New Techniques in Replica Encodings with Client Setup

Rachit Garg* George Lu† Brent Waters‡

May 26, 2020

Abstract

A proof of replication system is a cryptographic primitive that allows a server (or group of servers) to prove to a client that it is dedicated to storing multiple copies or replicas of a file. Until recently, all such protocols required fined-grained timing assumptions on the amount of time it takes for a server to produce such replicas.

Damgård, Ganesh, and Orlandi (CRYPTO' 19) [7] proposed a novel notion that we will call proof of replication with client setup. Here, a client first operates with secret coins to generate the replicas for a file. Such systems do not inherently have to require fine-grained timing assumptions. At the core of their solution to building proofs of replication with client setup is an abstraction called replica encodings. Briefly, these comprise a private coin scheme where a client algorithm given a file m can produce an encoding σ. The encodings have the property that, given any encoding σ, one can decode and retrieve the original file m. Secondly, if a server has significantly less than $n \cdot |m|$ bit of storage, it cannot reproduce n encodings. The authors give a construction of encodings from ideal permutations and trapdoor functions.

In this work, we make three central contributions.

- Our first contribution is that we discover and demonstrate that the security argument put forth by [7] is fundamentally flawed. Briefly, the security argument makes assumptions on the attacker's storage behavior that does not capture general attacker strategies. We demonstrate this issue by constructing a trapdoor permutation which is secure assuming indistinguishability obfuscation, serves as a counterexample to their claim (for the parameterization stated).
- In our second contribution we show that the DGO construction is actually secure in the ideal permutation model from any trapdoor permutation when parameterized correctly. In particular, when the number of rounds in the construction is equal to $\lambda \cdot n \cdot b$ where λ is the security parameter, n is the number of replicas and b is the number of blocks. To do so we build up a proof approach from the ground up that accounts for general attacker storage behavior where we create an analysis technique that we call "sequence-then-switch".
- Finally, we show a new construction that is provably secure in the random oracle (or random function) model. Thus requiring less structure on the ideal function.

1 Introduction

In a proof of replication system [2, 1] , a user wants to distribute a file m and ensure that a server or group of servers will dedicate the resources to storing multiple copies or replicas of it. That is, the server should either receive or generate n replicas $\sigma_1, \ldots, \sigma_n$ where the file m can be efficiently decoded from any single replica. In the original notion of proofs of replication, a server could take a file m as input and independently generate all the replicas $\sigma_1, \ldots, \sigma_n$. Later it could prove possession if challenged. Since the introduction of this concept, several such solutions [16, 3, 10, 5, 9] have emerged.

*University of Texas at Austin. Email: rachit0596@gmail.com. Supported by NSF CNS-1908611, CNS-1414082, Packard Foundation Fellowship, and Simons Investigator Award.

†University of Texas at Austin. Email: gclu@cs.utexas.edu. Supported by NSF CNS-1908611, CNS-1414082, Packard Foundation Fellowship, and Simons Investigator Award.

‡University of Texas at Austin and NTT Research. Email: bwaters@cs.utexas.edu.

However, in these solutions, there exist a tension that stems from the following attack. Consider a non-compliant server that stores just a single copy of m. When challenged to prove possession of replicas, it on the fly, generates $\sigma_1, \ldots, \sigma_n$ using the legitimate generation algorithm and proceeds to prove replication using the ephemeral values as though it were storing these replicas all along.

It is easy to see that achieving meaningful security against such an attack is impossible without imposing a concrete time-bound between when a server is challenged and when it must answer. The setting of this time-bound must be coupled with an understanding of how long it takes an honest system to retrieve the replicas and produce a proof and balanced against how fast a highly provisioned server might take to produce the replicas from scratch. This balancing act creates a certain tension in that more costly replica generation will help security, but also imposes a higher burden on initiation. Moreover, other issues can arise in the context of a more extensive system. For example, if audit challenges come out at a predictable time (e.g., daily), then a cheating server could start generating the response ahead of time.

To address these issues, Damgård, Ganesh, and Orlandi [7] proposed a novel notion that we will call proof of replication with client setup. In this notion, a client that wishes to store a file m will generate replicas $\sigma_1, \ldots, \sigma_n$, along with a (short) public verification key vk. The system will have the properties that (1) one can reconstruct the file from any replica along with the verification key, and (2) a server can prove possession of the replicas to any client that holds the verification key. Unlike the previous systems, proof of replication with client setup need not require fine-grained timing assumptions as a server will not be able to regenerate the replicas from only the message m and vk. Indeed the security definition says (informally) that any poly-time server that devotes significantly fewer resources than n times message length will not be able to pass the possession test.

The solution proposed in [7] combines two high-level ingredients. The first is a proof of retrievability system as proposed in prior work [18, 8, 4]. Roughly, if a server executes a proof of retrievability for data d with a client, this means that now, the server was capable of reconstructing d. However, a proof of retrievability in and of itself gives no guarantee about the amount of resources required to store d.

Second, the authors introduce a notion of a replica encoding. A replica encoding system consists of three algorithms: $(\mathsf{rSetup}, \mathsf{rEnc}, \mathsf{rDec})$. The setup algorithm on input, a security parameter κ and the maximum number of replicas n of a scheme, outputs a public and secret key pair as $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{rSetup}(1^\kappa, 1^n)$. The encoding algorithm takes as input the secret key and a message m to produce an encoding as $y \leftarrow \mathsf{rEnc}(\mathsf{sk}, m)$. Finally, the decoding algorithm takes as input an encoding y and the public key to retrieve the message as $m \leftarrow \mathsf{rDec}(\mathsf{pk}, y)$ or outputs $\perp$ to indicate failure. The algorithms are randomized, and the encoding procedure can be run multiple times to produce multiple encodings. The correctness of the scheme dictates that if one encodes any message m under a secret key and then decodes it under the corresponding public key, m will be decoded.

To capture security, we will consider a soundness game which uses a two-stage attacker $(\mathcal{A}_1, \mathcal{A}_2)$. In the first stage, $\mathcal{A}_1$ will be given a challenger-generated public key pk and reply with a message m. It is then given n encodings generated by the challenger as $y_1, \ldots, y_n$. The attacker outputs a state variable state, which we will generally think of as being smaller than $|m| \cdot n$. At the second phase, the algorithm $\mathcal{A}_2$ is given the input state and is tasked with outputting guesses $\tilde{y}_1, \ldots, \tilde{y}_n$. The security property intuitively states that if the size of the storage $|\mathsf{state}|$ is significantly less than $v \cdot |m|$, then the number of i where $y_i = \tilde{y}_i$ will be less than v. That is, the attacker cannot do much better than simply storing a set of values y_i.

Damgård, Ganesh, and Orlandi showed how a natural compilation of existing proof of retrievability schemes along with replica encodings gave way to proofs of storage with client setup. Also, they provided a candidate construction for replica encodings from trapdoor permutations under the ideal cipher model. We turn our attention to these.

The DGO construction: We now outline (a slight variant of the) construction for [7], which is given in the ideal permutation model. The building blocks will consists of a trapdoor permutation $\mathsf{f}, \mathsf{f}^{-1}$, along with the ideal cipher $\mathsf{T}, \mathsf{T}^{-1}$, and a random oracle H. We again let κ be the security parameter and let $\lambda = \lambda(\kappa)$ be the output length of the trapdoor permutation as well as the block length of an ideal permutation $\mathsf{T} : \{0,1\}^\lambda \to \{0,1\}^\lambda$. For pedagogical purposes, we will assume for the sketch below that messages consist of λ bits, but in our main body, we consider the more realistic case of many block messages.

The setup algorithm simply chooses a TDP public/secret key pair as $\mathsf{KeyGen}(1^\kappa)$ outputs $(\mathsf{pk}, \mathsf{sk})$ where KeyGen is the trapdoor permutation key generation algorithm. The public and secret key pair of the TDP serve as the keypair of the replicated encoding scheme.

The encoding algorithm $\mathsf{rEnc}(\mathsf{sk}, m)$ takes as input the TDP secret key and message m. It first chooses a string $\rho \xleftarrow{R} \{0,1\}^\kappa$. It then initializes a value $Y_0 = m \oplus \mathsf{H}(\rho)$ where H is modeled as a random oracle function. Then for $j = 1$ to r rounds it computes $Y_j = \mathsf{f}^{-1}(\mathsf{sk}, \mathsf{T}(Y_{j-1}))$ where r is the number of rounds, which grows linearly with the number of replicas. The encoding is output as (Y_r, ρ).

Finally, the decoding algorithm $\mathsf{rDec}(\mathsf{pk}, y = (Y_r, \rho))$ recovers a message as follows. For setting j from $r-1$ down to 0 compute $Y_j = \mathsf{T}^{-1}(\mathsf{f}(\mathsf{pk}, Y_{j+1}))$. Then output $m = Y_0 \oplus \mathsf{H}(\rho)$.

The fact that the decoding step recovers the message follows straightforwardly from the correctness of the trapdoor permutation and ideal permutation. We also observe that it is publicly computable since it uses the public key and forward direction of the trapdoor permutation.

1.1 Our Contributions

We make three core contributions to this area:

1. Our first contribution is that we discover and demonstrate that the security argument put forth by [7] is fundamentally flawed. The security argument makes implicit assumptions about an attacker's behavior which are not generally true. More specifically, in the security game applied to the DGO construction (in the ideal permutation model) an attacker works in two phases. The first stage attacker $\mathcal{A}_1$ receives the replicas, can make several queries to the ideal permutation and then records some state state of limited size. This state state is passed to a stage two attacker $\mathcal{A}_2$ which can make further permutation queries and attempts to reconstruct the queries. In general a first stage attacker can apply arbitrary strategies to breaking the scheme so long as it poly-time and state state is sufficiently small. However, the proof argument of [7] assume that the ideal permutation queries made by the attacker will be "uniquely stored". Roughly, they will argue that a query output bit will either be stored explicitly or not at all. This discounts the possibility of an attacher strategy such as making several oracle queries and storing the XOR of all the outputs together.

   We demonstrate that the above error manifests in a false theorem statement in [7]. The authors claim that the scheme is secure for *any* trapdoor permutation (TDP) if $r = \lambda \cdot n$ rounds are applied when doing n encodings of b blocks with security parameter λ . (I.e. Claim the number of rounds does not need to scale with b.) We provide an explicit counterexample to this claim in Section 7. We give a TDP that is secure assuming indistinguishability obfuscation, but for which the scheme is attackable using these parameters. The attacker strategy actually works by XORing several query values together and is thus directly tied to the flaw in the security proof. There does not appear to be any simple "fix" to the security argument of [7] as we will see that addressing general attacker storage strategies comprises the core difficulty of proving security.

2. For our second contribution we show that the DGO construction is actually secure when parameterized correctly. In particular, when the number of rounds is equal to $\lambda \cdot n \cdot b$. To do so we need to build up a proof approach from the ground up that accounts for general attacker storage behavior. We first develop an analysis technique that we call "sequence-then-switch". We show how in this framework, we can prove security against an attacker that arbitrarily assigns state. In particular, we show how to analyze the security of a close variant of the [7] construction in the ideal permutation model. In addition, we give an explicit construction of a trapdoor permutation using indistinguishability obfuscation which allows for an attack strategy not covered by their restricted model, showing the [7] construction as given is in fact explicitly *insecure* against general adversaries.

3. The prior construction and proof relies on the ideal permutation model. A perhaps better goal would be to have a construction secure in the random oracle or random function model as this assumes less structure on the ideal object. Typically, this is dealt with by building a random permutation from a function using a Feistel network and showing that this is "indifferentiable" in the indifferentiability

framework of Mauer et al. [13]. Prior works have shown this for 14 [11] and then 8 round Feistel [6]. However, Ristenpart, Shacham, and Shrimpton [17] show that the framework does not compose for multi-round games. Since the above construction relies on a multi-round game, proof from an ideal permutation cannot be reduced to a proof to an ideal function.

We give a new construction in the ideal function model and analyze its security. Our construction uses the random function to embed a Feistel like structure into the construction. However, instead of arguing in the indifferentiablity framework, we provide direct proof of security, which bypasses any composability issues. In both proofs, we allow the attacker to assign its storage arbitrarily.

1.2 Our Techniques

We begin by describing our analysis for the first construction using a TDP and ideal permutation. We focus on the construction producing many replicas on a single block, as described in the introduction for simplicity. Also, for simplicity, we consider the particular case where an attacker that asks for n replicas in the first stage and wants to produce all n of these replicas, but we significantly less than $n \cdot \lambda$ storage. In particular consider an adversary with state of length $n \cdot \lambda - n \cdot \omega(\log \kappa)$ bits of storage for security parameter κ and block length λ. Our central idea is to organize the proof into two parts where we first show that any storage bounded $\mathcal{A}_2$ must make "sequential" oracle queries on at least one replica. Then we show that on this particular replica, how one can swap out permutation output for another.

1. **Sequentiality:** In our security game, the challenger first creates n replicas of m. To create the i-th replica by choosing ρ_i randomly. It sets $Y_0^{(i)} = m \oplus \mathsf{H}(\rho_i)$. Then for $j = 1$ to r rounds it computes $Y_j^{(i)} = \mathsf{f}^{-1}(\mathsf{sk}, \mathsf{T}(Y_{j-1}(i)))$. The encoding is output to $\mathcal{A}_1$ as $(Y_r^{(i)}, \rho_i)$ for $i \in [n]$. The attacker $\mathcal{A}_1$ receives the encodings, makes some more oracles queries before producing state of $n \cdot \lambda - n \cdot \omega(\log \kappa)$ bits and passing it to $\mathcal{A}_2$.

   Let's examine the behavior of $\mathcal{A}_2$ whose job it is to output the encodings using the state plus oracle queries. We say that $\mathcal{A}_2$ "queries sequentially" on replica i if for all $j \in [0, r-1]$ it queries the oracle T on $Y_j^{(i)}$ before it queries the oracle on $Y_{j+1}^{(i)}$. (We will think of outputting the encoding $Y_r^{(i)}$ at the end as implicitly querying on the final value.) That is for $\mathcal{A}_2$ to query sequentially on replica i it must both make all $r+1$ oracle queries and make them in (relative) order. However, there could be many other queries outside the replica chain interspersed between $Y_j^{(i)}$ and $Y_{j+1}^{(i)}$.

   We will first argue that except with negligible probability whenever $\mathcal{A}_2$ produces all the encodings, it queries sequentially on at least one replica. Observe that we cannot hope to say that it queried sequentially on all replicas as state could directly store several on the replica encodings, which allows the algorithm to bypass any additional queries related to that replica.

   To prove this, we first define and prove a useful matching pairs lemma. Consider an algorithm $\mathcal{B}$ that takes as input a string advice of length $n \cdot \lambda - n \cdot \omega(\log \kappa)$ and gets access to a string oracle access to a randomly chosen permutation $\mathsf{T}(\cdot), \mathsf{T}^{-1}$ of block length λ. The goal of $\mathcal{B}$ is to provide n *distinct* pairs (x_i, y_i) such that $\mathsf{T}(x_i) = y_i$, but *without* querying the oracle a either x_i nor y_i. Thus $\mathcal{B}$ can make several oracle queries on many values; however, once a query is made on some x, it spoils using x as a value from one of the pairs. Note that to win in this game, $\mathcal{B}$ needs to produce the pairs— not just distinguish them from random. Also observe that $\mathcal{B}$ can use advice to help it win this game. For example, advice might encode the several pairs.

   We prove that no attacker $\mathcal{B}$ that makes a polynomially bounded number of queries can win in this game by a simple application of the union bound. Consider a fixed value of an advice string a — that is a is fixed before the permutation is chosen. We show that the probability of $\mathcal{B}(a)$ winning is at most $\frac{\mathsf{poly}(\lambda)}{2^{n\lambda}}$. Then by the union bound the probability that there exists *any* string a which it could win with is at most $2^{n \cdot \lambda - n \cdot \omega(\log \kappa)} \cdot \frac{\mathsf{poly}(\lambda)}{2^{n\lambda}}$ which is negligible in λ.

   Now we need to show that an attacker that wins but is not sequentially querying on any replica will break our matching pairs game. We consider $(\mathcal{A}_1, \mathcal{A}_2)$ that does this. Let's think of the algorithm

pair as deterministic. (If they are randomized for each security parameter, we can fix their coins that maximize success probability.) We construct an algorithm $\mathcal{B}$ along with the process of determining an advice string that does this. Conceptually we can think of a preprocessing algorithm $\mathcal{B}'$ that generates the advice. $\mathcal{B}'$ will first run $\mathcal{A}_1$, which makes several queries and then produce state. It then runs $\mathcal{A}_2$ on state. If $\mathcal{A}_2$ either did not produce all the replica encodings or it did sequentially query on some replica i, then abort. However, if it did not make sequential queries on all replicas, then there must be values $j_1, \ldots, j_n$ where $\mathcal{A}_2$ made an oracle query on $Y_{j_i}^{(i)}$ (or $\mathsf{f}(\mathsf{pk}, Y_{j_i}^{(i)})$), but had not yet made a query on $Y_{j_i-1}^{(i)}$. Let $q_1, \ldots, q_n$ be the indices of the queries (ordered chronologically) for which this occurs. Note the number of queries $\mathcal{A}_2$ can make is polynomial in κ, but in general, it could be much more than $r \cdot n \cdot \lambda$. The preprocessing algorithm will package its advice string as state along with $j_1, \ldots, j_n$ and $q_1, \ldots, q_n$. Importantly, the size of this information is bounded by $\lg(\mathsf{poly}(\kappa))$ for some polynomial poly since n, r, and the number of replicas is polynomially bounded. This means that if state is of size $n \cdot \lambda - n \cdot \omega(\log \kappa)$, then the advice string will be within $n \cdot \lambda - \omega(\log \kappa)$.

We now consider algorithm $\mathcal{B}$, which receives the advice string. $\mathcal{B}$ will run $\mathcal{A}_2$ with the following modifications. Suppose $\mathcal{A}_2$ makes its q-th query where $q = q_i$ for some i. This means that $\mathcal{A}_2$ is querying on $Y_{j_i}^{(i)}$, but had not yet made a query on $Y_{j_i-1}^{(i)}$. At this point $\mathcal{B}$ determines $Y_{j_i-1}^{(i)}$ by querying $Y_1^{(i)} = \mathsf{f}^{-1}(\mathsf{sk}, \mathsf{T}(Y_0^{(i)}))$ up to $Y_{j_i-1}^{(i)} = \mathsf{f}^{-1}(\mathsf{sk}, \mathsf{T}(Y_{j_i-2}^{(i)}))$. It then submits $(Y_{j_i-1}^{(i)}, \mathsf{f}(\mathsf{pk}, Y_{j_i}^{(i)}))$ as one of its matching pairs noting that neither $\mathsf{T}(Y_{j_i-1}^{(i)})$ nor $\mathsf{T}^{-1}(\mathsf{f}(\mathsf{pk}, Y_{j_i}^{(i)}))$ were made before. It can also continue to run $\mathcal{A}_2$ without making either of these queries to the oracles since it already knows the answers to them. As this process proceeds, $\mathcal{B}$ will eventually recover n such pairs which breaks our matching pairs lemma and arrives at a contradiction.

2. **Switching:**

   Once sequentiality is established, we will proceed to argue that the adversary must still be sequential with good probability even when we "switch" the random oracle output of some $Y_j^{(\gamma)}$ to a random value only for $\mathcal{A}_2$, allowing us to embed a trapdoor permutation challenge.

   In more detail, we now consider a new switched game that is almost equivalent to the prior one. In the switched game the challenger first chooses r random values $A_{i,b} \in \{0,1\}^\lambda$ for $j \in [1, r], b \in \{0, 1\}$ along with a bit string $x \in \{0, 1\}$. It programs the oracle T such that $Y_j^{(\gamma)} = A_{i,b}$. This game can be shown to be almost equivalent to the previous one.

   Next, we consider a game where the challenger answers queries according to a string x with $\mathcal{A}_1$, but switches to using a string x' (and keeps everything else the same) when responding to $\mathcal{A}_2$. The challenger chooses the string x' such that the output state given by $\mathcal{A}_1$ when the queries are answered according to x' in the first phase. The attacker is considered to win only if it would produce sequential queries *both* for when x was used with $\mathcal{A}_2$ and when x' was used with $\mathcal{A}_2$.

   With high probability, such an x' will exist from the fact that $|\mathsf{state}| \leq n \cdot \lambda - \omega(\log \kappa)$ and r is set to be $n \cdot \lambda$. We emphasize that to make this argument; we do not make any further assumptions on how $\mathcal{A}_1$ assigns state other than the bound on the size. We can then use the heavy row lemma [15] to argue that if an attacker wins with probability ϵ in the previous game, it wins with probability $\approx \epsilon$ in this game. We note that the game takes exponential time to find such an x', but this is not an issue as the closeness lemma is information-theoretic.

   Finally, in order to embed a TDP challenge, we need to move to a security game that can be efficiently simulated. While it might take exponential time to find x' from x above, we observe that this is not necessary. Instead, we can embed the challenge from just knowing the shortest common prefix of x and x'. Moreover, given x, we can simply guess what the prefix is with a $\frac{1}{r}$ loss. Thus we move to a final game where the challenger simply chooses a random value j and a random permutation T in the first phase and then replaces the oracle output of $Y_j^{(i)}$ with a random R in the second phase. The attacker wins if it queries $\mathsf{f}^{-1}(\mathsf{sk}, R)$. A simple reduction then shows that any attacker that wins in this game breaks the TDP security.

Extending to the ideal function model

We can now return to our goal of building a secure construction in the ideal function model as opposed to the ideal permutation model. As argued earlier, doing so is desirable as an ideal function imposes less structure and appears to be a less risky heuristic. Our solution will build upon the analysis principles established above, but proving security involves more complications.

We begin by sketching out the encoding construction. In this setting, we will have a TDP in the domain λ bits and use a random oracle T' that outputs λ bits. We will use blocks of length 2λ, and for this sketch, focus on the particular case where each replica consists of a single block message.

The setup algorithm again chooses a TDP public/secret key pair as $\mathsf{KeyGen}(1^\kappa) \rightarrow (\mathsf{pk}, \mathsf{sk})$ as before. The encoding algorithm $\mathsf{rEnc}(\mathsf{sk}, m)$ takes as input the TDP secret key and message $m \in \{0,1\}^{2\lambda}$. It first chooses a string $\rho \xleftarrow{R} \{0,1\}^\kappa$. It then initializes values $Y_0 = L(m \oplus \mathsf{H}(\rho))$ and $Y_1 = R(m \oplus \mathsf{H}(\rho))$ where H is a random oracle that produces an 2λ bit output and L, R are functions that take the left and right halves. Then on rounds j from 2 to r compute Y_j from Y_{j-1} and Y_{j-2} as

$$Y_j = \mathsf{f}^{-1}(\mathsf{sk}, Y_{j-2} \oplus \mathsf{T}'(Y_{j-1})).$$

The replica encoding value is 2λ bits long and consists of the last two values as $Y_{r-1}||Y_r$. The decoding algorithm rDec works backward down the Feistel structure to recover the message.

In this setting, we want to prove that in the security game, an attacker with $n \cdot 2\lambda - n \cdot \omega(\log \kappa)$ cannot produce n replica encodings. (The extra factor of two is solely due to blocks being 2λ bits here.)

Our proof will follow in the same theme of showing that there must be a form of sequential querying made on at least one replica. However, the new structure of the construction presents additional complications. For example, we could imagine an attacker $\mathcal{A}_1$, which stores all the values Y_j^i for some j. This is possible since storing these only take $n\lambda$ bits, and our assumption is only that the storage is less than $2n\lambda$ bits. On the one hand, it is unclear how the attacker can leverage storing all these values because one needs consecutive values (e.g., $Y_j^{(i)}$, $Y_{j+1}^{(i)}$) to propagate further. And, storing n different consecutive pairs requires $2n\lambda$ bits of storage. On the other hand, the attacker can store these means at the very least we need a new notion of sequentiality.

For our new notion of sequentiality, we say that the queries to replica i meet our requirements if the longest common subsequence of the queries made and $Y_1^{(i)}, Y_2^{(i)}, \ldots, Y_r^{(i)}$ is of length at least $r-3$. Intuitively, this is close to our original notion but allows for a little skipping. To prove this form of sequentiality, we invoke a random function analog of our matching pairs lemma from before. The reduction to matching pairs follows in a similar spirit to before but requires a more nuanced case analysis.

Once that is in place, our proof proceeds analogously, but again with more nuances and complications arising from the fact that we only can guarantee the weaker form of longest common subsequence.

1.3 Additional Prior Work

Proofs of Retrievability:

Proofs of retrievability guarantee to a verifier that a server is storing all of client's data. The notion was formalized in [12], where, in an audit protocol the verifier stores a (short) verification key locally and interacts with the server to enforce accountability of the storage provider. If the server can pass an audit, then there exists an extractor algorithm, that must be able to extract the file on interaction with the server. There are different constructions for this primitive, [8, 18, 4]. The construction of [18] showed how to do this in the random oracle model that allow public verifiability.

Proofs of Replicated Storage:

Damgärd, Ganesh and Orlandi [7] gave the formal treatment of proofs of replicated storage. The idea was introduced in [2, 1] where they proposed Filecoin, a decentralized storage network that performs consensus using proofs of replication. Recently, [16, 10, 3, 5, 9] have given constructions for proof of replication using timing assumptions (encoding process is much slower so that a server cannot replicate data on demand). On the other hand, the scheme of [7] (builds on the idea of repeated applications of trapdoor permutation from [19]) is not based on timing assumptions and introduces the notion of a replica encoding that can be

combined with a public verifiable proof of retrievability [18] to give a proof of replicated storage. Please see [7] for other related works to proofs of replication.

1.4 Concurrent Work

After completing our work we learned of a concurrent and independent work of Moran and Wichs [14]. They introduce a variant of replica encodings which they call incompressible encodings, and proceed to provide constructions in the random-oracle model using the Decisional Composite Residuosity or Learning with Errors assumptions. Their construction utilizes some new techniques to apply lossiness to construct said encodings. In addition, they introduce an additional "big-key" application for intrusion resilience which applies to our constructions and proofs as well.

At a very high level, our work depends on the general assumption of trapdoor permutations, whereas they use the specific number theoretic assumptions of Decisional Composite Residuosity and Learning with Errors. Comparing our construction instantiated with RSA trapdoor permutation to their DCR construction, their construction appears to be more practically efficient from a computational perspective due to the round complexity required for our construction, however, ours makes tighter use of space for small "s" values used in the DCR construction. An interesting future direction could be to explore concrete space and computational efficiency tradeoffs for increasing the s parameter in their DCR construction.

Similar to us, Moran and Wichs discovered foundational issues in the proof arguments of [7]. In a personal communication Wichs noted that there is a simple heuristic counterexample to the claim of [7] if one uses the heuristic of ideal obfuscation. We subsequently developed a counterexample based on the concrete assumption of indistinguishability obfuscation that we added as Section 7 of our work.

2 Preliminaries

Notation. These notations are used consistently throughout the text.
We use κ to denote the security parameter. $y \leftarrow \mathcal{B}(x)$ denotes the output of the algorithm $\mathcal{B}$ when we run x on it. A negligible function $\mathsf{negl}(x)$ is a function such that for every positive integer c, there exists an integer N_c such that for all $x > N_c$, $\mathsf{negl}(x) < \frac{1}{x^c}$. $[n]$ denotes the set $\{1, 2, \ldots, n\}$ and $[a, b]$ denotes the interval between a and b inclusive. $y \xleftarrow{R} \mathcal{D}$ implies that we are uniformly sampling y from a domain set $\mathcal{D}$. We say an adversary or an algorithm $\mathcal{A}$ is probabilistic poly time (PPT) if there is a polynomial $\mathsf{poly}(\cdot)$ such that for all κ, $\mathcal{A}$ will halt in $\leq \mathsf{poly}(\kappa)$ time in expectation on any input of length κ.

2.1 Trapdoor permutations

Definition 1. *A trapdoor permutation is defined as a collection of three PPT algorithms* $\mathsf{KeyGen}(.), \mathsf{f}(.,.), \mathsf{f}^{-1}(.,.)$ *with the following properties:*

- *Easy to sample:* $\mathsf{KeyGen}(1^\kappa)$ *outputs a public key* pk*, secret key* sk *pair* $(\mathsf{pk}, \mathsf{sk})$*. Size of both the keys* pk, sk *are polynomial in the security parameter* κ*.*
- *Easy to compute: A public key* pk *defines a family of trapdoor permutation functions denoted by* $\mathcal{F}$ *on domain* $\mathcal{D}_{\mathsf{pk}}$ *where* $\mathcal{F} = \{\mathsf{f}(\mathsf{pk}, .) : \mathcal{D}_{\mathsf{pk}} \to \mathcal{D}_{\mathsf{pk}}\}$ *as a collection of bijective polytime computable functions* $\mathsf{f}(\mathsf{pk}, .)$*.*
- *Inversion with trapdoor:* $\mathsf{f}^{-1}(\mathsf{sk}, .)$ *computes the inverse of* $\mathsf{f}(\mathsf{pk}, .)$*.*

$$\forall \kappa, (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa), \forall x \in \mathcal{D}_{\mathsf{pk}}, \mathsf{f}^{-1}(\mathsf{sk}, \mathsf{f}(\mathsf{pk}, x)) = x.$$

- *One wayness, i.e. it is hard to invert without the knowledge of the secret key* sk*. More formally, for all PPT algorithms* $\mathcal{A}$*, there exists a negligible function* negl *such that,*

$$\Pr\left[\begin{array}{c} \mathsf{f}(\mathsf{pk}, z) = y \ s.t. \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa), x \xleftarrow{R} \mathcal{D}_{\mathsf{pk}}, y = \mathsf{f}(\mathsf{pk}, x), z \leftarrow \mathcal{A}(\mathsf{pk}, y) \end{array}\right] \leq \mathsf{negl}(\kappa),$$

where the probability is over the coins to KeyGen *and sampling* x*.*

3 Defining Replica Encoding

A Replica Encoding scheme - ReplicaEncoding is defined as a tuple of algorithms $(\mathsf{rSetup}, \mathsf{rEnc}, \mathsf{rDec})$, where rSetup takes in the security parameter denoted by 1^κ and the maximum number of replicas a client wishes to replicate denoted by 1^n and outputs a public key secret key pair $(\mathsf{pk}, \mathsf{sk})$, rEnc is a randomized algorithm which takes a message $m \in \{0,1\}^*$, a secret key sk and outputs a replica encoding. rDec is a deterministic algorithm that takes as input a public key pk, a replica encoding and outputs a mesage m. Formally,

$$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{rSetup}(1^\kappa, 1^n),\ y \leftarrow \mathsf{rEnc}(\mathsf{sk}, m),\ m \leftarrow \mathsf{rDec}(\mathsf{pk}, y).$$

Definition 2. *A tuple (*rSetup,rEnc,rDec*) is a replica encoding if the following holds:*

- *Correctness: For any choice of coins of* rSetup*, the probability of incorrect decoding is*

$$\forall n, m,\ \Pr\begin{bmatrix}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{rSetup}(1^\kappa, 1^n) \\ \mathsf{rDec}(\mathsf{pk}, \mathsf{rEnc}(\mathsf{sk}, m)) \neq m\end{bmatrix} \leq \mathsf{negl}(\kappa)$$

  where the probability is over the coins of rEnc [1]*.*

- *Length of the encoding scheme is denoted by a function* $\mathsf{len}(\cdot,\cdot) : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ *that takes in the security parameter and the length of the message and outputs the length of the encoding, formally for any* κ, m*, choice of coins of* rSetup*,*

$$\forall \kappa, m,\ \Pr\begin{bmatrix}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{rSetup}(1^\kappa, 1^n) \\ \mathsf{len}(\kappa, |m|) \neq |\mathsf{rEnc}(\mathsf{sk}, m)|\end{bmatrix} \leq \mathsf{negl}(\kappa)$$

  where the probability is over the coins of rEnc*.*

- s*-Sound: Consider the game* $\mathsf{Sound}_{\mathcal{A}_1,\mathcal{A}_2}(\kappa, n)$ *between an adversary pair* $(\mathcal{A}_1, \mathcal{A}_2)$ *and a challenger defined in Figure 1. A replica encoding scheme is* s*-sound (*$\mathsf{s} : \mathbb{N} \times \mathbb{N} \to [0,1]$*), if for any probabilistic poly-time adversaries* $(\mathcal{A}_1, \mathcal{A}_2)$*, for all* $n \in \mathbb{N}$*, there exists a function* negl *such that the following holds.*

$$\Pr\begin{bmatrix}(v, \mathsf{state}, m) \leftarrow \mathsf{Sound}_{\mathcal{A}_1,\mathcal{A}_2}(\kappa, n), s.t. \\ |\mathsf{state}| < v \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)\end{bmatrix} \leq \mathsf{negl}(\kappa).$$

  where the probability is over the coins with the challenger and the two adversaries $\mathcal{A}_1, \mathcal{A}_2$*.*

A remark on the efficiency. We remark that there can exist trivial constructions of replica encoding by simply concatenating a string m with the randomness ρ i.e. let $\mathsf{rEnc}(\mathsf{sk}, m) = m||\rho$. These schemes are secure for $\mathsf{s} \in \frac{|\rho|}{|m|+|\rho|} - \frac{\omega(\log \kappa)}{|m|+|\rho|}$. If we consider long ρ, we can construct a sound replica encoding scheme for arbitrary $\mathsf{s}(\kappa, |m|)$. As a specific example, imagine $\mathsf{rEnc}(\mathsf{sk}, m) = m||\rho$ where $\rho \xleftarrow{R} \{0,1\}^{99|m|}$. This scheme is trivially correct as m is output in the clear and $\mathsf{len}(\kappa, |m|) = 100|m|$. For all functions s such that $\mathsf{s}(\kappa, |m|) \in \frac{99}{100} - \frac{\omega(\log \kappa)}{100|m|}$, the proposed scheme is s sound. Intuitively, for each encoding $\mathcal{A}_2$ has $99|m| - \omega(\log \kappa)$ information in state and is supposed to output $99|m|$ random bits. Even if they randomly guess the remaining bits the probability of success will be negligible in κ. For this reason we are interested in schemes that do better than the soundness efficiency tradeoffs of this trivial solution.

Definitions in prior work. We make a few brief remarks about some differences in our definition and the earlier one of [7]. The formal definition of proof of replication was introduced by Damgård et al [7].

[1] There exists a generic method for converting a scheme with negligible correctness error into a perfectly correct scheme. To do so augment the rEnc algorithm so that it first produces the encoding. Then the new rEnc algorithm run the deterministc rDec algorithm on the encoding to check that the message was recovered. If not, output the message in the clear and a flag bit indicating that the message is output in plain instead of the encoding. This adds a negligible hit in the security as opposed to the correctness.

Game $\mathsf{Sound}_{\mathcal{A}_1,\mathcal{A}_2}(\kappa, n)$

- **Setup:** The challenger(denoted by $\mathcal{C}$) runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{rSetup}(1^\kappa, 1^n)$ and sends pk to $\mathcal{A}_1$. It keeps the secret key sk for itself.
- **File Challenge:** The adversary $\mathcal{A}_1$, on input $(1^\kappa, \mathsf{pk})$, chooses a file $m \in \{0,1\}^*$. It sends m to $\mathcal{C}$.
- The challenger outputs n encodings of m by calling rEnc n times.

$$\forall i \in [n],\ y^{(i)} \leftarrow \mathsf{rEnc}(\mathsf{sk}, m)$$

  and returns $y^{(1)}, \ldots, y^{(n)}$ to $\mathcal{A}_1$.
- **State Sharing:** $\mathcal{A}_1$ outputs $\mathsf{state} \leftarrow \mathcal{A}_1(1^\kappa, \mathsf{pk}, y^{(1)}, \ldots, y^{(n)})$ and sends state, the number of replicas 1^n and message m to $\mathcal{A}_2$.
- **Guess:** $\mathcal{A}_2$ on receiving state state, outputs the replica guess to $\mathcal{C}$,

$$(\tilde{y}^{(1)}, \ldots, \tilde{y}^{(n)}) \leftarrow \mathcal{A}_2(1^\kappa, 1^n, \mathsf{pk}, m, \mathsf{state})$$

- **Verify:** Let $v_i = 1$ if $\tilde{y}^{(i)} = y^{(i)}$ and 0 otherwise. Output $(v = \sum_{i=1}^{n} v_i, \mathsf{state}, m)$.

Figure 1: The soundness game for the replica encoding scheme.

The earlier soundness definition is stated in terms of a constant c,

$$\Pr\left[|\mathsf{state}| < c \cdot v \cdot \mathsf{len}(\kappa, |m|) \mid (v, \mathsf{state}, m) \leftarrow \mathsf{Sound}_{\mathcal{A}_1,\mathcal{A}_2}(\kappa, n)\right] \leq \mathsf{negl}(\kappa).$$

To see the issue let's consider an attack algorithm that tries to guess the secret information used by $\mathcal{C}$ when constructing the challenge (e.g. it tries to guess the TDP secret key and the randomness used during the encryption algorithms). If its guess is correct, it can recover the replica encodings by running rEnc in the forward direction and outputs the n replicas; otherwise, it simply gives up and outputs all 0's. Clearly such an adversary should not be viewed as successful since it only succeeds a neglgible fraction of the time. However, if its guess is correct (which happens only with negligible probability) it wins the game with $v = n$ and no state bits used. Otherwise, if the guess is incorrect even for some encoding then $v < n$. This adversary succeeds when conditioned on $v = n$ with probability 1.

Tweaking their definition to include v, state as output of the game and not conditioning on events where the correct replica is output solves the issue. We also augment the soundness parameter s to be a function of the security parameter and message length as opposed to a constant. Relevance of this can be seen from our theorem statements.

Other minor differences between our definitions include a rSetup algorithm that sets up the parameters for the scheme. We do this to formalize the alignment and use the KeyGen environment of the underlying trapdoor permutation. The formal definition of replica encoding in DGO includes an efficiency condition defined as exactly $|m| + O(\kappa)$. We do not restrict the efficiency in the formal definition in our work and state it as a desired property that should be required for a practical replica encoding scheme.

4 Lemmas on Random Functions and Permutations

This section contains useful information theoretic lemmas on analyzing random functions and permutations. The first is a result on the hardness of outputting relations on the required ideal primtive given limited advice and restricted behavior. We will use this later in showing adversaries capable in distinguishing between certain games will be able to do the following with noticeable probability.

Lemma 1. *Let* T, $\mathsf{T}^{-1} : \{0,1\}^\lambda \to \{0,1\}^\lambda$ *be oracles to a random permutation and its inverse. Consider any computationally unbounded adversary* $\mathcal{B}$ *that makes polynomially bounded (in* λ*) queries to* $\mathsf{T}, \mathsf{T}^{-1}$ *on input a bounded* advice *and outputs* n *pairs* (x_i, y_i) *without querying them explicitly. If* advice *is bounded by* $n \cdot \lambda - \omega(\log \lambda)$ *bits where* n *is polynomial in* λ*, the probability that it succeeds is negligible in* λ*.*

More formally, let the inputs and outputs by $\mathcal{B}$ to oracle $\mathcal{O}$ be denoted by lists $\mathsf{s}_{\mathcal{B}}^{\mathcal{O}}$, $\mathsf{S}_{\mathcal{B}}^{\mathcal{O}}$ respectively. Then,

$$\Pr\left[\begin{array}{c}\exists\ \mathsf{advice}\in\{0,1\}^*\ s.t.\ |\mathsf{advice}|\leq n\cdot\lambda-\omega(\log\lambda),\\ \{(x_i,y_i)\}_{i=1}^n\leftarrow\mathcal{B}^{\mathsf{T}(\cdot),\mathsf{T}^{-1}(\cdot)}(\mathsf{advice})\ where\\ \forall i\neq j\in[n],x_i\neq x_j,\ \mathsf{T}(x_i)=y_i,\ x_i\notin\mathsf{s}_{\mathcal{B}}^{\mathsf{T}}\ and\ y_i\notin\mathsf{s}_{\mathcal{B}}^{\mathsf{T}^{-1}}\end{array}\right]\leq\mathsf{negl}(\lambda),$$

the probability is over the choice of the permutation T.

Proof. Let a be some fixed string , and let the event E_{a} be the event that $\mathcal{B}$ manages to produce n distinct (x_i,y_i) pairs such that $\mathsf{T}(x_i)=y_i$ without querying T on x_i or T^{-1} on y_i when run on a.

Claim 1. $\Pr[\mathsf{E}_{\mathsf{a}}]\leq\frac{\mathsf{poly}(\lambda)}{2^{n\lambda}}$ *over a random* T.

Proof. Consider the set of $\{x_i\}_{i=1}^n$ which $\mathcal{B}$ outputs. Suppose they are unique and not in $\mathsf{s}_{\mathcal{B}}^{\mathsf{T}}$ or $\mathsf{S}_{\mathcal{B}}^{\mathsf{T}^{-1}}$. Since each $\mathsf{T}(x_i)$ is uniform on permutations not fixed on $\mathsf{S}_{\mathcal{B}}^{\mathsf{T}}\cup\mathsf{s}_{\mathcal{B}}^{\mathsf{T}^{-1}}$, so the possible corresponding sequence of $\{y_i\}$ is on a set of size

$$\prod_{i=0}^{n-1}(2^\lambda-|\mathsf{S}_{\mathcal{B}}^{\mathsf{T}}\cup\mathsf{s}_{\mathcal{B}}^{\mathsf{T}^{-1}}|-i)\geq(2^\lambda-\mathsf{poly}(\lambda))^n\geq\frac{2^{n\lambda}}{\mathsf{poly}(\lambda)}.$$

We can upper bound the probability that the correct sequence was picked out of this uniform set with $\frac{\mathsf{poly}(\lambda)}{2^{n\lambda}}$. □

Following the above claim, we observe that $\exists\mathsf{advice}$ which solves Lemma 1, then some E_{a} must have occured. We can union bound over all $2^{n\cdot\lambda-\omega(\log\lambda)+1}-1$ strings of length $\leq n\cdot\lambda-\omega(\log\lambda)$.

$$\Pr\left[\bigcup_{|\mathsf{a}|\leq n\cdot\lambda-\omega(\log\lambda)}\mathsf{E}_{\mathsf{a}}\right]\leq\sum_{|\mathsf{a}|\leq n\cdot\lambda-\omega(\log\lambda)}\Pr\left[\mathsf{E}_{\mathsf{a}}\right]\leq\left(2^{n\cdot\lambda-\omega(\log(\lambda))+1}-1\right)\cdot\frac{\mathsf{poly}(\lambda)}{2^{n\lambda}}\leq$$

$$\frac{2^{n\lambda}}{2^{\omega(\log(\lambda))-1}}\frac{2^{O(\log(\lambda))}}{2^{n\lambda}}=\frac{1}{2^{\omega(\log(\lambda))-O(\log(\lambda))}}=\frac{1}{2^{\omega(\log(\lambda))}}\in\mathsf{negl}(\lambda).$$

□

Lemma 2. *Let $\mathsf{T}':\{0,1\}^\lambda\to\{0,1\}^\lambda$ be an oracle to a random function. Consider any computationally unbounded adversary $\mathcal{B}$ that makes queries to T' on input a bounded* advice *and outputs n pairs (x_i,y_i) without querying them explicitly. If* advice *is bounded by $n\cdot\lambda-\omega(\log\lambda)$ bits, the probability that it succeeds is negligible in λ.*

More formally, let the inputs and outputs by $\mathcal{B}$ to oracle $\mathcal{O}$ be denoted by lists $\mathsf{s}_{\mathcal{B}}^{\mathcal{O}}$, $\mathsf{S}_{\mathcal{B}}^{\mathcal{O}}$ respectively. Then,

$$\Pr\left[\begin{array}{c}\exists\ \mathsf{advice}\in\{0,1\}^*\ s.t.\ |\mathsf{advice}|\leq n\cdot\lambda-\omega(\log\lambda),\\ \{(x_i,y_i)\}_{i=1}^n\leftarrow\mathcal{B}^{\mathsf{T}'(\cdot)}(\mathsf{advice})\ where\\ \forall i\neq j\in[n],x_i\neq x_j,\ \mathsf{T}'(x_i)=y_i,\ x_i\notin\mathsf{s}_{\mathcal{B}}^{\mathsf{T}'}\end{array}\right]\leq\mathsf{negl}(\lambda),$$

the probability is over the choice of the random function T' picked.

Proof. Let a be some fixed string , and let the event E_{a} be the event that $\mathcal{B}$ manages to produce n distinct (x_i,y_i) pairs such that $\mathsf{T}(x_i)=y_i$ without querying T on x_i when run on a.

Claim 2. $\Pr[\mathsf{E}_{\mathsf{a}}]\leq\frac{1}{2^{n\lambda}}$ *over a random* T'.

Proof. Suppose the sequence $\{x_i\}$ are unique and not in $\mathsf{s}_{\mathcal{B}}^{\mathsf{T}'}$. We know from definition of a random function that $\mathsf{T}'(x_i)$ are uniform and independent of all other queries on $\{0,1\}^\lambda$, so the probability that all $\{y_i\}$ are correct is simply $(2^{-\lambda})^n$. □

Just as before, to complete the proof of the lemma, we observe that for $\mathcal{B}$ to succeed in Lemma 2, some $\mathsf{E_a}$ must have occured, which we can simply union bound the probability $\mathcal{B}$ is correct on any advice string with

$$\Pr\left[\bigcup_{|\mathsf{a}|\leq n\cdot\lambda-\omega(\log\lambda)}\mathsf{E_a}\right]\leq\sum_{|\mathsf{a}|\leq n\cdot\lambda-\omega(\log\lambda)}\Pr\left[\mathsf{E_a}\right]\leq\left(2^{n\cdot\lambda-\omega(\log(\lambda))+1}-1\right)\cdot\frac{1}{2^{n\lambda}}=\mathsf{negl}(\lambda).$$

□

Definition 3. *Let π be a permutation or permutation oracle with domain $\mathcal{D}$, and let $x_1, x_2 \in \mathcal{D}$. We define the notation $\pi' = \pi[\mathsf{swap}(x_1, x_2)]]$ to imply π' to be same as π but swapped on points $x_1, \pi^{-1}(x_2)$. Concretely,*

$$\pi'(x)=\begin{cases}x_2 & x=x_1\\ \pi(x_1) & x=\pi^{-1}(x_2)\\ \pi(x) & \textit{otherwise.}\end{cases}$$

Lemma 3. *Let $S_\mathcal{D}$ denote the symmetric group on $\mathcal{D}$. Let $x, r \xleftarrow{R} \mathcal{D}$, and $\pi \xleftarrow{R} S_\mathcal{D}$. Then $(x, \pi[\mathsf{swap}(x, r)])$ is uniform on $\mathcal{D} \times S_\mathcal{D}$ - i.e. x is independent of $\pi[\mathsf{swap}(x, r)]$.*

Proof. We compute the probability $\pi[\mathsf{swap}(x, r)]$ is equal to some fixed permutation π_0 conditional on x as

$$\Pr_{r,\pi}[\pi[\mathsf{swap}(x,r)]=\pi_0|x]=\sum_{i\in\mathcal{D}}\Pr_{\pi}[\pi=\pi_0[\mathsf{swap}(x,i)]|x]\cdot\Pr_{\pi}[i=\pi(x)|x]$$

by independence of r and π. Let $N = |\mathcal{D}|$, this is equal to

$$\sum_{i\in[N]}\frac{1}{N!}\cdot\frac{1}{N}=N\cdot\frac{1}{N!}\cdot\frac{1}{N}=\frac{1}{N!}.$$

This is exactly equal to the probability of picking a T uniformly at random from the space of permutations. □

Definition 4. *Multiple invocations of the swap notation are defined as following $\pi[\mathsf{swap}(x_1, y_1), \ldots, \mathsf{swap}(x_k, y_k)]$:*

- *Let $\pi_0 = \pi$.*
- *Iterate from $i = 1$ to k,*
  - *Perform i^{th} swap, $\pi_i = \pi_{i-1}[\mathsf{swap}(x_i, y_i)]$.*
- *Output π_k.*

Lemma 4. *Let $\{r_0, r_1, \ldots r_k\} \xleftarrow{R} \mathcal{D}$ and π be a random permutation. Let $S_\mathcal{D}$ be the set of all permutations. Let τ be a fixed permutation on $\mathcal{D}$. Then*

$$(r_k, \pi[\mathsf{swap}(r_0, \tau(r_1)), \ldots, \mathsf{swap}(r_{k-1}, \tau(r_k))])$$

is uniform on $\mathcal{D} \times S_\mathcal{D}$.

Proof. This proceeds via a simple induction argument. For our base case, it is easy to see (r_0, π) is uniform on $\mathcal{D} \times S_\mathcal{D}$ by their independence. In the induction step, we assume that

$$(r_i, \pi[\mathsf{swap}(r_0, \tau(r_1)), \ldots, \mathsf{swap}(r_{i-1}, \tau(r_i))])$$

is uniform on $\mathcal{D} \times S_{\mathcal{D}}$, so r_i is uniform and independent of

$$\pi[\mathsf{swap}(r_0, \tau(r_1)), \ldots, \mathsf{swap}(r_{i-1}, \tau(r_i))].$$

Since τ is a permutation, this means $\tau(r_{i+1})$ is still uniform and independent of

$$\pi[\mathsf{swap}(r_0, \tau(r_1)), \ldots, \mathsf{swap}(r_{i-1}, \tau(r_i))]$$

as r_{i+1} clearly is, so by Lemma 3, we can say

$$(r_{i+1}, \pi[\mathsf{swap}(r_0, \tau(r_1)), \ldots, \mathsf{swap}(r_{i-1}, \tau(r_i))][\mathsf{swap}(r_i, \tau(r_{i+1}))]$$

$$= \pi[\mathsf{swap}(r_0, \tau(r_1)), \ldots, \mathsf{swap}(r_i, \tau(r_{i+1}))])$$

is uniform on $\mathcal{D} \times S_{\mathcal{D}}$.

□

Definition 5. *Let F be a function with domain $\mathcal{D}$ and codomain $\mathcal{C}$, and let $x_1 \in \mathcal{D}, x_2 \in \mathcal{C}$. We define the reprogramming shorthand $F' = F[\mathsf{reprog}(x_1, x_2)]$ to mean*

$$F'(x) = \begin{cases} x_2 & x = x_1 \\ F(x) & \textit{otherwise.} \end{cases}$$

Definition 6. *Multiple invocations of the* reprogram *notation are defined as following $F[\mathsf{reprog}(x_1, y_1), \ldots, \mathsf{reprog}(x_k, y_k)]$:*

- *Let $F_0 = F$.*
- *Iterate from $i = 1$ to k,*
  - *Perform i^{th} reprogramming, $F_i = F_{i-1}[\mathsf{reprog}(x_i, y_i)]$.*
- *Output F_k.*

Lemma 5. *Let $x_1, x_2, r \xleftarrow{R} \mathcal{D}$, and $F \xleftarrow{R} (\mathcal{D} \to \mathcal{D})$[2]. If $F' = F[\mathsf{reprog}(x_1, x_2 \oplus r)]$, then (x_1, x_2, F') is uniform on $\mathcal{D} \times \mathcal{D} \times (\mathcal{D} \to \mathcal{D})$.*

Proof. We compute the probability $F[\mathsf{reprog}(x_1, x_2 \oplus r)]$ is equal to some fixed function F_0 conditional on x_1, x_2 as

$$\Pr_{r,F'}[F' = F_0 | x_1, x_2] = \sum_{i \in \mathcal{D}} \Pr_F[F = F_0[\mathsf{reprog}(x_1, i)] | x_1, x_2] \cdot \Pr_r[r = i \oplus x_2 | x_1, x_2]$$

by independence of r. Let $N = |\mathcal{D}|$, this is equal to

$$\sum_{i \in [N]} \frac{1}{N^N} \cdot \frac{1}{N} = N \cdot \frac{1}{N^N} \cdot \frac{1}{N} = \frac{1}{N^N}$$

which is exactly the probability of picking an F uniformly at random from the space of functions. □

Lemma 6. *Let $\{r_0, r_1, \ldots, r_k\} \xleftarrow{R} \mathcal{D}$ and $F \xleftarrow{R} (\mathcal{D} \to \mathcal{D})$. Let τ be a fixed permutation. If $F' = F[\mathsf{reprog}(r_1, r_0 \oplus \tau(r_2)), \ldots, \mathsf{reprog}(r_{k-1}, r_{k-2} \oplus \tau(r_k))]$, then (r_{k-1}, r_k, F') is uniform on $\mathcal{D} \times \mathcal{D} \times (\mathcal{D} \to \mathcal{D})$.*

[2] $(\mathcal{D} \to \mathcal{D})$ denotes the set of all functions from domain $\mathcal{D}$ to range $\mathcal{D}$

Proof. This proceeds via a simple induction argument. For our base case, it is easy to see (r_0, r_1, F) is uniform on $\mathcal{D} \times \mathcal{D} \times (\mathcal{D} \to \mathcal{D})$ by their independence.

In the induction step, If we assume that

$$(r_i, r_{i+1}, F[\mathsf{reprog}(r_1, r_0 \oplus \tau(r_2)), \ldots, \mathsf{reprog}(r_i, r_{i-1} \oplus \tau(r_{i+1}))])$$

is uniform on $\mathcal{D} \times \mathcal{D} \times (\mathcal{D} \to \mathcal{D})$, so r_i, r_{i+1} are uniform and independent of

$$F[\mathsf{reprog}(r_1, r_0 \oplus \tau(r_2)), \ldots, \mathsf{reprog}(r_i, r_{i-1} \oplus \tau(r_{i+1}))].$$

So we can apply our Lemma 5 to $x_1 = r_{i+1}, x_2 = \tau(r_{i+2})$ and $r = r_i$ to conclude that

$$(r_{i+1}, \tau(r_{i+2}), F[\mathsf{reprog}(r_1, r_0 \oplus \tau(r_2)), \ldots, \mathsf{reprog}(r_{i+1}, r_i \oplus \tau(r_{i+2}))])$$

is uniform. Since τ is a fixed permutation, this means

$$(r_{i+1}, r_{i+2}, F[\mathsf{reprog}(r_1, r_0 \oplus \tau(r_2)), \ldots, \mathsf{reprog}(r_{i+1}, r_i \oplus \tau(r_{i+2}))])$$

is as well. □

We introduce another useful result on the probability of finding collisions on a deterministic function h.

Lemma 7. *Let $\mathcal{D}(\kappa), \mathcal{R}(\kappa)$ represent domain,range respectively dependent on the security parameter. Let h be any deterministic function that maps values in domain $\mathcal{D}(\kappa)$ to range $\mathcal{R}(\kappa)$. Then,*

$$\Pr_a[\exists b \neq a \in \mathcal{D}(\kappa), \mathsf{h}(a) = \mathsf{h}(b)] \geq \frac{|\mathcal{D}(\kappa)| - |\mathcal{R}(\kappa)| + 1}{|\mathcal{D}(\kappa)|}.$$

Proof. Let S_{h} be the bad set for a function h. $S_{\mathsf{h}} = \{a | \mathsf{h}(a)$ has just one preimage$\}$. The cardinality of the set is upper bounded by $|\mathcal{R}| - 1$. If the set S_{h} was equal to the full range, each element in the range has one preimage and the rest of the elements in the domain cannot map to any choice in the range. Thus, $\Pr_a[a \in S_{\mathsf{h}}] = \frac{|\mathcal{R}(\kappa)|-1}{|\mathcal{D}(\kappa)|}$. □

5 Replica Encoding in the Ideal Permutation Model.

We now give the construction and proof of our replica encoding scheme from trapdoor permutations in the ideal permutation model. As stated in the introduction, the construction itself is a close variant of [7]. However, our proof will introduce new analysis techniques that account for an attacker that stores state in an arbitrary manner.

Let κ denote the security parameter. Let $\lambda(\kappa)$ (denoted by λ) be a function polynomial in κ and represents block length in our construction. We use a trapdoor permutation $(\mathsf{KeyGen}, \mathsf{f}(.,.)\mathsf{f}^{-1}(.,.))$ where the domain for the family of trapdoor functions is $\mathcal{D}_{\mathsf{pk}} = \{0,1\}^\lambda$ where KeyGen is setup with security parameter κ. Let $\mathsf{T}, \mathsf{T}^{-1}$ be random permutation oracles on the same domain $\{0,1\}^\lambda$.

5.1 Construction

Let $r(\kappa, n, |m|)$ (denoted by r) be the number of rounds in our scheme. For our construction, it depends on the security parameter, maximum number of replicas chosen during setup and the message length.

$\underline{\mathsf{rSetup}(1^\kappa, 1^n)}$**:**

Run $\mathsf{KeyGen}(1^\kappa) \to (\mathsf{pk}, \mathsf{sk})$. Output $(\mathsf{pk}' = (\mathsf{pk}, n), \mathsf{sk}' = (\mathsf{sk}, n))$.

$\underline{\mathsf{rEnc}(\mathsf{sk}', m)}$**:**

- Parse $\mathsf{sk}' = (\mathsf{sk}, n)$.

- Choose a string $\rho \xleftarrow{R} \{0,1\}^\kappa$.
- Divide m into b blocks of length λ i.e. $m = m_1||m_2||\dots||m_b$, $b = \lceil |m|/\lambda \rceil$.
- Set $r = n \cdot b \cdot \lambda$.
- Compute $\forall t \in [b]$,
$$Y_{t,0} = m_t \oplus \mathsf{H}(\rho||t).$$
  - For rounds j from 1 to r, compute:
$$Y_{t,j} = \mathsf{f}^{-1}(\mathsf{sk}, \mathsf{T}(Y_{t,j-1})).$$
- Let $y_r = Y_{1,r}||\dots||Y_{b,r}$ and output (y_r, ρ).

$\underline{\mathsf{rDec}(\mathsf{pk}', y)}$**:**

- Parse $\mathsf{pk}' = (\mathsf{pk}, n)$.
- Parse y as (y_r, ρ). Parse y_r as $Y_{1,r}||\dots||Y_{b,r}$, where $b = \lceil |y_r|/\lambda \rceil$ and $r = n \cdot b \cdot \lambda$.
- For rounds j from $r-1$ to 0:
  - Compute $\forall t \in [b]$,
$$Y_{t,j} = \mathsf{T}^{-1}(\mathsf{f}(\mathsf{pk}, Y_{t,j+1})).$$
- $\forall t \in [b]$ compute,
$$m_t = Y_{t,0} \oplus \mathsf{H}(\rho||t)$$
  Output $m = m_1||\dots||m_b$.

The encoding length for our scheme is $\mathsf{len}(\kappa, |m|) = |m| + O(\kappa)$.[3]

5.2 Security of Replica Encoding Scheme

Theorem 1. *Assuming* $(\mathsf{KeyGen}(1^\kappa), \mathsf{f}(\cdot,\cdot), \mathsf{f}^{-1}(\cdot,\cdot))$ *is a secure trapdoor permutation and* $\mathsf{T}, \mathsf{T}^{-1}$ *are oracles to a random permutation on domain and range* $\{0,1\}^\lambda$ *and* H *is a random oracle on the same range. Then our construction for* ReplicaEncoding *described above is* s*-sound according to Definition 2 for all* $\kappa, n \in \mathbb{N}$ *and* $\mathsf{s} \in 1 - \frac{\omega(\log \kappa)}{\lambda}$.

5.2.1 Sequence of Games

Our proof proceeds via a sequence of games as described below. We assume that adversaries have their randomness non-uniformly fixed in each game to maximize their success. The changes in each game in comparison to the previous one are indicated with red. Details of the previous game are copied without explicit rewriting.

Game 0: This is the original $\mathsf{Sound}_{\mathcal{A}_1,\mathcal{A}_2}(\kappa, n)$ security game where we record the queries made by the adversaries in lists. We also assume that any list is ordered and stores distinct elements. More concretely, when in Phase 1 a query x is made on $\mathcal{O}$, $\mathcal{C}$ checks if $x \notin \mathbf{u}^{\mathcal{O}}$ and updates the list $\mathbf{u}^{\mathcal{O}}$ if the condition is true. It performs this operation of maintaining the list for each Phase and oracle separately. Denote $\mathsf{q_1}^{\mathcal{O}}, \mathsf{q_2}^{\mathcal{O}}, \mathsf{q_3}^{\mathcal{O}}$ as the functions that take in the security parameter and output the total distinct queries made by the adversaries to oracle $\mathcal{O}$ during the three phases respectively.

[3] Upto additional rounding factors.

- **Setup:** The challenger(denoted by $\mathcal{C}$) runs $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{rSetup}(1^\kappa, 1^n)$ and sends public key pk' to $\mathcal{A}_1$. It keeps the secret key sk' for itself.
- **Phase 1:** The adversary $\mathcal{A}_1$ issues queries on $\mathsf{T}, \mathsf{T}^{-1}, \mathsf{H}$, $\mathcal{C}$ responds the query back to $\mathcal{A}_1$. Let the queries on oracle $\mathcal{O}$ be denoted by an ordered and distinct list $\mathbf{u}^{\mathcal{O}} = (\mathsf{u}_1^{\mathcal{O}}, \ldots, \mathsf{u}_{\mathsf{q}_1\mathcal{O}}^{\mathcal{O}})$ and their outputs be denoted by an ordered and distinct list $\mathbf{U}^{\mathcal{O}} = (\mathsf{U}_1^{\mathcal{O}}, \ldots, \mathsf{U}_{\mathsf{q}_1\mathcal{O}}^{\mathcal{O}})$.
- **File Challenge:** $m \in \{0,1\}^* \leftarrow \mathcal{A}_1^{\mathsf{H}(\cdot),\mathsf{T}(\cdot),\mathsf{T}^{-1}(\cdot)}(1^\kappa, \mathsf{pk}')$. It sends m to $\mathcal{C}$ who parses pk' as (pk, n); sk' as (sk, n) and does the following:
  - Divide m into b blocks of length λ i.e. $m = m_1||m_2||\ldots||m_b$, $b = \lceil |m|/\lambda \rceil$.
  - For $i \in [n]$,
    - Choose a string $\rho_i \xleftarrow{R} \{0,1\}^\kappa$.
    - Compute $\forall t \in [b]$,
    $$Y_{t,0}^{(i)} = m_t \oplus \mathsf{H}(\rho_i||t).$$
    - For rounds j from 1 to r and $\forall t \in [b]$,
      - Compute $Y_{t,j}^{(i)}$ from $Y_{t,j-1}^{(i)}$ as
      $$Y_{t,j}^{(i)} = \mathsf{f}^{-1}(\mathsf{sk}, \mathsf{T}(Y_{t,j-1}^{(i)})).$$
    - Let $y_r^{(i)} = Y_{1,r}^{(i)}||\ldots||Y_{b,r}^{(i)}$ and set $y^{(i)} = (y_r^{(i)}, \rho_i)$.

  $\mathcal{C}$ returns $y^{(1)}, y^{(2)}, \ldots y^{(n)}$ to $\mathcal{A}_1$.
- **Phase 2:** $\mathcal{A}_1$ issues additional queries on $\mathsf{T}, \mathsf{T}^{-1}, \mathsf{H}$, $\mathcal{C}$ responds the query back to $\mathcal{A}_1$. Let the queries on oracle $\mathcal{O}$ be denoted by an ordered and distinct list $\mathbf{v}^{\mathcal{O}} = (\mathsf{v}_1^{\mathcal{O}}, \ldots, \mathsf{v}_{\mathsf{q}_2\mathcal{O}}^{\mathcal{O}})$ and their outputs be denoted by an ordered and distinct list $\mathbf{V}^{\mathcal{O}} = (\mathsf{V}_1^{\mathcal{O}}, \ldots, \mathsf{V}_{\mathsf{q}_2\mathcal{O}}^{\mathcal{O}})$.
- **State Sharing:** $\mathcal{A}_1$ outputs state $\mathsf{state} \leftarrow \mathcal{A}_1^{\mathsf{H}(\cdot),\mathsf{T}(\cdot),\mathsf{T}^{-1}(\cdot)}(1^\kappa, \mathsf{pk}', y)$ and sends state to $\mathcal{A}_2$.
- **Phase 3:** The adversary $\mathcal{A}_2$ queries on $\mathsf{T}, \mathsf{T}^{-1}, \mathsf{H}$, $\mathcal{C}$ responds the query back to $\mathcal{A}_2$. Let the queries on oracle $\mathcal{O}$ be denoted by an ordered and distinct list $\mathbf{w}^{\mathcal{O}} = (\mathsf{w}_1^{\mathcal{O}}, \ldots, \mathsf{w}_{\mathsf{q}_3\mathcal{O}}^{\mathcal{O}})$ and their outputs be denoted by an ordered and distinct list $\mathbf{W}^{\mathcal{O}} = (\mathsf{W}_1^{\mathcal{O}}, \ldots, \mathsf{W}_{\mathsf{q}_3\mathcal{O}}^{\mathcal{O}})$.
- **Guess:** $\mathcal{A}_2$ outputs the replica guesses to $\mathcal{C}$.
$$\{\tilde{y}^{(i)}\} \leftarrow \mathcal{A}_2(1^\kappa, \mathsf{pk}', m, \mathsf{state}).$$
- **Verify:** Let $v_i = 1$ if $\tilde{y}^{(i)} = y^{(i)}$ and 0 otherwise. Adversary wins if $|\mathsf{state}| < \sum v_i \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)$.

Game 1 : In this game we remove the sk and rely on the public key with an additional reprogramming step at oracle H. This helps us further down the road in showing a reduction to the security of the trapdoor permutation.

- **Setup:** The challenger(denoted by $\mathcal{C}$) runs $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{rSetup}(1^\kappa, 1^n)$ and sends public key pk' to $\mathcal{A}_1$. It keeps the secret key sk' for itself. Set $\mathsf{flag} = 0$.
- **Phase 1:** ...
- **File Challenge:** $m \in \{0,1\}^* \leftarrow \mathcal{A}_1^{\mathsf{H}(\cdot),\mathsf{T}(\cdot),\mathsf{T}^{-1}(\cdot)}(1^\kappa, \mathsf{pk}')$. It sends m to $\mathcal{C}$ who parses pk' as (pk, n); sk' as (sk, n) and does the following:
  - Divide m into b blocks of length λ i.e. $m = m_1||m_2||\ldots||m_b$, $b = \lceil |m|/\lambda \rceil$.
  - For $i \in [n]$,
    - Choose a string $\rho_i \xleftarrow{R} \{0,1\}^\kappa$.
    Prequery Check H If $\exists t \in [b] : \ \rho_i||t \in \mathbf{u}^{\mathsf{H}}$, set $\mathsf{flag} = 1$.
    - Sample $\{Y_{t,r}^{(i)}\}_{t\in[b]} \xleftarrow{R} \{0,1\}^\lambda$
    - For rounds j from r to 1 and $\forall t \in [b]$,
      - Compute $Y_{t,j-1}^{(i)}$ from $Y_{t,j}^{(i)}$ as
      $$Y_{t,j-1}^{(i)} = \mathsf{T}^{-1}(\mathsf{f}(\mathsf{pk}, Y_{t,j-1}^{(i)})).$$

* For each block $\forall t \in [b]$, reprogram H

$$\mathsf{H}(\rho_i||t) = m_t \oplus Y_{t,0}^{(i)}$$

.

* Let $y_r^{(i)} = Y_{1,r}^{(i)}||\ldots||Y_{b,r}^{(i)}$ and set $y^{(i)} = (y_r^{(i)}, \rho_i)$.

$\mathcal{C}$ returns $y^{(1)}, y^{(2)}, \ldots y^{(n)}$ to $\mathcal{A}_1$.

- **Phase 2, State Sharing, Phase 3, Guess:** ...
- **Verify:** Let $v_i = 1$ if $\tilde{y}^{(i)} = y^{(i)}$ and 0 otherwise. Adversary wins if $\mathsf{flag} = 0$ and $|\mathsf{state}| < \sum v_i \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)$.

Game 2: In this game an adversary wins if they query on the oracle rather than outputting the replica. This helps us ease the notation by only focussing at the oracle query lists.

- **Setup, Phase 1, File Challenge, Phase 2, State Sharing, Phase 3:** ....
- **Guess:**....
  $\mathcal{C}$ adds the guess to $\mathcal{A}_2$'s lists of queries to T in Phase 3, i.e. $\forall i \in [n]$, let $\tilde{y}^{(i)} = (\tilde{Y}_{0,r}^{(i)}||\ldots||\tilde{Y}_{b,r}^{(i)}, \tilde{\rho}_i)$. $\forall t \in [b]$ add $\tilde{Y}_{t,r}^{(i)}$ to list of queries to T by $\mathcal{A}_2$ in Phase 3.
- **Verify:** Let $v_i = 1$ if $\forall t \in [b]$, T is queried on $Y_{t,r}^{(i)}$ and 0 otherwise. Adversary wins if $\mathsf{flag} = 0$ and $|\mathsf{state}| < \sum v_i \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)$.

Game 3: In this game, we look at the queries made by the adversary and require that it traverses atleast one block in some replica sequentially.

- **Setup, Phase 1, File Challenge, Phase 2, State Sharing, Phase 3, Guess:** ....
- **Sequentiality:**
  We consider going through $\mathcal{A}_2's$ ordered list of queries to T and T^{-1}. If $\forall i \in [n]$ $\forall t \in [b]$, there is a point in time such that some $Y_{t,j+1}^{(i)}$ was queried on T or $\mathsf{f}(\mathsf{pk}, Y_{t,j+1}^{(i)})$ was queried on T^{-1} when $\mathcal{A}_2$ has not made a query to T for $Y_{t,j}^{(i)}$), then set $\mathsf{flag} = 1$.
- **Verify:** Let $v_i = 1$ if $\forall t \in [b]$, T is queried on $Y_{t,r}^{(i)}$ and 0 otherwise. Adversary wins if $\mathsf{flag} = 0$ and $|\mathsf{state}| < \sum v_i \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)$.

Game 4: In this game, we guess the block which the adversary traversed sequentially. We concentrate on one randomly chosen block and replica and the adversary wins if it outputs the correct encoding for this block. We lose a multiplicative factor of $b \cdot n$ in the reduction due to this change.

- **Setup:** The challenger(denoted by $\mathcal{C}$) runs $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{rSetup}(1^\kappa, 1^n)$ and sends public key pk' to $\mathcal{A}_1$. It keeps the secret key sk' for itself. Set $\mathsf{flag} = 0$.
  Choose a random $\beta \in [b]$ and $\gamma \in [n]$.
- **Phase 1, File Challenge, Phase 2, State Sharing, Phase 3, Guess:** ....
- **Sequentiality:**
  We consider going through $\mathcal{A}_2's$ list of queries to T and T^{-1}. If there is a point in time such that some $Y_{\beta,j+1}^{(\gamma)}$ was queried on T or $\mathsf{f}(\mathsf{pk}, Y_{\beta,j+1}^{(\gamma)})$ was queried on T^{-1} when $\mathcal{A}_2$ has not made a query to T for $Y_{\beta,j}^{(\gamma)}$, then set $\mathsf{flag} = 1$.
- **Verify:** ~~Let $v_i = 1$ if $\forall t \in [b]$, T is queried on $Y_{t,r}^{(i)}$ and 0 otherwise.~~ Adversary wins if T is queried on $Y_{\beta,r}^{(\gamma)}$, $\mathsf{flag} = 0$, and $|\mathsf{state}| < n \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)$.

Game 5: In this game, we reprogramme the oracles H,T to have a permutation which we can analyze cleanly. The primary idea behind this game is that there will exist two sequences of values on the chosen block and replica for which any adversary $\mathcal{A}_1$ produces the same state. These possibilities for a "switch" are set up in this game. H is programmed to output $Y_{\beta,0}^{(\gamma)}$ and for $i \in [r]$, the values $A_{i,0}, A_{i,1}$ have a choice to be mapped to either of the two

$A_{i+1,0}, A_{i+1,1}$ depending on the sampled index x. The collision check makes sure that the reprogramming preserves the permutation property of T and the prequery check is done to make sure that none of the values were queried in the oracle lists in the previous phase. The oracle T_x is then reprogrammed according to the swap operation defined in Definition 4 where for $i \in [r]$, x_i is now mapped to $\mathsf{f}(\mathsf{pk}, \mathsf{x}_{i+1})$ where x_i is used to indicate the notation for $A_{i,\mathsf{x}[i]}$.

- **Setup, Phase 1:** ....
- **Sampling a new permutation:**
  - Sample, $Y^{(\gamma)}_{\beta,0}, A_{1,0}, \ \ldots\ , A_{r,0}, A_{1,1}, \ \ldots\ , A_{r,1} \xleftarrow{R} \{0,1\}^\lambda$.
    Let $\mathcal{Z}_1 = \{Y^{(\gamma)}_{\beta,0}, A_{1,0} \ldots, A_{r,0}, A_{1,1}, \ldots, A_{r,1}\}$.
    Let $\mathcal{Z}_2 = \{\mathsf{f}(\mathsf{pk}, A_{1,0}) \ldots, \mathsf{f}(\mathsf{pk}, A_{r,0}), \mathsf{f}(\mathsf{pk}, A_{1,1}), \ldots, \mathsf{f}(\mathsf{pk}, A_{r,1})\}$.
    Collision Check: If $|\mathcal{Z}_1| \neq 2r+1$, set $\mathsf{flag} = 1$.
    Prequery Check T: If $(\mathcal{Z}_1 \cup \mathcal{Z}_2) \cap \left(\mathbf{u}^\mathsf{T} \cup \mathbf{u}^{\mathsf{T}^{-1}} \cup \mathbf{U}^\mathsf{T} \cup \mathbf{U}^{\mathsf{T}^{-1}}\right) \neq \emptyset$, set $\mathsf{flag} = 1$.
  - Sample a random setting $\mathsf{x} \xleftarrow{R} \{0,1\}^r$. Let $\mathsf{x}[k]$ denote the k^{th} bit of x. We will write x_j to refer to $A_{j,\mathsf{x}[j-1]}$ and denote $A_{j,\bar{\mathsf{x}}[j-1]}$ with $\bar{\mathsf{x}}_j$. Set x_0 to denote $Y^{(\gamma)}_{\beta,0}$.
  - Define T_x using swap (Definition 4):
    $$\mathsf{T}_\mathsf{x} = \mathsf{T}[\mathsf{swap}(\mathsf{x}_0, \mathsf{f}(\mathsf{pk}, \mathsf{x}_1)), \ldots, \mathsf{swap}(\mathsf{x}_{r-1}, \mathsf{f}(\mathsf{pk}, \mathsf{x}_r))].$$
  - Let $\mathsf{T}_\mathsf{x}^{-1}$ be the inverse of T_x.
- **File Challenge:** $m \in \{0,1\}^* \leftarrow \mathcal{A}_1^{\mathsf{H}(\cdot),\mathsf{T}(\cdot),\mathsf{T}^{-1}(\cdot)}(1^\kappa, \mathsf{pk}')$. It sends m to $\mathcal{C}$ who parses pk' as (pk, n); sk' as (sk, n) and does the following:
  - Divide m into b blocks of length λ i.e. $m = m_1||m_2|| \ldots ||m_b$, $b = \lceil |m|/\lambda \rceil$.
  - For $i \in [n]$,
    - Choose a string $\rho_i \xleftarrow{R} \{0,1\}^\kappa$.
      Prequery Check H If $\exists t \in [b]:\ \rho_i||t \in \mathbf{u}^\mathsf{H}$, set $\mathsf{flag} = 1$.
    - Sample $\{Y^{(i)}_{t,r}\}_{t\in[b]} \xleftarrow{R} \{0,1\}^\lambda$
    - For rounds j from r to 1 and $\forall t \in [b]$, continue if $t \neq \beta$ or $i \neq \gamma$,
      - Compute $Y^{(i)}_{t,j-1}$ from $Y^{(i)}_{t,j}$ as
        $$Y^{(i)}_{t,j-1} = \mathsf{T}^{-1}(\mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j-1})).$$
    - For each block $\forall t \in [b]$, reprogram H
      $$\mathsf{H}(\rho_i||t) = m_t \oplus Y^{(i)}_{t,0}.$$
    - Let $y^{(i)}_r = Y^{(i)}_{1,r}|| \ldots ||Y^{(i)}_{b,r}$ and set $y^{(i)} = (y^{(i)}_r, \rho_i)$.

  $\mathcal{C}$ returns $y^{(1)}, y^{(2)}, \ldots y^{(n)}$ to $\mathcal{A}_1$.
- **Phase 2:** Use $\mathsf{T}_\mathsf{x}, \mathsf{T}_\mathsf{x}^{-1}$ to answer queries for $\mathsf{T}, \mathsf{T}^{-1}$ respectively.
- **State Sharing:** ....
- **Phase 3:** Use $\mathsf{T}_\mathsf{x}, \mathsf{T}_\mathsf{x}^{-1}$ to answer queries for $\mathsf{T}, \mathsf{T}^{-1}$ respectively.
- **Guess, Sequentiality:** ....
- **Verify:** Adversary wins if T is queried on $Y^{(\gamma)}_{\beta,r}$, $\mathsf{flag} = 0$, and $|\mathsf{state}| < n \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)$.

Game 6: In this game, $\mathcal{C}$ has unbounded computation time and calls $\mathcal{A}_1, \mathcal{A}_2$ exponentially many times to find a collision to state through the procedure search. The setting y' for which the procedure search outputs a collision in state is stored in a set which is outputted at the end of the procedure. $\mathsf{search}(1^\kappa, \mathsf{y}, \mathsf{state}; \zeta)$ takes input $\mathsf{y} \in \{0,1\}^r, \mathsf{state}$ and runs algorithms $\mathcal{A}_1, \mathcal{A}_2$ on Game 5. Let ζ be the randomness used by the procedure and denotes all the random coins (except those used to sample x) used by $\mathcal{C}$. The procedure is described in Figure 2.

$\mathsf{search}(1^\kappa, \mathsf{y}, \mathsf{state}; \zeta)$

Inputs: Security parameter - 1^κ
Oracle Settings on T - $\mathsf{y} \in \{0,1\}^r$
State - state
Randomness used in the game - ζ

Output: Set containing all oracle settings with collision in state - $\mathcal{S}$

- Set $\mathcal{S} = \emptyset$.
- $\forall \mathsf{y}' \neq \mathsf{y} \in \{0,1\}^r$,
  - Run $\mathcal{A}_1, \mathcal{A}_2$ on Game 5 with randomness defined by ζ and using y' instead of x in the game.
  - Let state' be the state shared between $\mathcal{A}_1, \mathcal{A}_2$.
  - If $\mathsf{state}' = \mathsf{state}$ and $\mathcal{A}_2$ wins Game 5, then $\mathcal{S} = \mathcal{S} \cup \{\mathsf{y}'\}$.
- Output $\mathcal{S}$.

Figure 2: Routine search

- **Setup, Phase 1, Sampling a New Permutation, File Challenge, Phase 2, State Sharing:** ....
- **Running search:** Let ζ be all the random coins (except those used to sample x) used by $\mathcal{C}$. Let $\mathcal{S} \leftarrow \mathsf{search}(1^\kappa, \mathsf{x}, \mathsf{state}; \zeta)$.
  If $\mathcal{S} = \emptyset$ set $\mathsf{flag} = 1$ and $\mathsf{x}' = \mathsf{x}$, otherwise sample $\mathsf{x}' \xleftarrow{R} \mathcal{S}$.
- **Setting switched oracle:**
  - Let $\mathsf{x}'[k]$ denote the k^{th} bit of x'. We will write x'_j to refer to $A_{j,\mathsf{x}'[j-1]}$ and denote $A_{j,\bar{\mathsf{x}}'[j-1]}$ with $\bar{\mathsf{x}}'_j$. Set x'_0 to denote $Y^{(\gamma)}_{\beta,0}$.
  - Define $\mathsf{T}_{\mathsf{x}'}$ to be:
  $$\mathsf{T}_{\mathsf{x}'} = \mathsf{T}[\mathsf{swap}(\mathsf{x}'_0, \mathsf{f}(\mathsf{pk}, \mathsf{x}'_1)), \ldots, \mathsf{swap}(\mathsf{x}'_{r-1}, \mathsf{f}(\mathsf{pk}, \mathsf{x}'_r))].$$
  - Let $\mathsf{T}^{-1}_{\mathsf{x}'}$ be the inverse of $\mathsf{T}_{\mathsf{x}'}$.
- **Phase 3:** Use $\mathsf{T}_{\mathsf{x}'}, \mathsf{T}^{-1}_{\mathsf{x}'}$ to answer queries for $\mathsf{T}, \mathsf{T}^{-1}$ respectively.
- **Guess:** ....
- **Sequentiality:**
  If $\exists j \in [0, r]$:(x'_{j+1} was queried on T or $\mathsf{f}(\mathsf{pk}, \mathsf{x}'_{j+1})$ was queried on T^{-1} while T had not been queried on x'_j), set $\mathsf{flag} = 1$.
- **Verify:** Adversary wins if T is queried on x'_r, $\mathsf{flag} = 0$, and $|\mathsf{state}| < n \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)$.

Game 7: In this game we modify the verification step for which an adversary can win this game. We increase it's winning probability so that the adversary can win if it doesn't query the full sequence, but queries at the point where the sequences x, x' diverge. Notice that we define another oracle $\mathsf{T}^{\delta}_{\mathsf{x}'}$ here that doesn't reprogram the complete sequence. This change is statistically indistinguishable to the adversary.

- **Setup, Phase 1, Sampling a New Permutation, File Challenge, Phase 2, State Sharing, Running search:** ....
- **Setting switched oracle:**
  - Let $\mathsf{x}'[k]$ denote the k^{th} bit of x'. We will write x'_j to refer to $A_{j,\mathsf{x}'[j-1]}$ and denote $A_{j,\bar{\mathsf{x}}'[j-1]}$ with $\bar{\mathsf{x}}'_j$. Set x'_0 to denote $Y^{(\gamma)}_{\beta,0}$.
  - Let δ be the first index for which $\mathsf{x}_\delta \neq \mathsf{x}'_\delta$.

– Define $\mathsf{T}^{\delta}_{\mathsf{x}'}$ to be:

$$\begin{aligned}\mathsf{T}^{\delta}_{\mathsf{x}'} &= \mathsf{T}[\mathsf{swap}(\mathsf{x}'_0, \mathsf{f}(\mathsf{pk}, \mathsf{x}'_1)), \ldots, \mathsf{swap}(\mathsf{x}'_{\delta-1}, \mathsf{f}(\mathsf{pk}, \mathsf{x}'_{\delta}))] \\ &= \mathsf{T}[\mathsf{swap}(\mathsf{x}_0, \mathsf{f}(\mathsf{pk}, \mathsf{x}_1)), \ldots, \mathsf{swap}(\mathsf{x}_{\delta-1}, \mathsf{f}(\mathsf{pk}, \bar{\mathsf{x}}_{\delta}))].\end{aligned}$$

– Let $\mathsf{T}^{-1}_{\mathsf{x}'}$ be the inverse of $\mathsf{T}_{\mathsf{x}'}$.

- **Phase 3, Guess:** ....
- ~~**Sequentiality:**~~
- **Verify:** Adversary wins of T is queried on $\bar{\mathsf{x}}_{\delta}$, $\mathsf{flag} = 0$ ~~and $|\mathsf{state}| < n \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)$.~~

Game 8: In this game we observe that $\mathcal{C}$ need not be unbounded computation time and only needs to the guess the first prefix at which x, x' differ to successfully output one sequential query.

- **Setup, Phase 1, Sampling a New Permutation, File Challenge, Phase 2, State Sharing:** ....
- ~~**Running search:** ....~~
- **Setting switched oracle:**
  - ~~Let $\mathsf{x}'[k]$ denote the k^{th} bit of x'. We will write x'_j to refer to $A_{j,\mathsf{x}'[j-1]}$ and denote $A_{j,\mathsf{x}'[j-1]}$ with $\bar{\mathsf{x}}'_j$. Set x'_0 to denote $Y^{(\gamma)}_{\beta,0}$.~~
  - Let $\delta \xleftarrow{R} [r]$
  - Define $\mathsf{T}^{\delta}_{\mathsf{x}'}$ to be:
$$\mathsf{T}^{\delta}_{\mathsf{x}'} = \mathsf{T}[\mathsf{swap}(\mathsf{x}_0, \mathsf{f}(\mathsf{pk}, \mathsf{x}_1)), \ldots, \mathsf{swap}(\mathsf{x}_{\delta-1}, \mathsf{f}(\mathsf{pk}, \bar{\mathsf{x}}_{\delta}))].$$
  - Let $\mathsf{T}^{-1}_{\mathsf{x}'}$ be the inverse of $\mathsf{T}_{\mathsf{x}'}$.
- **Phase 3, Guess:** ....
- **Verify:** Adversary wins of T is queried on $\bar{\mathsf{x}}_{\delta}$ and $\mathsf{flag} = 0$.

5.2.2 Indistinguishability of Games

Let $\mathsf{F}_i(\kappa)$ (denoted by F_i) be the probability that the adversaries win at the end of Game i.

Game 0

Proof. Game 0 is a restatement of the original $\mathsf{Sound}_{\mathcal{A}_1,\mathcal{A}_2}(\kappa, n)$ game with two differences, (i) the syntactical change to note down queries to each oracle, (ii) expands on rEnc. Both the syntactical changes do not change the functioning of the game. □

Game 1

Lemma 8. $\Pr[\mathsf{F}_1] \geq \Pr[\mathsf{F}_0] - \mathsf{negl}(\kappa)$.

Proof. Game 1 differs from Game 0 in how queries are answered to the adversaries and the possibility of flag being set. Let $\Pr[\mathsf{F}_0] = \epsilon$ be the probability of adversary winning in Game 0. Let E^{H} be the event flag is set due to **Prequery Check** H. We can upper bound the difference in probability of F_1 and F_0 by the sum of (i) the probability that E^{H} occurs, (ii) and the statistical difference of the output of $\mathcal{C}$ from the alternate method of encoding generation.

Claim 3. $\Pr[\mathsf{E}^{\mathsf{H}}] = \mathsf{negl}(\kappa)$.

Proof. Since each ρ_i is generated uniformly on $\{0,1\}^{\kappa}$ and independent of $\mathcal{A}_1$, we can bound the probability that any fixed query is equal to a particular ρ_i as $k \in [\mathsf{q_1}^{\mathsf{H}}]$ $i \in [n]$ $\Pr[\mathsf{u}^{\mathsf{H}}_k = \rho_i] = 2^{-\kappa}$. From this, we can union bound the

$$\Pr[\exists k \in [\mathsf{q_1}^{\mathsf{H}}]\ i \in [n]\ t \in [b] : \mathsf{u}^{\mathsf{H}}_k = \rho_i || t] \leq \mathsf{q_1}^{\mathsf{H}} \cdot n \cdot b \cdot 2^{-\kappa}$$

Since $\mathsf{q_1}^{\mathsf{H}}, n, b \in \mathsf{poly}(\kappa)$, this is negligible □

Claim 4. *The distribution of* $\mathsf{T} \times \{Y_{t,r}^{(i)}\}$ *in Game* 0 *is statistically close to uniform.*

Proof. First, observe that with probability $1 - \mathsf{negl}(\kappa)$, $\forall i \in [n], t \in [b]$ $\rho_i||t$ is not queried H on by $\mathcal{A}_1$ before submitting m. This is apparent as each ρ_i is uniformly random on a domain of size 2^κ and $\mathcal{A}_1$ can make at most $\mathsf{poly}(\kappa)$ queries. Since $Y_{t,0}^{(i)} = \mathsf{H}(\rho_i||t) \oplus m_t$, and m is independent of $\mathsf{H}(\rho_i||t)$, we can say $\{Y_{t,0}^{(i)}\}$ is uniform and independent of at least a $1 - \mathsf{negl}(\kappa)$ fraction of T, and once an T is fixed, this defines a bijective relation from $Y_{t,0}^{(i)}$ to $Y_{t,r}^{(i)}$, so the latter is also uniform and independent of at least a $1 - \mathsf{negl}(\kappa)$ fraction of T. Bounding the statistical distance with $\mathsf{negl}(\kappa)$. □

Since in Game 1, it is apparent that $\mathsf{T} \times \{Y_{t,r}^{(i)}\}$ is uniform by the fact that $\{Y_{t,r}^{(i)}\}$ are generated independent of T, Claim 4 bounds the statistical distance between the responses of $\mathcal{C}$ between Game 0 and 1 with $\mathsf{negl}(\kappa)$. Combined with the previous claim, we can conclude the total difference between Game 0 and 1 is negligible. □

Game 2

Claim 5. $\Pr[\mathsf{F}_2] \geq \Pr[\mathsf{F}_1]$.

Proof. Game 2 differs from Game 1 in the winning condition. Let $\Pr[\mathsf{F}_1] = \epsilon$ be the probability of adversary winning in winning Game 1. Then the adversary is sure to win Game 2 as $\mathcal{C}$ records the output in the query. □

Game 3

Lemma 9. $\Pr[\mathsf{F}_3] \geq \Pr[\mathsf{F}_2] - \mathsf{negl}(\kappa)$

Proof. Observe that these games only differ when flag is set to 1 and adversary ends up winning Game 2. Let ϵ be the probability that $(\mathcal{A}_1, \mathcal{A}_2)$ wins in such a manner. We will refer to this as winning non-sequentially.

Claim 6. $\mathcal{A}_1$ *will only query on* $Y_{t,j}^{(i)}$ *or* $\mathsf{f}(\mathsf{pk}, Y_{t,j}^{(i)})$ *in* ***Phase 1*** *with negligible probability.*

$$\Pr[\exists (i,j,t) : Y_{t,j}^{(i)} \in \mathbf{u}^{\mathsf{T}} \vee \mathsf{f}(\mathsf{pk}, Y_{t,j}^{(i)}) \in \mathbf{u}^{\mathsf{T}^{-1}}] = \mathsf{negl}(\kappa)$$

Proof. Once we fix T, any $Y_{t,j}^{(i)}$ and $\mathsf{f}(\mathsf{pk}, Y_{t,j}^{(i)})$ have a bijective relation onto $Y_{t,r}^{(i)}$, which is uniform and independently generated, so we can union bound the probability

$$\Pr[\exists (i,j,t) : Y_{t,j}^{(i)} \in \mathbf{u}^{\mathsf{T}} \vee \mathsf{f}(\mathsf{pk}, Y_{t,j}^{(i)}) \in \mathbf{u}^{\mathsf{T}^{-1}}]$$

$$\leq \sum_{(i,j,t) \in [n] \times [r] \times [b]} \left(\sum_{q=0}^{{\mathsf{q}_1}^{\mathsf{T}}} \Pr[Y_{t,j}^{(i)} = \mathsf{u}_q^{\mathsf{T}}] + \sum_{q=0}^{{\mathsf{q}_1}^{\mathsf{T}^{-1}}} \Pr[\mathsf{f}(\mathsf{pk}, Y_{t,j}^{(i)}) = \mathsf{u}_q^{\mathsf{T}^{-1}}] \right)$$

$$= n \cdot b \cdot r \left({\mathsf{q}_1}^{\mathsf{T}} \cdot 2^{-\lambda} + {\mathsf{q}_1}^{\mathsf{T}^{-1}} \cdot 2^{-\lambda} \right)$$

Since $b, r, n, {\mathsf{q}_1}^{\mathsf{T}}, {\mathsf{q}_1}^{\mathsf{T}^{-1}}$ are all $\mathsf{poly}(\kappa)$, this is negligible. □

Claim 7. *With all but negligible probability, the* $\{Y_{t,j}^{(i)}\}$ *are unique.*

$$\Pr[\exists (i_1, j_1, t_1) \neq (i_2, j_2, t_2) : Y_{t_1,j_1}^{(i_1)} = Y_{t_2,j_2}^{(i_2)}] \leq \mathsf{negl}(\kappa)$$

Proof. We do a case by case analysis,

- Assume $(i_1, t_1) = (i_2, t_2)$ - where both values are on the same block, but different rounds. Lets consider $j_1 \in [r]$. Let j_2 be the smallest index greater than j_1 such that $Y^{(i_1)}_{t_1,j_1} = Y^{(i_2)}_{t_2,j_2}$. Now consider the set of all permutations such that $\mathsf{T}(Y^{(i_1)}_{t_1,j}) = \mathsf{f}(\mathsf{pk}, Y^{(i_1)}_{t_1,j+1})$ for $j \in [j_1, j_2 - 2]$. This set fixes the permutation on $j_2 - j_1 - 1$ points. Thus the probability over the remaining permutations that $\mathsf{T}(Y^{(i_1)}_{t_1,j_2-1}) = Y^{(i_2)}_{t_2,j_1}$ is $1/(2^\lambda - j_2 + j_1 + 1)$. Union bounding over j_1, j_2, i_1, t_1, which are all $\mathsf{poly}(\kappa)$ the probability is still negligible.
- Assume $(i_1, t_1) \neq (i_2, t_2)$ - where the two values are on different blocks. Lets consider $j_1, j_2 \in [r]$. Observe that if $\max(j_1, j_2) < r$, $Y^{(i_1)}_{t_1,j_1} = Y^{(i_2)}_{t_2,j_2} \Rightarrow Y^{(i_1)}_{t_1,j_1+1} = Y^{(i_2)}_{t_2,j_2+1}$. Without loss of generality, assume $j_1 \leq j_2$, this implies $Y^{(i_1)}_{t_1,j_1-j_2+r} = Y^{(i_2)}_{t_2,r}$. However, since $Y^{(i_2)}_{t_2,r}$ was independently randomly chosen, the probability it is equal to any $Y^{(i_1)}_{t_1,j}$ occurs with probability $\leq \frac{r}{2^\lambda}$, which is negligible. We can union bound over t_1, i_1, t_2, i_2, which are all $\mathsf{poly}(\kappa)$, so the probability is still negligible.

□

Now consider the following computationally unbounded algorithm $\mathcal{B}'$ with access to oracle $\mathsf{T}, \mathsf{T}^{-1}$. This algorithm will translate a non-sequential $\mathcal{A}_2$ into a reduction to the game outlined in Lemma 1 by working backwards from m and using its unbounded computation to invert the trapdoor permutation and recover the output of a non-sequential random oracle query without ever querying on its input.

Reduction $\mathcal{B}'^{\mathsf{T}(\cdot),\mathsf{T}^{-1}(\cdot)}(\mathsf{advice})$:

Goal: Produce Input Output oracle pairs without explicitly querying the oracle.

- **Setup:**
  - Sample a random function $\mathsf{H}(\cdot)$ and use it to answer oracle queries made by $\mathcal{A}_1, \mathcal{A}_2$.
  - Perform, **Setup** and **Phase 1** as in Game 3.
  - Receive $m \leftarrow \mathcal{A}_1^{\mathsf{H}(\cdot),\mathsf{T}(\cdot),\mathsf{T}^{-1}(\cdot)}(1^\kappa, \mathsf{pk}')$ after **Phase 1**. Parse pk' as (pk, n).
  - Choose a set of random $\{\rho_i\}_{i\in[n]}$ and compute $\{Y^{(i)}_{t,0} = \mathsf{H}(\rho_i||t) \oplus m_t\}_{t\in[b],i\in[n]}$.
  - Parse advice as $(\mathsf{state}, \mathsf{Q} = \{(q_k, i_k, t_k, j_k)\}_k)$ where k is polynomial in security parameter and q_k represents query that will be made by algorithm $\mathcal{A}_2$, i_k, t_k, j_k represent the replica,block and round number respectively.
- **Simulate:**
  - Run $\mathcal{A}_2^{\mathsf{H}(\cdot),\mathsf{T}(\cdot),\mathsf{T}^{-1}(\cdot)}(1^\kappa, \mathsf{pk}', m, \mathsf{state})$, interacting with the random oracle queries it makes to $\mathsf{T}, \mathsf{T}^{-1}$. Let $\mathbf{w}^{\mathsf{T},\mathsf{T}^{-1}}$ denote the ordered and distinct list of queries $\mathcal{A}_2$ makes to T or T^{-1}, and let $\mathsf{w}_q^{\mathsf{T},\mathsf{T}^{-1}}$ refer to the q^{th} element in this list. We perform the operations below while running $\mathcal{A}_2$.
  - $\forall k$, let x_k be $\mathsf{w}_{q_k}^{\mathsf{T},\mathsf{T}^{-1}}$. Compute $Y^{(i_k)}_{t_k,j_k}$ from $Y^{(i_k)}_{t_k,0}$.
    - For j from 1 to j_k,
    $$Y^{(i_k)}_{t_k,j} = \mathsf{f}^{-1}(\mathsf{sk}, \mathsf{T}(Y^{(i_k)}_{t_k,j-1})).$$
    Let $y_k = x_k$ if the q_k^{th} query was made to T^{-1} and $y_k = \mathsf{f}(\mathsf{pk}, x_k)$ if it were made to T.
  - Any time $\mathcal{A}_2$ attempts to query T on $Y^{(i_k)}_{t_k,j_k}$ or T^{-1} on y_k, $\mathcal{B}'$ doesn't query the true oracle and instead returns y_k or $Y^{(i_k)}_{t_k,j_k}$ to $\mathcal{A}_2$ respectively.
  - For any other queries $\mathcal{A}_2$ makes to an oracle, $\mathcal{B}'$ simply queries the appropriate oracle, returns the query results and completes execution of $\mathcal{A}_2$.

- **Return:** Output the pairs $\{(Y^{(i_k)}_{t_k,j_k}, \mathsf{f}(\mathsf{pk}, x_k))\}_k$.

Claim 8. *Suppose $Y^{(i)}_{t,j+1}$ is queried on T or $\mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j+1})$ is queried on T^{-1} by $\mathcal{A}_2$ when $Y^{(i)}_{t,j}$ has not yet been queried on T. Then $\exists$ hint q' such that $\mathcal{B}'$ on input $\mathsf{advice} = (\mathsf{state}, \mathsf{Q})$ where $(q', i, t, j) \in \mathsf{Q}$ and $\mathcal{B}'$ outputs $(Y^{(i)}_{t,j}, \mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j+1}))$ and in **Simulate** phase never queries T on $Y^{(i)}_{t,j}$ or T^{-1} on $\mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j+1})$ and only queries T on $Y^{(i)}_{t,j'}$ for $j' < j$ (in addition to $\mathcal{A}_2$'s queries).*

Proof. Let q' be the smallest index in $\mathbf{w}^{\mathsf{T},\mathsf{T}^{-1}}$ which represent a query of $Y^{(i)}_{t,j+1}$ to T or $\mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j})$ to T^{-1} by $\mathcal{A}_2$. $\mathcal{B}'$ from our construction can from now on answer $\mathsf{T}(Y^{(i)}_{t,j})$ and $\mathsf{T}^{-1}(\mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j}))$ without querying the true oracle in **Simulate** phase. Since by assumption, $Y^{(i)}_{t,j}$ has not yet been queried on T by $\mathcal{A}_2$ yet, and q' was minimum, so T^{-1} was not queried on $\mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j})$ before either. □

Claim 9. *If $\mathcal{A}_2$ wins nonsequentially with probability ϵ,*

$$\Pr\left[\begin{array}{c} \exists\ \mathsf{advice} \in \{0,1\}^*\ s.t.\ |\mathsf{advice}| \le n' \cdot \lambda - \omega(\log\lambda), \\ \{(x_i, y_i)\}_{i=1}^{n'} \leftarrow \mathcal{B}'^{\mathsf{T}(\cdot),\mathsf{T}^{-1}(\cdot)}(\mathsf{advice})\ where \\ \forall i \neq j \in [n'], x_i \neq x_j,\ \mathsf{T}(x_i) = y_i,\ x_i \notin \mathsf{s}^{\mathsf{T}}_{\mathcal{B}'}\ and\ y_i \notin \mathsf{s}^{\mathsf{T}^{-1}}_{\mathcal{B}'} \end{array}\right] \ge \epsilon - \mathsf{negl}(\kappa),$$

Proof. We can take advice to be the state produced by $\mathcal{A}_1^{\mathsf{H}(\cdot),\mathsf{T},\mathsf{T}^{-1}}(1^\kappa, \mathsf{pk})$ and $(q, i, t, j) \in \mathsf{Q}$ to be a j such that $Y^{(i)}_{t,j+1}$ was queried before $Y^{(i)}_{t,j}$ for every $i \in [n], t \in [b]$.

By Claim 8, such a q exists for each (i, t, j), and $\mathcal{B}'$ has outputted $n \cdot b$ pairs $(Y^{(i)}_{t,j}, \mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j+1}))$ without querying on $\mathsf{T}, \mathsf{T}^{-1}$ in the **Simulate** phase. By Claim 7 and Claim 6 that these pairs (x_i, y_i) are distinct, and that $\mathcal{B}$ will not have queried any x_i or y_i in the **Setup** phase with all but negligible probability.

Since (q, i, t, j) are all $\le \mathsf{poly}(\kappa)$ by the running time of $\mathcal{A}_2$, Q only needs $b \cdot n \cdot O(\log\kappa)$ bits. As $\mathsf{s} \in 1 - \frac{\omega(\log\kappa)}{\lambda}$, we get,

$$|\mathsf{state}| \le b \cdot n \cdot \lambda - b \cdot n \cdot \omega(\log(\kappa)) \Rightarrow |\mathsf{advice}| \le b \cdot n \cdot \lambda - b \cdot n \cdot \omega(\log(\kappa)) + b \cdot n \cdot O(\log\kappa)$$

$$\le b \cdot n \cdot \lambda - \omega(\log(\kappa)).$$

This proves the claim for $n' = b \cdot n$ which is polynomial in κ and hence polynomial in λ. □

By Lemma 1, $\epsilon - \mathsf{negl}(\kappa) \in \mathsf{negl}(\lambda) = \mathsf{negl}(\kappa) \Rightarrow \epsilon \le \mathsf{negl}(\kappa)$.

□

Game 4

Claim 10. $\Pr[\mathsf{F}_4] \ge \frac{\Pr[\mathsf{F}_3]}{bn}$.

Proof. Game 4 differs from Game 3 in the winning condition. Let $\Pr[\mathsf{F}_3] = \epsilon$ be the probability of adversary winning in winning Game 3. Let this adversary be $\mathcal{A}_2$. From the sequentiality condition we have that $\mathcal{A}_2$ is sequential on at least one block and replica. The probability that this guess was made correctly in Game 4 is $\ge \frac{1}{bn}$. This adversary thus wins Game 4 with probability $\ge \frac{\epsilon}{bn}$. □

Game 5

Lemma 10. $\Pr[\mathsf{F}_5] \ge \Pr[\mathsf{F}_4] - \mathsf{negl}(\kappa)$.

Proof. Game 5 differs from Game 4 in how queries are answered to the adversaries and the possibility of flag being set. Let $\Pr[\mathsf{F}_4] = \epsilon$ be the probability of adversary winning in winning Game 4. Let E^{T} be the event that it is set due to **Prequery Check** T, and let $\mathsf{E}^{\times}$ be the probability that flag is due to **Collision Check**. We can bound the probability that the adversary wins Game 5 by the sum of (i) the probability that $\mathsf{E}^{\times}$ occurs, (ii) the probability E^{T} occurs, and (iii) the statistical difference of the output of $\mathcal{C}$ from using $\mathsf{T}, \mathsf{T}_\times$.

Claim 11. $\Pr[\mathsf{E}^{\mathsf{x}}] = \mathsf{negl}(\kappa)$.

We note that since $\mathcal{Z}_1$ are uniform and independently random, we can bound the probability that $z_a, z_b \in \mathcal{Z}_1$ collide for any fixed $a \neq b$ is $\frac{1}{2^\lambda}$, so we can union bound the probability that

$$\Pr[\exists a \neq b : z_a = z_b] \leq \sum_{a=0}^{2r+1} \sum_{b=a+1}^{2r+1} \Pr[z_a = z_b] = \binom{2r+1}{2} \frac{1}{2^\lambda} = \mathsf{negl}(\kappa)$$

Claim 12. $\Pr[\mathsf{E}^{\mathsf{T}}] = \mathsf{negl}(\kappa)$

Proof. We note that since all $4r+1$ elements of $\mathcal{Z}_1 \cup \mathcal{Z}_2$ are uniform and independent of T, we can bound the probability that some $z \in \mathcal{Z}_1 \cup \mathcal{Z}_2$ is equal to some $z' \in \left(\mathbf{u}^{\mathsf{T}} \cup \mathbf{u}^{\mathsf{T}^{-1}} \cup \mathbf{U}^{\mathsf{T}} \cup \mathbf{U}^{\mathsf{T}^{-1}}\right)$ with $\frac{1}{2^\lambda}$. Thus, we can union bound

$$\Pr[(\mathcal{Z}_1 \cup \mathcal{Z}_2) \cap \left(\mathbf{u}^{\mathsf{T}} \cup \mathbf{u}^{\mathsf{T}^{-1}} \cup \mathbf{U}^{\mathsf{T}} \cup \mathbf{U}^{\mathsf{T}^{-1}}\right) \neq \emptyset] \leq (4r+1) \cdot \left(2\mathsf{q_1}^{\mathsf{T}} + 2\mathsf{q_1}^{\mathsf{T}^{-1}}\right) \cdot 2^{-\lambda}$$

Since $\mathsf{q_1}^{\mathsf{T}}, \mathsf{q_1}^{\mathsf{T}^{-1}}, r$ are all $\mathsf{poly}(\kappa)$, this is $\mathsf{negl}(\kappa)$.

□

For (iii), applying Lemma 4 with $\pi = \mathsf{T}$, $\{r_0, \ldots r_k\} = \{\mathsf{x}_0, \ldots, \mathsf{x}_r\}$, and $\tau = \mathsf{f}(\mathsf{pk}, \cdot)$ tells us that $(\mathsf{x}_r, \mathsf{T}[\mathsf{swap}(\mathsf{x}_0, \mathsf{f}(\mathsf{pk}, \mathsf{x}_1)), \ldots, \mathsf{swap}(\mathsf{x}_{r-1}, \mathsf{f}(\mathsf{pk}, \mathsf{x}_r))])$ is uniformly random. Since the only place $\mathcal{C}$'s responses differ are answers to T_{x} and returning the encoding $Y^{(\gamma)}_{\beta,r} = \mathsf{x}_r$. We recall that $(Y^{(\gamma)}_{\beta,r}, \mathsf{T})$ is uniform in Game 4 by construction, so these distributions are identical.

Since (i), (ii), (iii) are all negligible, the adversary thus wins Game 5 with probability $\epsilon - \mathsf{negl}(\kappa)$. □

Game 6

Let $\Pr[\mathsf{F}_5] = \epsilon$ be the probability of adversary winning in winning Game 5.

Lemma 11. $\Pr[\mathsf{F}_6] \geq \left(\frac{\epsilon}{2}\right)\left(\frac{\epsilon 2^{r-1} - 2^{|\mathsf{state}|} + 1}{\epsilon 2^{r-1}}\right)$.

Here we apply the bound on state size to argue that x' which ensure that $\mathcal{A}$ wins must be fairly frequent.

Proof. Note that by construction, $\mathcal{A}$ will win Game 6 as long as an appropriate x' exists. We will lower bound said probability below. Let us consider the random coins used in Game 5. The randomness for $\mathcal{C}$ is over the choice of permutation picked T and $\gamma, \beta, \mathcal{Z}, \mathsf{x}, \delta, \rho'$ where ρ' denotes the randomness over ρ_i sampled for each replica and the coins used by the rSetup. Let us denote $\eta = (\gamma, \beta, \mathsf{T}, \mathcal{Z}, \rho')$ for ease of notation. Let $\mathcal{A}_1, \mathcal{A}_2$ denote the adversaries that solves Game 5 with ϵ probability.

Define the set $\mathcal{N} = \{(\eta, \mathsf{x}) | \mathsf{F}_5(\eta, \mathsf{x})\}$ where $\mathsf{F}_5(\eta, \mathsf{x})$ denotes the event that adversaries win game 5 with given parameters. Thus $\Pr_{\eta,\mathsf{x}}[(\eta, \mathsf{x}) \in \mathcal{N}] = \epsilon$.

Let us define a heavy set, $\mathcal{H} = \{(\eta, \mathsf{x}) | \Pr_{\eta,\mathsf{x}'}[(\eta, \mathsf{x}') \in \mathcal{N}] \geq \frac{\epsilon}{2}\}$. Then from the heavy row lemma of [15], $\Pr_{\eta,\mathsf{x}}[(\eta, \mathsf{x}) \in \mathcal{H} | (\eta, \mathsf{x}) \in \mathcal{N}] \geq 1/2$ i.e. probability that our winning instance lies on a heavy set is atleast half.

Note that for $\mathcal{A}_2$ to successfully solve Game 6, we must see the following events,

1. Adversary solves the Game 5 i.e. $(\eta, \mathsf{x}) \in \mathcal{N}$, and let this be denoted by event E_1.

$$\Pr_{\eta,\mathsf{x}}[\mathsf{E}_1] \geq \epsilon.$$

2. Let the event that $(\eta, \mathsf{x}) \in \mathcal{H}$ lies on a heavy set be denoted by E_2. By heavy row lemma, we have that,

$$\Pr_{\eta,\mathsf{x}}[\mathsf{E}_2 | \mathsf{E}_1] \geq 1/2$$
$$\Rightarrow \Pr_{\eta,\mathsf{x}}[\mathsf{E}_2] \geq \frac{\epsilon}{2}.$$

3. Let the event that there exists a collision of x consistent with state that is also successful on Game 5 be denoted by E_3.

Claim 13. $\Pr_{\eta,\mathsf{x}}[\mathsf{E}_3|\mathsf{E}_2] \geq \left(\frac{\epsilon 2^{r-1}-2^{|\mathsf{state}|}+1}{\epsilon 2^{r-1}}\right)$.

Proof. Given that $(\eta,\mathsf{x}) \in \mathcal{H}$ i.e. it lies on a heavy set. Define a possible set of switches by $\mathcal{Q} = \{\mathsf{x}'|(\eta,\mathsf{x}') \in \mathcal{N}\}$. From the definition of $\mathcal{H}$, clearly $|\mathcal{Q}| \geq \frac{\epsilon 2^r}{2}$.

From Lemma 7 (Section 4), define h_η as a function from $\mathcal{Q} \to \{0,1\}^{|\mathsf{state}|}$ where $\mathsf{h}_\eta(a) = \mathsf{state}_a$ where state_a denotes the state shared by $\mathcal{A}_1$ to $\mathcal{A}_2$ when playing Game 5 with parameters η and x set to a. Since $\mathcal{A}_1$'s coins are deterministically fixed, h_η is a deterministic function.

The lemma implies the statement that,

$$\Pr_{\mathsf{x}}[\exists \mathsf{x}' \neq \mathsf{x} \in \mathcal{Q}, \mathsf{h}_\eta(\mathsf{x}) = \mathsf{h}_\eta(\mathsf{x}')] \geq \frac{|\mathcal{D}| - |\mathcal{R}| + 1}{|\mathcal{D}|}.$$

This proves the claim. □

$$\begin{aligned}\Pr_{\eta,\mathsf{x}}(\mathcal{A}_2 \text{ wins Game 6}) &\geq \Pr[\mathsf{E}_2 \cap \mathsf{E}_3] \\ &\geq (\Pr[\mathsf{E}_2])\,(\Pr[\mathsf{E}_3|\mathsf{E}_2]) \\ &\geq \left(\frac{\epsilon}{2}\right)\left(\frac{\epsilon 2^{r-1} - 2^{|\mathsf{state}|}+1}{\epsilon 2^{r-1}}\right)\end{aligned}$$

□

We note that if F_5 is non negligible then so is F_6 - i.e. $\epsilon \geq \frac{1}{\mathsf{poly}(\kappa)}$, then

$$\epsilon 2^{r-1} \geq 2^{r-O(\log\kappa)} \in \omega(2^{|\mathsf{state}|}+1) \Rightarrow \frac{\epsilon 2^{r-1} - 2^{|\mathsf{state}|+1}}{\epsilon 2^{r-1}} \in (1 - \mathsf{negl}(\kappa)).$$

Game 7

Lemma 12. $\Pr[\mathsf{F}_7] \geq \Pr[\mathsf{F}_6] - \mathsf{negl}(\kappa)$.

Proof. Game 7 is played on a different permutation than Game 6, with a different winning condition. By our definition of $\mathsf{swap}(\cdot,\cdot)$, we can observe that these permutations differ when querying $\mathsf{T}^{\delta}_{\mathsf{x}'}$ on x'_j or $\mathsf{T}^{-1}(\mathsf{f}(\mathsf{pk},\mathsf{x}'_j))$ for $j > \delta$ and when querying $\mathsf{T}^{\delta}_{\mathsf{x}'}$ on $\mathsf{f}(\mathsf{pk},\mathsf{x}'_j)$ or $\mathsf{T}(\mathsf{f}(\mathsf{pk},\mathsf{x}'_j))$ again for $j > \delta$.

Claim 14. *Let $a,b,c,d \xleftarrow{R} \mathcal{D}$ and $\mathsf{T} \xleftarrow{R} S_{\mathcal{D}}$. Then the distributions of $(a,b,\mathsf{T}[\mathsf{swap}(a,b)])$ and $(a,b,\mathsf{T}[\mathsf{swap}(a,c),\mathsf{swap}(d,b),\mathsf{swap}(a,b)])$ are identical.*

Proof. By Lemma 3, $(a,\mathsf{T}[\mathsf{swap}(a,c)])$ is identical distributed to (a,T), so since b is independent of a, we can apply Lemma 3 again to say $(a,b,\mathsf{T}[\mathsf{swap}(a,c),\mathsf{swap}(b,d)])$ is identically distributed to (a,b,T), after the result follows from applying $\mathsf{swap}(a,b)$ to both distributions. □

By Claim 14, we know the distribution of

$$(\{\mathsf{x}'_j\},\mathsf{T}[\ldots,\mathsf{swap}(\mathsf{x}'_j,\mathsf{f}(\mathsf{pk},\mathsf{x}'_{j+1})),\ldots])$$

is identical to that of

$$(\{\mathsf{x}'_j\},\mathsf{T}[\ldots,\mathsf{swap}(\mathsf{x}'_j,c),\mathsf{swap}(d,\mathsf{f}(\mathsf{pk},\mathsf{x}'_{j+1})),\mathsf{swap}(\mathsf{x}'_j,\mathsf{f}(\mathsf{pk},\mathsf{x}'_{j+1})),\ldots])$$

for any $j \in [r]$, but observe in the latter case, d and c are the preimage and image of $\mathsf{f}(\mathsf{pk}, \mathsf{x}'_{j+1})$ and x'_j respectively before $\mathsf{swap}(\mathsf{x}'_j, \mathsf{f}(\mathsf{pk}, \mathsf{x}'_{j+1}))$, and are independently random, which tells us that $\mathsf{T}^{-1}(\mathsf{f}(\mathsf{pk}, \mathsf{x}'_j))$ and $\mathsf{T}(\mathsf{f}(\mathsf{pk}, \mathsf{x}'_j))$ are random independent of $\{(\mathsf{x}'_j), \mathsf{T}_{\mathsf{x}'}\}$, so the probability that some $z \in \mathsf{w}^{\mathsf{T}} \cup \mathsf{w}^{\mathsf{T}^{-1}}$ is equal a one of those is $2^{-\lambda}$, which allows us to union bound the total probability

$$\Pr[(\mathsf{T}^{-1}(\{\mathsf{f}(\mathsf{pk}, \mathsf{x}_j))\} \cup \{\mathsf{T}(\mathsf{f}(\mathsf{pk}, \mathsf{x}'_j))\}) \cap \left(\mathsf{w}^{\mathsf{T}} \cup \mathsf{w}^{\mathsf{T}^{-1}}\right) \neq \emptyset] \leq 2r \cdot \left(\mathsf{q_3}^{\mathsf{T}} + \mathsf{q_3}^{\mathsf{T}^{-1}}\right) \cdot 2^{-\lambda}$$

Since $\mathsf{q_3}^{\mathsf{T}}, \mathsf{q_3}^{\mathsf{T}^{-1}}, r$ are all $\mathsf{poly}(\kappa)$, this is $\mathsf{negl}(\kappa)$. In addition $\mathcal{A}$ wins Game 6, by sequentiality, $\mathcal{A}_2$ must have queried on x'_δ before querying on any x'_j or $\mathsf{f}(\mathsf{pk}, \mathsf{x}'_j)$ for $j > \delta$, so Game 6 and Game 7 have identical queries with all but negligible probability before x'_δ is queried on, at which point $\mathcal{A}$ wins Game 7. □

Game 8

Claim 15. $\Pr[\mathsf{F}_8] \geq \frac{\Pr[\mathsf{F}_7]}{r}$.

Proof. Our main observation here is that $\mathcal{C}$ does not need to guess the switch completely, $\mathcal{C}$ guesses δ. The probability that this index was corretly guessed in Game 8 is exactly equal to $\frac{1}{r}$. This adversary thus queries x'_δ and hence wins Game 8 with probability $\geq \frac{\Pr[\mathsf{F}_7]}{r}$.

□

Claim 16. $\Pr[\mathsf{F}_8] = \mathsf{negl}(\kappa)$

Proof. We will show through a reduction that an adversary able to win Game 8 with probability ϵ will also be able to invert our trapdoor permutation with the same probability.
Let $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa), y \xleftarrow{R} \mathcal{D}_{\mathsf{pk}} = \{0,1\}^\lambda, z = \mathsf{f}(\mathsf{pk}, y)$,

Reduction $\mathcal{B}(\mathsf{pk}, z)$

Goal: Output z such that, $\mathsf{f}(pk, y) = z$.

- **Setup:** The challenger(denoted by $\mathcal{C}$) ~~runs $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{rSetup}(1^\kappa, 1^n)$ and~~ sends public key pk' to $\mathcal{A}_1$. ~~It keeps the secret key sk' for itself.~~
- **Phase 1, Sampling a New Permutation, File Challenge, Phase 2, State Sharing:** ....
- **Setting switched oracle:**
  - ....
  - Let $\delta \xleftarrow{R} [r]$
  - Define T_x using swap (Definition 4):

$$\mathsf{T}^\delta_{\mathsf{x}'} = \mathsf{T}[\mathsf{swap}(\mathsf{x}_0, \mathsf{f}(\mathsf{pk}, \mathsf{x}_1)), \ldots, \mathsf{swap}(\mathsf{x}_{\delta-1}, z)]$$

  - Let $\mathsf{T}^{-1}_{\mathsf{x}'}$ be the inverse of $\mathsf{T}_{\mathsf{x}'}$.
- **Phase 3, Guess:** ....
- ~~**Verify:** Adversary wins of T is queried on $\bar{\mathsf{x}}_\delta$ and flag = 0.~~
- **Embed:** Output embedding $\mathsf{w}^{\mathsf{T}}_i$, where $\mathsf{w}^{\mathsf{T}}_i \in \mathbf{w}^{\mathsf{T}}$, s.t. $\mathsf{f}(\mathsf{pk}, \mathsf{w}^{\mathsf{T}}_i) = z$.

Since $\bar{\mathsf{x}}_\delta$ is uniformly distributed and not used before here, $\mathsf{f}(\mathsf{pk}, \bar{\mathsf{x}}_\delta)$ is uniform and independent, and indistinguishable from a uniformly random z. Note that since we no longer check for flag, it is no longer necessary to perform collision and prequery checks, the only place where x'_δ is used instead of $\mathsf{f}(\mathsf{pk}, \bar{\mathsf{x}}'_\delta)$. We can simulate the random permutation oracles and regular random oracle by picking uniformly random values on the appropriate domain whenever $\mathcal{A}$ makes a new query and maintaining a map of queries already made. Thus, if $\mathcal{A}_2$ won game 8, they must have queried

$$\bar{\mathsf{x}}_\delta = \mathsf{f}^{-1}(\mathsf{sk}, \mathsf{f}(\mathsf{pk}, \bar{\mathsf{x}}_\delta)) = \mathsf{f}^{-1}(\mathsf{sk}, z)$$

Which, from our definition of a secure trapdoor permutation, can only happen with negligible probability,

□

To complete our proof of Theorem 1, from our sequence of games, we conclude that $\Pr[\mathsf{F}_0] = \mathsf{negl}(\kappa)$, fulfilling our soundness definition.

6 Replica Encodings in the Random Function Model

We now turn toward building Replica Encodings from trapdoor permutations in the ideal function model. Our construction will embed a Feistel like structure into the replica encoding construction. We will directly prove security of this construction. Our construction makes use of the $\mathsf{KeyGen}, \mathsf{f}$, and f^{-1} defined for a trapdoor permutation on domain $\{0,1\}^\lambda$ and a random function T' on the same domain.

Define functions $\mathsf{L}, \mathsf{R} : \{0,1\}^* \to \{0,1\}^*$ on even length inputs as follows. If $x = y||z$, where $x, y, z \in \{0,1\}^*$, $|y| = |z|$, then the function $\mathsf{L}(.)$ denotes the left half of x i.e. $\mathsf{L}(x) = y$ and the function $\mathsf{R}(.)$ denotes the right half of x i.e. $\mathsf{R}(x) = z$.

6.1 Construction

$\mathsf{rSetup}(1^\kappa, 1^n)$**:**

Run $\mathsf{KeyGen}(1^\kappa) \to (\mathsf{pk}, \mathsf{sk})$. Output $(\mathsf{pk}' = (\mathsf{pk}, n), \mathsf{sk}' = (\mathsf{sk}, n))$.

$\mathsf{rEnc}(\mathsf{sk}', m)$**:**

- Parse $\mathsf{sk}' = (\mathsf{sk}, n)$.
- Choose a string $\rho \xleftarrow{R} \{0,1\}^\kappa$.
- Divide m into b blocks of length 2λ i.e. $m = m_1||m_2||\ldots||m_b$, $b = \lceil |m|/2\lambda \rceil$.
- Set $r = n \cdot b \cdot \lambda$.
- Compute $\forall t \in [b]$,
$$Y_{t,0} = L(m_t \oplus \mathsf{H}(\rho||t)).$$
$$Y_{t,1} = R(m_t \oplus \mathsf{H}(\rho||t)).$$
- For rounds j from 2 to r compute:
  - Compute $Y_{t,j}$ from $Y_{t,j-1}$ and $Y_{t,j-2}$ as
$$Y_{t,j} = \mathsf{f}^{-1}(\mathsf{sk}, Y_{t,j-2} \oplus \mathsf{T}'(Y_{t,j-1}))$$
- Let $Z_t = Y_{t,r-1}||Y_{t,r}$
- Let $y_r = Z_1||\ldots||Z_b$ and output (y_r, ρ).

$\mathsf{rDec}(\mathsf{pk}', y)$**:**

- Parse $\mathsf{pk}' = (\mathsf{pk}, n)$.
- Parse y as (y_r, ρ). Parse y_r as $Z_1||Z_2||\ldots||Z_b$,where $b = \lceil |y_r|/2\lambda \rceil$ and $r = n \cdot b \cdot \lambda$.
- For each $Z_t = Y_{t,r-1}||Y_{t,r}$ and for rounds j from $r-2$ to 0 compute:
  - Compute $Y_{t,j}$ from $Y_{t,j+1}$ and $Y_{t,j+2}$ as
$$Y_{t,j} = \mathsf{f}(\mathsf{pk}, Y_{t,j+2}) \oplus \mathsf{T}'(Y_{t,j+1})$$
- $\forall t \in [b]$ compute,
$$m_t = Y_{t,0}||Y_{t,1} \oplus \mathsf{H}(\rho||t)$$
  Output $m = m_1||\ldots||m_b$.

The encoding length for our scheme is $\mathsf{len}(\kappa, |m|) = |m| + O(\kappa)$.[4]

[4] Upto additional rounding factors.

6.2 Proof of Security

Theorem 2. *Assuming* $(\mathsf{KeyGen}(1^\kappa), \mathsf{f}(\cdot,\cdot), \mathsf{f}^{-1}(\cdot,\cdot))$ *is a secure trapdoor permutation on domain and range* $\{0,1\}^\lambda$ *and* T' *is an oracle to a random function on the same domain and range, and* H *is a random oracle with range* $\{0,1\}^{2\lambda}$. *Then our construction for* ReplicaEncoding *described above is* s*-sound according to Definition 2 for all* $\kappa, n \in \mathbb{N}$ *and* $\mathsf{s} \in 1 - \frac{\omega(\log \kappa)}{2\lambda}$.

6.2.1 Sequence of Games

Our proof proceeds via a sequence of games as described below. We assume that adversaries have their randomness non-uniformly fixed in each game to maximize their success. The changes in each game in comparison to the previous one are indicated with red. Details of the previous game are copied without explicit rewriting.

Game 0: This is the original $\mathsf{Sound}_{\mathcal{A}_1,\mathcal{A}_2}(\kappa, n)$ security game where we record the queries made by the adversaries in lists. We also assume that any list is ordered and stores distinct elements. More concretely, when in Phase 1 a query x is made on $\mathcal{O}$, $\mathcal{C}$ checks if $x \notin \mathbf{u}^{\mathcal{O}}$ and updates the list $\mathbf{u}^{\mathcal{O}}$ if the condition is true. It performs this operation of maintaining the list for each Phase and oracle separately. Denote $\mathsf{q}_1{}^{\mathcal{O}}, \mathsf{q}_2{}^{\mathcal{O}}, \mathsf{q}_3{}^{\mathcal{O}}$ as the functions that take in the security parameter and output the total distinct queries made by the adversaries to oracle $\mathcal{O}$ during the three phases respectively.

- **Setup:** The challenger(denoted by $\mathcal{C}$) runs $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{rSetup}(1^\kappa, 1^n)$ and sends public key pk' to $\mathcal{A}_1$. It keeps the secret key sk' for itself.
- **Phase 1:** The stateful adversary $\mathcal{A}_1$ issues queries on H and T', $\mathcal{C}$ responds the query back to $\mathcal{A}_1$. Let the queries on oracle $\mathcal{O}$ be denoted by an ordered and distinct list $\mathbf{u}^{\mathcal{O}} = (\mathsf{u}_1^{\mathcal{O}}, \ldots, \mathsf{u}_{\mathsf{q}_1\mathcal{O}}^{\mathcal{O}})$ and their outputs be denoted by an ordered and distinct list $\mathbf{U}^{\mathcal{O}} = (\mathsf{U}_1^{\mathcal{O}}, \ldots, \mathsf{U}_{\mathsf{q}_1\mathcal{O}}^{\mathcal{O}})$.
- **File Challenge:** $m \in \{0,1\}^* \leftarrow \mathcal{A}_1^{\mathsf{H}(\cdot),\mathsf{T}'(\cdot)}(1^\kappa, \mathsf{pk}')$. It sends m to $\mathcal{C}$ who parses pk' as (pk, n); sk' as (sk, n) and does the following:
  - Divide m into b blocks of length 2λ i.e. $m = m_1||m_2||\ldots||m_b$, $b = \lceil |m|/2\lambda \rceil$.
  - For $i \in [n]$,
    - Choose a string $\rho_i \xleftarrow{R} \{0,1\}^\kappa$.
    - Divide m into b blocks of length 2λ i.e. $m = m_1||m_2||\ldots||m_b$, $b = \lceil |m|/2\lambda \rceil$.
    - Compute $\forall t \in [b]$,
    $$Y_{t,0}^{(i)} = L(m_t \oplus \mathsf{H}(\rho_i||t)).$$
    $$Y_{t,1}^{(i)} = R(m_t \oplus \mathsf{H}(\rho_i||t)).$$
    - For rounds j from 2 to r compute:
      - Compute $Y_{t,j}^{(i)}$ from $Y_{t,j-1}^{(i)}$ and $Y_{t,j-2}^{(i)}$ as
      $$Y_{t,j}^{(i)} = \mathsf{f}^{-1}(\mathsf{sk}_1, Y_{t,j-2}^{(i)} \oplus \mathsf{T}'(Y_{t,j-1}^{(i)}))$$
    - Let $Z_t = Y_{t,r-1}^{(i)}||Y_{t,r}^{(i)}$
    - Let $y_r^{(i)} = Z_1^{(i)}||\ldots||Z_b^{(i)}$ and output $(y_r^{(i)}, \rho)$.

  $\mathcal{C}$ returns $y^{(1)}, y^{(2)}, \ldots y^{(n)}$ to $\mathcal{A}_1$.
- **Phase 2:** $\mathcal{A}_1$ issues additional queries on H and T', $\mathcal{C}$ responds the query back to $\mathcal{A}_1$. Let the queries on oracle $\mathcal{O}$ be denoted by an ordered and distinct list $\mathbf{v}^{\mathcal{O}} = (\mathsf{v}_1^{\mathcal{O}}, \ldots, \mathsf{v}_{\mathsf{q}_2\mathcal{O}}^{\mathcal{O}})$ and their outputs be denoted by an ordered and distinct list $\mathbf{V}^{\mathcal{O}} = (\mathsf{V}_1^{\mathcal{O}}, \ldots, \mathsf{V}_{\mathsf{q}_2\mathcal{O}}^{\mathcal{O}})$.
- **State Sharing:** $\mathcal{A}_1$ outputs state $\mathsf{state} \leftarrow \mathcal{A}_1^{\mathsf{H}(\cdot),\mathsf{T}'(\cdot)}(1^\kappa, \mathsf{pk}', y)$ and sends state to $\mathcal{A}_2$.
- **Phase 3:** Stateful adversary $\mathcal{A}_2$ queries on H and T', $\mathcal{C}$ responds the query back to $\mathcal{A}_2$. Let the queries on oracle $\mathcal{O}$ be denoted by an ordered and distinct list $\mathbf{w}^{\mathcal{O}} = (\mathsf{w}_1^{\mathcal{O}}, \ldots, \mathsf{w}_{\mathsf{q}_3\mathcal{O}}^{\mathcal{O}})$ and their outputs be denoted by an ordered and distinct list $\mathbf{W}^{\mathcal{O}} = (\mathsf{W}_1^{\mathcal{O}}, \ldots, \mathsf{W}_{\mathsf{q}_3\mathcal{O}}^{\mathcal{O}})$.

- **Guess:** $\mathcal{A}_2$ outputs the replica guesses to $\mathcal{C}$

$$\{\tilde{y}^{(i)}\} \leftarrow \mathcal{A}_2(1^\kappa, \mathsf{pk}', \mathsf{state}).$$

- **Verify:** Let $v_i = 1$ if $\tilde{y}^{(i)} = y^{(i)}$ and $v_i = 0$ otherwise. Adversary wins if $|\mathsf{state}| < \sum v_i \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)$.

Game 1: In this game we remove the sk and rely on the public key with an additional reprogramming step at oracle H. This helps us further down the road in showing a reduction to the security of the trapdoor permutation.

- **Setup:** The challenger(denoted by $\mathcal{C}$) runs $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{rSetup}(1^\kappa, 1^n)$ and sends public key pk' to $\mathcal{A}_1$. It keeps the secret key sk' for itself. Set $\mathsf{flag} = 0$.
- **Phase 1:** ...
- **File Challenge:** $m \in \{0,1\}^* \leftarrow \mathcal{A}_1^{\mathsf{H}(\cdot), \mathsf{T}'(\cdot)}(1^\kappa, \mathsf{pk}')$. It sends m to $\mathcal{C}$ who parses pk' as (pk, n); sk' as (sk, n) and does the following:
  - Divide m into b blocks of length 2λ i.e. $m = m_1||m_2||\dots||m_b$, $b = \lceil |m|/2\lambda \rceil$.
  - For $i \in [n]$,
    - Choose a string $\rho_i \xleftarrow{R} \{0,1\}^\kappa$.
      Prequery Check H: If $\exists t \in [b]: \ \rho_i||t \in \mathbf{u}^{\mathsf{H}}$, set $\mathsf{flag} = 1$.
    - Sample $\{(Y^{(i)}_{t,r-1}, Y^{(i)}_{t,r})\}_{t\in[b]} \xleftarrow{R} \{0,1\}^\lambda$.
    - For rounds j from r to 2 and $\forall t \in [b]$,
      - Compute $Y^{(i)}_{t,j-2}$ from $Y^{(i)}_{t,j}$, $Y^{(i)}_{t,j-1}$ as

$$Y^{(i)}_{t,j-2} = \mathsf{T}'(Y^{(i)}_{t,j-1}) \oplus \mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j}).$$

    - For each block $\forall t \in [b]$, reprogram H

$$\mathsf{H}(\rho_i||t) = m_t \oplus Y^{(i)}_{t,0}||Y^{(i)}_{t,1}.$$

    - Let $Z_t = Y^{(i)}_{t,r-1}||Y^{(i)}_{t,r}$.
    - Let $y^{(i)}_r = Z^{(i)}_1||\dots||Z^{(i)}_b$ and output $(y^{(i)}_r, \rho)$.

  $\mathcal{C}$ returns $y^{(1)}, y^{(2)}, \dots y^{(n)}$ to $\mathcal{A}_1$.
- **Phase 2, State Sharing, Phase 3, Guess:** ...
- **Verify:** Let $v_i = 1$ if $\tilde{y}^{(i)} = y^{(i)}$ and $v_i = 0$ otherwise. Adversary wins if $\mathsf{flag} = 0$ and $|\mathsf{state}| < \sum v_i \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)$.

Game 2: In this game an adversary wins if they query on the oracle rather than outputting the replica. This helps us ease the notation by only focussing at the oracle query lists. This is used later when we argue the order the adversary must make certain random oracle queries.

- **Setup, Phase 1, File Challenge, Phase 2, State Sharing, Phase 3:** ....
- **Guess:**....
  $\mathcal{C}$ adds the guess to $\mathcal{A}_2$'s lists of queries to T' in Phase 3, i.e. $\forall i \in [n]$, let $\tilde{y}^{(i)} = (\tilde{Z}^{(i)}_1||\dots||\tilde{Z}^{(i)}_r, \tilde{\rho}_i)$, where $\tilde{Z}^{(i)}_t = \tilde{Y}^{(i)}_{t,r-1}||\tilde{Y}^{(i)}_{t,r}$. $\forall t \in [b]$ add $\tilde{Y}^{(i)}_{t,r-1}$ and $\tilde{Y}^{(i)}_{t,r}$ to list of queries to T' by $\mathcal{A}_2$ in Phase 3.
- **Verify:** Let $v_i = 1$ if $\forall t \in [b]$, T' is queried on $Y^{(i)}_{t,r-1}$ and $Y^{(i)}_{t,r}$ and $v_i = 0$ otherwise. Adversary wins if $\mathsf{flag} = 0$ and $|\mathsf{state}| < \sum v_i \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)$.

Game 3: In this game, we look at the queries made by the adversary and show that if it didn't sequentially query (almost sequentially), then it has a negligible chance of winning. Let $\mathsf{LCS}[x, y]$ denote the longest common subsequence between two lists x, y. Since we want soundness for adversaries which can store information close to the encoding size (i.e. $< 2 \cdot \lambda \cdot b$). It should be able to query oracle T' on atmost one query out of place (it does not have enough to store the complete replica encoding or make two queries out of the sequence). The notion that it can query oracle T' out of place at one spot is captured by the LCS metric. For difficulties in the case analysis of our proof, we relax and allow the adversary to make two out of place queries. We claim that if the adversary wins with non negligible probability then there exists atleast one block in some replica that has LCS greater than equal to $r-2$.

- **Setup, Phase 1, File Challenge, Phase 2, State Sharing, Phase 3, Guess:** ....
- **Sequentiality:**
  If $\forall i \in [n]\ \forall t \in [b]$,
  $|\mathsf{LCS}[(Y^{(i)}_{t,1}, \ldots, Y^{(i)}_{t,r}), \mathbf{w}^{\mathsf{T}'}]| < r-2$, then set $\mathsf{flag} = 1$.
- **Verify:** Let $v_i = 1$ if $\forall t \in [b]$, T' is queried on $Y^{(i)}_{t,r-1}$ and $Y^{(i)}_{t,r}$ and $v_i = 0$ otherwise. Adversary wins if $\mathsf{flag} = 0$ and $|\mathsf{state}| < \sum v_i \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)$.

Game 4: In this game, we guess the block which the adversary traversed sequentially. We concentrate on one randomly chosen block and replica and the adversary wins if it outputs the correct encoding for this block. We lose a multiplicative factor of $b \cdot n$ in the reduction due to this change. In contrast with the permutation case, here the adversary also guesses the positions where the adversary was non-sequential. This looses an additional factor of $\binom{r}{2}$ in the reduction.

- **Setup:** The challenger(denoted by $\mathcal{C}$) runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{rSetup}(1^\kappa, 1^n)$ and sends public key pk to $\mathcal{A}_1$. It keeps the secret key sk for itself.
  Choose random $\alpha_1 < \alpha_2 \in [r]$, $\beta \in [b]$ and $\gamma \in [n]$.
- **Phase 1, File Challenge, Phase 2, State Sharing, Phase 3, Guess:** ....
- **Sequentiality:**
  If $\{Y^{(\gamma)}_{\beta,1}, \ldots, Y^{(\gamma)}_{\beta,\alpha_1-1}, Y^{(\gamma)}_{\beta,\alpha_1+1}, \ldots, Y^{(\gamma)}_{\beta,\alpha_2-1}, Y^{(\gamma)}_{\beta,\alpha_2+1}, \ldots, Y^{(\gamma)}_{\beta,r}\}$ is not a subsequence of $\mathbf{w}^{\mathsf{T}'}$, then set $\mathsf{flag} = 1$.
- **Verify:** ~~Let $v_i = 1$ if $\forall t \in [b]$, T' is queried on $Y^{(i)}_{t,r-1}$ and $Y^{(i)}_{t,r}$ and $v_i = 0$ otherwise.~~ Adversary wins if T' is queried on $Y^{(\gamma)}_{\beta,r-1}$ and $Y^{(\gamma)}_{\beta,r}$, $\mathsf{flag} = 0$, and $|\mathsf{state}| < n \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)$.

Game 5: In this game, we reprogramme the oracles H,T' to have a random function which we can analyze cleanly. The primary idea behind this game is that there will exist two sequences of values on the chosen block and replica for which any adversary $\mathcal{A}_1$ produces the same state. These possibilities for a "switch" are set up in this game. H is programmed to output $Y^{(\gamma)}_{\beta,0}, Y^{(\gamma)}_{\beta,1}$ and for $i \in [r-1]$, the values $A_{i-1,0}, A_{i-1,1}$ have a choice to be related to either of the two $A_{i,0}, A_{i,1}$ and one among $A_{i+1,0}, A_{i+1,1}$ depending on the sampled index x. The collision check makes sure that the reprogramming doesn't erroneously reprogram the same value twice and the prequery check is done to make sure that none of the values were queried in the oracle lists in the previous phase. The oracle T'_{x} is then reprogrammed according to the reprogram operation defined in Definition 6 where for $i \in [r-1]$, x_i is now mapped to $\mathsf{f}(\mathsf{pk}, \mathsf{x}_{i+1}) \oplus \mathsf{x}_{i-1}$ where x_i is used to indicate the notation for $A_{i,\mathsf{x}[i]}$.

- **Setup, Phase 1:** ....
- **Sampling a new function:**
  - Sample possible switches $Y^{(\gamma)}_{\beta,0}, Y^{(\gamma)}_{\beta,1}, A_{2,0}, \ \ldots\ , A_{r,0}, A_{2,1}, \ \ldots\ , A_{r,1} \xleftarrow{R} \{0,1\}^\lambda$.
    Let $\mathcal{Z}_1 = \{Y^{(\gamma)}_{\beta,0}, Y^{(\gamma)}_{\beta,1}, A_{2,0} \ldots, A_{r,0}, A_{2,1}, \ldots, A_{r,1}\}$.
    Collision Check: If $|\mathcal{Z}_1| \neq 2r$, set $\mathsf{flag} = 1$.
    Prequery Check T': If $\mathcal{Z}_1 \cap \mathbf{u}^{\mathsf{T}'} \neq \emptyset$, set $\mathsf{flag} = 1$.
  - Sample a random setting $\mathsf{x} \xleftarrow{R} \{0,1\}^{r-1}$. Let $\mathsf{x}[k]$ denote the k^{th} bit of x'. We will write x'_j to refer to $A_{j,\mathsf{x}'[j-2]}$ and denote $A_{j,\bar{\mathsf{x}}'[j-2]}$ with $\bar{\mathsf{x}}'_j$.
  - Define T'_{x} to be T' and perform the following changes:

$$\forall j \in [r-1]\ \mathsf{T}'_{\mathsf{x}}(\mathsf{x}_j) = \mathsf{x}_{j-1} \oplus \mathsf{f}(\mathsf{pk}, \mathsf{x}_{j+1}).$$

- **Phase 1:** Use T'_{x} to answer queries for T'.
- **File Challenge:** $(1^n, m \in \{0,1\}^*) \leftarrow \mathcal{A}_1^{\mathsf{H}(\cdot),\mathsf{T}'(\cdot)}(1^\kappa, \mathsf{pk})$. It sends $(1^n, m)$ to $\mathcal{C}$ who does the following:
  - Divide m into b blocks of length 2λ i.e. $m = m_1||m_2||\dots||m_b$, $b = \lceil |m|/2\lambda \rceil$.
  - For $i \in [n]$,
    - Choose a string $\rho_i \xleftarrow{R} \{0,1\}^\kappa$.
      Prequery Check H: If $\exists t \in [b]: \ \rho_i||t \in \mathbf{u}^{\mathsf{H}}$, set $\mathsf{flag} = 1$.
    - Sample $\{(Y^{(i)}_{t,r-1}, Y^{(i)}_{t,r})\}_{t\in[b]} \xleftarrow{R} \{0,1\}^\lambda$.
    - For rounds j from r to 2 and $\forall t \in [b]$, continue if $t \neq \beta$ or $i \neq \gamma$,
      - Compute $Y^{(i)}_{t,j-2}$ from $Y^{(i)}_{t,j}$, $Y^{(i)}_{t,j-1}$ as
      $$Y^{(i)}_{t,j-2} = \mathsf{T}'(Y^{(i)}_{t,j-1}) \oplus \mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j}).$$
    - For each block $\forall t \in [b]$, reprogram H
      $$\mathsf{H}(\rho_i||t) = m_t \oplus Y^{(i)}_{t,0}||Y^{(i)}_{t,1}.$$
    - Let $Z_t = Y^{(i)}_{t,r-1}||Y^{(i)}_{t,r}$.
    - Let $y^{(i)}_r = Z^{(i)}_1||\dots||Z^{(i)}_b$ and output $(y^{(i)}_r, \rho)$.

  $\mathcal{C}$ returns $y^{(1)}, y^{(2)}, \dots y^{(n)}$ to $\mathcal{A}_1$.
- **Phase 2:** Use T'_{x} to answer queries for T'.
- **State Sharing:** ....
- **Phase 3:** Use T'_{x} to answer queries for T'.
- **Guess, Sequentiality:** ....
- **Verify:** Adversary wins if T' is queried on $Y^{(\gamma)}_{\beta,r-1}$ and $Y^{(\gamma)}_{\beta,r}$, $\mathsf{flag} = 0$, and $|\mathsf{state}| < n \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)$.

Game 6: In this game, $\mathcal{C}$ has unbounded computation time and calls $\mathcal{A}_1, \mathcal{A}_2$ exponentially many times to find a collision to state through the procedure search. The setting y' for which the procedure search outputs a collision in state is stored in a set which is outputted at the end of the procedure. $\mathsf{search}(1^\kappa, \mathsf{y}, \mathsf{state}, \alpha_1, \alpha_2; \zeta)$ takes input $\mathsf{y}, \mathsf{state}, \alpha_1, \alpha_2$ and runs algorithms $\mathcal{A}_1, \mathcal{A}_2$ on Game 5. Let ζ be the randomness used by the procedure and denotes all the random coins (except those used to sample x) used by $\mathcal{C}$. The procedure is described in Figure 3. The additional complexity of this function in contrast to Figure 2 is due to the weaker conformity condition of $\mathsf{LCS}(Y_t) < r-2$ rather than $\mathsf{LCS}(Y_t) < r$. Since our goal is to argue indistinguishability via Lemma 2, we need to fix more points in the search space y'. The quantities α_1, α_2 represent the two places where the adversary might have queried out of the sequence. The six indices $\alpha_1 - 1, \alpha_1, \alpha_1 + 1, \alpha_2 - 1, \alpha_2, \alpha_2 + 1$ are all fixed so that no reprogramming is observed at α_1 and α_2.

- **Setup, Phase 1, Sampling a New Function, File Challenge, Phase 2, State Sharing:** ....
- **Running search:** Let ζ be all the random coins (except those used to sample x) used by $\mathcal{C}$. Let $\mathcal{S} \leftarrow \mathsf{search}(1^\kappa, \mathsf{x}, \mathsf{state}, \alpha_1, \alpha_2; \zeta)$.
  If $\mathcal{S} = \emptyset$ set $\mathsf{flag} = 1$ and $\mathsf{x}' = \mathsf{x}$, otherwise sample $\mathsf{x}' \xleftarrow{R} \mathcal{S}$.
- **Setting switched oracle:**
  - Let $\mathsf{x}'[k]$ denote the k^{th} bit of x'. We will write x'_j to refer to $A_{j,\mathsf{x}'[j-1]}$ and denote $A_{j,\bar{\mathsf{x}}'[j-1]}$ with $\bar{\mathsf{x}}'_j$. Let x'_0 denote $Y^{(\gamma)}_{\beta,0}$ and x'_1 denote $Y^{(\gamma)}_{\beta,1}$.
  - Define $\mathsf{T}'_{\mathsf{x}'}$ to be T' and perform the following changes:
  $$\forall j \in [r-1] \ \mathsf{T}'_{\mathsf{x}'}(\mathsf{x}'_j) = \mathsf{x}'_{j-1} \oplus \mathsf{f}(\mathsf{pk}, \mathsf{x}'_{j+1})$$
- **Phase 3:** Use $\mathsf{T}'_{\mathsf{x}'}$ to answer queries for T respectively.

$\mathsf{search}(1^\kappa, \mathsf{y}, \mathsf{state}, \alpha_1, \alpha_2; \zeta)$

Inputs: Security parameter - 1^κ
Oracle Settings on T - $\mathsf{y} \in \{0,1\}^{r-1}$
State - state
Out of Sequence Query indices - α_1, α_2
Randomness used in the game - ζ

Output: Set containing all oracle settings with collision in state - $\mathcal{S}$

- Set $\mathcal{S} = \emptyset$.
- $\forall \mathsf{y}' \neq \mathsf{y} \in \{0,1\}^{r-1} : \forall a \in \{\alpha_1, \alpha_2\}\ \forall k \in [3]\ \mathsf{y}'[a-k] = \mathsf{y}[a-k]$,
  - Run $\mathcal{A}_1, \mathcal{A}_2$ on Game 5 with randomness defined by ζ and using y' instead of x in the game.
  - Let state' be the state shared between $\mathcal{A}_1, \mathcal{A}_2$.
  - If $\mathsf{state}' = \mathsf{state}$ and $\mathcal{A}_2$ wins Game 5, then $\mathcal{S} = \mathcal{S} \cup \{\mathsf{y}'\}$.
- Output $\mathcal{S}$.

Figure 3: Routine search

- **Guess:** ....
- **Sequentiality:**
  If $\{\mathsf{x}'_0, \ldots, \mathsf{x}'_{\alpha_1-1}, \mathsf{x}'_{\alpha_1+1}, \ldots, \mathsf{x}'_{\alpha_2-1}, \mathsf{x}'_{\alpha_2+1}, \ldots, \mathsf{x}'_r\}$ is not a subsequence of $\mathbf{w}^{\mathsf{T}'}$, then set $\mathsf{flag} = 1$.
- **Verify:** Adversary wins if T' is queried on x'_{r-1} and x'_r, $\mathsf{flag} = 0$, and $|\mathsf{state}| < n \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)$.

Game 7: In this game we modify the verification step for which an adversary can win this game. We increase it's winning probability so that the adversary can win if it doesn't query the full sequence, but queries at the point where the sequences x, x' diverge. Notice that we define another oracle $\mathsf{T}'^{\delta}_{\mathsf{x}'}$ here that doesn't reprogram the complete sequence. This change is statistically indistinguishable to the adversary.

- **Setup, Phase 1, Sampling a New Function, File Challenge, Phase 2, State Sharing, Running search:** ....
- **Setting switched oracle:**
  - Let $\mathsf{x}'[k]$ denote the k^{th} bit of x'. We will write x'_j to refer to $A_{j,\mathsf{x}'[j-1]}$ and denote $A_{j,\bar{\mathsf{x}}'[j-1]}$ with $\bar{\mathsf{x}}'_j$. Set x'_0 to denote $Y^{(\gamma)}_{\beta,0}$.
  - Let δ be the first index for which $\mathsf{x}_\delta \neq \mathsf{x}'_\delta$.
  - Define $\mathsf{T}'^{\delta}_{\mathsf{x}'}$ to be T' and perform the following changes:

$$\begin{aligned}
\forall j \in [\delta-2], \mathsf{T}'^{\delta}_{\mathsf{x}}(\mathsf{x}_j) &= \mathsf{x}_{j-1} \oplus \mathsf{f}(\mathsf{pk}, \mathsf{x}_{j+1}) \\
\mathsf{T}'^{\delta}_{\mathsf{x}}(\mathsf{x}_{\delta-1}) &= \mathsf{x}_{\delta-2} \oplus \mathsf{f}(\mathsf{pk}, \bar{\mathsf{x}}_\delta) \\
\mathsf{T}'^{\delta}_{\mathsf{x}}(\mathsf{x}_{\alpha_1}) &= \mathsf{x}_{\alpha_1-1} \oplus \mathsf{f}(\mathsf{pk}, \mathsf{x}_{\alpha_1+1}) \\
\mathsf{T}'^{\delta}_{\mathsf{x}}(\mathsf{x}_{\alpha_2}) &= \mathsf{x}_{\alpha_2-1} \oplus \mathsf{f}(\mathsf{pk}, \mathsf{x}_{\alpha_2+1}).
\end{aligned}$$

  Note - this is equivalent to

$$\forall j \in [\delta-1] \cup \{\alpha_1, \alpha_2\}\ \mathsf{T}'^{\delta}_{\mathsf{x}'}(\mathsf{x}'_j) = \mathsf{x}'_{j-1} \oplus \mathsf{f}(\mathsf{pk}, \mathsf{x}'_{j+1})$$

- **Phase 3, Guess:** ....

- ~~**Sequentiality:**~~
- **Verify:** Adversary wins of T' is queried on $\bar{\mathsf{x}}_\delta (= \mathsf{x}'_\delta)$, $\mathsf{flag} = 0$ ~~and $|\mathsf{state}| < n \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)$~~

Game 8: In this game we observe that $\mathcal{C}$ need not be unbounded computation time and only needs to the guess the first prefix at which x, x' differ to successfully output one sequential query.

- **Setup, Phase 1, Sampling a New Function, File Challenge, Phase 2, State Sharing:** ....
- ~~**Running search:**~~
- **Setting switched oracle:**
  - ~~Let $\mathsf{x}'[k]$ denote the k^{th} bit of x'. We will write x'_j to refer to $A_{j,\mathsf{x}'[j-1]}$ and denote $A_{j,\mathsf{x}'[j-1]}$ with $\bar{\mathsf{x}}'_j$. Set x'_0 to denote $Y^{(\gamma)}_{\beta,0}$.~~
  - Let $\delta \xleftarrow{R} [2, r]$.
  - Define $\mathsf{T}'^{\delta}_{\mathsf{x}'}$ to be T' and perform the following changes:

$$\begin{aligned}\forall j \in [\delta - 2], \mathsf{T}'^{\delta}_{\mathsf{x}}(\mathsf{x}_j) &= \mathsf{x}_{j-1} \oplus \mathsf{f}(\mathsf{pk}, \mathsf{x}_{j+1}) \\ \mathsf{T}'^{\delta}_{\mathsf{x}}(\mathsf{x}_{\delta-1}) &= \mathsf{x}_{\delta-2} \oplus \mathsf{f}(\mathsf{pk}, \bar{\mathsf{x}}_\delta) \\ \mathsf{T}'^{\delta}_{\mathsf{x}}(\mathsf{x}_{\alpha_1}) &= \mathsf{x}_{\alpha_1-1} \oplus \mathsf{f}(\mathsf{pk}, \mathsf{x}_{\alpha_1+1}) \\ \mathsf{T}'^{\delta}_{\mathsf{x}}(\mathsf{x}_{\alpha_2}) &= \mathsf{x}_{\alpha_2-1} \oplus \mathsf{f}(\mathsf{pk}, \mathsf{x}_{\alpha_2+1}).\end{aligned}$$

- **Phase 3, Guess:** ....
- **Verify:** Adversary wins of T' is queried on $\bar{\mathsf{x}}_\delta$ and $\mathsf{flag} = 0$.

6.2.2 Indistinguishability of Games

Let $\mathsf{F}_i(\kappa)$ (denoted by F_i) be the probability that the adversaries win at the end of Game i.

Game 0

Proof. Game 0 is a restatement of the original $\mathsf{Sound}_{\mathcal{A}_1,\mathcal{A}_2}(\kappa, n)$ game with two differences, (i) the syntactical change to note down queries to each oracle, (ii) expands on rEnc. Both the syntactical changes do not change the functioning of the game. □

Game 1

Lemma 13. $\Pr[\mathsf{F}_1] \geq \Pr[\mathsf{F}_0] - \mathsf{negl}(\kappa)$.

Proof. Game 1 differs from Game − in how queries are answered to the adversaries and the possibility of flag being set. Let $\Pr[\mathsf{F}_0] = \epsilon$ be the probability of adversary winning in Game 0. Let E^{H} be the event flag is set due to **Prequery Check** H. We can upper bound the difference in probability of F_1 and F_0 by the sum of (i) the probability that E^{H} occurs, (ii) and the statistical difference of the output of $\mathcal{C}$ from the alternate method of encoding generation.

Claim 17. $\Pr[\mathsf{E}^{\mathsf{H}}] = \mathsf{negl}(\kappa)$.

Proof. Since each ρ_i is generated uniformly on $\{0,1\}^\kappa$ and independent of $\mathcal{A}_1$, we can bound the probability that any fixed query is equal to a particular ρ_i as $k \in [\mathsf{q}_1{}^{\mathsf{H}}]\ i \in [n]\ \Pr[\mathsf{u}^{\mathsf{H}}_k = \rho_i] = 2^{-\kappa}$. From this, we can union bound the

$$\Pr[\exists k \in [\mathsf{q}_1{}^{\mathsf{H}}]\ i \in [n]\ \mathsf{u}^{\mathsf{H}}_k = \rho_i] \leq \mathsf{q}_1{}^{\mathsf{H}} \cdot n \cdot 2^{-\kappa}$$

Since $\mathsf{q}_1{}^{\mathsf{H}}, n \in \mathsf{poly}(\kappa)$, this is negligible □

Claim 18. *The distribution of* $\mathsf{T}' \times \{Y^{(i)}_{t,r-1}, Y^{(i)}_{t,r}\}$ *in Game* 0 *is statistically close to uniform.*

Proof. First, observe that with probability $1 - \mathsf{negl}(\kappa)$, $\forall i \in [n], t \in [b]$ $\rho_i||t$ is not queried H on by $\mathcal{A}_1$ before submitting m. This is apparent as each ρ_i is uniformly random on a domain of size 2^κ and $\mathcal{A}_1$ can make at most $\mathsf{poly}(\kappa)$ queries. Since $Y^{(i)}_{t,0}||Y^{(i)}_{t,1} = \mathsf{H}(\rho_i||t) \oplus m_t$, and m is independent of $\mathsf{H}(\rho_i||t)$, we can say $\{(Y^{(i)}_{t,0}, Y^{(i)}_{t,1})\}$ is uniform and independent of at least a $1 - \mathsf{negl}(\kappa)$ fraction of T', and once an T' is fixed, this defines a bijective relation from $(Y^{(i)}_{t,0}, Y^{(i)}_{t,1})$ to $(Y^{(i)}_{t,r-1}, Y^{(i)}_{t,r})$, so the latter is also uniform and independent of at least a $1 - \mathsf{negl}(\kappa)$ fraction of T'. Bounding the statistical distance with $\mathsf{negl}(\kappa)$. □

Since in Game 1, it is apparent that $\mathsf{T}' \times \{Y^{(i)}_{t,r-1}, Y^{(i)}_{t,r}\}$ is uniform by the fact that $\{Y^{(i)}_{t,r-1}, Y^{(i)}_{t,r}\}$ are generated independent of T', Claim 18 bounds the statistical distance between the responses of $\mathcal{C}$ between Game 0 and 1 with $\mathsf{negl}(\kappa)$. Combined with the previous claim, we can conclude the total difference between Game 0 and 1 is negligible. □

Game 3

Lemma 14. $\Pr[\mathsf{F}_3] \geq \Pr[\mathsf{F}_2] - \mathsf{negl}(\kappa)$

Proof. Observe that these games only differ when flag is set to 1 and adversary ends up winning Game 2. Let ϵ be the probability that $(\mathcal{A}_1, \mathcal{A}_2)$ wins in such a manner. We will refer to this as winning non-sequentially.

Claim 19. $\mathcal{A}_1$ *will only query on* $Y^{(i)}_{t,j}$ *in* ***Phase 1*** *with negligible probability.*

$$\Pr[\exists (i,j,t) : Y^{(i)}_{t,j} \in \mathbf{u}^{\mathsf{T}'}] = \mathsf{negl}(\kappa).$$

Proof. Once we fix T', consider the bijective mapping from $Y^{(i)}_{t,r-1}, Y^{(i)}_{t,r}$ to $Y^{(i)}_{t,j}, Y^{(i)}_{t,j+1}$ defined by $Y^{(i)}_{t,j-2} = \mathsf{T}'(Y^{(i)}_{t,j-1}) \oplus \mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j})$. Since $Y^{(i)}_{t,r-1}, Y^{(i)}_{t,r}$ are uniformly random and independent of T' by construction, we conclude that $Y^{(i)}_{t,j}$ is as well, which lets us union bound the probability,

$$\Pr[\exists (i,j,t) : Y^{(i)}_{t,j} \in \mathbf{u}^{\mathsf{T}'}] \leq \sum_{(i,j,t) \in [n] \times [r] \times [b]} \left(\sum_{q=0}^{{\mathsf{q}_1}^{\mathsf{T}'}} \Pr[Y^{(i)}_{t,j} = \mathsf{u}^{\mathsf{T}'}_q] \right) = n \cdot b \cdot r \left({\mathsf{q}_1}^{\mathsf{T}'} \cdot 2^{-\lambda} \right)$$

Since $b, r, n, {\mathsf{q}_1}^{\mathsf{T}'}$ are all $\mathsf{poly}(\kappa)$, this is negligible. □

Claim 20. *With all but negligible probability, the* $\{Y^{(i)}_{t,j}\}$ *are distinct*

$$\Pr[\exists (i_1,j_1,t_1) \neq (i_2,j_2,t_2) : Y^{(i_1)}_{t_1,j_1} = Y^{(i_2)}_{t_2,j_2}] \leq \mathsf{negl}(\kappa).$$

Proof. We do a case by case analysis,

- Assume $(i_1, t_1) = (i_2, t_2)$ - that these two values are in the same block and replica, but on different rounds. We will proceed by an induction argument that any two such $Y^{(i_1)}_{t_1,j}$ will only be equal with negligible probability. In our base case, we note that since $Y^{(i_1)}_{t_1,r-1}$ and $Y^{(i_1)}_{t_1,r}$ are chosen independently at random, and so collide with probability $2^{-\lambda}$. In our induction step, consider the probability $Y^{(i_1)}_{t_1,j}$ collides with anything in $\{Y^{(i_1)}_{t_1,j+1}, \ldots, Y^{(i_1)}_{t_1,r}\}$. Since $Y^{(i_1)}_{t_1,j} = \mathsf{T}'(Y^{(i_1)}_{t_1,j+1}) \oplus \mathsf{f}(\mathsf{pk}, Y^{(i_1)}_{t_1,j+2})$, if $\{Y^{(i_1)}_{t_1,j+1}, \ldots, Y^{(i_1)}_{t_1,r}\}$ are unique, then they are all independent of $\mathsf{T}'(Y^{(i_1)}_{t_1,j+1})$, so we can union bound the probability that $Y^{(i_1)}_{t_1,j}$ collides with any of them with $(r-j)2^{-\lambda}$, which is negligible. Since the number of rounds, blocks, and replicas are polynomial, this remains negligible $\forall i_1 \in [n]$ $t_1 \in [b]$ $|\{Y^{(i_1)}_{t_1,j}\}_{j \in [b]}| < b$.

- Assume $(i_1, t_1) \neq (i_2, t_2)$ - Let's fix T' and some value for $Y^{(i_1)}_{t_1,j_1}$. Recall that fixing T' induces a bijective relation from $Y^{(i_2)}_{t_2,r-1}, Y^{(i_2)}_{t_2,r}$ to $Y^{(i_2)}_{t_2,j_2}, Y^{(i_2)}_{t_2,j_2+1}$ via $Y^{(i_2)}_{t_2,j-2} = \mathsf{T}'(Y^{(i_2)}_{t_2,j-1}) \oplus \mathsf{f}(\mathsf{pk}, Y^{(i_2)}_{t_2,j})$. Since $Y^{(i_2)}_{t_2,r-1}, Y^{(i_2)}_{t_2,r}$ are uniformly random and independent of $Y^{(i_1)}_{t_1,j_1}$, so is $Y^{(i_2)}_{t_2,j_2}$, so the probability of collision is $2^{-\lambda}$. We can union bound over $t_1, i_1, j_1, t_2, i_2, j_2$, which are all $\mathsf{poly}(\kappa)$, so the union is still negligible.

□

Claim 21. *Let $\mathcal{N}$ be a permutation of $(1, 2, \ldots, n)$ such that the longest increasing subsequence of $\mathcal{N}$ is of length $< n-2$. Let $\mathcal{N}_i$ be the mapping of i in $\mathcal{N}$. Then either (i) $\exists i_1 \neq i_2$ such that $\mathcal{N}_{i_1} > \max(\mathcal{N}_{i_1+1}, \mathcal{N}_{i_1-1})$ and $\mathcal{N}_{i_2} > \max(\mathcal{N}_{i_2+1}, \mathcal{N}_{i_2-1})$, (ii) $\exists i : \mathcal{N}_{i-1} > \mathcal{N}_i > \mathcal{N}_{i+1}$, or (iii) $\exists i : \max(\mathcal{N}_{i+1}, \mathcal{N}_{i+2}) < \min(\mathcal{N}_i, \mathcal{N}_{i-1})$*

Proof. Assume for sake of contradiction there is such a $\mathcal{N}$ with longest increasing subsequence $\mathcal{N}'$ where there is at most one i such that $\mathcal{N}_i > \max(\mathcal{N}_{i+1}, \mathcal{N}_{i-1})$ (corresponding to (i)) and no i such that either (ii) or (iii) are true. However, note that for any $\mathcal{N}_i < \max(\mathcal{N}_{i+1}, \mathcal{N}_{i-1})$ and $\mathcal{N}_{i-1} < \mathcal{N}_i \vee \mathcal{N}_i < \mathcal{N}_{i+1}$ (Complements of conditions (i) and (ii)) we can conclude that $\mathcal{N}_i < \mathcal{N}_{i+1}$. Thus, there exists only one i for which $\mathcal{N}_i > \mathcal{N}_{i+1}$. But by (iii), we know $\max(\mathcal{N}_{i+1}, \mathcal{N}_{i+2}) > \min(\mathcal{N}_i, \mathcal{N}_{i-1})$, so we can using at least one each of $(\mathcal{N}_i, \mathcal{N}_{i-1})$ and $(\mathcal{N}_{i+1}, \mathcal{N}_{i+2})$. Since all other elements are ordered, we have produced an increasing subsequence of length $n-2$, a contradiction. □

Now consider the following computationally unbounded algorithm $\mathcal{B}'$ with access to oracle T'. This algorithm will translate a non-sequential $\mathcal{A}_2$ into a reduction to the game outlined in Lemma 2. We will only consider 'conforming' adversaries $\mathcal{A}_2$ which, when successful, query all possible $Y^{(i)}_{t,j}$. We also observe that any adversary can be transformed into a 'conforming' adversary by simply using the $(Y^{(i)}_{t,r-1}, Y^{(i)}_{t,r})$ output to query on all lower $Y^{(i)}_{t,j}$. If the permutation of $(1, \ldots r)$ induced by the order of queries of $\{Y^{(i)}_{t,j}\}_{j\in[r]}$ to $\mathbf{w}^{\mathsf{T}'}$ on some block and replica falls into one of the 3 categories outlined in Claim 21, we present a strategy to output two $(x_i, \mathsf{T}'(x_i))$ pairs without querying x_i on T'. Roughly speaking, these strategies will work by computing backwards from non-sequential queries to recover the supposed output of certain oracle queries without ever making said queries.

Reduction $\mathcal{B}'^{\mathsf{T}'(\cdot)}(\mathsf{advice})$:

Goal: Produce Input Output oracle pairs without explicitly querying the oracle.

- **Setup:**
  - Sample a random function $\mathsf{H}(\cdot)$ and use it to answer oracle queries made by $\mathcal{A}_1, \mathcal{A}_2$.
  - Perform, **Setup** and **Phase 1** as in Game 3.
  - Receive $m \leftarrow \mathcal{A}_1^{\mathsf{H}(\cdot),\mathsf{T}'(\cdot)}(1^\kappa, \mathsf{pk}')$ after **Phase 1**. Parse pk' as (pk, n).
  - Choose a set of random $\{\rho_i\}_{i\in[n]}$ and compute $\{Y^{(i)}_{t,0} || Y^{(i)}_{t,1} = \mathsf{H}(\rho_i || t) \oplus m_t\}_{t\in[b],i\in[n]}$.
  - Parse advice as $(\mathsf{state}, \mathsf{Q} = \{\mathsf{hint}^{(i)}_t\})$ where $\mathsf{hint}^{(i)}_t$ is some $O(\log \kappa)$ length hint for each block and replica.
- **Simulate:**
  - Run $\mathcal{A}_2^{\mathsf{H}(\cdot),\mathsf{T}'(\cdot)}(1^\kappa, \mathsf{pk}', m, \mathsf{state})$, interacting with the random oracle queries it makes to T'. Let $\mathbf{w}^{\mathsf{T}'}$ denote the ordered and distinct list of queries $\mathcal{A}_2$ makes to T', and let $\mathbf{w}^{\mathsf{T}'}_q$ refer to the q^{th} element in this list. We perform the operations below while running $\mathcal{A}_2$.
  - Let $\mathcal{S} = \emptyset$. For each block/replica, the $\mathsf{hint}^{(i)}_t$ will fall into one of the 3 categories below:
  - **Case (i):** $\mathsf{hint}^{(i)}_t = (j, q_1, q_2, q_3, j', q'_1, q'_2, q'_3)$, where $q_2 > \max(q_1, q_3)$ and $q'_2 > \max(q'_1, q'_3)$. We will describe this procedure for q, j, but the same procedure is repeated for q', j'.

* Assume $Y^{(i)}_{t,j-1} = \mathsf{w}^{\mathsf{T}'}_{q_1}$ and $Y^{(i)}_{t,j+1} = \mathsf{w}^{\mathsf{T}'}_{q_3}$.
* Solve for $\mathsf{T}'(Y^{(i)}_{t,j}) = Y^{(i)}_{t,j-1} \oplus \mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j+1})$.
* Assume the q_2^{th} query is the first query to $Y^{(i)}_{t,j}$, and send $\mathsf{T}'(Y^{(i)}_{t,j})$ instead of querying the true T' whenver this value is queried.
* Add $(Y^{(i)}_{t,j}, \mathsf{T}'(Y^{(i)}_{t,j}))$ to $\mathcal{S}$.

– **Case (ii):** $\mathsf{hint}^{(i)}_t = (j, q_1, q_2, q_3)$, where $q_3 < q_2 < q_1$.
* Use $Y^{(i)}_{t,0}, Y^{(i)}_{t,1}$ to compute $Y^{(i)}_{t,j-1}$, querying T' on $Y^{(i)}_{t,1}, \dots Y^{(i)}_{t,j-2}$.
* Assume $Y^{(i)}_{t,j+1} = \mathsf{w}^{\mathsf{T}'}_{q_3}$.
* Solve for $\mathsf{T}'(Y^{(i)}_{t,j}) = Y^{(i)}_{t,j-1} \oplus \mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j+1})$.
* Assume the q_2^{th} query is the first query to $Y^{(i)}_{t,j}$, and send $\mathsf{T}'(Y^{(i)}_{t,j})$ instead of querying the true T' whenver this value is queried.
* Solve for $\mathsf{T}'(Y^{(i)}_{t,j-1}) = Y^{(i)}_{t,j-2} \oplus \mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j})$.
* Assume the q_1^{th} query is the first query to $Y^{(i)}_{t,j-1}$, and send $\mathsf{T}'(Y^{(i)}_{t,j-1})$ instead of querying the true T' whenver this value is queried.
* Add $(Y^{(i)}_{t,j-1}, \mathsf{T}'(Y^{(i)}_{t,j-1}))$ and $(Y^{(i)}_{t,j}, \mathsf{T}'(Y^{(i)}_{t,j}))$ to $\mathcal{S}$.

– **Case (iii):** $\mathsf{hint}^{(i)}_t = (j, q_3, q_4)$.
* Use $Y^{(i)}_{t,0}, Y^{(i)}_{t,1}$ to compute $Y^{(i)}_{t,j-1}$, querying T' on $Y^{(i)}_{t,1}, \dots Y^{(i)}_{t,j-2}$.
* Assume $Y^{(i)}_{t,j+1} = \mathsf{w}^{\mathsf{T}'}_{q_3}$ and $Y^{(i)}_{t,j+2} = \mathsf{w}^{\mathsf{T}'}_{q_4}$
* Solve for $Y^{(i)}_{t,j} = \mathsf{T}'(Y^{(i)}_{t,j+1}) \oplus \mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j+2})$.
* Solve for $\mathsf{T}'(Y^{(i)}_{t,j}) = \mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j+1}) \oplus Y^{(i)}_{t,j-1}$ and $\mathsf{T}'(Y^{(i)}_{t,j-1}) = \mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j}) \oplus Y^{(i)}_{t,j-2}$.
* Send our solved $\mathsf{T}'(Y^{(i)}_{t,j-1})$ and $\mathsf{T}'(Y^{(i)}_{t,j})$ whenever $\mathcal{A}_2$ queries on those inputs rather than querying the true T'.
* Add $(Y^{(i)}_{t,j-1}, \mathsf{T}'(Y^{(i)}_{t,j-1}))$ and $(Y^{(i)}_{t,j}, \mathsf{T}'(Y^{(i)}_{t,j}))$ to $\mathcal{S}$.

– For any other queries $\mathcal{A}_2$ makes to an oracle, $\mathcal{B}'$ simply queries the appropriate oracle, returns the query results and completes execution of $\mathcal{A}_2$.

- **Return:** Output the pairs $\mathcal{S}$.

Claim 22. *Suppose* $|\mathsf{LCS}[\mathsf{w}^{\mathsf{T}'}, \{Y^{(i)}_{t,1}, \dots, Y^{(i)}_{t,r}\}]| \leq r-3$. *Then if* $\mathcal{A}_2$ *wins,* $\exists$ *hints* $q^{(1)}, j^{(1)}, q^{(2)}, j^{(2)}, q^{(3)}, j^{(3)}$ *such that* $j^{(1)} \neq j^{(2)} \neq j^{(3)}$ *and* T' *was queried on* $Y^{(i)}_{t,j^{(1)}}$ *while* T' *was not queried on* $Y^{(i)}_{t,j^{(1)}-1}$ *and similarly for* $j^{(2)}$ *and* $j^{(3)}$. $\mathcal{B}'$ *on input* $\mathsf{advice} = (\mathsf{state}, Q)$ *where* $(i, t, q^{(1)}, j^{(1)}, q^{(2)}, j^{(2)}, q^{(3)}, j^{(3)}) \in \mathsf{Q}$ *outputs two* j, j' *such that* $(Y^{(i)}_{t,j-1}, \mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j-2} \oplus Y^{(i)}_{t,j}))$, $(Y^{(i)}_{t,j'-1}, \mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j'-2} \oplus Y^{(i)}_{t,j'}))$ *and in* ***Simulate*** *phase never queries* T' *on* $Y^{(i)}_{t,j-1}$ *and* $Y^{(i)}_{t,j'-1}$ *and additionally queries* T' *from the set* $\{Y^{(i)}_{t,j''}\}_{j'' \in [r]}$ *(in addition to* $\mathcal{A}_2$*'s queries).*

Proof. Consider the subsequence of $\mathsf{w}^{\mathsf{T}'}$ of elements in $\{Y^{(i)}_{t,1}, \dots, Y^{(i)}_{t,r}\}$. If $\mathcal{A}_2$ wins, then by the fact that $\mathcal{A}_2$ is conforming and this w^{T} is unique, this subsequence is a permutation of $\left(Y^{(i)}_{t,1}, \dots, Y^{(i)}_{t,r}\right)$ with longest increasing subsequence equal to the LCS described. By Claim 21, one of the three cases must occur. We can verify that in all cases, if $\mathcal{B}'$ is given the correct indices for said hint, it will be able to output those pairs without querying any additional pairs outside said (i, j) sequence. Since there is one hint per (i, j), we conclude that $\mathcal{B}$ will not query that pair in the **Simulate** phase. □

Claim 23. *If $\mathcal{A}_2$ wins nonsequentially with probability ϵ,*

$$\Pr\left[\begin{array}{c}\exists\ \mathsf{advice} \in \{0,1\}^*\ s.t.\ |\mathsf{advice}| \le n' \cdot \lambda - \omega(\log\lambda), \\ \{(x_i, y_i)\}_{i=1}^{n'} \leftarrow \mathcal{B}'^{\mathsf{T}'(\cdot)}(\mathsf{advice})\ where \\ \forall i \neq j \in [n'], x_i \neq x_j,\ \mathsf{T}'(x_i) = y_i,\ x_i \notin \mathsf{s}^{\mathsf{T}'}_{\mathcal{B}'}\end{array}\right] \ge \epsilon - \mathsf{negl}(\kappa).$$

Proof. We can take advice to be the state produced by $\mathcal{A}_1^{\mathsf{H}(\cdot),\mathsf{T}'}(1^\kappa, \mathsf{pk})$ and for every $i \in [n], t \in [b]$ we have $|\mathsf{LCS}[(Y^{(i)}_{t,1}, \ldots, Y^{(i)}_{t,r}), \mathbf{w}^{\mathsf{T}'}]| < r - 2$.

By Claim 22, a hint exist for each (i,t), and $\mathcal{B}'$ has outputted $2{\cdot}b{\cdot}n$ pairs $(Y^{(i)}_{t,j}, \mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j+1}))$, $(Y^{(i)}_{t,j'}, \mathsf{f}(\mathsf{pk}, Y^{(i)}_{t,j'+1}))$ without querying on T' in the **Simulate** phase. By Claim 20 and Claim 19, these pairs (x_i, y_i) are distinct, and that $\mathcal{B}'$ will not have queried any x_i or y_i in the **Setup** phase with all but negligible probability.

Since each hint outputs a constant number of q's and j's from domains of size $\le \mathsf{poly}(\kappa)$ by the running time of $\mathcal{A}_2$, Q only needs $b \cdot n \cdot O(\log\kappa)$ bits. As $\mathsf{s} \in 1 - \frac{\omega(\log\kappa)}{2\lambda}$, we get,

$$|\mathsf{state}| \le b \cdot n \cdot 2\lambda - b \cdot n \cdot \omega(\log(\kappa)) \Rightarrow |\mathsf{advice}| \le b \cdot n \cdot 2\lambda - b \cdot n \cdot \omega(\log(\kappa)) + b \cdot n \cdot O(\log\kappa)$$

$$\le b \cdot n \cdot \lambda - b \cdot n \cdot \omega(\log(\kappa)) \le b \cdot n \cdot 2\lambda - \omega(\log(\kappa)).$$

This proves the claim for $n' = 2b \cdot n$ which is polynomial in κ and hence polynomial in λ. □

By Lemma 2, $\epsilon - \mathsf{negl}(\kappa) \in \mathsf{negl}(\lambda) = \mathsf{negl}(\kappa) \Rightarrow \epsilon \le \mathsf{negl}(\kappa)$. □

Game 4

Claim 24. $\Pr[\mathsf{F}_4] \ge \frac{\Pr[\mathsf{F}_3]}{\binom{r}{2}bn}$.

Proof. Game 4 differs from Game 3 in the winning condition. Let $\Pr[\mathsf{F}_3] = \epsilon$ be the probability of adversary winning in winning Game 3. Let this adversary be $\mathcal{A}_2$. From the sequentiality condition we have that $\mathcal{A}_2$ on at least one block and replica, $\mathcal{A}_2$ queries all but possibly two $Y^{(i)}_{t,j}$ in order. The probability that these guesses were made correctly in Game 4 is $\ge \frac{1}{\binom{r}{2}bn}$. This adversary thus wins Game 4 with probability $\ge \frac{\epsilon}{\binom{r}{2}bn}$. □

Game 5

Claim 25. $\Pr[\mathsf{F}_5] \ge \Pr[\mathsf{F}_4] - \mathsf{negl}(\kappa)$.

Proof. Game 5 differs from Game 4 in how queries are answered to the adversaries and the possibility of flag being set. Let $\Pr[\mathsf{F}_4] = \epsilon$ be the probability of adversary winning in winning Game 4. Let $\mathsf{E}^{\mathsf{T}'}$ be the event flag is set due to **Prequery Check** T', and let E^{x} be the probability that flag is due to **Collision Check**. We can lower bound the probability that the adversary wins Game 5 relative to Game 4 by the sum of (i) the probability that E^{x} occurs, (ii) the probability $\mathsf{E}^{\mathsf{T}'}$ occurs, and (iii) the statistical difference of the output of $\mathcal{C}$ from using $\mathsf{T}', \mathsf{T}'_{\mathsf{x}}$.

Claim 26. $\Pr[\mathsf{E}^{\mathsf{x}}] = \mathsf{negl}(\kappa)$.

We note that since $\mathcal{Z}_1$ are uniform and independently random, we can bound the probability that $z_a, z_b \in \mathcal{Z}_1$ for any fixed $a \neq b$ is $\frac{1}{2^\lambda}$, so we can union bound the probability that

$$\Pr[\exists a \neq b : z_a = z_b] \le \sum_{a=0}^{2r} \sum_{b=a+1}^{2r} \Pr[z_a = z_b] = \binom{2r}{2} \frac{1}{2^\lambda} = \mathsf{negl}(\kappa)$$

Claim 27. $\Pr[\mathsf{E}^{\mathsf{T}'}] = \mathsf{negl}(\kappa)$

Proof. We note that since all $2r$ elements of $\mathcal{Z}_1$ are uniform and independent of T', we can bound the probability that some $z \in \mathcal{Z}_1$ is equal to some $z' \in \mathbf{u}^{\mathsf{T}'}$ with $\frac{1}{2^\lambda}$. Thus, we can union bound

$$\Pr[\mathcal{Z}_1 \cap \mathbf{u}^{\mathsf{T}'} \neq \emptyset] \leq 2r \cdot {\mathsf{q}_1}^{\mathsf{T}'} \cdot 2^{-\lambda}$$

Since ${\mathsf{q}_1}^{\mathsf{T}'}$ and r are $\mathsf{poly}(\kappa)$, this is $\mathsf{negl}(\kappa)$. □

Claim 28. *The statistical difference of the output of $\mathcal{C}$ using T' and using T'_{x} is $\mathsf{negl}(\kappa)$.*

Proof. Note the only outputs changed affected by the reprogramming of T'_{x} are the encoding $Y^{(\gamma)}_{\beta,r-1}, Y^{(\gamma)}_{\beta,r} = \mathsf{x}_{r-1}, \mathsf{x}_r$ and of course responses to $\mathcal{A}$'s queries to T'_{x}.

Applying Lemma 6 to $\{r_0, \dots r_k\} = \{\mathsf{x}_0 \dots \mathsf{x}_r\}$ and $\tau = \mathsf{f}(\mathsf{pk}, \cdot)$ gives us immediately that $(\mathsf{x}_{r-1}, \mathsf{x}_r, \mathsf{T}'_{\mathsf{x}})$ is uniform. Also recall by construction the distribution of $(Y^{(\gamma)}_{\beta,r-1}, Y^{(\gamma)}_{\beta,r}, \mathsf{T}')$ in Game 5 is uniform, so the responses distance of $\mathcal{C}$'s responses in Game 5 and 6 is negligible. □

Since (i), (ii), and (iii) are all negligible, the adversary thus wins Game 5 with probability $\epsilon - \mathsf{negl}(\kappa)$. □

Game 6

Let $\Pr[\mathsf{F}_5] = \epsilon$ be the probability of adversary winning in winning Game 5.

Lemma 15. $\Pr[\mathsf{F}_6] \geq \left(\frac{\epsilon}{2}\right)\left(\frac{\epsilon 2^{r-8} - 2^{|\mathsf{state}|} + 1}{\epsilon 2^{r-8}}\right)$.

Proof. Note that by construction, $\mathcal{A}$ will win Game 6 as long as an appropriate x' exists. We will lower bound said probability below. Let us consider the random coins used in Game 5. The randomness for $\mathcal{C}$ is over the choice of permutation picked T and $\gamma, \beta, \mathcal{Z}, \mathsf{x}, \delta, \rho'$ where ρ' denotes the randomness over ρ_i sampled for each replica and the coins used by the rSetup. Let us denote $\eta = (\gamma, \beta, \mathsf{T}, \mathcal{Z}, \rho')$ for ease of notation. Let $\mathcal{A}_1, \mathcal{A}_2$ denote the adversaries that solves Game 5 with ϵ probability.

Define the set $\mathcal{N} = \{(\eta, \mathsf{x}) | \mathsf{F}_5(\eta, \mathsf{x})\}$ where $\mathsf{F}_5(\eta, \mathsf{x})$ denotes the event that adversaries win game 5 with given parameters. Thus $\Pr_{\eta,\mathsf{x}}[(\eta, \mathsf{x}) \in \mathcal{N}] = \epsilon$.

Let us define a heavy set, $\mathcal{H} = \{(\eta, \mathsf{x}) | \Pr_{\eta,\mathsf{x}'}[(\eta, \mathsf{x}') \in \mathcal{N}] \geq \frac{\epsilon}{2}\}$. Then from the heavy row lemma of [15], $\Pr_{\eta,\mathsf{x}}[(\eta, \mathsf{x}) \in \mathcal{H} | (\eta, \mathsf{x}) \in \mathcal{N}] \geq 1/2$ i.e. probability that our winning instance lies on a heavy set is atleast half.

Note that for $\mathcal{A}_2$ to successfully solve Game 6, we must see the following events,

1. Adversary solves the Game 5 i.e. $(\eta, \mathsf{x}) \in \mathcal{N}$, and let this be denoted by event E_1.

$$\Pr_{\eta,\mathsf{x}}[\mathsf{E}_1] \geq \epsilon.$$

2. Let the event that $(\eta, \mathsf{x}) \in \mathcal{H}$ lies on a heavy set be denoted by E_2. By heavy row lemma, we have that,

$$\Pr_{\eta,\mathsf{x}}[\mathsf{E}_2 | \mathsf{E}_1] \geq 1/2 \Rightarrow \Pr_{\eta,\mathsf{x}}[\mathsf{E}_2] \geq \frac{\epsilon}{2}.$$

3. Let the event that there exists a collision of x consistent with state that is also successful on Game 5 be denoted by E_3.

   Claim 29. $\Pr_{\eta,\mathsf{x}}[\mathsf{E}_3 | \mathsf{E}_2] \geq \left(\frac{\epsilon 2^{r-8} - 2^{|\mathsf{state}|} + 1}{\epsilon 2^{r-8}}\right)$.

   Proof. Given that $(\eta, \mathsf{x}) \in \mathcal{H}$ i.e. it lies on a heavy set. Define a possible set of switches by $\mathcal{Q} = \{\mathsf{x}' | (\eta, \mathsf{x}') \in \mathcal{N}\}$. From the definition of $\mathcal{H}$, $|\mathcal{Q}| \geq \frac{\epsilon 2^{r-7}}{2}$ (Recall the total set of $\mathcal{H}$ considered by search are the set of $\{0,1\}^{r-1}$ which agree with x on up to 6 distinct indices $\alpha_1 - 1, \alpha_1 - 2, \alpha_1 - 3$, $\alpha_2 - 1, \alpha_2 - 2, \alpha_2 - 3$).

From Lemma 7, define h_η as a function from $\mathcal{Q} \to \{0,1\}^{|\mathsf{state}|}$ where $\mathsf{h}_\eta(a) = \mathsf{state}_a$ where state_a denotes the state shared by $\mathcal{A}_1$ to $\mathcal{A}_2$ when playing Game 5 with parameters η and x set to a. Since $\mathcal{A}_1$'s coins are deterministically fixed, h_η is a deterministic function.

The lemma implies the statement that,

$$\Pr_{\mathsf{x}}[\exists \mathsf{x}' \neq \mathsf{x} \in \mathcal{Q}, \mathsf{h}_\eta(\mathsf{x}) = \mathsf{h}_\eta(\mathsf{x}')] \geq \frac{|\mathcal{D}| - |\mathcal{R}| + 1}{|\mathcal{D}|}.$$

This proves the claim. □

$$\begin{aligned}\Pr_{\eta,\mathsf{x}}(\mathcal{A}_2 \text{ wins Game 6}) &\geq \Pr[\mathsf{E}_2 \cap \mathsf{E}_3] \\ &\geq (\Pr[\mathsf{E}_2])\,(\Pr[\mathsf{E}_3|\mathsf{E}_2]) \\ &\geq \left(\frac{\epsilon}{2}\right)\left(\frac{\epsilon 2^{r-8} - 2^{|\mathsf{state}|} + 1}{\epsilon 2^{r-8}}\right)\end{aligned}$$

□

We note that if F_5 is noticeable - i.e. $\epsilon \geq \frac{1}{\mathsf{poly}(\kappa)}$ and $|\mathsf{state}| < \mathsf{s}(\kappa, |m|) \cdot n \cdot \mathsf{len}(\kappa, |m|) \leq r - \omega(\log \kappa)$, then

$$\epsilon 2^{r-8} \geq 2^{r - O(\log \kappa)} \in \omega(2^{|\mathsf{state}|} + 1) \to \frac{\epsilon 2^{r-8} - 2^{|\mathsf{state}|} + 1}{\epsilon 2^{r-8}} \in (1 - o(1))$$

meaning F_6 is noticeable as well

Game 7

Claim 30. $\Pr[\mathsf{F}_7] \geq \Pr[\mathsf{F}_6]$.

Proof. Observe Game 7 differs from Game 6 in the winning condition and oracle responses to x'_j for $j \geq \delta$ and $j \neq \alpha_1, \alpha_2$.

If the adversary won Game 6, by sequentiality, $\mathcal{A}_2$ must have queried on x'_δ before querying on any x'_j for $j > \delta$ $j \neq \alpha_1, \alpha_2$. Since the adversary wins Game 7 when it queries x'_δ, the queries where $\mathsf{T}'^{\delta}_{\mathsf{x}'}$ differ from $\mathsf{T}'_{\mathsf{x}'}$ would only happen after $\mathcal{A}_2$ has already won - i.e. the queries before x'_δ are identical to those of Game 6, and so they will execute the same way. □

Game 8

Claim 31. $\Pr[\mathsf{F}_8] \geq \frac{\Pr[\mathsf{F}_7]}{r-1}$.

Proof. Our main observation here is that rather than guessing the switch completely, $\mathcal{C}$ can guess δ. The probability that this index was corretly guessed in Game 8 is equal to $\frac{1}{r-1}$. If guessed correctly, the games are identical, so the adversary queries x'_δ and hence wins Game 8 with probability $\geq \frac{\Pr[\mathsf{F}_7]}{r-1}$. □

Claim 32. $\Pr[\mathsf{F}_8] = \mathsf{negl}(\kappa)$

Proof. We will show through a reduction that an adversary able to win Game 8 with probability ϵ will also be able to invert our trapdoor permutation with the same probability.
Let $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa), y \xleftarrow{R} \mathcal{D}_{\mathsf{pk}} = \{0,1\}^\lambda, z = \mathsf{f}(\mathsf{pk}, y)$,

Reduction $\mathcal{B}(\mathsf{pk}, z)$

Goal: Output z such that, $\mathsf{f}(pk, y) = z$.

- **Setup:** The challenger(denoted by $\mathcal{C}$) ~~runs $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{rSetup}(1^\kappa, 1^n)$ and~~ sends public key pk' to $\mathcal{A}_1$. ~~It keeps the secret key sk' for itself.~~
- **Phase 1, Sampling a New Function, File Challenge, Phase 2, State Sharing:** ....
- **Setting switched oracle:**
  - Let $\delta \xleftarrow{R} [2, r]$.
  - Define $\mathsf{T}'^{\delta}_{\mathsf{x}'}$ to be T' and perform the following changes:

$$\begin{aligned}\forall j \in [\delta - 2], \mathsf{T}'^{\delta}_{\mathsf{x}}(\mathsf{x}_j) &= \mathsf{x}_{j-1} \oplus \mathsf{f}(\mathsf{pk}, \mathsf{x}_{j+1}) \\ \mathsf{T}'^{\delta}_{\mathsf{x}}(\mathsf{x}_{\delta-1}) &= \mathsf{x}_{\delta-2} \oplus z \\ \mathsf{T}'^{\delta}_{\mathsf{x}}(\mathsf{x}_{\alpha_1}) &= \mathsf{x}_{\alpha_1-1} \oplus \mathsf{f}(\mathsf{pk}, \mathsf{x}_{\alpha_1+1}) \\ \mathsf{T}'^{\delta}_{\mathsf{x}}(\mathsf{x}_{\alpha_2}) &= \mathsf{x}_{\alpha_2-1} \oplus \mathsf{f}(\mathsf{pk}, \mathsf{x}_{\alpha_2+1}).\end{aligned}$$

- **Phase 3:** ....
- ~~**Guess, Verify:**~~
- **Embed:** Output embedding $\mathsf{w}_i^{\mathsf{T}'}$, where $\mathsf{w}_i^{\mathsf{T}'} \in \mathbf{w}^{\mathsf{T}'}$, s.t. $\mathsf{f}(\mathsf{pk}, \mathsf{w}_i^{\mathsf{T}'}) = z$.

Since $\bar{\mathsf{x}}_\delta$ is uniformly distributed and not used before here, $\mathsf{f}(\mathsf{pk}, \bar{\mathsf{x}}_\delta)$ is uniform and independent, and indistinguishable from a uniformly random z. Note that since we no longer check for flag, it is no longer necessary to perform collision and prequery checks, the only place where x'_δ is used instead of $\mathsf{f}(\mathsf{pk}, \bar{\mathsf{x}}'_\delta)$. We can simulate the random oracles by picking uniformly random values on the appropriate domain whenever $\mathcal{A}$ makes a new query and maintaining a map of queries already made. Thus, if $\mathcal{A}_2$ won game 8, they must have queried

$$\bar{\mathsf{x}}_\delta = \mathsf{f}^{-1}(\mathsf{sk}, \mathsf{f}(\mathsf{pk}, \bar{\mathsf{x}}_\delta)) = \mathsf{f}^{-1}(\mathsf{sk}, z)$$

Which, from our definition of a secure trapdoor permutation, can only happen with negligible probability, □

To complete our proof of Theorem 2, from our sequence of games, we conclude that $\Pr[\mathsf{F}_0] = \mathsf{negl}(\kappa)$ as well, fulfilling our soundness definition.

7 Counterexample for Round Function Independent of Blocks

We show our construction is round optimal up to constant factors by constructing a secure trapdoor permutation scheme which is insecure for any number of rounds $\notin \Omega(b \cdot n)$ (i.e. $\in o(b \cdot n)$). Incidentally, this also shows that the construction provided by [7], which claims to only requires $O(n)$ rounds, is insecure against general adversaries.

We provide the intuition for our construction by considering the ideal VBB notion of obfuscation. The overall idea is to construct a trapdoor permutation family where we can amortize the 'state' space required to invert multiple independent instances. We will consider our permutations to be on domain $\{0,1\}^\lambda$. If we assume we have VBB obfuscation, then consider a program that takes in b many inputs $\{y_i\}_{i \in b}$ where $y_i \in \{0,1\}^\lambda$ and an advice string also in $\{0,1\}^\lambda$ and outputs the preimages of the messages $\{x_i = \mathsf{f}^{-1}(\mathsf{sk}, y_i)\}_{i \in b}$ iff the advice string that was input was equal to $\bigoplus_{i \in b} x_i$. The program has the secret key hardcorded and simply computes x_i and makes the check against the advice string and outputs $\{x_i\}_{i \in b}$ if the check succeeds. The VBB obfuscation of this program is then posted in the public parameters and provides a way for the adversary to compress $b \cdot \lambda$ bits to λ bits and still preserve information. Thus an adversary with outputting $r \cdot \lambda$ bits can recompute the replica from storing $o(b \cdot n\lambda)$ information. This would violate the security if we proved s soundness for the same parameters as our scheme. Below we formalize the notion by giving a construction from $i\mathcal{O}$ that captures this intuition formally and constructs a scheme in the end that breaks soundness security.

We assume the existence of a trapdoor permutation $(\mathsf{KeyGen}, \mathsf{f}(\cdot,\cdot), \mathsf{f}^{-1}(\cdot,\cdot))$ with domain $\{0,1\}^\lambda$ for $\lambda \in \omega(\kappa)$[5] , a puncturable PRF family $(\mathsf{PPRF.KeyGen_F}, \mathsf{PPRF.Eval_F}, \mathsf{PPRF.Puncture_F})$, indistinguishability obfuscation $i\mathcal{O}$ for all polynomial sized circuits.

7.1 Preliminaries

Definition 7. *We say a PPT algorithm $i\mathcal{O}(\kappa, C)$ is an indistinguishability obfuscator for a circuit class $\{\mathcal{C}_\kappa\}$ if the following hold*

Correctness - For all security parameters $\kappa \in \mathbb{N}$, for all $C \in \mathcal{C}_\kappa$, for all inputs x

$$\Pr[C'(x) = C(x) : C' \leftarrow i\mathcal{O}(\kappa, C)] = 1$$

Indistinguishability - For any PPT distinguisher $\mathcal{A}$, for all security parameters $\kappa \in \mathbb{N}$, for all pairs of circuits $C_0, C_1 \in \mathcal{C}_\kappa$ such that C_0 and C_1 are equivalent on all inputs x

$$|\Pr[\mathcal{A}(i\mathcal{O}(\kappa, C_0)) = 1] - \Pr[\mathcal{A}(i\mathcal{O}(\kappa, C_1)) = 1]| \leq \mathsf{negl}(\kappa)$$

Definition 8. *We say a set of algorithms $(\mathsf{PPRF.KeyGen_F}, \mathsf{PPRF.Eval_F}, \mathsf{PPRF.Puncture_F})$ puncturable pseudorandom function family (PPRF) on domain $\mathcal{D}_\kappa$ if it is a PRF and the following hold*

Function preserving - For every set polynomial (in κ) sized subset $S \subseteq \mathcal{D}_\kappa$, for all $x \in \mathcal{D}_\kappa \backslash S$

$$\Pr[\mathsf{PPRF.Eval_F}(K, x) = \mathsf{PPRF.Eval_F}(K_S, x) : K \leftarrow \mathsf{PPRF.KeyGen_F}(1^\kappa), K_S = \mathsf{PPRF.Puncture_F}(K, S)] = 1$$

Pseudorandom at puncture - For every PPT adversary (A_1, A_2) such that $A_1(1^\kappa)$ outputs a set $S \subseteq \mathcal{D}_\kappa$ and state σ, consider an experiment where $K \leftarrow \mathsf{KeyGen}(1^\kappa)$ and $K_S \leftarrow \mathsf{PPRF.Puncture_F}(K, S)$

$$\left|\Pr[A_2(\sigma, K_S, S, \mathsf{PPRF.Eval_F}(K, S)) = 1] - \Pr[A_2(\sigma, K_S, S, U_{\mathcal{D}_\kappa \cdot |S|}) = 1]\right|$$

where $\mathsf{PPRF.Eval_F}(K, S)$ denotes the concatenation of $\mathsf{PPRF.Eval_F}(K, x_1), \ldots, \mathsf{PPRF.Eval_F}(K, x_k)$ where $S = \{x_1, \ldots, x_k\}$ is the enumeration of the elements of S in lexicographic order.

For ease of notation, we will write $F(K, x)$ to represent $\mathsf{PPRF.Eval_F}(K, x)$ and will write $K(S)$ to represent punctured keys $\mathsf{PPRF.Puncture_F}(K, S)$

7.2 Construction

Let $(\mathsf{KeyGen}, \mathsf{f}(\cdot,\cdot), \mathsf{f}^{-1}(\cdot,\cdot))$ be a trapdoor permutation on $\{0,1\}^\kappa$, where KeyGen uses some $r(\kappa)$ bits of randomness. Let $(\mathsf{PPRF.KeyGen_F}, \mathsf{PPRF.Eval_F}, \mathsf{PPRF.Puncture_F})$ be a puncturable PRF on domain $\{0,1\}^\kappa$ and range $\{0,1\}^{r(\kappa)}$. We will construct a trapdoor permutation $(\mathsf{keygen}'_\mathsf{n}, \mathsf{f}'_\mathsf{n}(\cdot,\cdot), {\mathsf{f}'_\mathsf{n}}^{-1}(\cdot,\cdot))$ on domain $\{0,1\}^{2\kappa}$ parameterized by a quantity $n \in \mathsf{poly}(\kappa)$.

Program $\mathsf{f}(z, x)$

Inputs: Index $z \in \{0,1\}^\kappa$
Input $x \in \{0,1\}^\kappa$
Constants: Punctured PRF key K
Output: $y \in \{0,1\}^n$

1. Let $r \leftarrow \mathsf{PPRF.Eval_F}(K, z)$
2. Let $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}(1^\kappa; r)$
3. Output $\mathsf{f}(\mathsf{pk}', x)$

Figure 4: Routine Program f

[5]note it suffices to have some trapdoor permutation with domain $\lambda \in \omega(\kappa^\epsilon)$ for $\epsilon > 0$, and we can generically transform this by taking said TDP on security parameter $\kappa' = \kappa^{1/\epsilon}$

$\mathsf{Program}\ \mathsf{f}^{-1}((z_1, y_1), \ldots, (z_n, y_n) \in \{0,1\}^{2\kappa}, x \in \{0,1\}^{\kappa})$

Inputs: Images $(z_1, y_1), \ldots, (z_n, y_n) \in \{0,1\}^{2\kappa}$
Advice $x \in \{0,1\}^{\kappa}$
Constants: Punctured PRF key K
Output: Preimages $\{x_i \in \{0,1\}^{\kappa}\}_{i \in [n]}$

1. If $\exists i, j \in [n] : i \neq j \wedge z_i = z_j$ output $\perp$.
2. For $i \in [n]$
   (a) Let $r_i \leftarrow \mathsf{PPRF.Eval}_{\mathsf{F}}(K, z_i)$
   (b) Let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(1^{\kappa}; r_i)$
   (c) Let $x_i = \mathsf{f}^{-1}(\mathsf{sk}_i, y_i)$
3. If $\bigoplus_{i \in [n]} x_i \neq x$ output $\perp$.
4. If the above checks pass, output $\{x_i\}_{i \in [n]}$.

Figure 5: Routine $\mathsf{Program}\ \mathsf{f}^{-1}$

$\mathsf{keygen}'_{\mathsf{n}}(1^{\kappa})$

1. Sample $K \leftarrow \mathsf{PPRF.KeyGen}_{\mathsf{F}}(1^{\kappa})$
2. Let $\mathcal{O}\mathsf{Program}\ \mathsf{f} = i\mathcal{O}(\kappa, \mathsf{Program}\ \mathsf{f})$ and $\mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1} = i\mathcal{O}(\kappa, \mathsf{Program}\ \mathsf{f}^{-1})$.
3. Output $(\mathsf{pk} = (\mathcal{O}\mathsf{Program}\ \mathsf{f}, \mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1}), \mathsf{sk} = K)$

$\mathsf{f}'_{\mathsf{n}}(\mathsf{pk} = (\mathcal{O}\mathsf{Program}\ \mathsf{f}, \mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1}), (z, x))$

1. Let $y \leftarrow \mathcal{O}\mathsf{Program}\ \mathsf{f}(z, x)$ and output (z, y).

$\mathsf{f}'^{-1}_{\mathsf{n}}(\mathsf{sk} = K, (z, y))$

1. Let $r \leftarrow \mathsf{PPRF.Eval}_{\mathsf{F}}(K, z)$.
2. Let $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KeyGen}(1^{\kappa}; r)$.
3. Output $(z, \mathsf{f}^{-1}(\mathsf{sk}_0, y))$.

7.3 Proofs

Efficiency

Claim 33. $(\mathsf{keygen}'_{\mathsf{n}}, \mathsf{f}'_{\mathsf{n}}(\cdot,\cdot), \mathsf{f}'^{-1}_{\mathsf{n}}(\cdot,\cdot))$ *are polynomial time algorithms*

Proof. $\mathsf{keygen}'_{\mathsf{n}}$ simply calls $i\mathcal{O}$ twice. The programs $\mathsf{Program}\ \mathsf{f}$ and $\mathsf{Program}\ \mathsf{f}^{-1}$ simply call the underlying PRF and trapdoor primitives at most $n \in \mathsf{poly}(\kappa)$ times. By the efficiency of the underlying PRF and trapdoor permutation, $\mathsf{Program}\ \mathsf{f}$ and $\mathsf{Program}\ \mathsf{f}^{-1}$ are poly-sized circuits and $i\mathcal{O}$ runs in poly time, thus $\mathsf{keygen}'_{\mathsf{n}}$ runs in polynomial time.
f'_{n} simply evaluates a polynomial sized circuit, which is polynomial time.
$\mathsf{f}'^{-1}_{\mathsf{n}}$ does a single call to $\mathsf{PPRF.Eval}_{\mathsf{F}}, \mathsf{KeyGen}, \mathsf{f}^{-1}(\cdot,\cdot)$, which are all polynomial time algorithms by definition. □

Correctness

Claim 34. *$i\mathcal{O}$ is correct from definition (7) and $\mathsf{f}^{-1}(\mathsf{sk}', \cdot)$ computes inverse of $\mathsf{f}(\mathsf{pk}', \cdot)$ implies $\mathsf{f}'^{-1}_{\mathsf{n}}(\mathsf{sk}, \cdot)$ computes the inverse of $\mathsf{f}'_{\mathsf{n}}(\mathsf{pk}, \cdot)$, i.e.*

$$\forall \kappa, (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{keygen}'_{\mathsf{n}}(1^{\kappa}), \forall x' \in \{0,1\}^{2\kappa}, \mathsf{f}'^{-1}_{\mathsf{n}}(\mathsf{sk}, \mathsf{f}'_{\mathsf{n}}(\mathsf{pk}, x')) = x'.$$

Proof. Let $x' = (z, x) \in \{0,1\}^{2\kappa}$ be an arbitrary input to . Recall that f'_{n} simply runs $\mathcal{O}\mathsf{Program}\ \mathsf{f}$ on (z, x) and is same as the result of outputting $\mathsf{Program}\ \mathsf{f}$ on (z, x) from correctness of $i\mathcal{O}$. The output produced is $(z, \mathsf{f}(\mathsf{pk}', x))$ where $(\mathsf{pk}', \mathsf{sk}') = \mathsf{KeyGen}(1^\kappa, F(K, z))$. ${\mathsf{f}'_{\mathsf{n}}}^{-1}$ when run on $(z, \mathsf{f}(\mathsf{pk}', x))$ produces the same $(\mathsf{pk}', \mathsf{sk}')$ pair as $\mathcal{O}\mathsf{Program}\ \mathsf{f}$. Since,

$$\forall \kappa, (\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}(1^\kappa), \forall x \in \{0,1\}^\kappa, \mathsf{f}^{-1}(\mathsf{sk}', \mathsf{f}(\mathsf{pk}', x)) = x.$$

${\mathsf{f}'_{\mathsf{n}}}^{-1}$ returns $(z, \mathsf{f}^{-1}(\mathsf{sk}', \mathsf{f}(\mathsf{pk}', x))) = (z, x)$. □

Security

Theorem 3. *Assuming* $(\mathsf{KeyGen}, \mathsf{f}(\cdot,\cdot), \mathsf{f}^{-1}(\cdot,\cdot))$ *is a secure one way permutation from definition (1), indistinguishability of* $i\mathcal{O}$ *from definition (7) and a puncturable PRF family* $(\mathsf{PPRF.KeyGen_F}, \mathsf{PPRF.Eval_F}, \mathsf{PPRF.Puncture_F})$ *secure according to definition (8),* $(\mathsf{keygen}'_{\mathsf{n}}, \mathsf{f}'_{\mathsf{n}}, {\mathsf{f}'_{\mathsf{n}}}^{-1})$ *is a secure one way permutation - i.e., that for all PPT algorithms* $\mathcal{A}$

$$\Pr\left[\begin{array}{c} \mathsf{f}'_{\mathsf{n}}(\mathsf{pk}, (z_0, x_0)) = (z, y)\ s.t. \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{keygen}'_{\mathsf{n}}(1^\kappa), (z_0, x_0) \xleftarrow{R} \{0,1\}^{2\kappa}, (z, y) = \mathsf{f}'_{\mathsf{n}}(\mathsf{pk}, (z_0, x_0)), (z', x') \leftarrow \mathcal{A}(\mathsf{pk}, (z, y)) \end{array}\right] \leq \mathsf{negl}(\kappa),$$

over the random coins of $\mathsf{keygen}'_{\mathsf{n}}$ *and sampling of* (z_0, x_0).

We will show this via a sequence of games, where the view of the adversary between successive games is indistinguishable.

Game 0

This is the original security game.

1. Challenger samples a random $(z_0, x_0) \xleftarrow{R} \{0,1\}^{2\kappa}$
   (a) Sample $K \leftarrow \mathsf{PPRF.KeyGen_F}(1^\kappa)$
   (b) Let $\mathcal{O}\mathsf{Program}\ \mathsf{f} = i\mathcal{O}(\kappa, \mathsf{Program}\ \mathsf{f})$ and $\mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1} = i\mathcal{O}(\kappa, \mathsf{Program}\ \mathsf{f}^{-1})$. [6]
   (c) Output $(\mathsf{pk} = (\mathcal{O}\mathsf{Program}\ \mathsf{f}, \mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1}), \mathsf{sk} = K)$
2. Challenger runs $\mathsf{f}'_{\mathsf{n}}(\mathsf{pk}, (z_0, x_0))$
   (a) Let $y_0 \leftarrow \mathcal{O}\mathsf{Program}\ \mathsf{f}(z_0, x_0)$.
3. Adversary is given $(\mathsf{pk}, (z_0, y_0))$.
4. Adversary outputs (z', x')
5. If $\mathsf{f}'_{\mathsf{n}}(\mathsf{pk}, (z', x')) = (z_0, y_0)$ then output 1 else output 0

Game 1

In this game, we change the way $\mathsf{Program}\ \mathsf{f}$ is programmed.

1. Challenger samples a random $(z_0, x_0) \xleftarrow{R} \{0,1\}^{2\kappa}$
   (a) Sample $K \leftarrow \mathsf{PPRF.KeyGen_F}(1^\kappa)$
   (b) Compute $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KeyGen}(1^\kappa, F(K, z_0))$
   (c) Compute punctured key $K(\{z_0\})$

[6] The security parameter in the input to $i\mathcal{O}$ algorithm is the smallest λ for which $\mathsf{Program}\ \mathsf{f}, \mathsf{Program}\ \mathsf{f}^*$ are in $\mathcal{C}_\lambda$ which will be polynomial in κ as the circuits are polynomial. We denote this by κ here for notation clarity.

Program $\mathsf{f}^*(z, x)$

Inputs: Index $z \in \{0,1\}^\kappa$
Input $x \in \{0,1\}^\kappa$
Constants: Punctured PRF key $K(\{z_0\})$
Public Key pk_0
Output: $y \in \{0,1\}^n$

1. If $z = z_0$, output $\mathsf{f}(\mathsf{pk}_0, x)$.
2. Let $r \leftarrow \mathsf{PPRF.Eval}_\mathsf{F}(K(\{z_0\}), z)$
3. Let $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}(1^\kappa; r)$
4. Output $\mathsf{f}(\mathsf{pk}', x)$

Figure 6: Routine Program f^*

   (d) Let $\mathcal{O}\mathsf{Program}\ \mathsf{f}^* = i\mathcal{O}(\kappa, \mathsf{Program}\ \mathsf{f}^*)$ and $\mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1} = i\mathcal{O}(\kappa, \mathsf{Program}\ \mathsf{f}^{-1})$.

   (e) Output $(\mathsf{pk} = (\mathcal{O}\mathsf{Program}\ \mathsf{f}^*, \mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1}), \mathsf{sk} = K)$

2. Challenger runs $\mathsf{f}'_\mathsf{n}(\mathsf{pk}, (z_0, x_0))$

   (a) Let $y_0 \leftarrow \mathcal{O}\mathsf{Program}\ \mathsf{f}^*(z_0, x_0)$.

3. Adversary is given $(\mathsf{pk}, (z_0, y_0))$.

4. Adversary outputs (z', x')

5. If $\mathsf{f}'_\mathsf{n}(\mathsf{pk}, (z', x')) = (z_0, y_0)$ then output 1 else output 0

Game 2

In this game, we change the way Program f^{-1} is programmed.

1. Challenger samples a random $(z_0, x_0) \xleftarrow{R} \{0,1\}^{2\kappa}$

   (a) Sample $K \leftarrow \mathsf{PPRF.KeyGen}_\mathsf{F}(1^\kappa)$

   (b) Compute $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KeyGen}(1^\kappa, F(K, z_0))$

   (c) Compute punctured key $K(\{z_0\})$

   (d) Let $\mathcal{O}\mathsf{Program}\ \mathsf{f}^* = i\mathcal{O}(\kappa, \mathsf{Program}\ \mathsf{f}^*)$ and $\mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1^*} = i\mathcal{O}(\kappa, \mathsf{Program}\ \mathsf{f}^{-1^*})$.

   (e) Output $(\mathsf{pk} = (\mathcal{O}\mathsf{Program}\ \mathsf{f}^*, \mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1^*}), \mathsf{sk} = K)$

2. Challenger runs $\mathsf{f}'_\mathsf{n}(\mathsf{pk}, (z_0, x_0))$

   (a) Let $y_0 \leftarrow \mathcal{O}\mathsf{Program}\ \mathsf{f}^*(z_0, x_0)$.

3. Adversary is given $(\mathsf{pk}, (z_0, y_0))$.

4. Adversary outputs (z', x')

5. If $\mathsf{f}'_\mathsf{n}(\mathsf{pk}, (z', x')) = (z_0, y_0)$ then output 1 else output 0

$\mathsf{Program}\ \mathsf{f}^{-1^*}((z_1, y_1), \ldots, (z_n, y_n) \in \{0,1\}^{2\kappa}, x \in \{0,1\}^\kappa)$

Inputs: Images $(z_1, y_1), \ldots, (z_n, y_n) \in \{0,1\}^{2\kappa}$
Advice $x \in \{0,1\}^\kappa$
Constants: Punctured PRF key $K(\{z_0\})$
Public Key pk_0
Output: Preimages $\{x_i \in \{0,1\}^\kappa\}_{i\in[n]}$

1. If $\exists i, j \in [n] : i \neq j \wedge z_i = z_j$ output $\perp$.
2. For $i \in [n]$ when $z_i \neq z_0$
   (a) Let $r_i \leftarrow \mathsf{PPRF.Eval}_\mathsf{F}(K(\{z_0\}), z_i)$
   (b) Let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(1^\kappa; r_i)$
   (c) Let $x_i = \mathsf{f}^{-1}(\mathsf{sk}_i, y_i)$
3. If $\exists i' : z_{i'} = z_0$
   (a) Let $x_{i'} = x \oplus (\bigoplus_{i\in[n]\setminus\{i'\}} x_i)$.
   (b) If $\mathsf{f}(\mathsf{pk}_0, x_{i'}) \neq y_i$, output $\perp$.
4. If $\bigoplus_{i\in[n]} x_i \neq x$ output $\perp$.
5. If the above checks pass, output $\{x_i\}_{i\in[n]}$.

Figure 7: Routine $\mathsf{Program}\ \mathsf{f}^{-1^*}$

Game 3

In this game, we compute $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KeyGen}(1^\kappa, r_0)$ using true randomness r_0.

1. Challenger samples a random $(z_0, x_0) \xleftarrow{R} \{0,1\}^{2\kappa}$
   (a) Sample $K \leftarrow \mathsf{PPRF.KeyGen}_\mathsf{F}(1^\kappa)$
   (b) Sample $r_0 \xleftarrow{R} \{0,1\}^{r(\kappa)}$
   (c) Compute $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KeyGen}(1^\kappa, r_0)$
   (d) Compute punctured key $K(\{z_0\})$
   (e) Let $\mathcal{O}\mathsf{Program}\ \mathsf{f}^* = i\mathcal{O}(\kappa, \mathsf{Program}\ \mathsf{f}^*)$ and $\mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1^*} = i\mathcal{O}(\kappa, \mathsf{Program}\ \mathsf{f}^{-1^*})$.
   (f) Output $(\mathsf{pk} = (\mathcal{O}\mathsf{Program}\ \mathsf{f}^*, \mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1^*}), \mathsf{sk} = K)$
2. Challenger runs $\mathsf{f}'_\mathsf{n}(\mathsf{pk}, (z_0, x_0))$
   (a) Let $y_0 \leftarrow \mathcal{O}\mathsf{Program}\ \mathsf{f}^*(z_0, x_0)$.
3. Adversary is given $(\mathsf{pk}, (z_0, y_0))$.
4. Adversary outputs (z', x')
5. If $\mathsf{f}'_\mathsf{n}(\mathsf{pk}, (z', x')) = (z_0, y_0)$ then output 1 else output 0

Analysis

Let $\mathsf{adv}^i_\mathcal{A}$ be a function parameterized by κ and denote the probability of an adversary $\mathcal{A}$ winning Game i, i.e. the output of Game i is 1.

Lemma 16. *Assuming indistinguishability for PPT algorithm $i\mathcal{O}$ according to definition 7, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\forall\kappa \in \mathbb{N}, |\mathsf{adv}^0_\mathcal{A} - \mathsf{adv}^1_\mathcal{A}| = \mathsf{negl}(\kappa)$.*

Proof. The only difference between the two games is that obfuscation of $\mathsf{Program\ f}$ is changed to obfuscation of $\mathsf{Program\ f}^*$ respectively.

Claim 35. $\mathsf{Program\ f}, \mathsf{Program\ f}^*$ *functionally equivalent, i.e.*

$$\forall \kappa \in \mathbb{N}, (z,x) \in \{0,1\}^{2\kappa}, \mathsf{Program\ f}(z,x) = \mathsf{Program\ f}^*(z,x).$$

Proof. We separate inputs (z,x) into 2 cases

- Case 1 - $z = z_0$
  Here, we make an additional conditional check in bullet 1 in figure 6 and use pk_0 hardcoded to return the coputatation $\mathsf{f}(\mathsf{pk}_0, x)$ where pk_0 is computed from running keygen on randomness obtained by evaluating punctured prf on key K and input z_0. In figure 4 this is exactly how the computation proceeds.
- Case 2 - $z \neq z_0$
  Here, the conditional check inserted in bullet 1 in figure 6 does not hold, so by functionality preserving under puncturing, $\mathsf{PPRF.Eval_F}(K,z) = \mathsf{PPRF.Eval_F}(K(\{z_0\}), z)$, and the executions are identical.

□

Let $\mathcal{A}$ be an adversary that distinguishes between Game 0 and Game 1 with probability $\epsilon(\kappa)$ then we can construct an adversary $\mathcal{B}$ that is a distinguisher between $\mathcal{O}\mathsf{Program\ f}, \mathcal{O}\mathsf{Program\ f}^*$ with probability $\epsilon(\kappa)$. $\mathcal{B}$ on input (κ, P) interacts with $\mathcal{A}$ according to Game 0 but outputs $(\mathsf{pk} = (P, \mathcal{O}\mathsf{Program\ f}^{-1}), \mathsf{sk} = K)$. Clearly $\mathcal{B}$ on input $(\kappa, \mathcal{O}\mathsf{Program\ f})$ simulates Game 0 and on input $(\kappa, \mathcal{O}\mathsf{Program\ f}^*)$ simulates Game 1. By security of $i\mathcal{O}$ since $\mathcal{O}\mathsf{Program\ f}, \mathcal{O}\mathsf{Program\ f}^*$ are functionally equivalent from claim 35, ϵ is a negligible function.

□

Lemma 17. *Assuming indistinguishability for PPT algorithm $i\mathcal{O}$ according to definition 7, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that* $\forall \kappa \in \mathbb{N}, |\mathsf{adv}^1_{\mathcal{A}} - \mathsf{adv}^2_{\mathcal{A}}| = \mathsf{negl}(\kappa)$.

Proof. The only difference between the two games is that obfuscation of $\mathsf{Program\ f}^{-1}$ is changed to obfuscation of $\mathsf{Program\ f}^{-1^*}$ respectively.

Claim 36. $\mathsf{Program\ f}^{-1}, \mathsf{Program\ f}^{-1^*}$ *are functionally equivalent, i.e.*

$$\begin{bmatrix} \forall \kappa \in \mathbb{N}, (z_1,y_1), \ldots, (z_n,y_n) \in \{0,1\}^{2\kappa}, x \in \{0,1\}^{\kappa} \\ \mathsf{Program\ f}^{-1}((z_1,y_1), \ldots, (z_n,y_n), x) = \mathsf{Program\ f}^{-1^*}((z_1,y_1), \ldots, (z_n,y_n), x) \end{bmatrix}.$$

Proof. We first note that if there doesn't exist $i' \in [n], z_{i'} = z_0$, then the two programs are exactly identical except that the key is punctured in one of them. We observe by functionality preserving under puncturing, $\mathsf{PPRF.Eval_F}(K, z_i) = \mathsf{PPRF.Eval_F}(K(\{z_0\}), z_i)$, so $\{\mathsf{pk}_i\}_{i\in[n]}$ are identical in both programs when $z_i \neq z_0$.

In the case that there exists $i' \in [n], z_{i'} = z_0$, as observed above $\{\mathsf{pk}_i\}_{i\in[n]\setminus\{i'\}}$ are identical. Also as seen in the proof of claim 35, the hardcoded pk_0 is same as what was obtained by evaluating prf on key K. Let x^0_i, x^1_i denote the value of x_i computed in $\mathsf{Program\ f}^{-1}, \mathsf{Program\ f}^{-1^*}$ respectively. Note that we have shown above that $\forall i \in [n] \setminus \{i'\}\ x^0_i = x^1_i$.

- Case 1: If advice given is correct i.e. $x = \bigoplus_{i\in[n]}\{x^0_i\} \Rightarrow x^0_{i'} = x^1_{i'}$. This means check 3(b) in figure 7 passes as $y_{i'} = \mathsf{f}(\mathsf{pk}_0, x^0_{i'}) = \mathsf{f}(\mathsf{pk}_0, x^1_{i'})$ and both programme output the preimages $\{x^0_i\}_{i\in[n]} = \{x^1_i\}_{i\in[n]}$.
- Case 2: If advice given is incorrect i.e. $x \neq \bigoplus_{i\in[n]}\{x^0_i\}$ then $\mathsf{Program\ f}^{-1}$ clearly outputs $\perp$. Since $x^1_{i'} = x \oplus (\bigoplus_{i\in[n]\setminus\{i'\}} x^1_i) = x \oplus (\bigoplus_{i\in[n]\setminus\{i'\}} x^0_i)$. This implies $x^1_{i'} \neq x^0_{i'}$. Since $y_{i'} = \mathsf{f}(\mathsf{pk}_0, x^0_{i'}) \neq \mathsf{f}(\mathsf{pk}_0, x^1_{i'})$ as $\mathsf{f}(\cdot,\cdot)$ is a trapdoor permutation. $\mathsf{Program\ f}^{-1^*}$ will also output bot at check 3(b) in figure 7.

□

Let $\mathcal{A}$ be an adversary that distinguishes between Game 1 and Game 2 with probability $\epsilon(\kappa)$ then we can construct an adversary $\mathcal{B}$ that is a distinguisher between $\mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1}, \mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1^*}$ with probability $\epsilon(\kappa)$. $\mathcal{B}$ on input (κ, P) interacts with $\mathcal{A}$ according to Game 1 but outputs $(\mathsf{pk} = (\mathcal{O}\mathsf{Program}\ \mathsf{f}^*, P), \mathsf{sk} = K)$. Clearly $\mathcal{B}$ on input $(\kappa, \mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1})$ simulates Game 1 and on input $(\kappa, \mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1^*})$ simulates Game 2. By security of $i\mathcal{O}$ since $\mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1}, \mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1^*}$ are functionally equivalent from claim 36, ϵ is a negligible function.

□

Lemma 18. *Assuming pseudorandomness at puncturing property for a puncturable PRF family* $(\mathsf{PPRF.KeyGen_F}, \mathsf{PPRF.Eval_F}, \mathsf{PPRF.Puncture_F})$ *according to definition 8, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that* $\forall \kappa \in \mathbb{N}, |\mathsf{adv}^2_{\mathcal{A}} - \mathsf{adv}^3_{\mathcal{A}}| = \mathsf{negl}(\kappa)$.

Proof. We now argue that the advantage for any poly-time attacker must be negligibly close in Games 2 and 3. Otherwise, we can create a reduction algorithm $\mathcal{B}$ that breaks the selective security of the constrained pseudorandom function at the punctured points. $\mathcal{B}$ executes Game 2, except that it gets gets the punctured PRF key $K(\{z_0\})$ and challenge c. It continues to run as in Game 2 except that it generates $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KeyGen}(1^\kappa, c)$. If c is the output of the PRF at point z_0, then we are in Game 2. If it was chosen uniformly at random, we are in Game 3. Thus if $\mathcal{A}$ was an adversary with advantage $\epsilon(\kappa)$ in distinguishing between Game 2 and Game 3 then $\mathcal{B}$ is an adversary which distinguishes the pseudorandomness at puncturing property with the same advantage. Thus $\epsilon(\kappa)$ is $\mathsf{negl}(\kappa)$. Note, that we were able to reduce to selective security since z_0 is defined to be chosen at random by the challenger and outside the attacker's control and was chosen before the punctured PRF key was received. Below we have described the reduction explicitly.

$\mathcal{B}_1(1^\kappa)$

1. Sample a random z_0.
2. Output $S = \{z_0\}, \sigma = \epsilon$ (empty string)

$\mathcal{B}_2(\sigma, K(\{z_0\}), \{z_0\}, c)$

1. Challenger samples a random $x_0 \xleftarrow{R} \{0,1\}^\kappa$
   (a) Compute $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KeyGen}(1^\kappa, c)$
   (b) Let $\mathcal{O}\mathsf{Program}\ \mathsf{f}^* = i\mathcal{O}(\kappa, \mathsf{Program}\ \mathsf{f}^*)$ and $\underline{\mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1^*} = i\mathcal{O}(\kappa, \mathsf{Program}\ \mathsf{f}^{-1^*})}$.
   (c) Output $\mathsf{pk} = (\mathcal{O}\mathsf{Program}\ \mathsf{f}^*, \mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1^*})$
2. Challenger runs $\mathsf{f}'_{\mathsf{n}}(\mathsf{pk}, (z_0, x_0))$
   (a) Let $y_0 \leftarrow \mathcal{O}\mathsf{Program}\ \mathsf{f}^*(z_0, x_0)$.
3. Let $x'_0 = \mathcal{A}(\mathsf{pk}, (z_0, y_0))$.
4. If $x'_0 = x_0$ output 0, else output 1.

□

Lemma 19. *Assuming* $(\mathsf{KeyGen}, \mathsf{f}(\cdot,\cdot), \mathsf{f}^{-1}(\cdot,\cdot))$ *is a secure trapdoor function from definition 1, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that* $\forall \kappa \in \mathbb{N}, \mathsf{adv}^3_{\mathcal{A}} = \mathsf{negl}(\kappa)$.

Proof. Let $\mathcal{A}$ be an adversary that interacts with challenger in Game 3 with winning probability $\epsilon(\kappa)$. We describe a reduction to the security of a trapdoor permutation $(\mathsf{KeyGen}, \mathsf{f}(\cdot,\cdot), \mathsf{f}^{-1}(\cdot,\cdot))$.

Inputs: $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{KeyGen}(1^\kappa)$, $y = \mathsf{f}(\mathsf{pk}', x_0)$ where $x_0 \xleftarrow{R} \{0,1\}^\kappa$.
Reduction $\underline{\mathcal{B}(\mathsf{pk}', y)}$:

1. Challenger samples a random $(z_0) \xleftarrow{R} \{0,1\}^\kappa$
   (a) Sample $K \leftarrow \mathsf{PPRF.KeyGen_F}(1^\kappa)$
   (b) Set $\mathsf{pk}_0 = \mathsf{pk}'$
   (c) Compute punctured key $K(\{z_0\})$
   (d) Let $\mathcal{O}\mathsf{Program}\ \mathsf{f}^* = i\mathcal{O}(\kappa, \mathsf{Program}\ \mathsf{f}^*)$ and $\mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1^*} = i\mathcal{O}(\kappa, \mathsf{Program}\ \mathsf{f}^{-1^*})$.
   (e) Output $(\mathsf{pk} = (\mathcal{O}\mathsf{Program}\ \mathsf{f}^*, \mathcal{O}\mathsf{Program}\ \mathsf{f}^{-1^*}), \mathsf{sk} = K)$
2. Adversary is given $(\mathsf{pk}, (z_0, y))$.
3. Adversary outputs (z', x')
4. Output x'

$\mathcal{A}$ wins with probability ϵ which implies $\mathsf{f}'_\mathsf{n}(\mathsf{pk}, (z', x')) = (z_0, y)$. From the construction, it can be seen that this is correct if $\mathsf{f}(\mathsf{pk}_0, x') = y$. Thus $\mathcal{B}$ wins with probability $\geq \epsilon$. Thus $\epsilon = \mathsf{negl}(\kappa)$ from security of the trapdoor function. □

From the above lemmas, it is clear that $\mathsf{adv}^0_\mathcal{A} = \mathsf{negl}(\kappa)$ and hence proving stated theorem 3.

7.4 Attack on Replica Encoding Scheme

First, we restate the security game in the context of the above TDP. We consider a variation of our construction in Section 5 with $r \in o(b \cdot n)$ instantiated with the above trapdoor permutation and present a concrete attack adversary which breaks the s-soundness of our replica encoding scheme for any constant $\mathsf{s} \in (0,1)$. We remark that this attack also applies to the construction in Section 6.

Game 0: $\mathsf{Sound}_{\mathcal{A}_1,\mathcal{A}_2}(\kappa, n)$

- **Setup:** The challenger(denoted by $\mathcal{C}$) runs $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{rSetup}(1^\kappa, 1^n)$ and sends public key $\mathsf{pk}' = (C'_0, C'_1, n)$ to $\mathcal{A}_1$. It keeps the secret key $\mathsf{sk}' = (K, n)$ for itself.
- **Phase 1:** The adversary $\mathcal{A}_1$ issues queries on $\mathsf{T}, \mathsf{T}^{-1}, \mathsf{H}$, $\mathcal{C}$ responds the query back to $\mathcal{A}_1$.
- **File Challenge:** $m \in \{0,1\}^* \leftarrow \mathcal{A}_1^{\mathsf{H}(\cdot),\mathsf{T}(\cdot),\mathsf{T}^{-1}(\cdot)}(1^\kappa, \mathsf{pk}')$. It sends m to $\mathcal{C}$ who parses pk' as (C'_0, C'_1, n); sk' as (K, n) and does the following:
  - Divide m into b blocks of length λ i.e. $m = m_1||m_2||\ldots||m_b$, $b = \lceil |m|/\lambda \rceil$.
  - For $i \in [n]$,
    * Choose a string $\rho_i \xleftarrow{R} \{0,1\}^\kappa$.
    * Compute $\forall t \in [b]$,
    $$Y^{(i)}_{t,0} = m_t \oplus \mathsf{H}(\rho_i||t).$$
    * For rounds j from 1 to r and $\forall t \in [b]$,
      · Let $z^{(i)}_{t,j}, x^{(i)}_{t,j} \in \{0,1\}^{\lambda/2}$
      · Let $z^{(i)}_{t,j}||x^{(i)}_{t,j} = \mathsf{T}(Y^{(i)}_{t,j})$
      · Compute $Y^{(i)}_{t,j}$ from $Y^{(i)}_{t,j}$ as
      $$(\mathsf{pk}^{(i)}_{t,j}, \mathsf{sk}^{(i)}_{t,j}) = \mathsf{KeyGen}(1^\kappa; \mathsf{PPRF.Eval_F}(K, z^{(i)}_{t,j}).$$
      $$Y^{(i)}_{t,j} = z^{(i)}_{t,j}||\mathsf{f}^{-1}(\mathsf{sk}^{(i)}_{t,j}, x^{(i)}_{t,j})$$
    * Let $y^{(i)}_r = Y^{(i)}_{1,r}||\ldots||Y^{(i)}_{b,r}$ and set $y^{(i)} = (y^{(i)}_r, \rho_i)$.

  $\mathcal{C}$ returns $y^{(1)}, y^{(2)}, \ldots y^{(n)}$ to $\mathcal{A}_1$.
- **Phase 2:** $\mathcal{A}_1$ issues additional queries on $\mathsf{T}, \mathsf{T}^{-1}, \mathsf{H}$, $\mathcal{C}$ responds the query back to $\mathcal{A}_1$.
- **State Sharing:** $\mathcal{A}_1$ outputs state $\mathsf{state} \leftarrow \mathcal{A}_1^{\mathsf{H}(\cdot),\mathsf{T}(\cdot),\mathsf{T}^{-1}(\cdot)}(1^\kappa, \mathsf{pk}', y)$ and sends state to $\mathcal{A}_2$.

- **Phase 3:** The adversary $\mathcal{A}_2$ queries on $\mathsf{T}, \mathsf{T}^{-1}, \mathsf{H}$, $\mathcal{C}$ responds the query back to $\mathcal{A}_2$.
- **Guess:** $\mathcal{A}_2$ outputs the replica guesses to $\mathcal{C}$.

$$\{\tilde{y}^{(i)}\} \leftarrow \mathcal{A}_2(1^\kappa, \mathsf{pk}', m, \mathsf{state}).$$

- **Verify:** Let $v_i = 1$ if $\tilde{y}^{(i)} = y^{(i)}$ and 0 otherwise. Adversary wins if $|\mathsf{state}| < \sum v_i \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)$.

Now below, we present out construction of adversaries $\mathcal{A}_1, \mathcal{A}_2$.

$\mathcal{A}_1(1^\kappa, \mathsf{pk}' = (\mathsf{pk}, n))$

- Choose any message $m \in \{0,1\}^{b \cdot \lambda}$ where $b \geq 1$.
- Send m to challenger.
- Receive $\{y^{(i)} = \{Y_{t,r}^{(i)}\}_{t \in [b]}, \rho_i\}_{i \in [n]}$.
- For each $j \in [r]$, set $x_j = \bigoplus_{t \in [b], i \in [n]} Y_{t,j}^{(i)}$.
- Send $\{x_j\}, \{\rho_i\}$ as state.

$\mathcal{A}_2(1^\kappa, \mathsf{pk}' = (C_0', C_1', n), m, (\{x_j\}, \rho_i))$

- Divide m into b blocks of length λ, $m = m_1 || \ldots || m_b$
- For $i \in [r], t \in [b]$
  - Compute $Y_{t,0}^{(i)} = H(\rho_i || t) \oplus m_t$
- For $j \in [r]$
  - Set $\{Y_{t,j}^{(i)}\}_{i \in [r], t \in [b]} = C_1'(\{\mathsf{T}(Y_{t,j-1}^{(i)})\}, x_j)$
- For $i \in [r]$
  - Let $y_r^{(i)} = Y_{1,r}^{(i)} || \ldots || Y_{b,r}^{(i)}$ and output $(y_r^{(i)}, \rho_i)$

Lemma 20. *$(\mathcal{A}_1, \mathcal{A}_2)$ use $\lambda \cdot o(b \cdot n) + n \cdot o(\lambda)$ space.*

Proof. We observe that the state output is $\{x_j\}, \{\rho_i\}$, which use $r \cdot \lambda$, and $\kappa \cdot n$ space respectively. We use the fact that $r \in o(b \cdot n)$ and $\lambda \in \omega(\kappa)$ to give us our result. □

Lemma 21. *$\forall n \geq 1$, there exists a negligible function negl such that the probability that $\sum_i v_i = n$ in the verification stage of $\mathsf{Sound}_{\mathcal{A}_1, \mathcal{A}_2}(\kappa, n)$ for adversaries $(\mathcal{A}_1, \mathcal{A}_2)$ is $1 - \mathsf{negl}(\kappa)$.*

Proof. We recall that C_1' is simply an obfuscation of program $\mathsf{Program}\ \mathsf{f}^{-1}$. Thus, as long as the collection $\{z_{t,j}^{(i)}\}_{t \in [b], i \in [n]}$ is unique for every $j \in [r]$ and x_j is the $\oplus$ of $\{x_{t,j}^{(i)}\}_{t \in [b], i \in [n]}$, then C_1' will successfully invert. By the fact that H is a random oracle, and that T and f are permutations, we use the fact that a uniform random variable under a permutation is uniformly random to get that $\{z_{t,j}^{(i)}\}_{t \in [b], i \in [n]}$ is a uniform and independently random set. Thus, we can union bound the probability that any of them collide with $\binom{b \cdot n}{2} \cdot 2^{-\lambda/2} \in \mathsf{negl}(\kappa)$. In addition, we know that x_j is the aforementioned value by construction. Thus, we can inductively reason that our adversary computes $\{Y_{t,j}^{(i)}\}_{i \in [r], t \in [b]}$ correctly, and thus can recover the original encodings. □

Lemma 22. *When instantiated with a round function $r \in o(b \cdot n)$, the construction in Section 5 is not s-sound for any constant functions $\mathsf{s}(\kappa, |m|) = c \in (0,1)$.*

Proof. Recall the definition of s-soundness as

$$\Pr\left[\begin{array}{c}(v, \mathsf{state}, m) \leftarrow \mathsf{Sound}_{\mathcal{A}_1, \mathcal{A}_2}(\kappa, n), s.t. \\ |\mathsf{state}| < v \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)\end{array}\right] \leq \mathsf{negl}(\kappa).$$

We know by Lemma 21 that $v = n$ with all but negligible probability, and from Lemma 20 that $|\mathsf{state}| \in \lambda \cdot o(b \cdot n) + n \cdot o(\lambda)$. We recall that $\mathsf{len}(\kappa, |m|) = |m| + O(\kappa) > \lambda \cdot b$. From this, we can conclude that for sufficiently large $\kappa, |m|$, we know

$$\lambda \cdot o(b \cdot n) + n \cdot o(\lambda) < n \cdot c \cdot \lambda \cdot b < v \cdot \mathsf{s}(\kappa, |m|) \cdot \mathsf{len}(\kappa, |m|)$$

with all but negligible probability, and so this scheme is not s-sound. □

References

[1] J Benet and N Greco. Filecoin: A decentralized storage network. *Protoc. Labs*, 2018.

[2] Juan Benet, David Dalrymple, and Nicola Greco. Proof of replication. *Protoc. Labs*, 2017.

[3] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788. Springer, 2018.

[4] Kevin D Bowers, Ari Juels, and Alina Oprea. Proofs of retrievability: Theory and implementation. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 43–54, 2009.

[5] Ethan Cecchetti, Ben Fisch, Ian Miers, and Ari Juels. Pies: Public incompressible encodings for decentralized storage. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1351–1367, 2019.

[6] Yuanxi Dai and John Steinberger. Indifferentiability of 8-round feistel networks. In *Annual International Cryptology Conference*, pages 95–120. Springer, 2016.

[7] Ivan Damgård, Chaya Ganesh, and Claudio Orlandi. Proofs of replicated storage without timing assumptions. In *Advances in Cryptology – CRYPTO 2019*, pages 355–380, Cham, 2019. Springer International Publishing.

[8] Yevgeniy Dodis, Salil P. Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In Omer Reingold, editor, *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, volume 5444 of *Lecture Notes in Computer Science*, pages 109–127. Springer, 2009.

[9] Ben Fisch. Tight proofs of space and replication. Cryptology ePrint Archive, Report 2018/702, 2018. https://eprint.iacr.org/2018/702.

[10] Ben Fisch, Joseph Bonneau, Juan Benet, and Nicola Greco. Proofs of replication using depth robust graphs. *Blockchain Protocol Analysis and Security Engineering*, 2018, 2018.

[11] Thomas Holenstein, Robin Künzler, and Stefano Tessaro. The equivalence of the random oracle model and the ideal cipher model, revisited. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 89–98. ACM, 2011.

[12] Ari Juels and Burton S Kaliski Jr. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 584–597, 2007.

[13] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2004.

[14] Tal Moran and Daniel Wichs. Incompressible encodings. Manuscript, 2020.

[15] Kazuo Ohta and Tatsuaki Okamoto. On concrete security treatment of signatures derived from identification. In *Annual International Cryptology Conference*, pages 354–369. Springer, 1998.

[16] Krzysztof Pietrzak. Proofs of catalytic space. Cryptology ePrint Archive, Report 2018/194, 2018. https://eprint.iacr.org/2018/194.

[17] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 487–506. Springer, 2011.

[18] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 90–107. Springer, 2008.

[19] Marten Van Dijk, Ari Juels, Alina Oprea, Ronald L Rivest, Emil Stefanov, and Nikos Triandopoulos. Hourglass schemes: how to prove that cloud files are encrypted. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 265–280, 2012.