y') = \begin{cases} \mathcal{S}_{\mathrm{ro}}(y'), & \mathsf{hk} = \mathsf{hk}'; \\ \mathsf{Eval}(k, \mathsf{hk}' \| y'), & \text{otherwise.} \end{cases}$$

- When A chooses x, send it to $\mathcal{S}_{\mathrm{ro}}$ and receive π_{ro}. Return $(\mathsf{hk}, \pi_{\mathrm{ro}})$ to $\mathcal{A}$.
- Continue A with its oracle and output whatever A outputs.

Acknowledgments. Yevgeniy Dodis was supported by NSF CNS-2055578 and gifts from JP Morgan, Protocol Labs, Stellar, and Algorand Foundation. Aayush Jain was supported by gifts from CyLab of CMU and Google. Huijia Lin and Ji Luo were supported by NSF CNS-1936825 (CAREER), NSF CNS-2026774, a JP Morgan AI Research Award, a Cisco Research Award, and a Simons Collaboration on the Theory of Algorithmic Fairness. Daniel Wichs was supported by NSF CNS-1750795, NSF CNS-2055510, and the JP Morgan Faculty Research Award. The authors thank the anonymous reviewers of FOCS 2024 for their constructive feedback.

References

[ABR12] Benny Applebaum, Andrej Bogdanov, and Alon Rosen. A dichotomy for local small-bias generators. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 600–617. Springer, Berlin, Heidelberg, March 2012.

[ABW10] Benny Applebaum, Boaz Barak, and Avi Wigderson. Public-key cryptography from different assumptions. In Leonard J. Schulman, editor, *42nd ACM STOC*, pages 171–180. ACM Press, June 2010.

[AGHP90] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. In *31st FOCS*, pages 544–553. IEEE Computer Society Press, October 1990.

[AHK12] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: A meta-algorithm and applications. *Theory of Computing*, 8(6):121–164, 2012.

[AL16] Benny Applebaum and Shachar Lovett. Algebraic attacks against random local functions and their countermeasures. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 1087–1100. ACM Press, June 2016.

[AOW15] Sarah R. Allen, Ryan O'Donnell, and David Witmer. How to refute a random CSP. In Venkatesan Guruswami, editor, *56th FOCS*, pages 689–708. IEEE Computer Society Press, October 2015.

[App12] Benny Applebaum. Pseudorandom generators with long stretch and low locality from random local one-way functions. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 805–816. ACM Press, May 2012.

[App13] Benny Applebaum. Pseudorandom generators with long stretch and low locality from random local one-way functions. *SIAM Journal on Computing*, 42(5):2008–2037, 2013.

[AR16] Benny Applebaum and Pavel Raykov. Fast pseudorandom functions based on expander graphs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 27–56. Springer, Berlin, Heidelberg, October / November 2016.

[BQ09] Andrej Bogdanov and Youming Qiao. On the security of Goldreich's one-way function. In Irit Dinur, Klaus Jansen, Joseph Naor, and José D. P. Rolim, editors, *APPROX/RANDOM 2009*, volume 5687 of *Lecture Notes in Computer Science*, pages 392–405. Springer, 2009.

[BQ12] Andrej Bogdanov and Youming Qiao. On the security of Goldreich's one-way function. *Computational Complexity*, 21(1):83–127, 2012.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

[CCL18] Yi-Hsiu Chen, Kai-Min Chung, and Jyun-Jie Liao. On the complexity of simulating auxiliary input. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 371–390. Springer, Cham, April / May 2018.

[CDGS18] Sandro Coretti, Yevgeniy Dodis, Siyao Guo, and John P. Steinberger. Random oracles and non-uniformity. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 227–258. Springer, Cham, April / May 2018.

[CEI96] Matthew Clegg, Jeff Edmonds, and Russell Impagliazzo. Using the Groebner basis algorithm to find proofs of unsatisfiability. In *28th ACM STOC*, pages 174–183. ACM Press, May 1996.

[CEMT09] James Cook, Omid Etesami, Rachel Miller, and Luca Trevisan. Goldreich's one-way function candidate and myopic backtracking algorithms. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 521–538. Springer, Berlin, Heidelberg, March 2009.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998.

[CM01] Mary Cryan and Peter Bro Miltersen. On pseudorandom generators in NC^0. In Jiri Sgall, Ales Pultr, and Petr Kolman, editors, *MFCS 2001*, volume 2136 of *Lecture Notes in Computer Science*, pages 272–284. Springer, 2001.

[DFMT20] Yevgeniy Dodis, Pooya Farshim, Sogol Mazaheri, and Stefano Tessaro. Towards defeating backdoored random oracles: Indifferentiability with bounded adaptivity. In Rafael Pass and Krzysztof Pietrzak, editors,

TCC 2020, Part III, volume 12552 of *LNCS*, pages 241–273. Springer, Cham, November 2020.

[DGK17] Yevgeniy Dodis, Siyao Guo, and Jonathan Katz. Fixing cracks in the concrete: Random oracles with auxiliary input, revisited. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 473–495. Springer, Cham, April / May 2017.

[DTT10] Anindya De, Luca Trevisan, and Madhur Tulsiani. Time space tradeoffs for attacks against one-way functions and PRGs. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 649–665. Springer, Berlin, Heidelberg, August 2010.

[Fei02] Uriel Feige. Relations between average case complexity and approximation complexity. In *34th ACM STOC*, pages 534–543. ACM Press, May 2002.

[FKO06] Uriel Feige, Jeong Han Kim, and Eran Ofek. Witnesses for non-satisfiability of dense random 3CNF formulas. In *47th FOCS*, pages 497–508. IEEE Computer Society Press, October 2006.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *31st FOCS*, pages 308–317. IEEE Computer Society Press, October 1990.

[FN91] Amos Fiat and Moni Naor. Rigorous time/space tradeoffs for inverting functions. In *23rd ACM STOC*, pages 534–541. ACM Press, May 1991.

[GGKL21] Nick Gravin, Siyao Guo, Tsz Chiu Kwok, and Pinyan Lu. Concentration bounds for almost k-wise independence with applications to non-uniform security. In Dániel Marx, editor, *32nd SODA*, pages 2404–2423. ACM-SIAM, January 2021.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *44th FOCS*, pages 102–115. IEEE Computer Society Press, October 2003.

[GLLZ21] Siyao Guo, Qian Li, Qipeng Liu, and Jiapeng Zhang. Unifying presampling via concentration bounds. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 177–208. Springer, Cham, November 2021.

[Gol00] Oded Goldreich. Candidate one-way functions based on expander graphs, 2000.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for NIZK. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 97–111. Springer, Berlin, Heidelberg, August 2006.

[GT00] Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *41st FOCS*, pages 305–313. IEEE Computer Society Press, November 2000.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011.

[Hel80] Martin Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.

[JP14] Dimitar Jetchev and Krzysztof Pietrzak. How to fake auxiliary input. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 566–590. Springer, Berlin, Heidelberg, February 2014.

[KMOW17] Pravesh K. Kothari, Ryuhei Mori, Ryan O'Donnell, and David Witmer. Sum of squares lower bounds for refuting any CSP. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th ACM STOC*, pages 132–145. ACM Press, June 2017.

[KU11] Kiran S. Kedlaya and Christopher Umans. Fast polynomial factorization and modular composition. *SIAM Journal on Computing*, 40(6):1767–1802, 2011.

[Lee24] Keewoo Lee. Bit security as cost to demonstrate advantage. *CiC*, 1(1):1, 2024.

[LMW23] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. In Barna Saha and Rocco A. Servedio, editors, *55th ACM STOC*, pages 595–608. ACM Press, June 2023.

[MST03] Elchanan Mossel, Amir Shpilka, and Luca Trevisan. On e-biased generators in NC0. In *44th FOCS*, pages 136–145. IEEE Computer Society Press, October 2003.

[MSW24] Daniele Micciancio and Mark Schultz-Wu. Bit security: Optimal adversaries, equivalence results, and a toolbox for computational-statistical security analysis. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part II*, volume 15365 of *LNCS*, pages 224–254. Springer, Cham, December 2024.

[MT79] Robert Morris and Ken Thompson. Password security: A case history. *Communications of the ACM*, 22(11):594–597, 1979.

[MV15] Eric Miles and Emanuele Viola. Substitution-permutation networks, pseudorandom functions, and natural proofs. *Journal of the ACM*, 62(6):46:1–46:29, 2015.

[MW18] Daniele Micciancio and Michael Walter. On the bit security of cryptographic primitives. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 3–28. Springer, Cham, April / May 2018.

[Oec03] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 617–630. Springer, Berlin, Heidelberg, August 2003.

[OW14] Ryan O'Donnell and David Witmer. Goldreich's PRG: Evidence for near-optimal polynomial stretch. In *CCC 2014*, pages 1–12. IEEE Computer Society, 2014.

[Unr07a] Dominique Unruh. Random oracles and auxiliary input. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 205–223. Springer, Berlin, Heidelberg, August 2007.

[Unr07b] Dominique Unruh. Random oracles and auxiliary input. Cryptology ePrint Archive, Report 2007/168, 2007.

[WY21] Shun Watanabe and Kenji Yasunaga. Bit security as computational cost for winning games with high probability. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 161–188. Springer, Cham, December 2021.

[WY23] Shun Watanabe and Kenji Yasunaga. Unified view for notions of bit security. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part VI*, volume 14443 of *LNCS*, pages 361–389. Springer, Singapore, December 2023.

A Leakage Simulation Lemma

In this section, we prove Lemma 1 (generalized version of leakage simulation) with exact constants. It is mostly adjusting the proof in [CCL18] to work with sets that are not necessarily finite and verifying that there are no measure-theoretic issues.

Let $(\mathcal{X}, \Sigma_{\mathcal{X}})$ be a non-empty measurable space, $(\mathcal{Z}, 2^{\mathcal{Z}})$ a non-empty finite discrete measurable space, where $2^{\mathcal{Z}}$ is the power set of $\mathcal{Z}$, and $0 < \varepsilon \leq 1$ the desired error. Naturally, $\mathcal{X} \times \mathcal{Z}$ is equipped with the product σ-algebra $\Sigma_{\mathcal{X}} \otimes 2^{\mathcal{Z}}$. Let $\mathcal{F}$ be a set of measurable functions $\mathcal{X} \times \mathcal{Z} \to [0,1]$ and $\mathcal{D}$ a distribution over $\mathcal{X} \times \mathcal{Z}$.

We follow the two-step approach in [CCL18].

1. We show how to construct the list $\mathcal{L}$ of functions in $\mathcal{F}$ and a bit-string NegBits of length $|\mathcal{L}|$ (due to completion of $\mathcal{F}$ under negation), which define a distribution $\mathcal{D}'$ that is $(\mathcal{F}, \varepsilon)$-indistinguishable from $\mathcal{D}$. This step works with the general setting stated above — $\mathcal{Z}$ is not necessarily a set of bit-strings and distinguishers in $\mathcal{F}$ are real-valued instead of just zero/one.

2. We cite [CCL18] for how to approximately sample from $\mathcal{D}'$ within error ε, using $\mathcal{L}$, NegBits, and PowTable for fast arithmetics. This step assumes that the distinguishers in $\mathcal{F}$ output zero/one.

The overall advice z' is an encoding of NegBits and PowTable.

Multiplicative Weights Update. We rely on the following lemma about MWU.

Lemma 12 (exact version of Corollary 1 of [CCL18])**.** *Let $\mathcal{Z}$ be a finite set with at least two elements and $0 < \varepsilon \leq 1$ a desired error bound. Set $I = \left\lceil \frac{4 \log |\mathcal{Z}|}{\varepsilon^2} \right\rceil$ and $\eta = \sqrt{\frac{\log |\mathcal{Z}|}{I}}$, and let $g_1, \ldots, g_I$ be functions $\mathcal{Z} \to [0,1]$. In MWU (Figure 1), for all $z \in \mathcal{Z}$, it holds that*

$$\frac{1}{I} \sum_{i=1}^{I} \mathbb{E}[g_i(Z_{i-1})] \leq \varepsilon + \frac{1}{I} \sum_{i=1}^{I} g_i(z) \qquad \text{for } Z_{i-1} \sim \mathcal{D}_i \text{ for all } i \in [I].$$

$\mathsf{MWU}(\mathcal{Z}, g_1, \ldots, g_I, \eta)$

- Once w_{i-1} is given, define the discrete distribution $\mathcal{D}_i$ over $\mathcal{Z}$ as

$$\Pr_{Z_{i-1} \sim \mathcal{D}_i}[Z_{i-1} \in Z] = \frac{\sum_{z \in Z} w_{i-1}(z)}{\sum_{z \in \mathcal{Z}} w_{i-1}(z)} \qquad \text{for all } Z \subseteq \mathcal{Z}.$$

- Set

$$w_i(z) = \begin{cases} 1, & \text{if } i = 0; \\ w_{i-1}(z) \cdot (1-\eta)^{g_i(z)}, & \text{if } i = 1, \ldots, I; \end{cases} \qquad \text{for all } z \in \mathcal{Z},$$

  hence defining $D_1, \ldots, D_I$ as well as (unused) D_{I+1}.

Figure 1. The multiplicative weights update algorithm [AHK12].

Constructing $\mathcal{L}$ and NegBits. We reword Algorithm 2 of [CCL18] with its typos[26] fixed in Figure 2. The rewording[27] is in how the loop (Step 5) chooses $f_i^{\pm}$.

Lemma 13 (Lemma 3 of [CCL18]; ¶)**.** *In* MakeList *(Figure 2), for all $f \in \mathcal{F}$,*

$$\left|\mathbb{E}[f(X, Z_{I'})] - \mathbb{E}[f(X, Z)]\right| \le \varepsilon \qquad \text{for } (X, Z) \sim \mathcal{D} \text{ and } Z_{I'} \sim h_{I'}(X).$$

MakeList$(\mathcal{X}, \mathcal{Z}, \mathcal{F}, \mathcal{D}, \varepsilon)$

1. Once w_i is given, define the regular conditional probability
$$h_i(x, Z) = \frac{\sum_{z \in Z} w_i(x, z)}{\sum_{z \in \mathcal{Z}} w_i(x, z)} \qquad \text{for all } x \in \mathcal{X},\ Z \subseteq \mathcal{Z}.$$
2. Set $w_0(x, z) = 1$ for all $x \in \mathcal{X}$ and $z \in \mathcal{Z}$, hence defining h_0.
3. If $|\mathcal{Z}| = 1$, then set $I' = 0$ and return empty $\mathcal{L}$ and NegBits.
4. Set $I = \left\lceil \frac{4 \log |\mathcal{Z}|}{\varepsilon^2} \right\rceil$ and $\eta = \sqrt{\frac{\log |\mathcal{Z}|}{I}}$. Define $\mathcal{F}_{\pm} = \{ f, 1 - f \mid f \in \mathcal{F} \}$.
5. For $i = 1, \ldots, I$, do the following.
   (a) If for all $g_i \in \mathcal{F}_{\pm}$, it holds that
   $$\mathbb{E}[g_i(X, Z_{i-1})] - \mathbb{E}[g_i(X, Z)] \le \varepsilon \qquad \text{for } (X, Z) \sim \mathcal{D} \text{ and } Z_{i-1} \sim h_{i-1}(X),$$
   then set $I' = i - 1$ and break out of the loop.
   (b) Otherwise, pick one g_i violating the above. Set
   $$w_i(x, z) = w_{i-1}(x, z) \cdot (1 - \eta)^{g_i(x, z)} \qquad \text{for all } x \in \mathcal{X},\ z \in \mathcal{Z},$$
   hence defining h_i.
6. If the loop ended naturally, set $I' = I$. (Never the case — see proof.)
7. Represent g_i's by $(\mathcal{L}[i], \mathsf{NegBits}[i]) \in \mathcal{F} \times \{0, 1\}$. Namely, define
$$\begin{cases} \mathcal{L}[i] = f_i,\ \mathsf{NegBits}[i] = 0, & \text{if } g_i = f_i \in \mathcal{F}; \\ \mathcal{L}[i] = f_i,\ \mathsf{NegBits}[i] = 1, & \text{if } g_i = 1 - f_i \text{ for some } f_i \in \mathcal{F}. \end{cases}$$

Return $\mathcal{L}$ and NegBits, which are of length I'.

Figure 2. The procedure MakeList to construct $\mathcal{L}$ and NegBits, fixed and reworded from Algorithm 2 of [CCL18].

[26] The exposition in [CCL18] conflates which function $x \in \mathcal{X}$ is an input to (x is not an input to MakeList, but h). Its assignment of T, η (in our notation, I, η) has typos of symbols.

[27] It does not require maximizing the advantage. For infinite $\mathcal{F}$, the supremum advantage might be unattainable. Nevertheless, the rewording bears no difference for computer science since $\mathcal{F}$ is always finite even for infinite $\mathcal{X}$.

We adapt the proof in [CCL18] to our version, with its typos[28] fixed.

Proof (Lemma 13). We first verify that h_i's are regular conditional probabilities so that MakeList and Lemma 13 are meaningful. Clearly each $h_i(x, \cdot)$ is a probability measure. Due to the fact that functions in $\mathcal{F}$ (hence $\mathcal{F}_\pm$) are measurable, by induction on i, each $w_i(\cdot, z)$, hence each $h_i(\cdot, Z)$, is measurable.

If $|\mathcal{Z}| = 1$, then $Z, Z_{I'}$ only take one value, the only element in $\mathcal{Z}$, so the simulation is perfect. Otherwise, we will show that MakeList always breaks at Step 5a out of the loop. Suppose for the sake contradiction that the loop ended naturally, then $g_1, \ldots, g_I$ would all be defined and

$$\mathbb{E}[g_i(X, Z_{i-1})] - \mathbb{E}[g_i(X, Z)] > \varepsilon \qquad \text{for } (X, Z) \sim \mathcal{D} \text{ and } Z_{i-1} \sim h_{i-1}(X)$$

for all $i \in [I]$. Averaging the above over i would yield ($\lightning$)

$$\frac{1}{I}\sum_{i=1}^{I} \mathbb{E}[g_i(X, Z_{i-1})] > \varepsilon + \frac{1}{I}\sum_{i=1}^{I} \mathbb{E}[g_i(X, Z)] \qquad \text{for } (X, Z) \sim \mathcal{D} \text{ and } Z_i \sim h_i(X) \text{ for all } i \in [I].$$

For each $x \in \mathcal{X}$, consider the effect of $\mathsf{MWU}(\mathcal{Z}, g_1(x, \cdot), \ldots, g_I(x, \cdot), \eta)$ in Figure 1. By inspection, $\mathcal{D}_i$ would exactly be $h_{i-1}(x, \cdot)$. Therefore, by Lemma 12, for each $z \in \mathcal{Z}$,

$$\frac{1}{I}\sum_{i=1}^{I} \mathbb{E}[g_i(x, Z_{i-1})] \le \varepsilon + \frac{1}{I}\sum_{i=1}^{I} g_i(x, z) \qquad \text{for } Z_{i-1} \sim h_{i-1}(x) \text{ for all } i \in [I].$$

Taking the expectation over $(X, Z) \sim \mathcal{D}$ and applying the law of total expectation for h_{i-1}'s would lead to a contradiction against the previous inequality ($\lightning$).

Now that MakeList breaks at Step 5a out of the loop, we conclude the proof, as the desired conclusion is just the condition of breaking. □

Approximate Sampling. We describe the simulator of [CCL18] and derive the exact complexity of several components. For other analysis, see [CCL18]. We now confine ourselves to the case when the functions in $\mathcal{F}$ have zero/one output, and assume $|\mathcal{Z}| > 1$ as otherwise the task is trivial.

Let $0 < \delta \le 1$ be an integer power of two (the precision) and PowTable a look-up table mapping $y \in \mathbb{N}$ to $\lfloor (1-\eta)^y / \delta \rfloor$. Given $x \in \mathcal{X}$, the simulator approximates the conditional distribution $h_{I'}(x, \cdot)$ as follows (not the final version yet).

1. Compute $F_x(z) = \sum_{i=1}^{I'} g_i(z)$ for each z.
2. Compute $\overline{F}_x = \min_z F_x(z)$.
3. Sample $\widetilde{Z}$ such that $\Pr[\widetilde{Z} = z] \propto \mathsf{PowTable}[F_x(z) - \overline{F}_x]$ as follows.
   (a) Let $\mathcal{Z} = \{z_1, \ldots, z_{|\mathcal{Z}|}\}$ be enumerated.
   (b) Let $W_x(0) = 0$ and $W_x(j) = W_x(j-1) + \mathsf{PowTable}[F_x(z) - \overline{F}_x]$ for $j \in [|\mathcal{Z}|]$.
   (c) Sample $r \xleftarrow{\$} \{0, \ldots, W_x(|\mathcal{Z}|) - 1\}$. (Not the final version yet.)
   (d) Output $\widetilde{Z}$ being the unique z_j such that $W_x(j-1) \le r < W_x(j)$.

[28] The proof in [CCL18] causes some confusion as to when x is fixed or taken expectation over.

The work of [CCL18] shows that for the statistical distance with $h_{I'}(x,\cdot)$ to be bounded by $\eta \le \frac{\varepsilon}{2}$ (and for good efficiency), we should set

$$\frac{\eta}{2|\mathcal{Z}|(1+\eta)} < \delta \le \frac{\eta}{|\mathcal{Z}|(1+\eta)}.$$

For the final version, Step 3c is done with the usual trick of enlarging randomness. To sample r within statistical error $\frac{\varepsilon}{2}$, it suffices to sample $R \xleftarrow{\$} \{0,\dots,W'-1\}$ and setting $r = R \bmod W_x(|\mathcal{Z}|)$, where W' is the unique integer power of two satisfying

$$\frac{2I'|\mathcal{Z}|}{\delta\varepsilon} \le W' < \frac{4I'|\mathcal{Z}|}{\delta\varepsilon}.$$

The two approximations (PowTable and W') causes a total error of no more than ε.

Component Sizes. Jumping ahead, the entries for zero input or zero output are not explicit in PowTable, and every output value in PowTable can be represented using exactly $\log_2(1/\delta)$ bits. Let Y be the number of explicit entries in PowTable. The values of $\log_2(1/\delta)$, Y, $\log_2 W'$ will be useful later. They are

$$\log_2(1/\delta) = \left\lceil \log_2 \frac{|\mathcal{Z}|(1+\eta)}{\eta} \right\rceil, \qquad Y = 1 + \left\lfloor \frac{\log \delta}{\log(1-\eta)} \right\rfloor, \qquad \log_2 W' = 1 + \left\lceil \log_2 \frac{I'|\mathcal{Z}|}{\delta\varepsilon} \right\rceil.$$

Putting Things Together. The following bit-strings are concatenated into z'.

- The prefix-free encodings of I', $\log_2(1/\delta)$, Y, $\log_2 W'$.
- The bit-string NegBits of length I'.
- The values of $\lfloor (1-\eta)^y/\delta \rfloor$ for $y \in [Y]$, each being exactly $\log_2(1/\delta)$ bits long.

However, if $|\mathcal{Z}| = 1$, the string z' is empty.

Assuming $\mathcal{Z} = \{0,1\}^S$, we describe the algorithm $h^{x,\mathcal{L}}(z')$ along with its efficiency of each step (for the circuit model, the time to generate the $\mathcal{L}^x$-aided circuit from z'). In this description, "linear in X" (for some $X \ge 0$) means "bounded by $C_{\text{model}}(X+1)$".

1. If z' is empty, output the empty string (RAM) or the empty circuit (circuits) and terminate. Otherwise, parse the first portion of z' for I', $\log_2(1/\delta)$, Y, $\log_2 W'$.

   For both models, the time is linear in the length of the parsed content.

2. Compute $F_x(z)$ for each z. Accumulate the results of oracle calls as follows.

   - (RAM.) Use a fixed-length, $\lceil \log_2(I'+1) \rceil$-bit encoding for the numbers and add one naïvely. For each $F_x(z)$, the total time for accumulation is linear in I' plus the total number of bit-flips in the add-one operations, the latter of which is again linear in I'.
   - (Circuits.) Group (parenthesize) additions in a balanced way, use linear-time addition circuits, and use i bits for the sum of $(2^i - 1)$ calls. For each $F_x(z)$, the total time for accumulation is linear in I'.

   The total time excluding the queries to $\mathcal{L}$ is linear in $2^S(I'+1)$, hence linear in $2^S(S+1)\varepsilon^{-2}$. (The concrete implementation needs to maintain a loop index, whose cost is amortized constant, again by counting the number of bit-flips.)

3. Compute $\overline{F}_x$.

   For both models, this takes time linear in $2^S \log(I' + 2)$.

4. Align PowTable as follows. Zero-pad the output values so that their lengths are the same power of two and put the resized table to an address that is a large power of two. This ensures that look-up can be done with fast bit operations (shifting and setting lower bits) instead of integer arithmetics.

   In both models, this step takes time poly(|PowTable|).

5. Compute $W_x(j)$ for $j \in \{0, \dots, 2^S\}$.

   For RAM, this takes time linear in $(2^S + 1) \log_2 W'$, where $\log_2 W'$ is for subtraction, look-up, and addition, absorbing the amortized cost of maintaining the loop index. As will be shown later, $\log_2 W' \le 9 + 3S + 4\varepsilon^{-1}$, so overall this step finishes in time linear in $2^S(S + \varepsilon^{-1})$.

   For circuits, each subtraction-look-up-addition takes time linear in

   $$\log(I' + 2) + Y \log_2(1/\delta) + \log_2 W',$$

   which is linear in $(S + \log \varepsilon^{-1})^2 \varepsilon^{-1}$ (shown later) and hence $(S^2 + 1)\varepsilon^{-2}$. The total time is linear in $2^S(S^2 + 1)\varepsilon^{-2}$.

6. Read a random bit-string R of length $\log_2 W'$ and compute $r \leftarrow R \bmod W_x(2^S)$.

   In both models, this takes time

   $$\log_2^{C_{\text{model}}}(2W') \le (1 + 3S + 8 + 4 \log_2 \varepsilon)^{C_{\text{model}}} \le C_{\text{model}} 2^S / \varepsilon.$$

7. Use linear search to find the output. (For RAM, binary search also works.)

   In both models, the time is linear in $2^S \log_2 W'$, i.e., $2^S(S + \varepsilon^{-1})$.

Excluding the oracle calls, Step 2 is the bottleneck for RAM, and Step 5, for circuits. Note that in the circuit model, functions in $\mathcal{F}$ are often represented by circuits, and each oracle call to $\mathcal{L}^x$ is implemented by a circuit of size at least S since it has to read the entire z. This makes Step 2 the real bottleneck for circuits once we replace the oracle gates by their circuits.

The whole process yields distinguishing advantage bounded by 2ε, which is then adjusted for Lemma 1. This does not change the expression of the running time, but it affects the exact advice length. We conclude with the concrete absolute constants.

Lemma 1 (¶). *When $S \ge 1$, set $I = \left\lceil \frac{8S \log 2}{\varepsilon^2} \right\rceil$, $\eta = \sqrt{\frac{S \log 2}{I}}$,*

$$\log_2(1/\delta) = S + \left\lceil \log_2 \left(1 + \frac{1}{\eta}\right) \right\rceil, \quad Y = 1 + \left\lfloor \frac{\log \delta}{\log(1 - \eta)} \right\rfloor, \quad \log_2 W'_+ = 1 + S + \left\lceil \log_2 \frac{I}{\delta \varepsilon} \right\rceil.$$

Here, W'_+ is an upper bound of W'. The number of functions in $\mathcal{L}$ is bounded by $I \le 7S\varepsilon^{-2}$. The length of z' is bounded by

$$2\Big(\overbrace{\lceil \log_2(I+1) \rceil}^{I'} + \overbrace{\lceil \log_2(\log_2(1/\delta) + 1) \rceil}^{\log_2(1/\delta)} + \overbrace{\lceil \log_2(Y+1) \rceil}^{Y} + \overbrace{\lceil \log_2(\log_2 W'_+ + 1) \rceil}^{\log_2 W'}\Big) + 4$$

$$+ \underbrace{I}_{\text{NegBits}} + \underbrace{Y \log_2(1/\delta)}_{\text{PowTable}} \qquad \le 36S\varepsilon^{-2} + 108(S + \log_2 \varepsilon^{-1})^2 \varepsilon^{-1}$$

$$\le 225(S\varepsilon^{-2} + (S + \log \varepsilon^{-1})^2 \varepsilon^{-1}).$$

The bounds also work for $S = 0$.

Proof (Lemma 1). When $S = 0$, both $\mathcal{L}$ and z' are empty so the bounds hold trivially. Otherwise, $S\varepsilon^{-2} \geq 1$ and

$$I < \frac{8S\log 2}{\varepsilon^2} + 1 \leq \frac{8S\log 2}{\varepsilon^2} + \frac{S}{\varepsilon^2} = (1 + 8\log 2)S\varepsilon^{-2} < \frac{20}{3}S\varepsilon^{-2} < 7S\varepsilon^{-2} < 8S\varepsilon^{-2}.$$

By definition, $\eta = \sqrt{\frac{S\log 2}{I}} \geq \varepsilon\sqrt{\frac{\log 2}{1+8\log 2}} > \frac{\varepsilon\log 2}{3} > \varepsilon/7$. Note that $\varepsilon^{-1} \geq 1$, so

$$\begin{aligned}
\log_2(1/\delta) &= S + \lceil \log_2(1+\eta^{-1}) \rceil < S + 1 + \log_2(1 + 7\varepsilon^{-1}) \\
&\leq S + 1 + \log_2(\varepsilon^{-1} + 7\varepsilon^{-1}) = 4 + S + \log_2 \varepsilon^{-1} \\
&\leq -2 + 6(S + \log_2 \varepsilon^{-1}), \\
Y &= 1 + \left\lfloor \frac{\log \delta}{\log(1-\eta)} \right\rfloor \leq 1 + \frac{\log_2(1/\delta)}{-\log_2(1-\eta)} \\
&\leq 1 + \frac{\log 2}{\eta} \cdot \log_2(1/\delta) < 1 + 3\varepsilon^{-1}(4 + S + \log_2 \varepsilon^{-1}) \\
&< -2 + 18\varepsilon^{-1}(S + \log_2 \varepsilon^{-1}).
\end{aligned}$$

Coming to W'_+, we have

$$\begin{aligned}
\log_2 W'_+ &= 1 + S \quad + \quad \log_2(1/\delta) \quad + \quad \lceil \log_2(I/\varepsilon) \rceil \\
&< 1 + S \quad + \quad 4 + S + \log_2 \varepsilon^{-1} \quad + \quad 1 + \log_2(8S\varepsilon^{-2}/\varepsilon) \\
&< 9 + 2S + \log_2 S + 4\log_2 \varepsilon^{-1} \leq 9 + 2S + S + 4\varepsilon^{-1} \leq 16S\varepsilon^{-2}.
\end{aligned}$$

Using $\lceil \log_2(n+1) \rceil \leq n$ for $n \in \mathbb{N}$, we obtain

$$\begin{aligned}
|z'| &\leq 2(I + \log_2(1/\delta) + Y + \log_2 W'_+) + 4 + I + Y\log_2(1/\delta) \\
&= 3I + \log_2 W'_+ + (2+Y)(2 + \log_2(1/\delta)) \\
&\leq 20S\varepsilon^{-2} + 16S\varepsilon^{-2} + 18 \cdot 6 \cdot \varepsilon^{-1}(S + \log_2 \varepsilon^{-1})^2 \\
&= 36S\varepsilon^{-2} + 108(S + \log_2 \varepsilon^{-1})^2\varepsilon^{-1} \\
&\leq \frac{108}{\log^2 2}(S\varepsilon^{-2} + (S + \log \varepsilon^{-1})^2\varepsilon^{-1}) \leq 225(S\varepsilon^{-2} + (S + \log \varepsilon^{-1})^2\varepsilon^{-1}).
\end{aligned}$$

□