Adaptive Multiparty Non-interactive Key Exchange Without Setup In The Standard Model

Vanishree Rao*

Abstract

Non-interactive key exchange (NIKE) is a fundamental notion in Cryptography. This notion was introduced by Diffie and Hellman in 1976. They proposed the celebrated 2-party NIKE protocol and left open as a fascinating question, whether NIKE could be realized in the multiparty setting. NIKE has since then been an active area of research with an ultimate goal of obtaining best possible security in the multiparty setting. Although this has evaded researchers for many decades, advancements have been made through relaxations in multiple directions such as restricting to 3-parties, static/semi-static model (where the adversary needs to commit to the set of parties he wishes to be challenged upon *ahead of time*), random-oracle model, allowing initial setup, etc.

In this work, we settle the longstanding open question: we present the first multiparty NIKE protocol that is adaptively secure with no setup and in the standard model.

Our construction is based on indistinguishability obfuscation and *obliviously-patchable puncturable pseudorandom functions*, a new notion that we introduce.

We employ novel techniques of using indistinguishability obfuscation, which are interesting in their own right and which we believe would find wider applications in other settings. One such technique pertains overcoming, the somewhat inherent, drawback of non-adaptivity of the puncturing technique introduced by Sahai and Waters [STOC'14]. Central to this technique is our new notion of obliviously-patchable puncturable pseudorandom functions. We present a concrete construction of these pseudorandom functions using multilinear maps and their recent approximations – the leveled-graded encoding schemes.

Note that pseudorandom functions amount to an interactive assumption. We shall establish via a meta-reduction technique that, in natural settings, an interactive assumption is necessary (even with setup).

*University of California, Los Angeles. vanishri@cs.ucla.edu.

1 Introduction

In a seminal work, Diffie and Hellman introduced the fundamental notion of non-interactive key exchange [DH76]. This notion enables any set of parties to come together and just by knowing one another's public keys be able to derive a common secret key *without requiring any interaction*. Although this notion sounds infeasible on the face value, in the same work, Diffie and Hellman presented a NIKE protocol for two parties; the question of generalizing it to a multiparty setting was left as an important open problem.

Before we proceed, let us review at a high level the utility and security of multiparty NIKE. We consider a setting of N parties, who to begin with, do not share any common information. Each party simply publishes a public value.

> *Utility:* Consider any subset of the parties who wish to derive a common secret key. It is required that every party in the subset derives the *same* common key as every other party in the subset.
>
> *Security:* For security, we require that, in the view of every party who is not a part of a subset, the common key for the subset is indistinguishable from a uniform random string. This requirement is formalized by the following security game between an adversary and a challenger.
>
> - The game is structured as the adversary making multiple kinds of queries to the challenger. We detail the different kinds of queries in the ensuing.
> - Query the challenger to register an honest party – upon this query, the challenger outputs a public value. The adversary can also register a (malicious) party – by publishing a public value by himself.
> - Query the challenger to reveal the secret value of some honest party (the party is then deemed 'corrupted').
> - Query the challenger to reveal the common key for some subset as derived by some honest party.
> - At some point along the way, present a challenge query, which is a subset of honest parties, upon which the challenger responds via either the common key for this subset or a random string, corresponding respectively to the so-called 'real' and 'ideal' worlds.
>
> Finally, the adversary presents its guess for the world it is playing in, and succeeds if it guesses correctly. A multiparty NIKE protocol is said to be secure if no efficient adversary can succeed with probability non-negligibly more than $1/2$.

The notion of NIKE has found immense applications in practice [DH76, CGP+13, DKSW09, BMP04, DR06, JSI96]. Besides the theoretical importance of this fascinating notion, the wide applications also triggered active research in the area. However, achieving the aforementioned holy grail has proven to be extremely challenging and has evaded researchers for over three decades. Consequently, multiple relaxations have appeared in the literature towards advancing the state-of-the-art. We shall review in the following some of the important relaxations and the results achieved with these relaxations.

One of the first improvements was by Joux who gave the first 3-party NIKE protocol [Jou04] based on Weil and Tate bilinear maps. Boneh and Silverberg [BS02] showed how to obtain a multiparty NIKE protocol if multilinear maps existed. The candidate multilinear map constructions

by Garg, Gentry, and Halevi [GGH13a] using ideal lattices, and by Coron, Lepoint, and Tibouchi [CLT13] over the integers, provide the first implementations for N parties, but also require a trusted setup phase. However, the security notion in these works was restricted to the public values being only honestly generated. Cash, Kiltz, and Shoup [CKS09] and Freire, Hofheinz, Kiltz, and Paterson [FHKP13] studied the stronger security model that accounts for adversarial key-registrations of parties, in the random oracle model and in the standard model, respectively (albeit in just the 2-party setting). Finally, another line of relaxation is to have the adversary commit to the subset of parties he wishes to be challenged upon – the so-called 'static/semi-static' model; only recently, Boneh and Zhandry [BZ14] showed how to obtain the first multiparty NIKE without setup in the static/semi-static model.

Our Contribution. In this work, we solve the longstanding open problem and give the first multiparty NIKE protocol that is adaptively-secure without any setup in the standard model. Our construction is based on hybrid trapdoor commitments [CV07], indistinguishability obfuscation, and obliviously-patchable puncturable pseudorandom functions (PRFs), a new notion that we introduce. Stated informally, following is our main result. (A formal version appears as Theorem 3).

Theorem 1 (Informal)**.** Assume that there exists an indistinguishability obfuscator, a hybrid trapdoor commitment scheme, and an obliviously-patchable puncturable PRF, then there exists a multiparty non-interactive key exchange protocol with no setup that is *adaptively* secure in the standard model.

One of the main tools used in our work is the puncturing technique, of employing indistinguishability obfuscation, introduced in the influential work of [SW14]. This technique has found immense applications since its inception [BZ14, CGP14, DSKR14, GGHR14, GGHR14, HSW14]. However, a shortcoming of this technique is the somewhat inherent non-adaptivity of the security that can be achieved with this technique. This shortcoming has had strong implications, for instance, in the aforementioned work of Boneh and Zhandry who constructed multiparty NIKE but only in the semi-static model. We introduce novel techniques to overcome this shortcoming in employing indistinguishability obfuscation via the puncturing technique. These techniques are interesting in their own right and we believe would find wider applications. A central tool to our techniques is the new notion of obliviously-patchable puncturable PRFs.

Furthermore, one might hope to improve the setting by relying on non-interactive assumptions alone (Note that obliviously-patchable puncturable PRFs are an interactive primitive). However, we ascertain via a meta-reduction technique (see [HJK12] and references therein) that an interactive assumption is *necessary* to achieve adaptively-secure multiparty NIKE in the standard model (even with setup) in most natural settings which we shall characterize shortly.

Other Related Works. The following is a summary of some of the important works relevant in one aspect or the other.

- As mentioned earlier, static/semi-static security is easier to achieve than the full security of the adaptive model. However, adaptive security is the notion that captures aptly the real-life scenarios. Indeed, attackers in a computer network (hackers, viruses, insiders) may break into the network during the course of the computation, based on partial information that was already garnered. Moreover, even theoretically, it is easy to come up with examples of protocols that are secure in a static-corruption model, but that are trivially insecure in the adaptive setting. This was also evidenced in the setting of generic secure multiparty

computation – active research has resulted in many advancements only in the recent past [CGP14, DPR14, DSKR14, GP14, GS12]. However, these generic results cannot be extended to NIKE since the former need at least two-rounds of interaction. For NIKE, as mentioned earlier, advancements in adaptive security was made in stronger settings like random oracle model, etc. [HJK+14, SOK00] (See and references therein). Furthermore, no-interaction setting is extremely crucial for key exchange as discussed and argued in many important works including [DKSW09, FHKP13, JSI96].

- While some works have shown how to obfuscate simple functionalities [Can97, CRV10, CMR98, BR14a, LPS04, Wee05], it is only recently that obfuscation for poly-size circuits became possible [GGH+13b, BR14b, BGK+14] and was applied to building higher level cryptographic primitives [BZ14, HSW14, SW14].

- A different setting, where there is trusted third-party who generates a master public-key and the secret values are derived for identities of parties using the master secret-key, called identity-based NIKE (ID-based NIKE) has also been extensively studied [DE06, FHPS13, HKKW14, PS09, SOK00]. Sakai, Ohgishi, and Kasahara [SOK00] proposed the first efficient and secure ID-based NIKE scheme in the random oracle model, (with security models and formal proofs in follow up works [DE06, PS09]). Freire, Hofheinz, Paterson, and Striecks constructed programmable hash functions (PHFs) from multilinear maps [FHPS13]. By substituting the random oracle in the [SOK00] scheme with PHFs, they obtained the first ID-based NIKE scheme in the standard model. Although [SOK00] was only in the 2-party setting, [FHPS13] additionally gave the first multiparty ID-based NIKE scheme in the standard model. Although, like in the simpler ID-based public-key encryption setting [BF03], adaptive corruption is an issue in the ID-based NIKE setting, a differentiating factor in ID-based NIKE is that identities can be any arbitrary strings and hence adversarial key-registration is not an issue, unlike in the setting we consider. [GL03, GK10] studied password-based authenticated key exchange protocols.

- Freire, Hesse, and Hofheinz [FHH14] studied universal composability (UC) of NIKE for a modular treatment of the important primitive. They also showed how to achieve adaptive security in the random oracle model.

- Boneh and Zhandry were the first to present a multiparty NIKE protocol in the semi-static model without any setup [BZ14].

1.1 Our Techniques

The crux of our techniques is the new notion of obliviously-patchable puncturable PRFs, a variant of puncturable PRFs. Before we proceed to present our approach, let us quickly take stock of the current situation.

The background story, Part 1 – puncturable PRFs and indistinguishability obfuscation. Puncturable PRFs are a specific version of constrained PRFs introduced concurrently by [BW13, KPTZ13, BGI14]. Intuitively, puncturability of a PRF F allows one to 'puncture' a PRF key K at some input x to obtain a punctured key $K[x]$ which satisfies the following properties: Given $K[x]$, one can easily evaluate F at any input other than x; however, even when given $K[x]$, no adversary can distinguish between $F(K, x)$ and a random string. The same concept can be extended

towards puncturing at a set of input points. Based on the recent breakthrough results on candidate multilinear maps [GGH13a, CLT13], [GGH+13b] showed how to construct indistinguishability obfuscators for all polynomial-size circuits. Following this, Sahai and Waters, in their influential work [SW14], showed how to realize a host of primitives using indistinguishability obfuscation. The core technique introduced in this work is called the puncturing technique that puts together the strengths of puncturable PRFs and indistinguishability obfuscators. This work spawned a multitude of results that realized various other primitives [BZ14, CGP14, DSKR14, GGHR14, GP14, HSW14]; the central tool in all these results is the puncturing technique. Let us review this technique by considering a natural application to multiparty NIKE (in the semi-static model) as shown in [BZ14].

The background story, Part 2 – The puncturing technique. Multiparty NIKE protocol of [BZ14] in the semi-static model is a natural application of the puncturing technique. To see this, let us try to arrive at this protocol from ground up. Note that each party needs to publish a public value such that this public value computationally hides the corresponding secret value. A natural candidate would be to use one-way functions (OWFs); however, we shall use pseudorandom generators (PRGs) for the reason that will be evident very shortly. Thus, for a party P_i, PRG seed s_i will be the secret value and $x_i = PRG(s_i)$ would be the public value. Furthermore, we require that even if an adversary learns the common keys of some subsets of parties, for any other subset of honest parties, the actual common key is indistinguishable from a uniform string. Observe that this requirement is quite akin to the notion of PRFs – namely, even if an adversary knows the output of the PRF on some inputs, the PRF output on any other input is indistinguishable from a random string. Thus, a natural solution would be to have the common key for a subset S computed as PRF output at $(x_i)_{i \in S}$ as the input.

Now, what remains is to enforce that only those parties P_j with $j \in S$ are able to evaluate the PRF at $(x_i)_{i \in S}$. This is achieved by obfuscating the program that first checks whether an input to the program contains the PRG seed s_j for a PRG output x_j for some $j \in S$ and only then evaluates the PRF at $(x_i)_{i \in S}$. This obfuscated program would be the common reference string (CRS)/setup. Ignoring for the moment that the adversary also gets to see this obfuscated program, we observe that just the security of PRFs would have sufficed to achieve the security requirement of multiparty NIKE. However, to establish security even in the presence of this obfuscated program that contains the PRF key hardcoded within, we need to employ the security of puncturable PRFs. Essentially, the idea for proof of security would be to move to a hybrid where the PRF key hardcoded is actually punctured at the input $(x_i)_{i \in S^*}$, where S^* is the challenge subset of honest parties. However, now that the program cannot compute the output for an input corresponding to $(x_i)_{i \in S^*}$, we have actually modified the input/output functionality of the program. This is undesirable as indistinguishability obfuscation guarantees indistinguishability of obfuscations of only those programs that are functionally equivalent. In order to still employ the security of indistinguishability obfuscation, we instead will puncture at $(x_i^*)_{i \in S^*}$, where each x_i^* thereof is a random string (instead of a PRG output). This ensures that, with all but negligible probability, the program never evaluates the PRF on this input (since, with all but negligible probability, there does not exist any PRG seed that would pass the initial check performed by the obfuscated program) (This is the point where we needed PRG instead of just any one-way functions).

A crucial point to note in the above illustration of the puncturing technique is the following. $(x_i^*)_{i \in S^*}$, and in particular S^*, need to be fixed before generating the obfuscated program which is a part of the initial setup. The implication on the best achievable security would be that the adversary needs to commit to the subset S^* of honest parties it wishes to be challenged upon even before it gets to see the CRS. This is the very point that restricts the puncturing technique from

giving us the desired adaptive security.

Now we shall see how we solve the generic problem of adaptivity. We shall first deal with the problem with a setup and then build on it to remove the setup.

The story ahead, with setup. *The problem at hand.* Now let us dissect the problem at hand. Recall that to prove secure any multiparty NIKE protocol, we need to construct a reduction $\mathcal{R}$ that when given access to any adversary $\mathcal{A}$ that breaks the protocol (in the sense of the security game described above), it should be able to break some underlying cryptographic hard problem Π. Now note that such an adversary $\mathcal{A}$ makes many queries on the secret values and common keys for some subset of honest parties and finally wishes to be challenged upon some other subset S^*. Thus, intuitively, $\mathcal{R}$ must be able to somehow incorporate his own challenge for Π into *exactly* the challenge subset of honest parties. Clearly, there is a very narrow bottleneck: the reduction does not know ahead of time what S^* would be; if $\mathcal{R}$ fails to guess it correctly and embed his Π challenge somewhere else, then he might not be able to answer the rest of the queries made by $\mathcal{A}$. Furthermore, the total number of subsets can be exponential in the security parameter, implying that $\mathcal{R}$ can correctly guess with only negligible probability, in turn constricting $\mathcal{R}$ in exploiting the power of $\mathcal{A}$.

Our solution. Now we have just the right background to get to our solution. Let us start with the natural idea of enforcing that only those with a secret value can compute the common key, using indistinguishability obfuscation of the program that first checks for such a secret value and only then evaluates the PRF. Since the reduction does not know what the challenge subset would be, puncture at all possible challenge subsets. Now, we are faced with the usual problem of maintaining functional equivalence of the punctured program: when an adversary corrupts an honest party and gets its secret value, it needs to be able to run the obfuscated punctured program and the program should output the common key, which means that the program should somehow get information about the PRF outputs at the already punctured points. The idea to solve this problem is extremely simple yet novel: *Embed a 'patch' to the puncture into the secret value itself.* We do it in such a way that the adversary will not be able to distinguish between an honestly sampled secret value and a secret value that has a patch embedded within itself. On the other hand, the punctured program would have some trapdoor hardcoded within itself using which it can recover that patch and still be able to evaluate the PRF. This trapdoor is hidden, thanks to obfuscation. Now, one may feel apprehensive that the functional equivalence may still not be entirely established as the PRF could be evaluated only when the input contains a secret value with a patch embedded in it. Note that in fact our original program also needs to evaluate the PRF only when the input contains a secret value. In our solution, we ascertain that the program gets the patch every time it gets a valid secret value by ensuring that there is only one unique secret value for every public value – and it is the secret value that has the patch embedded in it.

One slight detail of our solution. Observe that the reduction would not even know for which all i, the adversary would have honest parties registered and for which all i the adversary himself would register malicious parties. However, the reduction needs to have computed the public values of honest parties ahead of time – i.e., before computing the obfuscated program in the CRS – so that he can puncture at these values. The solution to this problem is extremely simple. We shall have the reduction compute the public-value secret-value pair for *every* i ahead of time and puncture at all the possible subset of honest parties with those honestly generated public values, and generate the CRS. Then, the honestly generated public-value secret-value pairs are used only if the adversary requests to register an honest party for i; otherwise, the reduction would simply ignore the pair.

The core subtlety – a circularity. With the aforementioned idea being one part of the whole idea, the core subtlety is yet to be met. Note that the inputs on which the PRF would be computed are the public values. So, in effect, we would be puncturing at honestly generated public values. However, the core idea is to embed within the secret values, the patch to the PRF key at the punctured points. But the sequence should be that the patches are first sampled and then they are embedded within the secret values and then the corresponding public values are computed. Clearly, we have run into a circularity that simply seems, as yet, unsurmountable.

Breaking the circularity. In breaking the circularity is where the further crux of our solution lies – *obliviously-patchable* puncturable PRFs. Before we proceed with our solution, a quick look at the properties of this new primitive that we introduce is in order. This primitive allows one to sample patches without knowledge of at what inputs of the PRF these will serve as patches. Then, post sampling the patches, given any input (or a set of inputs), one can sample a key that is punctured at these inputs in such a way that the patches sampled earlier patch the key punctured at these inputs. Stepping back, this is precisely what we need to break the circularity – the reduction shall first sample the patches (obliviously of the public values), embed these patches within the secret value, and then compute the corresponding public values. Then, a PRF key that is punctured at these public values is sampled and hardcoded in the program that will be obfuscated.

Putting ideas into effect. We shall now explicate how exactly we shall implement the aforementioned ideas. Our aim is to use the minimally strong tools. Let us begin with the public-value secret-value pair. These will be just commitment and (message, opening), respectively. (The reason to use commitments instead of PRGs or OWFs would be evident shortly). Thus, the check performed within the obfuscated program is whether the secret value (m_i, open_i) w.r.t. a public value com_i is such that $\mathrm{com}_i = \mathsf{Commit}(m_i; \mathrm{open}_i)$. By the hiding property of the commitment scheme, we have that the public value hides the secret value. Now, in our protocol, we could have let m_i to be just some random string. However, note that our idea is to somehow embed a patch within m_i. Thus, the idea is to instead prescribe m_i to be an encryption ct_i of some random plaintext pt_i. During the reduction, we will set pt_i to be a patch. Since the secret value is just $(\mathrm{ct}_i, \mathrm{open}_i)$, the modification will not be detected by the adversary owing to the CPA security of the encryption scheme. Furthermore, by using a statistically binding commitment scheme Commit, we are also guaranteed that the only secret value corresponding to com_i is the $(\mathrm{ct}_i, \mathrm{open}_i)$ where ct_i is an encryption of a patch.

The core subtlety – yet another circularity. It is instructive at this point to step back and assess whether we have constructed the reduction we had desired for – a reduction from breaking our protocol to breaking the security of the obliviously-patchable puncturable PRF. Before we proceed, in detail, the reduction needed is as follows: It first invokes the adversary with the obfuscated program punctured at public values (corresponding to honest parties) and gives the adversary the public values if and when it requests to register honest parties; then, when the adversary asks to reveal the secret value of any honest party, we would like that the reduction query the PRF challenger to obtain a patch, encrypt it, and then somehow give an opening too for the public value, namely the commitment. On the other hand, assessing the hybrid we are currently at, patches are computed before computing the public values (the sequence is patches $\rightarrow$ encryption of patches $\rightarrow$ commitments (public values) to those ciphertexts). Clearly, we have run into yet another circularity. A naïve solution to this would be to have the commitments equivocable – this is so that the commitments can be sampled obliviously of what they commit to, and can later be opened to any desired value, which in our case would be encryptions of patches. However, the problem is that, as we recall, we needed Commit to be statistically binding, and, equivocability and statistically binding are two conflicting properties of a commitment scheme.

Breaking the second circularity. We employ hybrid trapdoor commitments introduced by [CV07]. Just as required, these commitments work in two modes – one, the statistically-binding mode, and the other, the equivocable mode. A commitment's mode is decided by the CRS used to commit. The two distributions of the CRS corresponding to the two modes are indistinguishable.

All in all, the way we use these commitments is as follows: in the protocol, the setup would include the CRS of the commitment scheme for the statistically-binding mode. In the proof, we shall move to a hybrid where the committed value is a ciphertext encrypting a patch. Therein, we shall argue that the ciphertext is the only value the commitment can be opened to, owing to the statistically-binding property. Later, we shall move to a hybrid where we switch the commitment CRS to the equivocable mode. This would enable us to reach a hybrid which can be deemed our final reduction – therein, the public values (commitments) are first given to the adversary; later, when the adversary queries for a secret value, the reduction would query for the corresponding patch, encrypt it, and equivocate the commitment to the resulting ciphertext.

Constructing obliviously-patchable puncturable PRFs. We give a concrete construction of the novel primitive, obliviously-patchable puncturable PRFs, using graded encoding schemes, a candidate version of multilinear maps, first constructed by [GGH13a]. This construction is proven secure based on a weak 1-more variant of the Multilinear Decisional Diffie-Hellman Inversion (MDDHI) assumption. Furthermore, using multilinear maps, we show how to relax the assumption to just a weak 1-more variant of the Multilinear Decisional Diffie-Hellman (MDDH) assumption. We prove these assumptions secure in the usual generic group model. We remark here that, in literature, one of the simplest assumptions used for multilinear maps is the Multilinear Decisional Diffie-Hellman (MDDH) assumption. 1-weak MDDH assumption (that we shall state later) is a natural variant of the MDDH assumption in the interactive setting.

Necessity of an interactive assumption. Observe that obliviously-patchable puncturable PRFs is an interactive assumption. We ascertain via a meta-reduction technique that in natural settings, an interactive assumption would be necessary to prove secure any multiparty NIKE protocol (even with setup). More specifically, the natural settings we consider are identical or even weaker than the conventional settings: here, we rule out that all reductions – that would use the adversary in a black-box manner and are allowed to even rewind the adversary – to any hard problem corresponding to a non-interactive assumption. In fact, we provide a bound on the amount of interaction needed depending on the number of parties in the multiparty NIKE system.

Removing the setup. Recall that by now we have only dealt with setup (obfuscated program and CRS of the commitment scheme were parts of the setup). Now we shall show how to remove this setup altogether. A traditional solution to such a problem is to let the parties themselves compute the setup. Now with multiple setups, the question next is whose setup the parties should use for computation.

In [BZ14], a simple solution was proposed – to compute the common key for any subset $\mathcal{S}$ of parties, use the setup generated by P_{i^*}, where i^* is the smallest element in $\mathcal{S}$. (Many other choices could have worked, and the above is one such simple choice). However, as pointed out by [BZ14], this modification introduces a weak link in security: consider an adversary who registers P_{i^*} by himself. Note that a party is expected to output an obfuscated program as a part of its public value. Potentially, the adversary may compute the program that is part of the public value of P_{i^*} in some malicious manner that the program leaks some information about the secret value that forms a part of the input on which the program is run. Now note that an adversary may present a query to receive the common key of some set $\mathcal{S}$ as derived by some honest party P_i; furthermore, let i^* be the smallest element in $\mathcal{S}$. Then, the challenger would run the malformed obfuscated program of P_{i^*}

using the secret value sv_i of P_i and present to the adversary the resulting value; this value may leak information about sv_i. An elegant solution proposed in [BZ14] is the following. Before using sv_i to run a program, transform it into a "non-reusable" form. More specifically, in the no-setup case, public-value secret-value pair is modified into a verification-key signing-key pair. Then, instead of running the obfuscated program with the very secret value, a party is instructed to run the program with a signature on $\mathcal{S}$ computed with its secret value. The check inside the program now is verification of the signature. Unforgeability of the signature scheme ensures non-reusability of the value on which the program is run. [BZ14] observe that any signature scheme would not suffice; they need what they call constrained signatures and provide a construction for this primitive using indistinguishability obfuscation.

Now let us see how we can solve the problem. Towards providing a simple solution, here is an insightful observation on the problem at hand: it is not against the desired security if the maliciously generated program by P_{i^*} leaks information about the secret value that might be reusable for any other $\mathcal{S}'$ which contains i^*. To see this, roughly speaking, note that the adversary (who is playing the part of P_{i^*}) could himself have derived the common key by himself. Moreover, the security game rightly captures the requirement by specifying that the challenge set of parties needs to be a set of all honest parties. Thus, if suffices to ensure that the leaked information, although reusable for sets like $\mathcal{S}'$, is not reusable for the challenge set of all honest parties (which does not include the malicious P_{i^*}). To this end, our solution is the following. Every party will have a (commitment, opening) pair dedicated towards every other party. Then, when a party needs to run the program generated by P_{i^*}, it would use the (commitment, opening) pair dedicated towards P_{i^*}.

A final subtlety in removing the setup is in constructing ciphertexts. It stems from two reasons:

1. Recall that in our no-setup case, the public value of a party consists of an obfuscated program and some other values. Looking ahead, it will be required in the proof of security that the simulator computes these other values using encryptions w.r.t. public keys corresponding to *other* honest parties.
2. We are in the non-interactive setting.

Thus, in the protocol itself, it is not possible to instruct the parties to construct their public values using the public keys present in other parties' public values. We overcome this by using PKE schemes with pseudorandom ciphertexts (i.e., ciphertexts that are indistinguishable from random strings). In the protocol, instead of the ciphertexts, we would have random strings (A crucial point to note is that no decryption will be performed in the protocol itself: The secret value is simply an opening to the commitment in the public value; the opened value is cast as a ciphertext only in the proof of security). Then in the proof, these random strings will be indistinguishably replaced with ciphertexts.

Although there are certain other subtleties, the aforementioned ones capture the crux of the idea. We shall deal with the subtleties in detail in the formal proof.

On our assumptions. Recall that our construction is based on PKE with pseudorandom ciphertexts, hybrid trapdoor commitments, indistinguishability obfuscation and obliviously-patchable puncturable PRFs. [CV07] show that hybrid trapdoor commitments can be constructed from just OWFs. Furthermore, under the assumption that NP is not solvable in probabilistic polynomial time in the worst case, Moran and Rosen [MR13] show that indistinguishability obfuscation implies OWFs. PKE with pseudorandom ciphertexts can be based on Decisional Diffie-Hellman (DDH) assumption. Also, we show how to obtain obliviously-patchable puncturable PRFs from a variant of the DDH assumption in the multilinear setting. Thus, effectively, on the same assumption as

[MR13], our construction can be based on indistinguishability obfuscation and on the variant of the DDH assumption in the multilinear setting.

Recently, [CHL+14] showed that certain assumptions based on existing approximate multilinear maps [GGH13a, CLT13] can be broken in polynomial time. Our assumptions *do not* succumb to the attacks by [CHL+14].

Roadmap. In Section 2, we shall recall the necessary background. In Section 3, we introduce formally our new notion of obliviously-patchable puncturable PRFs. In Section 4, we present our protocol with setup, followed by a high-level structure of the proof of security in Section 5. In Section 6, we present our protocol without any setup, and provide a high-level structure of the proof of security in Section 7. Finally, in Section 8, we establish via a meta-reduction technique, the necessity of interactive assumptions (such as PRFs) for achieving adaptive NIKE even with setup.

In Appendix A, we provide a formal proof of security of our protocol with setup. Then, in Appendix B, we provide a formal proof of security of our protocol without any setup. In Appendix C, we provide concrete constructions of obliviously-patchable puncturable PRFs using multilinear maps and their approximations (i.e., leveled graded-encoding schemes first constructed by [GGH13a]). In Appendix D, we prove security of the assumptions based on which we construct our PRFs, in the generic group model.

2 Preliminaries

Notations. We denote concatenation of two bit-strings a and b by $a \circ b$. U_ℓ denotes the uniform distribution over ℓ bits. Let $\mathcal{D}_1, \mathcal{D}_2$ be two distributions over the same domain. We denote by $\mathcal{D}_1 \equiv \mathcal{D}_2$ that they are identical, by $\mathcal{D}_1 \approx_s \mathcal{D}_2$ that they are statistically close, and $\mathcal{D}_1 \approx_c \mathcal{D}_2$ that they are computationally indistinguishable.

- We often consider a bit string x divided into blocks of sub-strings; we denote the ith block by $x|_i$.
- Let $n, N \in \mathbb{N}$. W e consider strings where some of the blocks can be just $\perp$: $x \in (\{0,1\}^n \cup \perp^n)^N$. For simplicity, we represent such strings x, where $x|_i = x_i$ for $i \in \mathcal{S}$ and $x|_i = (\perp, \ldots, \perp)$ for $i \notin \mathcal{S}$, by $(\mathcal{S}, (x_i)_{i \in \mathcal{S}})$.
- Let $1 \leq G \leq N$. Also, let $n \in \mathbb{N}$. We often consider the set of strings in $x' = (\mathcal{S}, (x'_i)_{i \in \mathcal{S}}) \in (\{0,1\}^n \cup \perp^n)^N$ such that $|\mathcal{S}| \leq G$. We denote this set by $[[n, G, N]]$. (Looking ahead, this set will form the domain of our PRFs).
- Consider any $x \in (\{0,1\}^n)^N$. We consider the set of all elements $x' \in [[n, G, N]]$ where, for every $x'|_i \neq (\perp, \ldots, \perp)$, we have $x'|_i = x|_i$. We denote the set of such elements by 2^x_G.

In all games that we consider, it can be efficiently verified whether an adversary's query is valid (as specified by the experiment) or not; thus, w.l.o.g. we shall assume that an adversary only presents valid queries unless specified otherwise.

Indistinguishability Obfuscation. For any two circuits C_0, C_1 over the same input space, if for every input x, $C_0(x) = C_1(x)$, then this is denoted by $C_0 \equiv C_1$; such circuits are said to be 'functionally equivalent'.

Definition 1 (Indistinguishability Obfuscator ($\mathsf{i}\mathcal{O}$))**.** A uniform PPT machine $\mathsf{i}\mathcal{O}$ is called an *indistinguishability obfuscator* for a circuit class $\{\mathcal{C}_\lambda\}$ if the following conditions are satisfied:

Preserving functionality: For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, we have that

$$\Pr[\mathsf{i}\mathcal{O}(\lambda, C) \equiv C] = 1$$

Polynomial slowdown: There exists a universal polynomial p such that for every λ, $C \in \{\mathcal{C}_\lambda\}$, we have $|C'| \leq p|C|$, where, $C' \leftarrow \mathsf{i}\mathcal{O}(\lambda, C)$.

Indistinguishability: For any (not necessarily uniform) PPT adversaries Samp, $\mathcal{D}$, there exists a negligible function η such that the following holds: if $\Pr[C_0 \equiv C_1 \ : \ (C_0, C_1, \mathrm{aux}) \leftarrow \mathsf{Samp}(1^\lambda)] > 1 - \eta(\lambda)$, then we have:

$$\Big| \Pr\left[\mathcal{D}(\mathrm{aux}, \mathsf{i}\mathcal{O}(\lambda, C_0)) = 1 \ : \ (C_0, C_1, \mathrm{aux}) \leftarrow \mathsf{Samp}(1^\lambda)\right] - \Pr\left[\mathcal{D}(\mathrm{aux}, \mathsf{i}\mathcal{O}(\lambda, C_1)) = 1 \ : \ (C_0, C_1, \mathrm{aux}) \leftarrow \mathsf{Samp}(1^\lambda)\right] \Big| \leq \eta(\lambda)$$

When clear from context, we will often drop λ as an input to $\mathsf{i}\mathcal{O}$ and as a subscript for C.

In this paper, we will make use of such indistinguishability obfuscators for all polynomial-size circuits:

Definition 2 (Indistinguishability Obfuscator for $P/poly$)**.** A uniform PPT machine $\mathsf{i}\mathcal{O}$ is called an *indistinguishability obfuscator* for $P/poly$ if the following holds: Let $\mathcal{C}_\lambda$ be the class of circuits of size at most λ. Then $\mathsf{i}\mathcal{O}$ is an indistinguishability obfuscator for the class $\{\mathcal{C}_\lambda\}$.

The first candidate construction of such obfuscators is due to [GGH+13b].

Hybrid Trapdoor Commitments. We now recall the notion of hybrid trapdoor commitments introduced by [CV07]. Intuitively an hybrid trapdoor commitment scheme can be either an unconditionally binding commitment scheme or a trapdoor commitment scheme depending on the distribution of the CRS. They are somewhat weaker than the more widely employed mixed commitments [DN02]: hybrid trapdoor commitments can either be polynomially trapdoor commitments or unconditionally binding commitments, while mixed commitment can either be trapdoor commitments or extractable commitments.

Definition 3 (Commitments)**.** A triplet of PPT algorithms $(\mathsf{Gen}, \mathsf{Commit}, \mathsf{Ver})$ is a commitment scheme if the following conditions hold.

Completeness. For all m it holds that

$$\Pr\left[\mathsf{Ver}(\mathrm{crs}, \mathrm{com}, (m, \mathrm{open})) = 1 \ : \ \begin{array}{c} \mathrm{crs} \leftarrow \mathsf{Gen}(1^\lambda) \\ \mathrm{com} \leftarrow \mathsf{Commit}(\mathrm{crs}, m; \mathrm{open}) \end{array}\right] = 1$$

Binding. For any PPT adversary $\mathcal{A}$ there is a negligible function negl such that for all sufficiently large λ it holds that for all but negligible fraction of $\mathrm{crs} \in \mathsf{Gen}(1^\lambda)$,

$$\Pr\left[(\mathrm{com}, m_0, m_1, \mathrm{open}_0, \mathrm{open}_1) \leftarrow \mathcal{A}(\mathrm{crs}) \ : \ \mathsf{Ver}(\mathrm{crs}, \mathrm{com}_0, (m_0, \mathrm{open}_0)) = \mathsf{Ver}(\mathrm{crs}, \mathrm{com}_1, (m_1, \mathrm{open}_1)) = 1\right] \leq$$

Hiding. For any PPT adversary $\mathcal{A}$ there is a negligible function negl such that, for all m_0, m_1 where $|m_0| = |m_1|$, and for sufficiently large λ, it holds that

$$\Pr\left[b \leftarrow \mathcal{A}(\mathrm{crs}, \mathrm{com}) \ : \ \begin{array}{c} \mathrm{crs} \leftarrow \mathsf{Gen}(1^\lambda) \\ b \leftarrow \{0,1\} \\ \mathrm{com} \leftarrow \mathrm{com}(\mathrm{crs}, m_b) \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$$

If the binding property holds with respect to a computationally unbounded algorithm $\mathcal{A}$, the commitment scheme is said to be statistically binding.

Definition 4 (Hybrid trapdoor commitments)**.** $\mathsf{HCOM} = (\mathsf{HGen}, \mathsf{HTGen}, \mathsf{HCommit}, \mathsf{HTCommit}, \mathsf{HTDec}, \mathsf{HVer})$ is a hybrid trapdoor commitment scheme (HTCS, for short) if the following conditions hold.

Binding. $(\mathsf{HGen}, \mathsf{HCommit}, \mathsf{HVer})$ is a statistically binding commitment scheme;

Trapdoor property. $(\mathsf{HTGen}, \mathsf{HCommit}, \mathsf{HVer})$ is a commitment scheme, and $\mathsf{HTCommit}$ and HTDec are polynomial-time algorithms such that, for all m, the following distributions are computationally indistinguishable even to an adversary that is given aux:

$$(\mathrm{crs}, \mathrm{aux}, \mathrm{com}, \mathrm{open}, m) \ : \ \begin{array}{c} (\mathrm{crs}, \mathrm{aux}) \leftarrow \mathsf{HTGen}(1^\lambda), \\ (\mathrm{com}, \mathrm{open}) \leftarrow \mathsf{HCommit}(\mathrm{crs}, m) \end{array}$$

and

$$(\mathrm{crs}, \mathrm{aux}, \mathrm{com}', \mathrm{open}', m) \ : \ \begin{array}{c} (\mathrm{crs}, \mathrm{aux}) \leftarrow \mathsf{HTGen}(1^\lambda), \\ (\mathrm{com}', \mathrm{aux}_{\mathrm{com}'}) \leftarrow \mathsf{HTCommit}(\mathrm{crs}, \mathrm{aux}), \\ \mathrm{open}' \leftarrow \mathsf{HTDec}(\mathrm{crs}, \mathrm{aux}_{\mathrm{com}'}, m) \end{array}$$

Hybrid property. Let HTGen' be an algorithm that restricts the output $(\mathrm{crs}, \mathrm{aux})$ of $\mathsf{HTGen}(1^\lambda)$ to crs, then for every PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{hyb}}_{\mathcal{A}}$ is negligible, where,

$$\mathsf{Adv}^{\mathsf{hyb}}_{\mathcal{A}}(\lambda) := |\Pr[1 \leftarrow \mathcal{A}(\mathrm{crs}_0) \mid \mathrm{crs}_0 \leftarrow \mathsf{HGen}(1^\lambda)] - \Pr[1 \leftarrow \mathcal{A}(\mathrm{crs}_1) \mid \mathrm{crs}_1 \leftarrow \mathsf{HTGen}(1^\lambda)]|.$$

2.1 PKE With Pseudorandom Ciphertexts

We now define PKE schemes with pseudorandom ciphertexts [CLOS02, BC05]. Roughly, these are the schemes with a property that for any plaintext message a randomly generated ciphertext is computationally indistinguishable from a uniform random string of the same length.

Definition 5 (PKE with pseudorandom ciphertexts)**.** A public-key encryption scheme $\Sigma = (\mathsf{E.Gen}, \mathsf{Enc}, \mathsf{Dec})$ is said to have pseudorandom ciphertexts if, for $(\mathrm{pk}, \cdot) \leftarrow \mathsf{E.Gen}(1^\lambda)$, for any plaintext message m, the distribution ensembles $\mathsf{Enc}(\mathrm{pk}, m)$ and $U_{\ell'}$ are all computationally indistinguishable, where the ciphertexts of Σ are of length ℓ'.

In [CLOS02], Canetti et al. also provide a simple construction of such schemes based on trapdoor permutations. Briefly, the construction in [CLOS02] is as follows. With the public key as the description of f, a trapdoor function, encryption of a bit b is: $f(x), b \oplus HC(x)$, where x is chosen at random from the domain of f and $HC(\cdot)$ is a hard-core predicate of f. Notice that for this scheme, the distribution of encryption of a random bit b is itself a uniform distribution over strings of the same length as the ciphertexts. This primitive can also be constructed based on Decisional Diffie-Hellman assumption.

Adaptive Multiparty Non-interactive Key Exchange. An adaptive multiparty NIKE protocol has the following three algorithms:

$\mathsf{Setup}(\lambda, N, G)$**:** The setup algorithm takes a security parameter λ and two integers G and N. G is the maximum number of parties that can derive a common secret key, and N is an upper bound on the number of parties in the system. It outputs public parameters params.

$\mathsf{Publish}(\lambda, \mathsf{params}, i)$**:** Each party executes the publishing algorithm, which takes as input the public parameters and the index of the party, and generates two values: a secret key sv_i and a public value pv_i. Party P_i keeps sv_i as his secret value, and publishes pv_i to the other parties.

$\mathsf{KeyDerive}(\mathsf{params}, \mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, j, \mathrm{sv}_j)$**:** Finally, to derive the common key $k_{\mathcal{S}}$ for a subset $\mathcal{S} \subseteq [N]$, each party in $\mathcal{S}$ runs KeyGen with params, its secret value sv_j, and the public values $\{\mathrm{pv}_i\}_{i\in\mathcal{S}}$ of the parties in $\mathcal{S}$.

For correctness, we require that each party derives the same secret key. That is, for all $\mathcal{S} \subseteq [N]$, $|\mathcal{S}| \leq G$, $i, i' \in \mathcal{S}$,

$$\mathsf{KeyDerive}(\mathsf{params}, \mathcal{S}, (\mathrm{pv}_j)_{j\in\mathcal{S}}, i, \mathrm{sv}_i) = \mathsf{KeyDerive}(\mathsf{params}, \mathcal{S}, (\mathrm{pv}_j)_{j\in\mathcal{S}}, i', \mathrm{sv}_{i'}). \tag{1}$$

One can talk about a secret value sv_i for pv_i satisfying correctness *w.r.t.* some $\mathrm{sv}_{i'}$ for $\mathrm{pv}_{i'}$. This would mean that for every $\mathcal{S} \subseteq [N]$, $|\mathcal{S}| \leq G$, such that $i, i' \in \mathcal{S}$,

$$\mathsf{KeyDerive}(\mathsf{params}, \mathcal{S}, (\mathrm{pv}_j)_{j\in\mathcal{S}}, i, \mathrm{sv}_i) = \mathsf{KeyDerive}(\mathsf{params}, \mathcal{S}, (\mathrm{pv}_j)_{j\in\mathcal{S}}, i', \mathrm{sv}_{i'}).$$

The security notion is captured by the game below. Herein, the adversary can corrupt parties and reveal the common secret for arbitrary subsets of parties of its choice.

More formally, for an adversary $\mathcal{A}$, consider the experiments $\textsc{real}_{\mathcal{A}}^{\mathsf{aNIKE}}$ and $\textsc{rand}_{\mathcal{A}}^{\mathsf{aNIKE}}$ described below. For for $\textsc{expt} \in \{\textsc{real}, \textsc{rand}\}$, $\textsc{expt}_{\mathcal{A}}^{\mathsf{aNIKE}}$ is defined as follows. The experiment is parameterized by the total number of parties N, the maximal group size G (where potentially G is the same as N), and an adversary $\mathcal{A}$:

$\mathsf{params} \leftarrow \mathsf{Setup}(\lambda, N)$

$b' \leftarrow \mathcal{A}^{\mathsf{Reg}(\cdot),\mathsf{RegCorr}(\cdot,\cdot),\mathsf{Ext}(\cdot),\mathsf{Rev}(\cdots),\mathsf{Test}(\cdots)}(\lambda, N, \mathsf{params})$

where,

$\mathsf{Reg}(i \in [N])$ registers an honest party P_i. It takes an index i, and runs $(\mathrm{sv}_i, \mathrm{pv}_i) \leftarrow \mathsf{Publish}(\mathsf{params}, i)$. The challenger records the tuple $(i, \mathrm{sv}_i, \mathrm{pv}_i, \mathit{honest})$, and sends pv_i to $\mathcal{A}$.

$\mathsf{RegCorr}(i \in [N], \mathrm{pv}_i)$ registers a corrupt party P_i^*. It takes an index i and a public value pv_i. The challenger records $(i, \perp, \mathrm{pv}_i, \mathit{corrupt})$. The adversary may make multiple queries for a particular identity, in which case the challenger only uses the most recent record.

$\mathsf{Ext}(i)$ extracts the secret key for an honest registered party. The challenger looks up the tuple $(i, \mathrm{sv}_i, \mathrm{pv}_i, \mathit{honest})$, and returns sv_i to the challenger.

$\mathsf{Rev}(\mathcal{S}, i)$ reveals the common secret for a group $\mathcal{S} \subseteq [N]$, $|\mathcal{S}| \leq G$ of parties, as computed by the ith party, where $i \in \mathcal{S}$. We require that party P_i was registered as honest. The challenger uses the secret key for party P_i to derive the common secret key $k_{\mathcal{S}}$, which it returns to the adversary.

$\mathsf{Test}(\mathsf{ChQ})$ takes a set $\mathsf{ChQ} \subseteq [N]$, $|\mathsf{ChQ}| \leq G$ of parties, all of which were registered as honest. If $\text{EXPT} = \text{REAL}$, the challenger runs KeyGen to determine the common secret key (arbitrarily choosing which party to calculate the key), which it returns to the adversary. Otherwise (i.e., if $\text{EXPT} = \text{RAND}$), the challenger generates a random key k to return to the adversary.

We require that all reveal and test queries are for distinct sets, and no extract query is allowed on any party in a reveal query. We require that all register queries and register-corrupt queries are for distinct i, and that $\text{pv}_i \neq \text{pv}_j$ for any $i \neq j$. We define

$$\mathsf{Adv}^{\mathsf{aNIKE}}_{\mathcal{A}}(\lambda) := |\Pr[\text{REAL}^{\mathsf{aNIKE}}_{\mathcal{A}} \to 1] - \Pr[\text{RAND}^{\mathsf{aNIKE}}_{\mathcal{A}} \to 1]|$$

Definition 6 (Adaptively-secure NIKE)**.** A multiparty key exchange protocol $(\mathsf{Setup}, \mathsf{Publish}, \mathsf{KeyGen})$ is adaptively secure if, for any polynomial N, and any PPT adversary $\mathcal{A}$, the function $\mathsf{Adv}^{\mathsf{aNIKE}}_{\mathcal{A}}(\lambda)$ is negligible.

Definition 7 (Adaptively-secure NIKE with no trusted setup)**.** A NIKE protocol has untrusted setup if the random coins used by Setup are part of the public parameters params. That is, $\mathsf{Setup}(\lambda, N, G; r) = (\mathsf{params}, r)$.

An even stronger notion is that of no setup:

Definition 8 (Adaptively-secure NIKE with no setup)**.** A NIKE protocol has no setup if Setup does nothing. That is, $\mathsf{Setup}(\lambda, N, G; r) = (\lambda, N, G)$.

Note that identities in the scheme and security model are merely used to track which public keys are associated with which users – we are *not* in the identity-based setting.

3 Obliviously-Patchable Puncturable PRFs

Definition 9 (Obliviously-patchable puncturable PRFs)**.** Let n, G, N, m be polynomials in λ such that $1 \leq G \leq N$. Let $F : \mathcal{K} \times [[n, G, N]] \to \mathcal{Z}$, where $\mathcal{Z} \subset \{0,1\}^m$. We call F an *obliviously-patchable puncturable PRF* if there exist associated algorithms $(\mathsf{F.ParamGen}, \mathsf{F.KeyGen}, \mathsf{Puncture}, \mathsf{PatchGen}, \mathsf{Eval}, \mathsf{OPatchGen}, \mathsf{OPuncture})$, such that $\mathsf{PatchGen}$ is deterministic and the following properties hold.

- **[Functionality preserved under puncturing.]** For every set[1] $S \subseteq [[n, G, N]]$, for every $x \in [[n, G, N]]$ where $x \notin S$, we have that:

$$\Pr\left[F(K,x) = \mathsf{Eval}(K[S], x) : \begin{array}{c} \mathsf{F.params} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N), \\ K \leftarrow \mathsf{F.KeyGen}(1^\lambda, \mathsf{F.params}), \\ K[S] = \mathsf{Puncture}(K, S) \end{array}\right] = 1$$

- **[Block-wise patchable.]** For every $i \in [N]$, $x_i \in \{0,1\}^n$, $K \in \mathcal{K}$,

$$\mathsf{patch}(K, (i, x_i)) \leftarrow \mathsf{PatchGen}(K, i, x_i)$$

  for every $x \in (\{0,1\}^n)^N$ such that $x|_i = x_i$, we have that

$$F(K, x') = \mathsf{Eval}((K[2^x_G], \mathsf{patch}(K, (i, x_i))), x')$$

  where, $x' \in 2^x_G$. (Recall that 2^x_G is the set of all $x' \in [[n, G, N]]$ where, for every $x'|_i \neq (\bot, \ldots, \bot)$, we have $x'|_i = x|_i$.)

[1] Although we present a generic definition of puncturing a key at any subset of the domain, in this work, it would suffice to be able to puncture only at subsets of the form 2^x_G.

- **[Obliviously patchable.]** For every $x \in (\{0,1\}^n)^N$, output distributions of the following two processes are identical:

$\mathsf{F.params} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$	$\mathsf{F.params} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$
$K \leftarrow \mathsf{F.KeyGen}(1^\lambda, \mathsf{F.params})$	
$K[2^x_G] \leftarrow \mathsf{Puncture}(K, 2^x_G)$	
$\forall i,\ \mathsf{patch}(K,(i,x_i)) \leftarrow \mathsf{PatchGen}(K,i,x_i)$	$(\{\mathsf{patch}_i\}_{i\in[N]}, \mathsf{o.state}) \leftarrow \mathsf{OPatchGen}(1^\lambda, \mathsf{F.params})$
	$K[2^x_G] \leftarrow \mathsf{OPuncture}(1^\lambda, \mathsf{o.state}, x)$
Return $(K[2^x_G], \{\mathsf{patch}(K,(i,x_i))\}_{i\in[N]})$	Return $(K[2^x_G], \{\mathsf{patch}_i\}_{i\in[N]})$

- **[Succinct patches.]** There exists a polynomial P such that, for any λ, any polynomials G, N in λ, polynomial $n = n(\lambda, G, N)$, $K \in \mathcal{K}$, the size of a patch $|\mathsf{patch}(K,(i,x_i))|$ is bounded by $P(\lambda, G, N)$ (i.e., the size of the patches can be bounded above by a single polynomial P for all polynomials $n = n(\lambda, G, N)$).

- **[Pseudorandom at punctured points.]** For every PPT adversary $\mathcal{A}$, for every polynomials n, G, N (in the security parameter), $\mathsf{Adv}_{F,\mathcal{A}}(\lambda)$ is negligible in λ, where,

$$\mathsf{Adv}_{F,\mathcal{A}}(\lambda) := |\Pr[\textsc{real}^{\mathsf{PRF}}_{F,\mathcal{A}}(\lambda) \to 1] - \Pr[\textsc{rand}^{\mathsf{PRF}}_{F,\mathcal{A}}(\lambda) \to 1]|$$

and for $\textsc{expt} \in \{\textsc{real}, \textsc{rand}\}$, $\textsc{expt}^{\mathsf{PRF}}_{F,\mathcal{A}}$ is defined as follows.

1. $\mathsf{F.params} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$
2. $\tilde{x} \leftarrow \mathcal{A}(\mathsf{F.params})$, where, $\tilde{x} \in (\{0,1\}^n)^N$ and $\tilde{x} = (\tilde{x}_1, \ldots, \tilde{x}_N)$
3. $K \leftarrow \mathsf{F.KeyGen}(1^\lambda, \mathsf{F.params})$, $K[2^{\tilde{x}}_G] \leftarrow \mathsf{Puncture}(K, 2^{\tilde{x}}_G)$
4. $b \leftarrow \mathcal{A}^{\mathcal{O}}(K[2^{\tilde{x}}_G])$, where, $\mathcal{O}$ takes three kinds of queries and responds to them as follows:
   - On query ($\textsc{patch-at}\ i$), where $i \in [N]$, respond via $\mathsf{patch}(K,(i,\tilde{x}_i))$;
   - On query ($\textsc{eval-at}\ \mathcal{S}$), where $\mathcal{S} \subseteq [N]$ and $|\mathcal{S}| \leq G$, respond via $F(K,(\mathcal{S},(\tilde{x}_i)_{i\in\mathcal{S}}))$;
   - On query ($\textsc{chal-at}\ \mathsf{ChQ}$), where $\mathsf{ChQ} \subseteq [N]$ and $|\mathsf{ChQ}| \leq G$, respond via

$$\begin{cases} F(K,(\mathsf{ChQ},(\tilde{x}_i)_{i\in\mathsf{ChQ}})) & \text{if } \textsc{expt} = \textsc{real} \\ y & \text{if } \textsc{expt} = \textsc{rand} \end{cases}$$

   where y is a random element from the co-domain, $\{0,1\}^m$, of F.

   By the time $\mathcal{A}$ outputs b, let Q_1 be the set of i for which $\mathcal{A}$ makes $\textsc{patch-at}$ queries and Q_2 be the set of $\mathcal{S}$ for which $\mathcal{A}$ makes $\textsc{eval-at}$ queries. We require that $\mathsf{ChQ} \subseteq [N] \setminus Q_1$ and $\mathsf{ChQ} \notin Q_2$.

4 Construction With Setup

In this section, we present our first construction of adaptively secure multiparty non-interactive key exchange with setup. We shall begin by listing our ingredients. Then we shall set the parameters of our construction followed by the description of our construction. It is instructive to observe that while we set our parameters, we crucially rely on succinctness of patches so as not to run into circularity of setting the parameters. More specifically, the size of patches being independent of the block-size n is crucial since n will be set as some polynomial in the size of patches, as will be evident shortly.

Ingredients.

– **CPA-secure encryption scheme.** Let $\Sigma = (\mathsf{E.Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure encryption scheme.

– **Obliviously-patchable puncturable PRF.** Let F, associated with algorithms $(\mathsf{F.ParamGen}, \mathsf{F.KeyGen}, \mathsf{Puncture}, \mathsf{PatchGen}, \mathsf{Eval}, \mathsf{OPatchGen}, \mathsf{OPuncture})$, be an obliviously-patchable puncturable PRF.

– **Hybrid trapdoor commitment scheme.** Let $\mathsf{HCOM} = (\mathsf{HGen}, \mathsf{HTGen}, \mathsf{HCommit}, \mathsf{HTCommit}, \mathsf{HTDec}, \mathsf{HVer})$ be a hybrid trapdoor commitment scheme.

– **Indistinguishability obfuscator.** Let $\mathsf{i}\mathcal{O}$ be an indistinguishability obfuscator for all $P/poly$ circuits.

Setting up parameters. Given the following parameters:

λ**:** security parameter,

N**:** total number of parties,

G**:** maximum number of parties that can derive a common secret key,

we define the following parameters.

Patch size ℓ**:** Let $\ell = \ell(\lambda, G, N)$ be the number of bits required to represent any patch generated with parameters (λ, n, G, N), where, n is any polynomial in λ, G, N.

Ciphertext size ℓ'**:** The message space of the encryption scheme Σ is $\{0,1\}^{\ell}$. Let the corresponding ciphertext space be a subset of $\{0,1\}^{\ell'}$.

Commitment size n**:** The message space of the hybrid trapdoor commitment scheme HCOM is $\{0,1\}^{\ell'}$. Let the size of any commitment be n.

Block size n**:** For the obliviously-patchable puncturable PRF F, the block-sizes are set to be n.

Number of blocks N**:** For the obliviously-patchable puncturable PRF F, the number of blocks is set to be N.

CONSTRUCTION **1.** The protocol $\mathsf{aNIKE} = (\mathsf{Setup}, \mathsf{Publish}, \mathsf{KeyDerive})$ is described as follows.

$\mathsf{Setup}(\lambda, N, G)$.

- For λ, let ℓ be the size of patches as defined above. Run the parameter-generation algorithms of the PRF, HCOM and Σ: let $\mathsf{F.params} \leftarrow \mathsf{F.ParamGen}(1^{\lambda}, n, G, N)$, $\mathrm{crs} \leftarrow \mathsf{HGen}(1^{\lambda})$ and $(\mathrm{pk}, \cdot) \leftarrow \mathsf{E.Gen}(1^{\lambda})$.
- Choose $K \leftarrow \mathsf{F.KeyGen}(1^{\lambda}, \mathsf{F.params})$.
- For a relation $\mathrm{R}^{\mathsf{HCOM}}$ defined later in Equation (2), build the key-derivation program $\mathsf{KD\text{-}Prog}$ in Figure 1, padded to the appropriate length[2]. Output $\widetilde{\mathsf{KD\text{-}Prog}} \leftarrow \mathsf{i}\mathcal{O}(\mathsf{KD\text{-}Prog})$ and $\mathsf{F.params}$, crs, pk as the public parameters params.

[2]To prove security, we will replace $\mathsf{KD\text{-}Prog}$ with the obfuscation of other programs which may be of size larger than $\mathsf{KD\text{-}Prog}$. In order to be able to employ the security property of the indistinguishability obfuscation, all the programs must be of the same size.

$\mathsf{Publish}(\mathsf{params}, \lambda, i)$. Party P_i computes its public and secret values as follows. Choose a random element $\mathrm{pt}_i \leftarrow \{0,1\}^\ell$ and encrypt it with pk as $\mathrm{ct}_i \leftarrow \mathsf{Enc}(\mathrm{pk}, \mathrm{pt}_i)$. Commit to ct_i: $\mathrm{com}_i = \mathsf{HCommit}(\mathrm{crs}, \mathrm{ct}_i; \mathrm{open}_i)$ for uniformly chosen random coins open_i. Publish i as the ID and com_i as pv_i. Save $(\mathrm{ct}_i, \mathrm{open}_i)$ as the secret value sv_i.

We shall denote the public-value and secret-value relation as $\mathrm{R}^{\mathsf{HCOM}}$; more precisely, parsing $\mathrm{pv} = \mathrm{com}$ and $\mathrm{sv} = (\mathrm{ct}, \mathrm{open})$,

$$\boxed{\mathrm{R}^{\mathsf{HCOM}}(\mathrm{pv}, \mathrm{sv}) = 1, \text{ if and only if } \mathsf{HVer}(\mathrm{crs}, \mathrm{pv}, \mathrm{sv}) = 1} \tag{2}$$

$\mathsf{KeyDerive}((\widetilde{\mathsf{KD\text{-}Prog}}, \mathsf{F.params}, \mathrm{crs}, \mathrm{pk}), \mathcal{S}, \{(i, \mathrm{pv}_i)\}_{i\in\mathcal{S}}, j, \mathrm{sv}_j)$: Run $\widetilde{\mathsf{KD\text{-}Prog}}$ on input $(\mathcal{S}, \{(i, \mathrm{pv}_i)\}_{i\in\mathcal{S}}, j, \mathrm{sv}_j)$. The resultant value is defined to be the '*common key*' derived by parties in $\mathcal{S}$.

KD-Prog

Constants:

- $\mathsf{F.params}, \mathrm{crs}, \mathrm{pk}$
- K

Input: $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, j, \mathrm{sv}_j)$, where $\mathcal{S} \subseteq [N]$, $|\mathcal{S}| \leq G$.
Procedure:

1. Check whether $j \in \mathcal{S}$; if not, output $\perp$. Then, check whether $(\mathrm{pv}_j, \mathrm{sv}_j) \in \mathrm{R}^{\mathsf{HCOM}}$ (defined in Equation (2)); if not, output $\perp$. Otherwise, proceed as follows.
2. Compute and output $F(K, (\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}))$.

Figure 1: Key-derivation Program

5 Structure Of Security Proof Of Construction With Setup

In this Section, we put together the many subtleties discussed in Section 1.1 and present the resulting proof structure at a high level.

Proof structure. Assume for contradiction that there exists an adversary $\mathcal{A}$ that breaks the security of our protocol with a non-negligible advantage. Then, basing on security of PKE scheme, hybrid trapdoor commitments, and $\mathsf{i}\mathcal{O}$, we shall show that there exists an adversary $\mathcal{R}$ (namely, a reduction) that breaks the security of our obliviously-patchable puncturable PRF. We shall do so through the following carefully designed sequence of hybrids that we outline below. We shall begin with the original security game, specify the modifications for every hop of a hybrid, and finally arrive at a hybrid that works as our reduction.

Hyb_0. This experiment is the same as the original security experiment.

Hyb_1. The modification we introduce in this hybrid is the following. Instead of computing a public-value secret-value pair $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$ only when an adversary requests to register an honest party, the challenger computes $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$ for all $i \in [N]$, where N is the total number of parties. Furthermore, it hardcodes the set of all thus generated public values in the obfuscated program. (Use the pair only when the adversary requests to register an honest party P_i, and ignore the pair, otherwise).

Note that including constants to a program does not disturb its input/output functionality. Thus, by applying the security of $\mathsf{i}\mathcal{O}$, we have that $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_1$.

Hyb_2. Puncture the PRF key K at all possible inputs that correspond to subsets of honestly registered parties. Generate patches for the puncture. Although we have punctured at super-polynomially many points, note that our primitive allows these patches to be represented in a succinct manner as $(\mathsf{patch}_1, \ldots, \mathsf{patch}_N)$. Replace K in the program with the punctured key and its patches. (Recall that the punctured key and the patches are such that they satisfy the following. Given just the punctured key, one can compute the PRF on any $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}})$ where there exists $j \in \mathcal{S}$ such that $\mathrm{pv}_j \neq \widetilde{\mathrm{pv}}_j$. On the other hand, given the punctured key *and* a patch patch_j, then, one can easily compute the PRF on any input $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}})$ where $\mathrm{pv}_j = \widetilde{\mathrm{pv}}_j$.)

Note that the obfuscated program still computes the same PRF. Hence, its input/output functionality still remains unaltered. Thus, by applying the security of $\mathsf{i}\mathcal{O}$, we have that $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_2$.

Hyb_3. Begin by obliviously generating the patches and then generate the public values $\{\widetilde{\mathrm{pv}}_i\}_{i\in[N]}$.

From the property of oblivious patchability of F, the change in the sequence of sampling the patches still results in identical joint distributions. Hence, $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_3$.

Hyb_4. Towards generating the public values $\{\widetilde{\mathrm{pv}}_i\}_{i\in[N]}$, recall that plaintexts were sampled uniformly at random in the previous hybrids. The modification here is to set $\mathrm{pt}_i \leftarrow \mathsf{patch}_i$.

From the CPA security of the encryption scheme, $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_4$.

Hyb_5. Hardcode the secret key sk of the public key into the obfuscated program. Also, remove the patches that were hardcoded in the program in the previous hybrid. Within the program, whenever a patch is needed to evaluate the PRF, parse the secret value in the input to contain a ciphertext ct_j. Decrypt it with sk to obtain the required patch patch_j.

We shall argue that this modification has not altered the input/output functionality of the obfuscated program. Note that the public value $\widetilde{\mathrm{pv}}_i$ is computed as a commitment to a secret. Also recall that crs used to commit is in the statistically-binding mode. Thus, there exists only one opening to this commitment. Since the secret value needs to be fed into the program for the program to evaluate the PRF, we have that whenever the program needs to evaluate the PRF, it can recover a patch (from the secret value) and compute the output. Thus, applying the security of $\mathsf{i}\mathcal{O}$, $\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_5$.

Hyb_6. Switch the commitment crs to being sampled in the equivocable mode.

Since the two modes of the hybrid trapdoor commitment scheme are guaranteed to be indistinguishable, $\mathsf{Hyb}_5 \approx_c \mathsf{Hyb}_6$.

Hyb_7. Generate the public values $\widetilde{\mathrm{pv}}_i$ that are commitments as equivocable commitments. Later, when an adversary requests for the secret value, open them to the required value.

Again applying the security of the hybrid trapdoor commitment scheme, $\mathsf{Hyb}_6 \approx_c \mathsf{Hyb}_7$.

Hyb_8. Switch from obliviously generating the patches to generating them together with puncturing. As a sanity check for the modified sequence being well-defined: Note that patches are needed by the challenger not during the computation of the public values (anymore), since public values are now just equivocable commitments. Patches are generated only when the adversary requests for revealing secret values/common keys, which is anyway post publishing the setup.

We simply have gone back on the modification introduced while transitioning from Hyb_2 to Hyb_3. On the similar lines as before, we have that $\mathsf{Hyb}_7 \equiv \mathsf{Hyb}_8$.

Hyb_9. As mentioned in the previous hybrid, patches are needed only upon adversary's queries for revealing secret values/common keys. Observe that all the patches are anyway computed and used only when needed in the previous hybrid. The modification here is to compute them only if and when they are required.

Clearly, $\mathsf{Hyb}_8 \equiv \mathsf{Hyb}_9$.

Hyb_{10}. The modification here is only syntactic and laid out as computing the responses to the adversary's queries by evaluating the PRF directly.

Note that the final hybrid Hyb_{10} can be recast as a reduction who behaves just like the challenger in the hybrid, but obtains the PRF outputs through its own challenger.

This completes the structure of our proof.

6 Construction Without Any Setup

In this section, we present our construction of adaptively secure multiparty non-interactive key exchange without setup assumption. The only additional tool we need to remove setup is PKE scheme with pseudorandom ciphertexts.

Ingredients.

– **CPA-secure encryption scheme with pseudorandom ciphertexts.** Let $\Sigma = (\mathsf{E.Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure encryption scheme with pseudorandom ciphertexts (See Definition 5).

– **Obliviously-patchable puncturable PRF.** Let F, associated with algorithms ($\mathsf{F.ParamGen}$, $\mathsf{F.KeyGen}$, $\mathsf{Puncture}$, $\mathsf{PatchGen}$, Eval, $\mathsf{OPatchGen}$, $\mathsf{OPuncture}$), be an obliviously-patchable puncturable PRF.

– **Hybrid trapdoor commitment scheme.** Let $\mathsf{HCOM} = (\mathsf{HGen}, \mathsf{HTGen}, \mathsf{HCommit}, \mathsf{HTCommit}, \mathsf{HTDec}, \mathsf{HVer})$ be a hybrid trapdoor commitment scheme.

– **Indistinguishability obfuscator.** Let $i\mathcal{O}$ be an indistinguishability obfuscator for all $P/poly$ circuits.

Setting up parameters. Given the following parameters:

λ**:** security parameter,

N**:** total number of parties,

G**:** maximum number of parties that can derive a common secret key,

we define the following parameters.

Patch size ℓ**:** Let $\ell = \ell(\lambda, G, N)$ be the number of bits required to represent any patch generated with parameters (λ, n, G, N), where, n is any polynomial in λ, G, N.

Ciphertext size ℓ'**:** The message space of the encryption scheme Σ is $\{0,1\}^{\ell}$. Let the corresponding size of ciphertexts ℓ'.

CRS size crsLen **and commitment size** comLen**:** The message space of the hybrid trapdoor commitment scheme HCOM is $\{0,1\}^{\ell'}$. Let the size of the CRS (from algorithm HGen or HTGen) be crsLen. Let the size of commitments be comLen = comLen(λ).

n**:** Define

$$n := \text{crsLen} + N \cdot \text{comLen}.$$

Construction **2.** The scheme $\mathsf{aNIKE\text{-}noSetup} = (\mathsf{Setup}, \mathsf{Publish}, \mathsf{KeyDerive})$ is described as follows.

$\mathsf{Setup}(\lambda, N, G)$.

- Simply output $\mathsf{params} = (\lambda, N, G)$.

$\mathsf{Publish}(\mathsf{params}, \lambda, \hat{i})$**.** Party $P_{\hat{i}}$ computes its public value $\text{pv}^{(\hat{i})}$ and secret value $\text{sv}^{(\hat{i})}$, where, $\text{pv}^{(\hat{i})} = (x^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$, as follows.
Computing $(x^{(\hat{i})}, \text{sv}^{(\hat{i})})$**.**

- Run the statistically-binding parameter-generation algorithm of HCOM: $\text{crs}^{(\hat{i})} \leftarrow \mathsf{HGen}(1^{\lambda})$. Sample uniformly at random $\text{ct}_1^{(\hat{i})}, \ldots, \text{ct}_N^{(\hat{i})} \leftarrow \{0,1\}^{\ell'}$. Commit to $\text{ct}_j^{(\hat{i})}$: $\text{com}_j^{(\hat{i})} = \mathsf{HCommit}(\text{crs}^{(\hat{i})}, \text{ct}_j^{(\hat{i})}; \text{open}_j^{(\hat{i})})$ using uniformly chosen random coins $\text{open}_j^{(\hat{i})}$.

Define

$$x^{(\hat{i})} := (\text{crs}^{(\hat{i})}, \text{com}_1^{(\hat{i})}, \ldots, \text{com}_N^{(\hat{i})})$$

and

$$\text{sv}^{(\hat{i})} := ((\text{ct}_1^{(\hat{i})}, \text{open}_1^{(\hat{i})}), \ldots, (\text{ct}_N^{(\hat{i})}, \text{open}_N^{(\hat{i})})).$$

Computing the program $\mathsf{ioP}^{(\hat{i})}$**.**

- Run the parameter-generation algorithms of the PRF: $\mathsf{F.params}^{(\hat{i})} \leftarrow \mathsf{F.ParamGen}(1^{\lambda}, n, G, N)$.
- Choose $K^{(\hat{i})} \leftarrow \mathsf{F.KeyGen}(1^{\lambda}, \mathsf{F.params}^{(\hat{i})})$.

- For $K^{(\hat{i})}$, build the key-derivation program $\mathsf{KD\text{-}Prog}^{(\hat{i})}$ in Figure 2, padded to the appropriate length[3]. (The function $\mathrm{R}^{\mathsf{HCOM}}_{(\cdot)}$ that $\mathsf{KD\text{-}Prog}^{(\hat{i})}$ uses is defined shortly.) Compute $\mathsf{ioP}^{(\hat{i})} \leftarrow \mathsf{i}\mathcal{O}(\mathsf{KD\text{-}Prog}^{(\hat{i})})$.

Let

$$\mathrm{pv}^{(\hat{i})} := (x^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$$

Publish $\hat{i}$ as the ID and also publish $\mathrm{pv}^{(\hat{i})}$. Save the secret value $\mathrm{sv}^{(\hat{i})}$.

$\mathsf{KeyDerive}(\mathcal{S}, (\hat{i}, \mathrm{pv}^{(\hat{i})})_{\hat{i}\in\mathcal{S}}, \hat{j}, \mathrm{sv}^{(\hat{j})})$:

- If $\hat{j} \notin \mathcal{S}$, then output $\perp$. Otherwise, proceed as follows.
- For every $\hat{i} \in \mathcal{S}$, parse $\mathrm{pv}^{(\hat{i})} = (x^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$ and $x^{(\hat{i})} = (\mathrm{crs}^{(\hat{i})}, \mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})})$. Also parse $\mathrm{sv}^{(\hat{j})} = (\mathrm{ct}_1^{(\hat{j})}, \mathrm{open}_1^{(\hat{j})}), \ldots, (\mathrm{ct}_N^{(\hat{j})}, \mathrm{open}_N^{(\hat{j})})$. Define

$$\boxed{\mathrm{sv}^{(\hat{j})}_{\hat{i}^*} := (\mathrm{ct}^{(\hat{j})}_{\hat{i}^*}, \mathrm{open}^{(\hat{j})}_{\hat{i}^*})} \tag{3}$$

  Output $\mathsf{ioP}^{(\hat{i}^*)}(\mathcal{S}, \{(\hat{i}, x^{(\hat{i})})\}_{\hat{i}\in\mathcal{S}}, \hat{j}, \mathrm{sv}^{(\hat{j})}_{\hat{i}^*})$.
- Let us define the following relation, $\mathrm{R}^{\mathsf{HCOM}}_{(\cdot)}$: For x, y parsed as $x = (\mathrm{crs}, \mathrm{com}_1, \ldots, \mathrm{com}_N)$ and $y = (\mathrm{ct}_{\hat{i}^*}, \mathrm{open}_{\hat{i}^*})$,

$$\boxed{\mathrm{R}^{\mathsf{HCOM}}_{\mathcal{S}}(x, y) = 1, \text{ if and only if } \begin{array}{c} \hat{i^*} \text{ is the smallest element in } \mathcal{S} \\ \& \\ \mathsf{HVer}(\mathrm{crs}, \mathrm{com}_{\hat{i}^*}, \mathrm{ct}_{\hat{i}^*}, \mathrm{open}_{\hat{i}^*}) = 1 \end{array}} \tag{4}$$

$\mathsf{KD\text{-}Prog}^{(\hat{i})}$

Constants:

- $\mathsf{F.params}^{(\hat{i})}$
- $K^{(\hat{i})}$

Input: $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{j}, \mathrm{sv}^{(\hat{j})}_{\hat{i}})$, where $\mathcal{S} \subseteq [N]$, $|\mathcal{S}| \leq G$.
Procedure:

1. Check whether $\hat{j} \in \mathcal{S}$; if not, output $\perp$. Then, check whether $(x^{(\hat{j})}, \mathrm{sv}^{(\hat{j})}_{\hat{i}}) \in \mathrm{R}^{\mathsf{HCOM}}_{\mathcal{S}}$ (defined in Equation (4)); if not, output $\perp$. Otherwise, proceed as follows.
2. Compute and output $F(K^{(\hat{i})}, (\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}))$.

Figure 2: Key-derivation Program

[3]To prove security, we will replace $\mathsf{KD\text{-}Prog}^{(\hat{i})}$ with the obfuscation of another program $\mathsf{KD\text{-}Prog}^{(\hat{i})}$, which may be larger than $\mathsf{KD\text{-}Prog}^{(\hat{i})}$. In order to be able to employ the security property of the differing-input obfuscation, both programs must be of the same size.

7 Structure Of Security Proof Of Construction Without Any Setup

In this Section, we present a high-level structure of the proof whose subtleties were explained in Section 1.1. The structure is similar to that of the proof for the construction with setup. The additional subtlety, as we recall from Section 1.1, is solved by using an encryption scheme with pseudorandom ciphertexts. Furthermore, we shall see in the proof clearly how the idea of having N commitments in the public value of each party suffices to get around the core subtlety of removing the setup – the adversary designing obfuscated program in a malicious way that might leak information about its inputs (cf. Section 1.1).

Proof structure. Assume for contradiction that there exists an adversary $\mathcal{A}$ that breaks the security of our protocol with a non-negligible advantage. Then, basing on security of PKE scheme, hybrid trapdoor commitments, and $\mathsf{i}\mathcal{O}$, we shall show that there exists an adversary $\mathcal{R}$ (namely, a reduction) that breaks the security of our obliviously-patchable puncturable PRF. We shall do so through the following carefully designed sequence of hybrids that we outline below. We shall begin with the original security game, specify the modifications for every hop of a hybrid, and finally arrive at a hybrid that works as our reduction.

Hyb_0. This experiment is the same as the original security experiment.

Hyb_1. Recall that the public value of each party $P_{\hat{i}}$ consists of two parts – one, an obfuscated program $\mathsf{ioP}^{(\hat{i})}$, and two, a commitment CRS together with a tuple of N commitments, denoted by $\widetilde{x}^{(\hat{i})}$. Also recall that in the previous hybrid, the sequence of computation of these values is that the challenger iterates from $\hat{i} = 1$ through N, and for each $\hat{i}$, it computes the public value $(\mathsf{ioP}^{(\hat{i})}, \widetilde{x}^{(\hat{i})})$. The modification we introduce in this hybrid is the following. Firstly, the challenger iterates from $\hat{i} = 1$ through N, and computes only $\widetilde{x}^{(\hat{i})}$ (together with $\widetilde{\mathrm{sv}}_{\hat{j}}$). Then, after completing this iteration, it again iterates from $\hat{i} = 1$ through N now computing $\mathsf{ioP}^{(\hat{i})}$. A further modification is that in every $\mathsf{ioP}^{(\hat{i})}$, the tuple $\{\widetilde{x}^{(\hat{k})}\}_{\hat{k}\in[N]}$ is hardcoded as a constant. Finally, the thus generated public-value secret-value pair $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$ is used only if the adversary requests to register an honest party $P_{\hat{i}}$; otherwise, the pair is ignored.

Note that including constants to a program does not disturb its input/output functionality. Thus, by applying the security of $\mathsf{i}\mathcal{O}$, we have that $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_1$.

Hyb_2. Let $K^{(\hat{i})}$ be the PRF key used in $\mathsf{ioP}^{(\hat{i})}$ by the challenger in the previous hybrid. In this hybrid, the modification is as follows. Instead of hardcoding the key as is, the key is punctured at all possible inputs that correspond to subsets of honestly registered parties; furthermore, complementing the puncture, patches $\mathsf{patch}_1^{(\hat{i})}, \ldots, \mathsf{patch}_N^{(\hat{i})}$ are generated; the program is hardcoded with the punctured key and the patches.

Note that the obfuscated program still computes the same PRF. Hence, its input/output functionality still remains unaltered. Thus, by applying the security of $\mathsf{i}\mathcal{O}$, we have that $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_2$.

Hyb_3. The modification again is in the sequence of computation. Recall that in the previous hybrid, we sampled the PRF key for any party only after computing all $\widetilde{x}^{(\hat{1})}, \ldots, \widetilde{x}^{(\hat{N})}$. Now the modification is the following. The challenger begins by obliviously sampling the patches for PRF key for every $\hat{i}$. Then, $\widetilde{x}^{(\hat{1})}, \ldots, \widetilde{x}^{(\hat{N})}$ are computed as before. Next, $\mathsf{ioP}^{(\hat{1})}, \ldots, \mathsf{ioP}^{(\hat{N})}$ are computed by sampling the punctured PRF keys consistent with the already sampled patches.

From the property of oblivious patchability of F, the change in the sequence of sampling the patches still results in identical joint distributions. Hence, $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_3$.

Hyb_4. The modification is in the distribution of the values that are committed to in generating the commitments in $\widetilde{x}^{(\hat{i})}$. Recall that in the previous hybrid, $\mathrm{com}_{\hat{j}}^{(\hat{i})}$, that is a part of $\widetilde{x}^{(\hat{i})}$, is a commitment to the a random value $\mathrm{ct}_{\hat{j}}^{(\hat{i})} \leftarrow \{0,1\}^{\ell'}$. The modification now is the following: for every $\hat{j} \in [N]$, we sample a public key $\mathrm{pk}^{(\hat{j})}$. Then the patch $\mathsf{patch}_{\hat{i}}^{(\hat{j})}$ (for the key corresponding to the program of party $P_{\hat{j}}$) is encrypted with $\mathrm{pk}^{(\hat{j})}$ and the resulting ciphertext is set to be $\mathrm{ct}_{\hat{j}}^{(\hat{i})}$. This value is then committed and set as $\mathrm{com}_{\hat{j}}^{(\hat{i})}$ as a part of $\widetilde{x}^{(\hat{i})}$.

From the CPA security of the encryption scheme with pseudorandom ciphertexts, $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_4$.

Hyb_5. Hardcode the secret key $\mathrm{sk}^{(\hat{j})}$ of the public key into the obfuscated program $\mathsf{ioP}^{(\hat{j})}$. Also, remove the patches that were hardcoded in the program $\mathsf{ioP}^{(\hat{j})}$ in the previous hybrid. Within the program, whenever a patch is needed to evaluate the PRF, parse the secret value in the input to contain a ciphertext $\mathrm{ct}_{\hat{j}}^{(\hat{i})}$. Decrypt it with $\mathrm{sk}^{(\hat{j})}$ to obtain the required patch $\mathsf{patch}_{\hat{i}}^{(\hat{j})}$.

We shall argue that this modification has not altered the input/output functionality of the obfuscated program. Note that the public value $\widetilde{x}^{(\hat{i})}$ is computed as a set of commitments to some values. Also recall that the $\mathrm{crs}^{(\hat{i})}$ used to commit is in the statistically-binding mode. Thus, there exists only one opening to any of these commitments. Since the secret value needs to be fed into the program for the program to evaluate the PRF, we have that whenever the program needs to evaluate the PRF, it can recover a patch (from the secret value) and compute the output. Thus, applying the security of $\mathsf{i}\mathcal{O}$, $\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_5$.

Hyb_6. For every $\hat{i} \in [N]$, switch the commitment $\mathrm{crs}^{(\hat{i})}$ to being sampled in the equivocable mode.

Since the two modes of the hybrid trapdoor commitment scheme are guaranteed to be indistinguishable, $\mathsf{Hyb}_5 \approx_c \mathsf{Hyb}_6$.

Hyb_7. For every $\hat{i} \in [N]$, generate the commitments belonging to $\widetilde{x}^{(\hat{i})}$ as equivocable commitments. Later, when an adversary requests for the secret value, open them to the required value.

Again applying the security of the hybrid trapdoor commitment scheme, $\mathsf{Hyb}_6 \approx_c \mathsf{Hyb}_7$.

Hyb_8. Switch from obliviously generating the patches to generating them together with puncturing. As a sanity check for the modified sequence being well-defined: Note that patches are needed by the challenger not during the computation of the public values (anymore), since public values now just consist of equivocable commitments. Patches are generated only when the adversary requests for revealing secret values/common keys, which is anyway post publishing the setup.

We simply have gone back on the modification introduced while transitioning from Hyb_2 to Hyb_3. On the similar lines as before, we have that $\mathsf{Hyb}_7 \equiv \mathsf{Hyb}_8$.

Hyb_9. As mentioned in the previous hybrid, patches are needed only upon adversary's queries for revealing secret values/common keys. Observe that all the patches are anyway computed and used only when needed in the previous hybrid. The modification here is to compute them only if and when they are required.

Clearly, $\mathsf{Hyb}_8 \equiv \mathsf{Hyb}_9$.

Hyb_{10}. The modification here is only syntactic and laid out as computing the responses to the adversary's queries by evaluating the PRF directly.

Note that the final hybrid Hyb_{10} can be recast as a reduction who behaves just like the challenger in the hybrid, but obtains the PRF outputs through its own challenger.

This completes the structure of our proof.

8 Interactive Cryptographic Assumption is Necessary

In this Section, we shall ascertain that an interactive assumption would be necessary in natural settings to achieve adaptive NIKE *even with setup*. We shall establish this necessity via a meta-reduction technique, a common technique employed in results that rule out certain classes of reductions.

While this is the main technique in showing that interactive cryptographic assumptions are necessary, we run into certain subtleties. To see one such subtlety, we review how the meta-reduction technique is generally employed [Cor02, FJS14, HJK12, KK12]. For concreteness, let us consider [KK12], who rule out reductions from breaking security of a signature scheme to a hard problem with security loss less than some specific value dependent on the number of queries by the adersary. Here, for a proof by contradiction, they suppose that there exists such a reduction $\mathcal{R}$. Then, they construct another adversary – called meta-reduction $\mathcal{B}$ – that given access to that reduction breaks that underlying hard problem. The proof approach would be the following: Recall that we are assured that if $\mathcal{R}$ is given access to an adversary $\mathcal{A}$ that breaks unforgeability of the signature scheme, then $\mathcal{R}$ can break the underlying hard problem. To this end, $\mathcal{B}$ emulates an adversary $\mathcal{A}$ that breaks the unforgeability of the signature scheme. It does so simply by rewinding $\mathcal{R}$ *for the same verification* key but for a different message to be signed. Then it uses the signature on this message provided by the reduction himself to answer the challenge query in the main thread. Intuitively, this is a good simulation since the rewinding is performed for the same verification key; this implies that the valid signature in the rewound thread would be valid in the main thread too.

Now let us see how this applies to our setting. Consider the no-setup case. Here, there is no fixed parameters (such as the verification key in [KK12]) which could fix the response of the reduction, so that the value in the rewound thread can be used in the main thread. Thus, it is a critical task to clearly abstract out the properties the reduction needs to satisfy in order for us to employ the meta-reduction technique. In the following, we shall review certain preliminaries, formally specify the required but natural properties of a class of reductions that we rule out, and finally provide our proof.

8.1 Cryptographic Problems

We recall the formalism of cryptographic problems from [FF13].

Definition 10 (Cryptographic Problem)**.** A cryptographic problem Π is characterized by four algorithms, $\Pi = (\mathsf{InsGen}, \mathsf{Orcl}, \mathsf{Vrfy}, \mathsf{Thresh})$:

InsGen: The instance generator InsGen takes as input the security parameter 1^λ and outputs a problem instance $\mathtt{ins}$. The set of all possible instances output by InsGen is denoted by $\mathbf{I}$.

Orcl: The computationally unbounded and stateful oracle algorithm Orcl takes as input a query $\mathtt{q} \in \{0,1\}^*$ and outputs a response $\mathtt{resp} \in \{0,1\}^*$ or a special symbol $\perp$ indicating that $\mathtt{q}$ was not a valid query.

Vrfy: The deterministic verification algorithm Vrfy takes as input a problem instance $\mathtt{ins} \in \mathbf{I}$ and a candidate solution $\mathtt{sol} \in \mathbf{S}$. The algorithm outputs $b \in \{0,1\}$. We say $\mathtt{ins}$ is a valid solution to instance $\mathtt{ins}$ if and only if $b = 1$.

Thresh: The efficient threshold algorithm Thresh takes as input a problem instance $\mathtt{ins}$ and outputs some $\mathtt{sol}$. The threshold algorithm is a special adversary and as such also has access to Orcl.

We note that the algorithms $\mathsf{InsGen}, \mathsf{Orcl}, \mathsf{Vrfy}$ potentially have access to shared state that persists for the duration of an experiment.

Definition 11 (Hard Cryptographic Problem)**.** For a cryptographic problem $\Pi = (\mathsf{InsGen}, \mathsf{Orcl}, \mathsf{Vrfy}, \mathsf{Thresh})$ and an adversary $\mathcal{A}$ we define the following experiment:

$$\boxed{\begin{array}{c}\textbf{Experiment } \textsc{expt}^{\Pi}_{\mathcal{A}}(\lambda)\text{:} \\ \mathtt{ins} \leftarrow \mathsf{InsGen}(1^\lambda) \\ \mathtt{sol} \leftarrow \mathcal{A}^{\mathsf{Orcl}} \\ b \leftarrow \mathsf{Vrfy}(\mathtt{ins}, \mathtt{sol}) \\ \text{Output } b.\end{array}}$$

The problem Π is said to be hard if and only if for all probabilistic polynomial-time algorithms $\mathcal{A}$ the following advantage function is negligible in the security parameter λ

$$\mathsf{Adv}^{\Pi}_{\mathcal{A}}(\lambda) = \Pr[\textsc{expt}^{\Pi}_{\mathcal{A}}(\lambda) \to 1] - \Pr[\textsc{expt}^{\Pi}_{\mathsf{Thresh}}(\lambda) \to 1]$$

where the probability is taken over the random tapes of InsGen and $\mathcal{A}$.

Remark 2 (Monotone Problem)**.** We shall only consider the natural class of problems, namely, monotone problems; namely, the ones in which whether any $\mathtt{sol}$ for a problem instance is verified to be a solution or not does not depend on the number of queries made by $\mathcal{A}$ to the oracle Orcl. More specifically, a problem is said to be monotone if and only if for all instances $\mathtt{ins} \leftarrow \mathsf{InsGen}(1^\lambda)$, all solutions $\mathtt{sol} \in \mathbf{S}$, all $\lambda \in \mathbb{N}$, and all sequences of queries $q_1, \ldots, q_n$, the following holds: If $\mathsf{Vrfy}(\mathtt{ins}, \mathtt{sol}) = 1$ holds after executing the queries $\mathsf{Orcl}(q_1), \ldots, \mathsf{Orcl}(q_n)$ then this already held before $\mathsf{Orcl}(q_1)$ was executed.

We shall only consider monotone problems in this work.

Below, we recall certain relevant aspects of the black-box model.

Black-box reductions. An algorithm $\mathcal{R}$ is said to be a black-box reduction from a problem Π_2 to a problem Π_1 if for any algorithm $\mathcal{A}$ solving Π_1, algorithm $\mathcal{R}^{\mathcal{A}}$ solves Π_2 thanks to an oracle (a.k.a. black-box) access to $\mathcal{A}$. Below, we provide more details about our black-box model. Namely, we recall what is meant by "oracle access" and give a characterization of the classes of algorithms $\mathcal{A}$ we

will consider. In other words, we specify which algorithms are transformed by $\mathcal{R}$ and how $\mathcal{R}$ can interact with them.

Oracle access. A black-box access essentially means that $\mathcal{R}$ is allowed to use $\mathcal{A}$ as a subroutine without taking advantage of its internal structure (code). $\mathcal{R}$ can only provide the inputs to $\mathcal{A}$ and observe the resulting outputs. If $\mathcal{A}$ has access to an oracle, the corresponding queries must be answered by $\mathcal{R}$. Below, we specify what interactions $\mathcal{R}$ can perform with $\mathcal{A}$.

We allow $\mathcal{R}$ to rewind $\mathcal{A}$ with a previously used random tape. Our approach is formalized by restricting the reduction $\mathcal{R}$ to sequentially execute some of the following operations when interacting with $\mathcal{A}$:

- [Launch/Relaunch.] Any previously launched execution of $\mathcal{A}$ is aborted. $\mathcal{R}$ launches a new execution of $\mathcal{A}$ with a fresh random tape on an input of its choice.
- [Rewind.] Any previously launched execution of $\mathcal{A}$ is aborted. $\mathcal{R}$ restarts $\mathcal{A}$ with a previously used random tape and an input of its choice.
- [Stop.] $\mathcal{R}$ definitely stops the interaction with $\mathcal{A}$.

We shall reserve the term "stop" to only interactions of $\mathcal{R}$ with $\mathcal{A}$. For other interactions such as those between $\mathcal{R}$ and the external challenger of Π, we use the term "abort".

Definition 12 (Black-box Reductions)**.** A PPT algorithm $\mathcal{R}$ is said to be a black-box $(\varepsilon_{\mathcal{A}}, \varepsilon_{\mathcal{R}}, \tau_{\mathcal{A}}, \tau_{\mathcal{R}})$-reduction from problem Π_1 to problem Π_2, if given an oracle access to a PPT algorithm $\mathcal{A}$ that solves Π_2 with probability $\varepsilon_{\mathcal{A}}$ and time complexity $\tau_{\mathcal{A}}$, $\mathcal{R}$ solves Π_1 with probability $\varepsilon_{\mathcal{R}}$ and time complexity $\tau_{\mathcal{R}}$.

Admissible Reductions. We shall rule out a class of black-box reductions, that we refer to as 'admissible reductions'. Intuitively, in some respect, we wish to fix the common key derived by secret values that the reduction presents to the adversary when the adversary makes Ext queries. This is so that we can rewind $\mathcal{R}$ and obtain some trapdoor information such that this trapdoor can be used in simulating $\mathcal{A}$ in the main thread; fixing the common key somehow will keep $\mathcal{R}$ from detecting that we have used trapdoor derived from rewinding $\mathcal{R}$. We explain this in detail in the ensuing.

More specifically, an admissible reduction would be such that the parameters and public values (for honest parties) generated by the reduction (information-theoretically) determine the common keys for every subset such that the following holds: For any set of secret values corresponding to the public values, if all the secret values satisfy the correctness property w.r.t. one another, as formalized in Equation 1, then they always derive the unique common key that is determined by the parameters and the public values; let us refer to such secret values as 'valid secret values'. Furthermore, the secret values of honest parties provided by an admissible reduction to the adversary are valid secret values. We shall define the class of admissible reductions more formally in the ensuing. This requirement on the reductions can be relaxed in multiple ways and still have our impossibility result apply. We shall discuss the relaxations later in the Section. However, for a cleaner presentation of the proof, we shall stick to the following class of admissible reductions.

Definition 13 (Admissible Reductions)**.** A reduction $\mathcal{R}$ from an adaptively-secure non-interactive key exchange protocol aNIKE to a computational problem Π is said to be admissible if for every parameters params and public values $\{\widetilde{\mathrm{pv}}_i\}_{i\in[N]}$ (for honest parties) on which it runs an adversary satisfy the following properties.

- For every params and $\{\widetilde{\mathrm{pv}}_i\}_{i\in[N]}$ output by $\mathcal{R}$, there exists a unique common key for every set of honest parties $\mathcal{S}$ such that the following holds: For every set[4] of secret values corresponding to the public values of the honest parties, where, the secret values satisfy correctness w.r.t. one other (cf. Equation 1), the common key derived using these secret values is the unique common key. Such secret values are referred to as '*valid secret values*'.

- For every public value $\widetilde{\mathrm{pv}}_i$ output by the reduction for an ith honest party, the corresponding secret-value $\widetilde{\mathrm{sv}}_i$ revealed by the reduction (upon every query $\mathsf{Ext}(i)$) is a valid secret value.

Theorem 2. Let $\Pi = (\mathsf{InsGen}, \mathsf{Orcl}, \mathsf{Vrfy}, \mathsf{Thresh})$ be a hard cryptographic problem. Let aNIKE be an adaptively-secure $N(\lambda)$-party non-interactive key exchange protocol. If there exists an admissible reduction $\mathcal{R}$ that, for infinitely many $\lambda \in \mathbb{N}$, black-box $(\varepsilon_{\mathcal{A}}, \varepsilon_{\mathcal{R}}, \tau_{\mathcal{A}}, \tau_{\mathcal{R}})$-reduces breaking adaptive security of aNIKE to solving Π such that the total number of relaunches (or rounds) of $\mathcal{A}$ executed by $\mathcal{R}$ is r, the set of queries made by $\mathcal{R}$ to Orcl in an ith round with $\mathcal{A}$ is Q_i, then there exists a PPT machine $\mathcal{B}$ that solves Π with probability $\varepsilon_\Pi := \sum_{\mathtt{ins}\in\mathsf{InsGen}} \varepsilon_{\mathtt{ins}} \Pr[\mathtt{ins}]$, where, $\varepsilon_{\mathtt{ins}} = \varepsilon^{(1)}_{\mathtt{ins}}\eta_{\mathtt{ins}} + \varepsilon^{(2)}_{\mathtt{ins}}(1-\eta_{\mathtt{ins}})\left(1 - \sum_{i=1}^{r} \eta_{\mathtt{ins}}^{(N-G)-|Q_i|}\right)$, $\varepsilon_{\mathcal{R}} = \sum_{\mathtt{ins}\in\mathsf{InsGen}} \left(\varepsilon^{(1)}_{\mathtt{ins}}\eta_{\mathtt{ins}} + \varepsilon^{(2)}_{\mathtt{ins}}(1-\eta_{\mathtt{ins}})\right)\Pr[\mathtt{ins}]$, for some $0 \le \varepsilon^{(1)}_{\mathtt{ins}}, \varepsilon^{(2)}_{\mathtt{ins}}, \eta_{\mathtt{ins}} \le 1$.

Remark 3 (Implication of Theorem 2)**.** Intuitively, Theorem 2 asserts that in order to have a reduction to a hard problem the reduction $\mathcal{R}$ needs to make queries at least of the order of $N - G$ in at least one of the rounds.

Proof of Theorem 2. We shall first provide an intuition for the proof.

Intuition. Let $\mathcal{R}$ be an admissible reduction. Let $\mathcal{A}$ be an adversary that runs in time $\tau_{\mathcal{A}}$ and breaks the adaptive security of aNIKE with probability $\varepsilon_{\mathcal{A}}$. Then we show existence of a meta-reduction $\mathcal{B}$ that given just an oracle access to $\mathcal{R}$ will break the hardness of Π with polynomial probability, if $\varepsilon_{\mathcal{A}}$ is also polynomial (in the security parameter). We shall present this proof of contradiction for a bound $G \in \omega(\log N)$.

We shall first prove the theorem for the case when the reduction runs only a single copy of $\mathcal{A}$ without rewinding. Therein, we shall first prove that before the reduction gives params to the adversary, it needs to have received the problem instance from InsGen of Π. This intuitively follows from the following rationale. Recall that the challenge common key $\mathcal{A}$ is challenged upon is already determined by params and public values. If $\mathcal{R}$ receives the problem instance $\mathtt{ins}$ after giving params and public values to $\mathcal{A}$, then the challenge common key is distributed independently of $\mathtt{ins}$. Thus, intuitively, distinguishing the challenge common key from random should not facilitate the reduction in solving $\mathtt{ins}$.

We shall next treat the case when the reduction launches the adversary just once and does not rewind it, like before, but receives problem instance from InsGen after it provides params and public values to the adversary. In this execution, we shall recognize the 'slots' at which it is 'safe' to rewind $\mathcal{R}$ without risking rewinding the external challenger of problem Π. Then, we shall show that by rewinding at these slots, one can query the reduction for some information that suffices to break the adaptive security of aNIKE in the main thread. In particular, in the rewound thread we can have the adversary $\mathcal{A}$ query $\mathcal{R}$ to reveal a secret value $\widetilde{\mathrm{sv}}_i$ corresponding to the public value $\widetilde{\mathrm{pv}}_i$ for an $i \in \mathsf{ChQ}$, where, ChQ is the set of parties on which the adversary has chosen to be challenged upon

[4]Note that it might be possible that a NIKE protocol is such that there are multiple valid secret values for a public value.

in the main thread. Recall that, since the reduction is admissible, there is a unique common key for ChQ; also, the secret values, for public values of honest parties, revealed by the reduction are valid – i.e., the common keys recovered using these secret values are the ones information-theoretically determined by the parameters and the public values. Hence, without being 'detected' by $\mathcal{R}$ that we are using the value received by $\mathcal{R}$ from the rewound thread in the main thread, we can solve the aNIKE challenge in the main thread. By considering an adversary that queries $\mathcal{R}$ to reveal the secret values of $(N - G) \in (N - \omega(\log(N)))$ public values chosen uniformly random depending on the given parameters and the set of public keys, we can solve Π with probability $\geq \varepsilon_{\Pi}$.

Then, we show how to deal with an admissible reduction that might rewind the adversary. We shall show that by rewinding, the reduction can derive no further advantage. This is because, even when rewound the adversary would query the reduction with the same set of queries as in the thread from which the reduction has rewound $\mathcal{A}$. Thus, rewinding will not help $\mathcal{R}$. Furthermore, in the threads where the meta-reduction rewinds $\mathcal{R}$, the meta-reduction is not required to answer the challenge queries since the trapdoor information that it hopes to get from the rewound thread would most definitely be obtained before it reaches the stage where it needs to answer the challenge query; thus, it can simply abort the interaction in the rewound threads before the challenge phase. A further detailed discussion will appear later in the proof.

Next, we shall deal with the case when $\mathcal{R}$, besides rewinding $\mathcal{A}$ (by running $\mathcal{A}$ with the same random coins but with different inputs), also relaunches $\mathcal{A}$ by running it with different random coins and different inputs. The idea is to simply maintain a list of queries queried by $\mathcal{A}$ in any round for specific random coins. Then, when rewound with the same coins and same parameters and public values, $\mathcal{A}$ makes the same queries as before. If rewound with the same coins but with different parameters or public values, then $\mathcal{A}$ queries with a freshly chosen random set of indices of the public values. When relaunched with distinct coins, we would again query the reduction with a freshly chosen random set of indices of the public values. Observe that our final goal is to build a meta-reduction that can simulate $\mathcal{A}$ to $\mathcal{R}$. Even when the reduction rewinds or relaunches the adversary, the meta-reduction will still be able to simulate the adversary since it would only need to query on the same queries it has already queried for any particular parameters and set of public values. Now, we identify slots that are safe to rewind just as in the previous case of rewinding – we identify such slots for every round like in the case of rewinding. More details would follow later in the proof.

We shall now build upon the above intuition and provide a formal proof.

Formal Proof.

Case 1 – Single round (i.e., no relaunching), no rewinding. We shall begin with the case that the reduction runs $\mathcal{A}$ only once. Herein, we shall consider two cases.

Case 1a – $\mathtt{ins}$ is independent of $\mathsf{params}, \{\widetilde{\mathrm{pv}}_i\}$. Firstly, we shall focus on the following simple case: $\mathcal{R}$ gives the adversary params, and $\{\widetilde{\mathrm{pv}}_i\}_{i \in S}$ for some set $S \subseteq [N]$ before it receives a Π problem instance $\mathtt{ins}$ from the external challenger. Then, it gives the rest of the public values $\widetilde{\mathrm{pv}}_i$. If the set of parties ChQ that the adversary wishes to be challenged upon is such that $\mathsf{ChQ} \not\subseteq S$, then the reduction simply stops interacting with the adversary.

For this case, we shall first build an ideal adversary $\mathcal{A}_{\mathrm{ideal}}$ which breaks the security of aNIKE with probability $\varepsilon_{\mathcal{A}}$. Then, we shall construct a meta-reduction $\mathcal{B}$ that simulates $\mathcal{A}_{\mathrm{ideal}}$ to $\mathcal{R}$. Note that one cannot naïvely follow the traditional way of solving such problem, namely, compute the response to the challenge ahead of time and give it as a succinct "advice" to the adversary. This subtlety is because, since we are in the adaptive setting, the challenge set of parties is not known ahead of time before the reduction gives the adversary the instance $\mathtt{ins}$; moreover, there are

exponentially-many possible challenge sets ChQ of parties. This is where we employ admissibility of the reduction $\mathcal{R}$. The idea would be to compute, not the common key, but a valid secret value for every $\widetilde{\mathrm{pv}}_i$ for $i \in S$, before $\mathtt{ins}$ is given to $\mathcal{R}$. We are assured from the admissibility property that the common key derived using valid secret values is unique. Upon the meta-reduction being given the challenge key, it can then match it with the common key it can compute using the secret values received as advice, and distinguish the actual key from random. Given such an adversary, by the hypothesis of the theorem, the reduction $\mathcal{R}$ solves Π with probability $\varepsilon_{\mathcal{R}}$; hence, so does $\mathcal{B}$. Since the meta-reduction runs in polynomial time, it is valid to regard it as a PPT adversary that solves Π with some probability. Details follow.

Ideal Adversary $\mathcal{A}_{\mathrm{ideal}}$. The ideal adversary $\mathcal{A}_{\mathrm{ideal}}$ that breaks the security of aNIKE with probability $\varepsilon_{\mathcal{A}}$ is defined as follows.

- Upon given params, $\mathcal{A}_{\mathrm{ideal}}$ queries to register N-many honest parties. Let the public values output by $\mathcal{R}$ be $\{\widetilde{\mathrm{pv}}_i\}_{i\in[N]}$.
- $\mathcal{A}_{\mathrm{ideal}}$ chooses a random set of parties $\mathsf{ChQ} \leftarrow 2^{[N]}$, such that $|\mathsf{ChQ}| \leq G$.
- $\mathcal{A}_{\mathrm{ideal}}$ queries ChQ as the challenge set of parties.
- $\mathcal{A}_{\mathrm{ideal}}$ tosses a coin that takes 1 with probability $\varepsilon_{\mathcal{A}}$ and 0 otherwise.
- Upon receiving k^*, if the coin-toss outcome has resulted in a 1, then output 1 if $k^* = \mathsf{KeyDerive}(\mathsf{params}, \mathsf{ChQ}, (\mathrm{pv}_j)_{j\in\mathsf{ChQ}}, i, \mathrm{sv}_i)$ for some arbitrary $i \in \mathsf{ChQ}$ and $\widetilde{\mathrm{sv}}_i$ being a valid secret value, and 0 otherwise. If the coin-toss outcome has resulted in a 0, then output a random bit.

This completes the description of $\mathcal{A}_{\mathrm{ideal}}$. Observe that $\mathcal{A}_{\mathrm{ideal}}$ breaks the security of aNIKE with probability $\varepsilon_{\mathcal{A}}$. Thus, from the hypothesis, the reduction $\mathcal{R}$, when given a black-box access to $\mathcal{A}_{\mathrm{ideal}}$, should solve the problem Π with probability $\varepsilon_{\mathcal{R}}$. We now describe an algorithm $\mathcal{B}$ that interacts with $\mathcal{R}$ and simulates $\mathcal{A}_{\mathrm{ideal}}$ to $\mathcal{R}$.

Ideal Adversary $\mathcal{A}_{\mathrm{ideal}}$. The ideal adversary $\mathcal{A}_{\mathrm{ideal}}$ that breaks the security of aNIKE with probability $\varepsilon_{\mathcal{A}}$ is defined as follows.

Meta-reduction $\mathcal{B}$. $\mathcal{B}$ is described as follows.

- Upon $\mathcal{R}$ invoking the simulated adversary with params, query $\mathcal{R}$ to register N-many honest parties.
- Let $\mathcal{R}$ respond via $\{\widetilde{\mathrm{pv}}_i\}_{i\in S}$ for some set $S \subseteq [N]$, before it requests to receive $\mathtt{ins}$.
- Run in exponential time and compute a valid secret value $\widetilde{\mathrm{sv}}_i$ corresponding to $\widetilde{\mathrm{pv}}_i$ for every $i \in \mathcal{S}$.
- Upon the reduction requesting for the problem instance of Π, relay its request to the external challenger to receive $\mathtt{ins}$.
- Upon giving $\mathtt{ins}$ to $\mathcal{R}$, let $\mathcal{R}$ proceed to present the rest of the public values.
- Choose a random set of parties $\mathsf{ChQ} \leftarrow 2^{[N]}$, such that $|\mathsf{ChQ}| \leq G$. Abort interaction with the external challenger of Π if $\mathsf{ChQ} \not\subseteq S$. Otherwise proceed as follows.
- Compute $k_{\mathrm{REAL}} := \mathsf{KeyDerive}(\mathsf{params}, \mathsf{ChQ}, (\widetilde{\mathrm{pv}}_j)_{j\in\mathsf{ChQ}}, i, \widetilde{\mathrm{sv}}_i)$ for some arbitrary $i \in \mathsf{ChQ}$.

- Query the reduction with ChQ. Let the response from $\mathcal{R}$ be k^*.
- Toss a coin that takes 1 with probability $\varepsilon_{\mathcal{A}}$ and 0 otherwise.
- If the coin-toss outcome has resulted in a 1, then output 1 if $k^* = k_{\text{REAL}}$ for some arbitrary $i \in \mathsf{ChQ}$ and 0 otherwise. If the coin-toss outcome has resulted in a 0, then output a random bit to $\mathcal{R}$.
- Finally, output whatever $\mathcal{R}$ outputs.

Analysis of $\mathcal{B}$. Let Q be the set of queries made by $\mathcal{R}$ to the external oracle Orcl. Observe that from the admissibility of the reduction, $\mathsf{params}, \{\widetilde{\text{pv}}_i\}_{i \in S}$ information-theoretically determine the common key derived for ChQ (under the interesting case of $\mathsf{ChQ} \subseteq S$). Thus, $\mathcal{B}$ is able to compute the actual common key, namely, k_{REAL}. Finally, when challenged to distinguish this key from random, $\mathcal{B}$ can do so. Since, by definition, it distinguishes with probability $\varepsilon_{\mathcal{A}}$, we have that $\mathcal{B}$ identically simulates $\mathcal{A}_{\text{ideal}}$, conditioned on $\mathcal{A}_{\text{ideal}}$ querying for ChQ that satisfies $\mathsf{ChQ} \subseteq S$. Furthermore, by supposition, $\mathcal{R}$ aborts the interaction with the external challenger when $\mathsf{ChQ} \not\subseteq S$. Thus, the view of the external challenger when it interacts with $\mathcal{R}^{\mathcal{A}_{\text{ideal}}}$ is identical to that when it interacts with $\mathcal{B}^{\mathcal{R}}$. Also, by supposition, recall that, when given oracle access to $\mathcal{A}_{\text{ideal}}$, $\mathcal{R}$ solves Π with probability $\varepsilon_{\mathcal{R}}$. Since $\mathcal{B}$ simply outputs the output of $\mathcal{R}$, we have that $\mathcal{B}$ also solves Π with probability $\varepsilon_{\mathcal{R}}$ by running in polynomial time upon given the instance. Since $\varepsilon_{\mathcal{R}}^{(1)} + \varepsilon_{\mathcal{R}}^{(2)} = \varepsilon_{\mathcal{R}} \geq \varepsilon_{\mathcal{R}}^{(1)} + \varepsilon_{\mathcal{R}}^{(2)} \left(1 - \left(1 - \varepsilon_{\mathcal{R}}^{(2)}\right)^{(N-G)-|Q|}\right) = \varepsilon_{\Pi}$, where $G \in \omega(\log N)$, we meet the bounds of the theorem.

Case 1b – $\mathsf{params}, \{\widetilde{\text{pv}}_i\}$ are constructed based on `ins`. We now consider the case when the reduction first receives `ins` from the external challenger and then invokes the adversary with params. (This can be easily generalized to the case where the reduction first sends the parameters, and some public values $\{\widetilde{\text{pv}}_i\}_{i \in S}$ before receiving `ins` and sends rest of the public values after receiving `ins`; furthermore, the reduction may not stop interacting with the adversary if $\mathsf{ChQ} \not\subseteq S$.) We shall first build an ideal adversary that we shall later show how to simulate.

Ideal Adversary $\mathcal{A}_{\text{ideal}}$. The ideal adversary $\mathcal{A}_{\text{ideal}}$ that breaks the security of aNIKE with probability $\varepsilon_{\mathcal{A}}$ is defined as follows.

- Upon given params, $\mathcal{A}_{\text{ideal}}$ queries to register N-many honest parties. Let the public values output by $\mathcal{R}$ be $\{\widetilde{\text{pv}}_i\}_{i \in [N]}$.
- $\mathcal{A}_{\text{ideal}}$ chooses a random sequence $(i_1, \ldots, i_{N-G})$ of $N - G$) distinct parties and queries for the secret values of these honest parties in the same sequence. Let $\overline{\mathsf{ChQ}} \subset [N]$ be the thus chosen set of parties.
- $\mathcal{A}_{\text{ideal}}$ then queries ChQ as the challenge set of parties, where ChQ is defined as $[N] \setminus \overline{\mathsf{ChQ}}$.
- $\mathcal{A}_{\text{ideal}}$ tosses a coin that takes 1 with probability $\varepsilon_{\mathcal{A}}$ and 0 otherwise.
- Upon receiving k^*, if the coin-toss outcome has resulted in a 1, then output 1 if $k^* = \mathsf{KeyDerive}(\mathsf{params}, \mathsf{ChQ}, (\text{pv}_j)_{j \in \mathsf{ChQ}}, i, \widetilde{\text{sv}}_i)$ for some $i \in \mathsf{ChQ}$ and $\widetilde{\text{sv}}_i$ being a valid secret value, and 0 otherwise. If the coin-toss outcome has resulted in a 0, then output a random bit.

This completes the description of $\mathcal{A}_{\text{ideal}}$. Observe that $\mathcal{A}_{\text{ideal}}$ breaks the security of aNIKE with probability $\varepsilon_{\mathcal{A}}$. Thus, from the hypothesis, the reduction $\mathcal{R}$, when given a black-box access to such

an adversary, should solve the problem Π with probability $\varepsilon_{\mathcal{R}}$. We now describe an algorithm $\mathcal{B}$ that interacts with $\mathcal{R}$ and simulates $\mathcal{A}_{\text{ideal}}$ to $\mathcal{R}$.

Before we describe $\mathcal{B}$, we shall first set some terminologies.

Slot: In an execution of an adversary by $\mathcal{R}$, we shall consider the pair of messages – a query made by the adversary $\mathtt{q}$ and the response $\mathtt{resp}$ given by $\mathcal{R}$. We call this pair of messages a 'slot'. We shall identify this slot by the sequence of query-response pairs till then.

Safe slot: In an execution of an adversary by $\mathcal{R}$, we shall call a slot $s = (\mathtt{q}, \mathtt{resp})$ 'safe', if after the adversary presents the query $\mathtt{q}$ and before it receives the response $\mathtt{resp}$ to $\mathtt{q}$, $\mathcal{R}$ does not send any message to the external challenger.

Meta-reduction $\mathcal{B}$. $\mathcal{B}$ is described as follows.

- Upon $\mathcal{R}$ invoking the adversary with params, query $\mathcal{R}$ to register N-many honest parties. Let the public values output by $\mathcal{R}$ be $\{\widetilde{\mathrm{pv}}_i\}_{i\in[N]}$.
- Choose a random sequence $(i_1, \ldots, i_{N-G})$ of $N-G$ parties. Let $\overline{\mathsf{ChQ}}$ be the set of parties thus chosen. Define ChQ as $[N] \setminus \overline{\mathsf{ChQ}}$.
- Query $\mathcal{R}$ to reveal the secret values of the $N-G$ parties one by one as per the above sequence. As soon as a safe slot $((\mathsf{Ext}(i_1), \widetilde{\mathrm{sv}}_{i_1}), \ldots, (\mathsf{Ext}(i_j), \widetilde{\mathrm{sv}}_{i_j}))$ is encountered, choose a random $i^* \leftarrow \mathsf{ChQ}$ and rewind $\mathcal{R}$ to the point after $((\mathsf{Ext}(i_1), \widetilde{\mathrm{sv}}_{i_1}), \ldots, (\mathsf{Ext}(i_{j-1}), \widetilde{\mathrm{sv}}_{i_{j-1}}))$ and query with $\mathsf{Ext}(i^*)$. If the reduction does not abort, then let $\widetilde{\mathrm{sv}}_{i^*}$ as a response. Otherwise, continue to execute the main thread.
- If $\mathcal{R}$ aborts interaction with the simulated $\mathcal{A}_{\text{ideal}}$ at any point in time, then output whatever $\mathcal{R}$ outputs. Otherwise, upon completing all the queries $(\mathsf{Ext}(i_1), \ldots, \mathsf{Ext}(i_{N-G}))$ and after having received the corresponding responses, present ChQ as the challenge query. Let k^* be the received value.
- If there exit does not exist any rewound slot $((\mathsf{Ext}(i_1), \widetilde{\mathrm{sv}}_{i_1}), \ldots, (\mathsf{Ext}(i^*), \widetilde{\mathrm{sv}}_{i^*}))$ where $\widetilde{\mathrm{sv}}_{i^*} \neq \perp$, then output a random bit to $\mathcal{R}$. Otherwise, compute k_{REAL} as $\mathsf{KeyDerive}(\mathsf{params}, \mathsf{ChQ}, (\widetilde{\mathrm{pv}}_j)_{j\in\mathsf{ChQ}}, i^*, \widetilde{\mathrm{sv}}_{i^*})$.
- Query the reduction with ChQ, where ChQ is defined as $[N] \setminus \overline{\mathsf{ChQ}}$. Let the response from $\mathcal{R}$ be k^*.
- Toss a coin that takes 1 with probability $\varepsilon_{\mathcal{A}}$ and 0 otherwise.
- If the coin toss results in 1, then check if $k^* \stackrel{?}{=} k_{\text{REAL}}$. If yes, then output 1, and 0 otherwise. On the other hand, if the coin toss is 0, then output a random bit to $\mathcal{R}$.
- Finally, output whatever $\mathcal{R}$ outputs.

Analysis of $\mathcal{B}$. Recall that $\mathcal{R}^{\mathcal{A}_{\text{ideal}}}$ denotes the execution of the reduction with the ideal adversary, and $\mathcal{R}^{\mathcal{B}}$ denotes the execution of the reduction with the adversary simulated by $\mathcal{B}$. We begin by defining the following events.

`Abort:` `Abort` is the event that $\mathcal{R}$ aborts an interaction $\mathcal{A}_{\text{ideal}}$ before it responds to the challenge query of $\mathcal{A}_{\text{ideal}}$.

$\mathcal{E}^{(1)}_{\mathtt{ins}}$: $\mathcal{E}^{(1)}_{\mathtt{ins}}$ is the event that the reduction $\mathcal{R}$ solves the instance $\mathtt{ins}$ conditioned on $\mathcal{R}$ aborting its interaction with $\mathcal{A}_{\text{ideal}}$. Define $\varepsilon^{(1)}_{\mathtt{ins}} := \Pr[\text{EXPT}^{\Pi}_{\mathcal{R}^{\mathcal{A}_{\text{ideal}}}}(\lambda) = 1 | \mathtt{Abort} \wedge \mathtt{ins}]$.

$\mathcal{E}^{(2)}_{\mathtt{ins}}$: $\mathcal{E}^{(2)}_{\mathtt{ins}}$ is the event that the reduction $\mathcal{R}$ solves the instance $\mathtt{ins}$ conditioned on $\mathcal{R}$ not aborting its interaction with $\mathcal{A}_{\text{ideal}}$. Define $\varepsilon^{(2)}_{\mathtt{ins}} := \Pr[\text{EXPT}^{\Pi}_{\mathcal{R}^{\mathcal{A}_{\text{ideal}}}}(\lambda) = 1 | \neg\mathtt{Abort} \wedge \mathtt{ins}]$.

Furthermore, define $\eta_{\mathtt{ins}} := \Pr[\mathtt{Abort} | \mathtt{ins}]$.

We have:

$$
\begin{aligned}
&\Pr[\text{EXPT}^{\Pi}_{\mathcal{R}^{\mathcal{A}_{\text{ideal}}}}(\lambda) = 1 | \mathtt{ins}] \\
&\quad = \Pr[\text{EXPT}^{\Pi}_{\mathcal{R}^{\mathcal{A}_{\text{ideal}}}}(\lambda) = 1 | \mathtt{Abort} \wedge \mathtt{ins}] \Pr[\mathtt{Abort} | \mathtt{ins}] + \Pr[\text{EXPT}^{\Pi}_{\mathcal{R}^{\mathcal{A}_{\text{ideal}}}}(\lambda) = 1 | \neg\mathtt{Abort} \wedge \mathtt{ins}] \Pr[\neg\mathtt{Abort} | \mathtt{ins}] \\
&\quad = \varepsilon^{(1)}_{\mathtt{ins}} \eta_{\mathtt{ins}} + \varepsilon^{(2)}_{\mathtt{ins}} (1 - \eta_{\mathtt{ins}})
\end{aligned}
$$

From above, we get:

$$
\begin{aligned}
&\Pr[\mathcal{R}^{\mathcal{A}_{\text{ideal}}}(\mathtt{ins}) \to \mathtt{sol} \ : \ \mathsf{Vrfy}(\mathtt{ins}, \mathtt{sol})] - \Pr[\mathcal{R}^{\mathcal{B}}(\mathtt{ins}) \to \mathtt{sol} \ : \ \mathsf{Vrfy}(\mathtt{ins}, \mathtt{sol})] \\
&\leq \Pr[\text{EXPT}^{\Pi}_{\mathcal{R}^{\mathcal{A}_{\text{ideal}}}}(\lambda) = 1 | \neg\mathtt{Abort} \wedge \mathtt{ins}] \Pr[\neg\mathtt{Abort} | \mathtt{ins}] \cdot (\Pr[\mathtt{Abort} | \mathtt{ins}])^{(N-G)-|Q|} \\
&= \varepsilon^{(2)}_{\mathtt{ins}} (1 - \eta_{\mathtt{ins}}) \eta^{(N-G)-|Q|}_{\mathtt{ins}}
\end{aligned}
$$

Overall, we gather that the success probability of $\mathcal{R}^{\mathcal{B}}$ for an instance $\mathtt{ins}$ is:

$$
\begin{aligned}
\varepsilon_{\mathtt{ins}} &= \Pr[\mathcal{R}^{\mathcal{B}}(\mathtt{ins}) \to \mathtt{sol} \ : \ \mathsf{Vrfy}(\mathtt{ins}, \mathtt{sol})] \\
&\geq \Pr[\mathcal{R}^{\mathcal{A}_{\text{ideal}}}(\mathtt{ins}) \to \mathtt{sol} \ : \ \mathsf{Vrfy}(\mathtt{ins}, \mathtt{sol})] - \varepsilon^{(2)}_{\mathtt{ins}} (1 - \eta_{\mathtt{ins}}) \eta^{(N-G)-|Q|}_{\mathtt{ins}} \\
&= \varepsilon^{(1)}_{\mathtt{ins}} \eta_{\mathtt{ins}} + \varepsilon^{(2)}_{\mathtt{ins}} (1 - \eta_{\mathtt{ins}}) - \varepsilon^{(2)}_{\mathtt{ins}} (1 - \eta_{\mathtt{ins}}) \eta^{(N-G)-|Q|}_{\mathtt{ins}} \\
&= \varepsilon^{(1)}_{\mathtt{ins}} \eta_{\mathtt{ins}} + \varepsilon^{(2)}_{\mathtt{ins}} (1 - \eta_{\mathtt{ins}}) \left(1 - \eta^{(N-G)-|Q|}_{\mathtt{ins}}\right)
\end{aligned}
$$

Finally, weighing the above probability with that of the occurrence of $\mathtt{ins}$ gives us that $\varepsilon_{\Pi} = \sum_{\mathtt{ins} \in \mathsf{InsGen}} \varepsilon_{\mathtt{ins}} \Pr[\mathtt{ins}]$. This establishes the theorem for the current case.

Case 2 – Rewinding/Relaunching. We shall next move to the case when the adversary is rewound for the same/multiple initial values namely, params, $\{\widetilde{\text{pv}}_i\}_{i \in [N]}$, and the adversary's random coins. When rewound for the same initial values, the adversary is rewound for different responses to its queries. For instance, if a slot $((\mathsf{Ext}(i_1), \widetilde{\text{sv}}_{i_1}), \ldots, (\mathsf{Ext}(i_j), \widetilde{\text{sv}}_{i_j}))$ appears in the main thread, the rewound thread corresponds to running the adversary with $((\mathsf{Ext}(i_1), \widetilde{\text{sv}}_{i_1}), \ldots, (\mathsf{Ext}(i_j), \widetilde{\text{sv}}'_{i_j}))$ where $\widetilde{\text{sv}}'_{i_j} \neq \widetilde{\text{sv}}_{i_j}$. The ideal adversary for this case is similar to that for Case 1; we shall highlight the differences here below.

Case 2a – $\mathtt{ins}$ is independent of $\mathsf{params}, \{\widetilde{\text{pv}}_i\}$, only rewinding. Observe that in the case of single execution, roughly speaking, the simulation of the ideal adversary works regardless of whether the adversary was rewound or not. Herein too, the ideal adversary would proceed in the same way. Moreover, the meta-reduction simply ran in exponential time to compute the secret values before the reduction continued any further interaction with the adversary. The meta-reduction in this case would also proceed in the same way.

Case 2b – $\mathsf{params}, \{\widetilde{\text{pv}}_i\}$ are constructed based on $\mathtt{ins}$. The ideal adversary and the meta-reduction $\mathcal{B}$ in this case are similar to the ones in Case 1b, except for a few subtleties that we

shall explore shortly. The ideal adversary is identical to the one in Case 1b. A crucial point that needs attention in this case is that the sequence of Ext queries by $\mathcal{A}_{\text{ideal}}$ is only a function of the parameters and the public values; in particular, this sequence does not depend on the responses to previous Ext queries made by $\mathcal{A}_{\text{ideal}}$. This implies that the set to parties on which $\mathcal{A}_{\text{ideal}}$ wishes to be challenged upon does not vary when $\mathcal{R}$ rewinds $\mathcal{A}_{\text{ideal}}$. This in turn implies that it suffices if $\mathcal{B}$ (somehow) obtains $\widetilde{\text{sv}}_{i^*}$ (for $i^* \in \mathsf{ChQ}$) in any of the rewound threads of execution of (simulated) $\mathcal{A}_{\text{ideal}}$ by $\mathcal{R}$; $\widetilde{\text{sv}}_{i^*}$ would allow the simulated $\mathcal{A}_{\text{ideal}}$ to compute k_{REAL} and complete every thread that is not stopped by $\mathcal{R}$.

Another point of cruciality concerns rewinding $\mathcal{R}$ itself by the meta-reduction $\mathcal{B}$; it is pivotal to bear in mind the exact reason why our meta-reduction needs to rewind $\mathcal{R}$, (lest any overkill will lead to a more complex proof or even disallow us from obtaining a full proof). Our meta-reduction rewinds $\mathcal{R}$ in a hope to obtain $\widetilde{\text{sv}}_{i^*}$ for just any single $i^* \in \mathsf{ChQ}$. Note thus, that it is never the case that the meta-reduction needs to answer any challenge query in the threads corresponding to $\mathcal{R}$ being rewound; this is because, in those threads, by the time $\mathcal{A}_{\text{ideal}}$ reaches a point where it needs to answer

Now we shall see the difference in the meta-reduction when compared to Case 1b: The only difference would be in its behavior when the adversary is rewound. Observe that the ideal adversary when rewound presents the same queries. Thus, $\mathcal{B}$ simulates the ideal adversary in the rewound threads also by presenting the same sequence of queries as in the main thread. The safe slot for which the meta-reduction would rewind the reduction is redefined as follows: $((\mathsf{Ext}(i_1), \widetilde{\text{sv}}_{i_1}), \ldots, (\mathsf{Ext}(i_j), \widetilde{\text{sv}}_{i_j}))$ is a safe slot if $\mathcal{R}$ does not query the external oracle Orcl after the adversary sends $\mathsf{Ext}(i_j)$ and before $\mathcal{R}$ responds with $\widetilde{\text{sv}}_j \neq \perp$, and if the adversary (in any of the executions maintained by $\mathcal{R}$) has not queried $\mathsf{Ext}(i_j)$ before (This is because, our ideal adversary decides on the sequence of Ext queries as a function of the initial parameters; once the parameters are fixed, and if the simulated adversary reveals to the reduction a part of the sequence in one thread, it cannot change the sequence in another thread as the reduction then clearly distinguish the simulated adversary from the ideal one). Upon encountering such a safe slot, $\mathcal{B}$ rewinds $\mathcal{R}$ to present another query $\mathsf{Ext}(i^*)$, where $i^* \leftarrow \mathsf{ChQ}$. If $\mathcal{R}$ aborts this thread, then $\mathcal{B}$ simply switches back to the main thread and continues the interaction with $\mathcal{R}$. Otherwise, it saves the response as $\widetilde{\text{sv}}_{i^*}$. Finally, $\mathcal{B}$ presents ChQ as the challenge set and derives k_{REAL} (if any of the rewindings have resulted in a non-aborting response from $\mathcal{R}$) and responds to the challenge as before. Before we analyze the success probability of $\mathcal{B}$ in solving the given instance of problem Π, we shall treat the even more general case when $\mathcal{R}$ not only rewinds the adversary but relaunches it. After we specify the modifications to $\mathcal{B}$ for this general case, we shall analyze $\mathcal{B}$.

Case 2c – $\mathsf{params}, \{\widetilde{\text{pv}}_i\}$ are constructed based on `ins`, rewinding and relaunching. Following [KK12], we shall refer to every execution of $\mathcal{A}_{\text{ideal}}$ with distinct params and $\{\widetilde{\text{pv}}_i\}_{i\in[N]}$ as a '*round*'. The ideal adversary would remain the same as before in Case 2b: upon given params and $\{\widetilde{\text{pv}}_i\}_{i\in[N]}$, it would freshly choose a random sequence of $\mathsf{Ext}(\cdot)$ queries and finally answer the challenge query correctly with probability $\varepsilon_{\mathcal{A}}$ and randomly otherwise. The corresponding meta-reduction would simulate the ideal adversary as follows. Instead of choosing the random sequences of queries uniformly at random for every new round, $\mathcal{B}$ chooses a random key for a PRF and then chooses the pseudorandom sequence depending on the PRF output for params, $\{\widetilde{\text{pv}}_i\}_{i\in[N]}$ as inputs. Then, in any execution of the adversary by $\mathcal{R}$ for a particular params, $\{\widetilde{\text{pv}}_i\}_{i\in[N]}$, the meta-reduction would use the same sequence except for the following case: when meta-reduction reaches a safe slot, which is just as defined for Case 2b, it rewinds $\mathcal{R}$ and presents a different query like in Case 2b. Finally, the response to the challenge query for every relaunch is computed also the same way as in Case 2b.

Analysis of $\mathcal{B}$. The analysis is much similar to that of Case 1b. Like in Case 1b, we shall begin by (re)defining the following events.

`Abort`: `Abort` is the event that the reduction aborts the interaction with $\mathcal{A}_{\text{ideal}}$ in all the relaunched rounds.

$\mathcal{E}^{(1)}_{\mathtt{ins}}$: $\mathcal{E}^{(1)}_{\mathtt{ins}}$ is the event that the reduction $\mathcal{R}$ solves the instance `ins` conditioned on $\mathcal{R}$ aborting the interaction with $\mathcal{A}_{\text{ideal}}$ in all the relaunched rounds. Define $\varepsilon^{(1)}_{\mathtt{ins}} := \Pr[\text{EXPT}^{\Pi}_{\mathcal{R}^{\mathcal{A}_{\text{ideal}}}}(\lambda) = 1 | \mathtt{Abort} \wedge \mathtt{ins}]$.

$\mathcal{E}^{(2)}_{\mathtt{ins}}$: $\mathcal{E}^{(2)}_{\mathtt{ins}}$ is the event that the reduction $\mathcal{R}$ solves the instance `ins` conditioned on $\mathcal{R}$ not aborting all the interaction with $\mathcal{A}_{\text{ideal}}$. Define $\varepsilon^{(2)}_{\mathtt{ins}} := \Pr[\text{EXPT}^{\Pi}_{\mathcal{R}^{\mathcal{A}_{\text{ideal}}}}(\lambda) = 1 | \neg\mathtt{Abort} \wedge \mathtt{ins}]$.

Furthermore, define $\eta_{\mathtt{ins}} := \Pr[\mathtt{Abort} | \mathtt{ins}]$.

Just like in Case 1b, we have:

$$\begin{aligned}
&\Pr[\text{EXPT}^{\Pi}_{\mathcal{R}^{\mathcal{A}_{\text{ideal}}}}(\lambda) = 1 | \mathtt{ins}] \\
&\quad = \Pr[\text{EXPT}^{\Pi}_{\mathcal{R}^{\mathcal{A}_{\text{ideal}}}}(\lambda) = 1 | \mathtt{Abort} \wedge \mathtt{ins}] \Pr[\mathtt{Abort} | \mathtt{ins}] + \Pr[\text{EXPT}^{\Pi}_{\mathcal{R}^{\mathcal{A}_{\text{ideal}}}}(\lambda) = 1 | \neg\mathtt{Abort} \wedge \mathtt{ins}] \Pr[\neg\mathtt{Abort} | \mathtt{ins}] \\
&\quad = \varepsilon^{(1)}_{\mathtt{ins}} \eta_{\mathtt{ins}} + \varepsilon^{(2)}_{\mathtt{ins}} (1 - \eta_{\mathtt{ins}})
\end{aligned}$$

Let Q_i be the set of queries made by $\mathcal{R}$ to the external oracle Orcl during the ith round. With r denoting the total number of rounds, from above, we get:

$$\begin{aligned}
&\Pr[\mathcal{R}^{\mathcal{A}_{\text{ideal}}}(\mathtt{ins}) \to \mathtt{sol} \ : \ \mathsf{Vrfy}(\mathtt{ins}, \mathtt{sol})] - \Pr[\mathcal{R}^{\mathcal{B}}(\mathtt{ins}) \to \mathtt{sol} \ : \ \mathsf{Vrfy}(\mathtt{ins}, \mathtt{sol})] \\
&\leq \Pr[\text{EXPT}^{\Pi}_{\mathcal{R}^{\mathcal{A}_{\text{ideal}}}}(\lambda) = 1 | \neg\mathtt{Abort} \wedge \mathtt{ins}] \Pr[\neg\mathtt{Abort} | \mathtt{ins}] \cdot \sum_{i=1}^{r} (\Pr[\mathtt{Abort} | \mathtt{ins}])^{(N-G)-|Q_i|} \\
&= \varepsilon^{(2)}_{\mathtt{ins}} (1 - \eta_{\mathtt{ins}}) \sum_{i=1}^{r} \eta_{\mathtt{ins}}^{(N-G)-|Q_i|}
\end{aligned}$$

Whence, the success probability of $\mathcal{R}^{\mathcal{B}}$ for an instance `ins` is:

$$\begin{aligned}
\varepsilon_{\mathtt{ins}} &= \Pr[\mathcal{R}^{\mathcal{B}}(\mathtt{ins}) \to \mathtt{sol} \ : \ \mathsf{Vrfy}(\mathtt{ins}, \mathtt{sol})] \\
&\geq \Pr[\mathcal{R}^{\mathcal{A}_{\text{ideal}}}(\mathtt{ins}) \to \mathtt{sol} \ : \ \mathsf{Vrfy}(\mathtt{ins}, \mathtt{sol})] - \varepsilon^{(2)}_{\mathtt{ins}} (1 - \eta_{\mathtt{ins}}) \sum_{i=1}^{r} \eta_{\mathtt{ins}}^{(N-G)-|Q_i|} \\
&= \varepsilon^{(1)}_{\mathtt{ins}} \eta_{\mathtt{ins}} + \varepsilon^{(2)}_{\mathtt{ins}} (1 - \eta_{\mathtt{ins}}) - \varepsilon^{(2)}_{\mathtt{ins}} (1 - \eta_{\mathtt{ins}}) \sum_{i=1}^{r} \eta_{\mathtt{ins}}^{(N-G)-|Q_i|} \\
&= \varepsilon^{(1)}_{\mathtt{ins}} \eta_{\mathtt{ins}} + \varepsilon^{(2)}_{\mathtt{ins}} (1 - \eta_{\mathtt{ins}}) \left(1 - \sum_{i=1}^{r} \eta_{\mathtt{ins}}^{(N-G)-|Q_i|} \right)
\end{aligned}$$

Finally, accounting the success for every `ins` as per the probability of occurrence of `ins`, we have, $\varepsilon_{\Pi} = \sum_{\mathtt{ins} \in \mathsf{InsGen}} \varepsilon_{\mathtt{ins}} \Pr[\mathtt{ins}]$. This establishes the theorem for the generic case too.

■

Acknowledgments

The author is thankful to Dana Dachman-Soled, Vipul Goyal, Jonathan Katz, Dakshita Khurana, Adam O'Neill, Kenny Paterson, Amit Sahai, Samarinder Sharma, and Ramarathnam Venkatesan for valuable comments.

References

[BC05] Michael Backes and Christian Cachin. Public-key steganography with active attacks. In Joe Kilian, editor, *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 210–226. Springer, 2005.

[BF03] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.

[BG04] Daniel R. L. Brown and Robert P. Gallant. The static diffie-hellman problem. *IACR Cryptology ePrint Archive*, 2004:306, 2004.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, pages 501–519, 2014.

[BGK+14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 221–238. Springer, 2014.

[BMP04] Colin Boyd, Wenbo Mao, and Kenneth G. Paterson. Key agreement using statically keyed authenticators. In Markus Jakobsson, Moti Yung, and Jianying Zhou, editors, *Applied Cryptography and Network Security, Second International Conference, ACNS 2004, Yellow Mountain, China, June 8-11, 2004, Proceedings*, pages 248–262. Springer, 2004.

[BMV08] Emmanuel Bresson, Jean Monnerat, and Damien Vergnaud. Separation results on the "one-more" computational problems. In Tal Malkin, editor, *Topics in Cryptology - CT-RSA 2008, The Cryptographers' Track at the RSA Conference 2008, San Francisco, CA, USA, April 8-11, 2008. Proceedings*, pages 71–87. Springer, 2008.

[BN05] Mihir Bellare and Gregory Neven. Transitive signatures: new schemes and proofs. *IEEE Transactions on Information Theory*, 51(6):2133–2151, 2005.

[BNPS03] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *J. Cryptology*, 16(3):185–215, 2003.

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In Yvo Desmedt, editor, *Public Key*

Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings, pages 31–46. Springer, 2003.

[BP02] Mihir Bellare and Adriana Palacio. GQ and schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, pages 162–177. Springer, 2002.

[BR14a] Zvika Brakerski and Guy N. Rothblum. Black-box obfuscation for d-cnfs. In Moni Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 235–250. ACM, 2014.

[BR14b] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In Yehuda Lindell, editor, *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, pages 1–25. Springer, 2014.

[BS01] Mihir Bellare and Ravi S. Sandhu. The security of practical two-party RSA signature schemes. *IACR Cryptology ePrint Archive*, 2001:60, 2001.

[BS02] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324:71–90, 2002.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, pages 280–300. Springer, 2013.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 480–499. Springer, 2014.

[Can97] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, pages 455–469. Springer, 1997.

[CGP+13] Cagatay Capar, Dennis Goeckel, Kenneth G. Paterson, Elizabeth A. Quaglia, Don Towsley, and Murtaza Zafer. Signal-flow-based analysis of wireless security protocols. *Inf. Comput.*, 226:37–56, 2013.

[CGP14] Ran Canetti, Shafi Goldwasser, and Oxana Poburinnaya. Adaptively secure two-party computation from indistinguishability obfuscation. Cryptology ePrint Archive, Report 2014/845, 2014. http://eprint.iacr.org/.

[CHL+14] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. Cryptology ePrint Archive, Report 2014/906, 2014. http://eprint.iacr.org/.

[CKS09] David Cash, Eike Kiltz, and Victor Shoup. The twin diffie-hellman problem and applications. *J. Cryptology*, 22(4):470–504, 2009.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 494–503. ACM, 2002.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 476–493. Springer, 2013.

[CMR98] Ran Canetti, Daniele Micciancio, and Omer Reingold. Perfectly one-way probabilistic hash functions (preliminary version). In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 131–140. ACM, 1998.

[Cor02] Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, pages 272–287. Springer, 2002.

[CRV10] Ran Canetti, Guy N. Rothblum, and Mayank Varia. Obfuscation of hyperplane membership. In Daniele Micciancio, editor, *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, pages 72–89. Springer, 2010.

[CV07] Dario Catalano and Ivan Visconti. Hybrid commitments and their applications to zero-knowledge proof systems. *Theor. Comput. Sci.*, 374(1-3):229–260, 2007.

[DE06] Régis Dupont and Andreas Enge. Provably secure non-interactive key distribution based on pairings. *Discrete Applied Mathematics*, 154(2):270–276, 2006.

[DH76] Whitfield Diffie and Martin E. Hellman. Multiuser cryptographic techniques. In *AFIPS National Computer Conference*, pages 109–112, 1976.

[DKSW09] Yevgeniy Dodis, Jonathan Katz, Adam Smith, and Shabsi Walfish. Composability and on-line deniability of authentication. In Omer Reingold, editor, *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, pages 146–162. Springer, 2009.

[DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, pages 581–596. Springer, 2002.

[DPR14] Ivan Damgård, Antigoni Polychroniadou, and Vanishree Rao. Adaptively secure uc constant round multi-party computation. Cryptology ePrint Archive, Report 2014/845, 2014. http://eprint.iacr.org/.

[DR06] T. Dierks and E. Rescorla. The transport layer security (tls) protocol. In *IETF RFC 4346*, 2006.

[DSKR14] Dana Dachman-Soled, Jonathan Katz, and Vanishree Rao. Adaptively secure, universally composable, multi-party computation in constant rounds. Cryptology ePrint Archive, Report 2014/858, 2014. http://eprint.iacr.org/.

[FF13] Marc Fischlin and Nils Fleischhacker. Limitations of the meta-reduction technique: The case of schnorr signatures. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 444–460. Springer, 2013.

[FHH14] Eduarda S. V. Freire, Julia Hesse, and Dennis Hofheinz. Universally composable non-interactive key exchange. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, pages 1–20. Springer, 2014.

[FHKP13] Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. Non-interactive key exchange. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings*, pages 254–271. Springer, 2013.

[FHPS13] Eduarda S. V. Freire, Dennis Hofheinz, Kenneth G. Paterson, and Christoph Striecks. Programmable hash functions in the multilinear setting. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 513–530. Springer, 2013.

[FJS14] Nils Fleischhacker, Tibor Jager, and Dominique Schröder. On tight security proofs for schnorr signatures. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, pages 512–531. Springer, 2014.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 1–17. Springer, 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In Yehuda Lindell, editor, *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, pages 74–94. Springer, 2014.

[GK10] Adam Groce and Jonathan Katz. A new framework for efficient password-based authenticated key exchange. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 516–525. ACM, 2010.

[GL03] Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, pages 524–543. Springer, 2003.

[GP14] Sanjam Garg and Antigoni Polychroniadou. Two-round adaptively secure mpc from indistinguishability obfuscation. Cryptology ePrint Archive, Report 2014/844, 2014. http://eprint.iacr.org/.

[GS12] Sanjam Garg and Amit Sahai. Adaptively secure multi-party computation with dishonest majority. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 105–123. Springer, 2012.

[HJK12] Dennis Hofheinz, Tibor Jager, and Edward Knapp. Waters signatures with optimal security reduction. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings*, pages 66–83. Springer, 2012.

[HJK+14] Dennis Hofheinz, Tibor Jager, Dakshita Khurana, Amit Sahai, Brent Waters, and Mark Zhandry. How to generate and use universal parameters. Cryptology ePrint Archive, Report 2014/507, 2014. http://eprint.iacr.org/.

[HKKW14] Dennis Hofheinz, Akshay Kamath, Venkata Koppula, and Brent Waters. Adaptively secure constrained pseudorandom functions. *IACR Cryptology ePrint Archive*, 2014:720, 2014.

[HSW14] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 201–220. Springer, 2014.

[Jou04] Antoine Joux. A one round protocol for tripartite diffie-hellman. *J. Cryptology*, 17(4):263–276, 2004.

[JSI96] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In Ueli M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, pages 143–154. Springer, 1996.

[KK12] Saqib A. Kakvi and Eike Kiltz. Optimal security proofs for full domain hash, revisited. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology -*

EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings, pages 537–553. Springer, 2012.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 669–684, 2013.

[LPS04] Ben Lynn, Manoj Prabhakaran, and Amit Sahai. Positive results and techniques for obfuscation. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 20–39. Springer, 2004.

[MR13] Tal Moran and Alon Rosen. There is no indistinguishability obfuscation in pessiland. *IACR Cryptology ePrint Archive*, 2013:643, 2013.

[PS09] Kenneth G. Paterson and Sriramkrishnan Srinivasan. On the relations between non-interactive key distribution, identity-based encryption and trapdoor discrete log groups. *Des. Codes Cryptography*, 52(2):219–241, 2009.

[PV05] Pascal Paillier and Damien Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In Bimal K. Roy, editor, *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings*, pages 1–20. Springer, 2005.

[SOK00] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing, 2000. SCIS 2000, The 2000 Symposium on Cryptography and Information Security, Okinawa, Japan, January 26–28.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 475–484. ACM, 2014.

[Wee05] Hoeteck Wee. On obfuscating point functions. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 523–532. ACM, 2005.

A Security Of Construction With Setup

Theorem 3. If $i\mathcal{O}$ is indistinguishably secure, Σ is a CPA-secure encryption scheme, HCOM is a hybrid trapdoor commitment scheme, and F is an obliviously-patchable puncturable PRF, then Construction 1 is an *adaptively* secure multiparty non-interactive key exchange protocol.

Proof. Assume towards contradiction that an adversary $\mathcal{A}$ has non-negligible advantage in breaking the adaptive security of our Construction 1 as in Definition 6. We arrive at a contradiction through a carefully designed sequence of several hybrids $\mathsf{Hyb}_0, \mathsf{Hyb}_1, \ldots, \mathsf{Hyb}_{10}$. If the challenger in Hyb_i

chooses to play the real experiment (respectively, the random experiment) and $\mathcal{A}$ outputs 1, then we denote the event by $\text{REAL}_{\mathcal{A}}^{\mathsf{Hyb}_i} \to 1$ (respectively, $\text{RAND}_{\mathcal{A}}^{\mathsf{Hyb}_i} \to 1$). The advantage of $\mathcal{A}$ in Hyb_i is defined as

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_i}(\lambda) := |\Pr[\text{REAL}_{\mathcal{A}}^{\mathsf{Hyb}_i} \to 1] - \Pr[\text{RAND}_{\mathcal{A}}^{\mathsf{Hyb}_i} \to 1]| \tag{5}$$

Furthermore, we shall use Hyb_i to also denote the view of the adversary in the hybrid whenever the connotation is unambiguous.

To maintain ease of verification for the reader, we present a full description of each hybrid experiment, each one given on a separate page. The modification introduced in the current hybrid in comparison with the previous hybrid will be highlighted in red underlined font. Furthermore, if a value is removed as we move from one hybrid to the next, then in the latter hybrid, we highlight the ~~value~~ within a red frame with a red strike-through. Also, let's say that while switching to the next hybrid, a value is moved from one step of execution to a later step; then in the latter hybrid, ~~value~~ shall appear at its position in the previous hybrid and value shall appear at its new position in the current hybrid.

Hyb_0. This experiment is the same as the original experiment in Definition 6. Here, we simply unwrap the original experiment as per our construction. The challenger behaves as follows. It begins by choosing $\text{EXPT} \in \{\text{REAL}, \text{RAND}\}$ uniformly at random, and executes with $\mathcal{A}$ the experiment $\text{EXPT}_{\mathcal{A}}^{\mathsf{Hyb}_0}$ defined as follows.

1. Run $\mathsf{F.params} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$, $\text{crs} \leftarrow \mathsf{HGen}(1^\lambda)$, and $(\text{pk}, \cdot) \leftarrow \mathsf{E.Gen}(1^\lambda)$.
2. Compute $\widetilde{\mathsf{KD\text{-}Prog}}$ as follows.
   - $K \leftarrow \mathsf{F.KeyGen}(1^\lambda, \mathsf{F.params})$.
   - For K, generate $\mathsf{KD\text{-}Prog}_0$ as in Figure 3, and compute $\widetilde{\mathsf{KD\text{-}Prog}} \leftarrow \mathsf{i}\mathcal{O}(\mathsf{KD\text{-}Prog}_0)$.

Thus, set $\mathsf{params} = (\widetilde{\mathsf{KD\text{-}Prog}}, \mathsf{F.params}, \text{crs}, \text{pk})$ and give it to $\mathcal{A}$. Then respond to the adversary's queries as follows.

3. Upon receiving $\mathsf{Reg}(i \in [N])$, run $\mathsf{Publish}((\widetilde{\mathsf{KD\text{-}Prog}}, \mathsf{F.params}, \text{crs}, \text{pk}), \lambda, i)$. Namely, sample $\text{pt}_i \leftarrow \{0,1\}^\ell$, encrypt pt_i as $\text{ct}_i \leftarrow \mathsf{Enc}(\text{pk}, \text{pt}_i)$, and commit to ct_i as $\text{com}_i = \mathsf{HCommit}(\text{crs}, \text{ct}_i; \text{open}_i)$. Set $\text{sv}_i = (\text{ct}_i, \text{open}_i)$ and $\text{pv}_i = \text{com}_i$. Respond via pv_i.
4. Upon receiving $\mathsf{Ext}(i)$ for a registered honest P_i, respond via sv_i.
5. Upon receiving $\mathsf{Rev}(\mathcal{S}, j)$, run $\widetilde{\mathsf{KD\text{-}Prog}}$ on input $(\mathcal{S}, (\text{pv}_i)_{i \in \mathcal{S}}, j, \text{sv}_j)$ and respond via the output.
6. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, choose $j \leftarrow \mathsf{ChQ}$, compute $Y_{\text{REAL}} := \widetilde{\mathsf{KD\text{-}Prog}}(\mathsf{ChQ}, (\text{pv}_i)_{i \in \mathsf{ChQ}}, j, \text{sv}_j)$ and sample $Y_{\text{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
7. Finally, output whatever $\mathcal{A}$ outputs.

$\mathsf{KD\text{-}Prog}_0$

Constants:

- $\mathsf{F.params}, \mathrm{crs}, \mathrm{pk}$
- K

Input: $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, j, \mathrm{sv}_j)$, where $\mathcal{S} \subseteq [N]$, $|\mathcal{S}| \le G$.
Procedure:

1. Check whether $j \in \mathcal{S}$; if not, output $\perp$. Then, check whether $(\mathrm{pv}_j, \mathrm{sv}_j) \in \mathrm{R}^{\mathsf{HCOM}}$ (defined in Equation (2)); if not, output $\perp$. Otherwise, proceed as follows.
2. Compute and output $F(K, (\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}))$.

Figure 3: Key-derivation Program

Hyb_1. This experiment is the same as Hyb_0 except for the following modification. Essentially, this modification corresponds to the sequence in which the challenger computes certain values. Namely, before the adversary begins presenting queries, the challenger samples the public-value secret-value pairs ahead of time – we shall denote these pairs as $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$. Recall that the challenger does not know ahead of time for which i the adversary would issue $\mathsf{Reg}(i \in [N])$ (for which the challenger is supposed to compute a public-value) and for which other i it would issue $\mathsf{RegCorr}(i \in [N], \mathrm{pv}_i)$ (for which the challenger need not compute a public-value). Thus, in this hybrid, the challenger, who samples the pairs ahead of time, samples the pairs for for *every* $i \in [N]$. Later, it would use $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$ only if the adversary issues $\mathsf{Reg}(i \in [N])$. Details follow.

The challenger behaves as follows. It begins by choosing $\textsc{expt} \in \{\textsc{real}, \textsc{rand}\}$ uniformly at random, and executes with $\mathcal{A}$ the experiment $\textsc{expt}_{\mathcal{A}}^{\mathsf{Hyb}_1}$ defined as follows.

1. Run $\mathsf{F.params} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$, $\mathrm{crs} \leftarrow \mathsf{HGen}(1^\lambda)$, and $(\mathrm{pk}, \cdot) \leftarrow \mathsf{E.Gen}(1^\lambda)$. (Defer generating the program $\widetilde{\mathsf{KD\text{-}Prog}}$ to later).
2. Then, for every $i \in [N]$, compute $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$ as follows and store $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$: sample $\mathrm{pt}_i \leftarrow \{0,1\}^\ell$, encrypt pt_i as $\mathrm{ct}_i \leftarrow \mathsf{Enc}(\mathrm{pk}, \mathrm{pt}_i)$, and commit to ct_i as $\mathrm{com}_i = \mathsf{HCommit}(\mathrm{crs}, \mathrm{ct}_i; \mathrm{open}_i)$. Set $\widetilde{\mathrm{sv}}_i = (\mathrm{ct}_i, \mathrm{open}_i)$ and $\widetilde{\mathrm{pv}}_i = \mathrm{com}_i$. (Observe that, in our protocol, the algorithm $\mathsf{Publish}(\mathsf{params}, \lambda, i)$ depends only on $\mathrm{crs}, \mathrm{pk}$, and in particular, does not depend on $\widetilde{\mathsf{KD\text{-}Prog}}$. Thus, this step of the challenger is still well-defined.)
3. Compute $\widetilde{\mathsf{KD\text{-}Prog}}$ as follows.
   - $K \leftarrow \mathsf{F.KeyGen}(1^\lambda, \mathsf{F.params})$.
   - For K, generate $\mathsf{KD\text{-}Prog}_1$ as in Figure 4 and compute $\widetilde{\mathsf{KD\text{-}Prog}} \leftarrow \mathsf{i}\mathcal{O}(\mathsf{KD\text{-}Prog}_1)$.

Thus, set $\mathsf{params} = (\widetilde{\mathsf{KD\text{-}Prog}}, \mathsf{F.params}, \mathrm{crs}, \mathrm{pk})$ and give it to $\mathcal{A}$. Then respond to the adversary's queries as follows.

4. Upon receiving $\mathsf{Reg}(i \in [N])$, respond via $\mathrm{pv}_i = \widetilde{\mathrm{pv}}_i$. (On the other hand, if the adversary himself registers a (corrupted) party P_i^* by presenting $\mathsf{RegCorr}(i \in [N], \mathrm{pv}_i)$, then simply ignore $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$.)
5. Upon receiving $\mathsf{Ext}(i)$ for a registered honest P_i, respond via $\mathrm{sv}_i = \widetilde{\mathrm{sv}}_i$.
6. Upon receiving $\mathsf{Rev}(\mathcal{S}, j)$, run $\widetilde{\mathsf{KD\text{-}Prog}}$ on input $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, j, \mathrm{sv}_j)$ and respond via the output.

7. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, choose $j \leftarrow \mathsf{ChQ}$, compute $Y_{\text{REAL}} := \widetilde{\mathsf{KD\text{-}Prog}}(\mathsf{ChQ}, (\text{pv}_i)_{i \in \mathsf{ChQ}}, j, \text{sv}_j)$ and sample $Y_{\text{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
8. Finally, output whatever $\mathcal{A}$ outputs.

$\mathsf{KD\text{-}Prog}_1$

Constants:

- $\mathsf{F.params}, \text{crs}, \text{pk}$
- K
- $\{\widetilde{\text{pv}}_i\}_{i \in [N]}$

Input: $(\mathcal{S}, (\text{pv}_i)_{i \in \mathcal{S}}, j, \text{sv}_j)$, where $\mathcal{S} \subseteq [N]$, $|\mathcal{S}| \leq G$.
Procedure:

1. Check whether $j \in \mathcal{S}$; if not, output $\perp$. Then, check whether $(\text{pv}_j, \text{sv}_j) \in \text{R}^{\mathsf{HCOM}}$; if not, output $\perp$. Otherwise, proceed as follows.
2. Compute and output $F(K, (\mathcal{S}, (\text{pv}_i)_{i \in \mathcal{S}}))$.

Figure 4: Key-derivation Program

Lemma 1. $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_1$.

Proof. Observe that the only difference between Hyb_0 and Hyb_1 is in the sequence of the values computed and an additional constant, $\{\widetilde{\text{pv}}_i\}_{i \in [N]}$, in the program $\mathsf{KD\text{-}Prog}_1$ of Hyb_1, that is not present in program $\mathsf{KD\text{-}Prog}_0$ of Hyb_0. As for the first difference, in Hyb_1 the challenger computes $(\widetilde{\text{pv}}_i, \widetilde{\text{sv}}_i)$ for every i ahead of time and uses them only if necessary (i.e., if the adversary requests to register i as an honest party). Observe that this does not cause any deviation in the distribution of $(\widetilde{\text{pv}}_i, \widetilde{\text{sv}}_i)$ for identities i under which the adversary requests to register honest parties. As for the next difference, since the input/output relation of both the programs $\mathsf{KD\text{-}Prog}_0$ and $\mathsf{KD\text{-}Prog}_1$ are identical, from the security of $\mathsf{i}\mathcal{O}$, we have that their obfuscations are computationally indistinguishable. Hence, we have that $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_1$.

■

Hyb_2. This experiment is the same as Hyb_1 except for the following modification in the way the key is sampled for constructing $\mathsf{KD\text{-}Prog}_2$.

The challenger behaves as follows. It begins by choosing $\text{EXPT} \in \{\text{REAL}, \text{RAND}\}$ uniformly at random, and executes with $\mathcal{A}$ the experiment $\text{EXPT}_{\mathcal{A}}^{\mathsf{Hyb}_2}$ defined as follows.

1. Run $\mathsf{F.params} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$, $\text{crs} \leftarrow \mathsf{HGen}(1^\lambda)$, and $(\text{pk}, \cdot) \leftarrow \mathsf{E.Gen}(1^\lambda)$. (Defer generating the program $\widetilde{\mathsf{KD\text{-}Prog}}$ to later).
2. Then, for every $i \in [N]$, compute $(\widetilde{\text{pv}}_i, \widetilde{\text{sv}}_i)$ as follows and store $(\widetilde{\text{pv}}_i, \widetilde{\text{sv}}_i)$: sample $\text{pt}_i \leftarrow \{0,1\}^\ell$, encrypt pt_i as $\text{ct}_i \leftarrow \mathsf{Enc}(\text{pk}, \text{pt}_i)$, and commit to ct_i as $\text{com}_i = \mathsf{HCommit}(\text{crs}, \text{ct}_i; \text{open}_i)$. Set $\widetilde{\text{sv}}_i = (\text{ct}_i, \text{open}_i)$ and $\widetilde{\text{pv}}_i = \text{com}_i$. (Observe that, in our scheme, the algorithm $\mathsf{Publish}(\mathsf{params}, \lambda, i)$ depends only on crs, pk, and in particular, does not depend on $\widetilde{\mathsf{KD\text{-}Prog}}$. Thus, this step of the challenger is still well-defined.)
3. Compute $\widetilde{\mathsf{KD\text{-}Prog}}$ as follows.
   - Sample a key, $K \leftarrow \mathsf{F.KeyGen}(1^\lambda, \mathsf{F.params})$.
   - Puncture K at $2_G^{\widetilde{\text{pv}}}$: $K[2_G^{\widetilde{\text{pv}}}] \leftarrow \mathsf{Puncture}(K, 2_G^{\widetilde{\text{pv}}})$, where $\widetilde{\text{pv}} = (\widetilde{\text{pv}}_1, \ldots, \widetilde{\text{pv}}_N)$.
   - Generate patches for $K[2_G^{\widetilde{\text{pv}}}]$ at every ith block: $\mathsf{patch}(K, (i, \widetilde{\text{pv}}_i)) \leftarrow \mathsf{PatchGen}(K, i, \widetilde{\text{pv}}_i)$.
   - For $(K[2_G^{\widetilde{\text{pv}}}], \{\mathsf{patch}(K, (i, \widetilde{\text{pv}}_i))\}_{i \in [N]})$, generate $\mathsf{KD\text{-}Prog}_1$ as in Figure 4 and compute $\widetilde{\mathsf{KD\text{-}Prog}} \leftarrow \mathsf{i}\mathcal{O}(\mathsf{KD\text{-}Prog}_1)$.

Thus, set $\mathsf{params} = (\widetilde{\mathsf{KD\text{-}Prog}}, \mathsf{F.params}, \text{crs}, \text{pk})$ and give it to $\mathcal{A}$. Then respond to the adversary's queries as follows.

4. Upon receiving $\mathsf{Reg}(i \in [N])$, respond via $\text{pv}_i = \widetilde{\text{pv}}_i$. (On the other hand, if the adversary himself registers a (corrupted) party P_i^* by presenting $\mathsf{RegCorr}(i \in [N], \text{pv}_i)$, then simply ignore $(\widetilde{\text{pv}}_i, \widetilde{\text{sv}}_i)$.)
5. Upon receiving $\mathsf{Ext}(i)$ for a registered honest P_i, respond via $\text{sv}_i = \widetilde{\text{sv}}_i$.
6. Upon receiving $\mathsf{Rev}(\mathcal{S}, j)$, run $\widetilde{\mathsf{KD\text{-}Prog}}$ on input $(\mathcal{S}, (\text{pv}_i)_{i \in \mathcal{S}}, j, \text{sv}_j)$ and respond via the output.
7. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, choose $j \leftarrow \mathsf{ChQ}$, compute $Y_{\text{REAL}} := \widetilde{\mathsf{KD\text{-}Prog}}(\mathsf{ChQ}, (\text{pv}_i)_{i \in \mathsf{ChQ}}, j, \text{sv}_j)$ and sample $Y_{\text{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
8. Finally, output whatever $\mathcal{A}$ outputs.

Lemma 2. $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_2$.

Proof. Observe that the only difference between Hyb_1 and Hyb_2 is in the structure of the key and the way it is used in computing the final outcome. We shall show that, despite the change, the input-output relations of the programs $\mathsf{KD\text{-}Prog}_1$ and $\mathsf{KD\text{-}Prog}_2$ are identical. Thence, by applying the security property of $\mathsf{i}\mathcal{O}$, we have that $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_2$. Now to prove the functional equivalence of $\mathsf{KD\text{-}Prog}_1$ and $\mathsf{KD\text{-}Prog}_2$, we shall show that, there exists a one-to-one relation between all possible keys K of $\mathsf{KD\text{-}Prog}_1$ and all possible keys $(K[2_G^{\widetilde{\text{pv}}}], \{\mathsf{patch}(K, (i, \widetilde{\text{pv}}_i))\}_{i \in [N]})$ of $\mathsf{KD\text{-}Prog}_2$, and that the input-output relations of the programs $\mathsf{KD\text{-}Prog}_1$ and $\mathsf{KD\text{-}Prog}_2$ are identical for the mapped keys. Towards establishing the one-to-one mapping, consider any $\{\widetilde{\text{pv}}_i\}_{i \in [N]}$ (which is a part of the constants in both the programs). For any K for program $\mathsf{KD\text{-}Prog}_1$, the corresponding key for $\mathsf{KD\text{-}Prog}_2$ is $(K[2_G^{\widetilde{\text{pv}}}], \{\mathsf{patch}(K, (i, \widetilde{\text{pv}}_i))\}_{i \in [N]})$. Now consider any input $(\mathcal{S}, (\text{pv}_i)_{i \in \mathcal{S}}, j, \text{sv}_j)$. Let us

$\mathsf{KD\text{-}Prog}_2$

Constants:

- $\mathsf{F.params}, \mathrm{crs}, \mathrm{pk}$
- $(K[2^{\widetilde{\mathrm{pv}}}_G], \{\mathsf{patch}(K, (i, \widetilde{\mathrm{pv}}_i))\}_{i\in[N]})$
- $\{\widetilde{\mathrm{pv}}_i\}_{i\in[N]}$

Input: $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, j, \mathrm{sv}_j)$, where $\mathcal{S} \subseteq [N]$, $|\mathcal{S}| \leq G$.
Procedure:

1. Check whether $j \in \mathcal{S}$; if not, output $\perp$. Then, check whether $(\mathrm{pv}_j, \mathrm{sv}_j) \in \mathrm{R}^{\mathsf{HCOM}}$; if not, output $\perp$. Otherwise, proceed as follows.
2. - If $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}) \notin 2^{\widetilde{\mathrm{pv}}}_G$, then compute and output $\mathsf{Eval}(K[2^{\widetilde{\mathrm{pv}}}_G], (\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}))$.
   - If $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}) \in 2^{\widetilde{\mathrm{pv}}}_G$, then compute and output $\mathsf{Eval}((K[2^{\widetilde{\mathrm{pv}}}_G], \mathsf{patch}(K, (j, \widetilde{\mathrm{pv}}_j))), (\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}))$.

Figure 5: Key-derivation Program

analyze the outputs for this input as computed by programs $\mathsf{KD\text{-}Prog}_1$ and $\mathsf{KD\text{-}Prog}_2$. Observe that the checks performed at Step 1 in both the programs are identical. In the event that this check does not go through, both the programs output the same value, namely, $\perp$. However, if the check goes through, $\mathsf{KD\text{-}Prog}_1$ computes the function F on $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}})$ with key K; on the other hand, $\mathsf{KD\text{-}Prog}_2$ works under two cases:

(a). When $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}) \notin 2^{\widetilde{\mathrm{pv}}}_G$, it outputs $\mathsf{Eval}(K[2^{\widetilde{\mathrm{pv}}}_G], (\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}))$. Note that $\mathsf{Eval}(K[2^{\widetilde{\mathrm{pv}}}_G], (\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}})) = F(K, (\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}))$.

(b). When $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}) \in 2^{\widetilde{\mathrm{pv}}}_G$, it outputs $\mathsf{Eval}((K[2^{\widetilde{\mathrm{pv}}}_G], \mathsf{patch}(K, (j, \widetilde{\mathrm{pv}}_j))), (\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}))$. Note that

$\mathsf{Eval}((K[2^{\widetilde{\mathrm{pv}}}_G], \mathsf{patch}(K, (j, \widetilde{\mathrm{pv}}_j))), (\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}})) = F(K, (\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}))$.

Hence, the programs $\mathsf{KD\text{-}Prog}_1$ and $\mathsf{KD\text{-}Prog}_2$ are functionally equivalent, thus allowing us to apply security of $\mathsf{i}\mathcal{O}$ and concluding that $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_2$. ■

Hyb_3. This experiment is the same as Hyb_2 except for the following modification in the *sequence* of the computations performed by the challenger. At a high level, the challenger first obliviously samples the patches for every i (i.e., oblivious of $\widetilde{\mathrm{pv}}_i$). Then it computes $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$ for every i, after which it computes $K[2^{\widetilde{\mathrm{pv}}}_G]$ for $\widetilde{\mathrm{pv}} := (\widetilde{\mathrm{pv}}_1, \ldots, \widetilde{\mathrm{pv}}_N)$. Details follow.

The challenger behaves as follows. It begins by choosing $\text{EXPT} \in \{\text{REAL}, \text{RAND}\}$ uniformly at random, and executes with $\mathcal{A}$ the experiment $\text{EXPT}^{\mathsf{Hyb}_3}_{\mathcal{A}}$ defined as follows.

1. Run $\mathsf{F.params} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$, $\mathrm{crs} \leftarrow \mathsf{HGen}(1^\lambda)$, and $(\mathrm{pk}, \cdot) \leftarrow \mathsf{E.Gen}(1^\lambda)$.
2. Obliviously (of $\widetilde{\mathrm{pv}}_i$) generate the patches: For every $i \in [N]$, compute
$$(\{\mathsf{patch}_i\}_{i\in[N]}, \mathsf{o.state}) \leftarrow \mathsf{OPatchGen}(1^\lambda, \mathsf{F.params}))$$
3. Compute $\widetilde{\mathsf{KD\text{-}Prog}}$ as follows.
   - Then, for every $i \in [N]$, sample $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$ as follows and store $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$: sample $\mathrm{pt}_i \leftarrow \{0,1\}^\ell$, encrypt pt_i as $\mathrm{ct}_i \leftarrow \mathsf{Enc}(\mathrm{pk}, \mathrm{pt}_i)$, and commit to ct_i as $\mathrm{com}_i = \mathsf{HCommit}(\mathrm{crs}, \mathrm{ct}_i; \mathrm{open}_i)$. Set $\widetilde{\mathrm{sv}}_i = (\mathrm{ct}_i, \mathrm{open}_i)$ and $\widetilde{\mathrm{pv}}_i = \mathrm{com}_i$.
   - Recall that $\widetilde{\mathrm{pv}}_i \in \{0,1\}^n$. Using $\mathsf{OPuncture}$, consistent with the already generated patches and with all $\widetilde{\mathrm{pv}}_i$, generate a key punctured at $2^{\widetilde{\mathrm{pv}}}_G$, where, $\widetilde{\mathrm{pv}} := (\widetilde{\mathrm{pv}}_1, \ldots, \widetilde{\mathrm{pv}}_N)$:
$$K[2^{\widetilde{\mathrm{pv}}}_G] \leftarrow \mathsf{OPuncture}(1^\lambda, \mathsf{o.state}, \widetilde{\mathrm{pv}})$$
   - $(K[2^{\widetilde{\mathrm{pv}}}_G], \{\mathsf{patch}_i\}_{i\in[N]})$, generate $\mathsf{KD\text{-}Prog}_3$ exactly the same way as $\mathsf{KD\text{-}Prog}_2$ and compute $\widetilde{\mathsf{KD\text{-}Prog}} \leftarrow i\mathcal{O}(\mathsf{KD\text{-}Prog}_3)$.

Thus, set $\mathsf{params} = (\widetilde{\mathsf{KD\text{-}Prog}}, \mathsf{F.params}, \mathrm{crs}, \mathrm{pk})$ and give it to $\mathcal{A}$. Then respond to the adversary's queries as follows.

4. Upon receiving $\mathsf{Reg}(i \in [N])$, respond via $\mathrm{pv}_i = \widetilde{\mathrm{pv}}_i$. (On the other hand, if the adversary himself registers a (corrupted) party P^*_i by presenting $\mathsf{RegCorr}(i \in [N], \mathrm{pv}_i)$, then simply ignore $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$.)
5. Upon receiving $\mathsf{Ext}(i)$ for a registered honest P_i, respond via $\mathrm{sv}_i = \widetilde{\mathrm{sv}}_i$.
6. Upon receiving $\mathsf{Rev}(\mathcal{S}, j)$, run $\widetilde{\mathsf{KD\text{-}Prog}}$ on input $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, j, \mathrm{sv}_j)$ and respond via the output.
7. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, choose $j \leftarrow \mathsf{ChQ}$, compute $Y_{\text{REAL}} := \widetilde{\mathsf{KD\text{-}Prog}}(\mathsf{ChQ}, (\mathrm{pv}_i)_{i\in\mathsf{ChQ}}, j, \mathrm{sv}_j)$ and sample $Y_{\text{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
8. Finally, output whatever $\mathcal{A}$ outputs.

Lemma 3. $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_3$.

Proof. Observe that the only difference between Hyb_2 and Hyb_3 is in the *sequence* in which the challenger computes the values. Namely, in Hyb_3, the challenger first samples the patches obliviously of $\widetilde{\mathrm{pv}}_i$ using the algorithm $\mathsf{OPatchGen}$, while in the previous hybrid, the patches were generated as a function of $\widetilde{\mathrm{pv}}_i$ using the algorithm $\mathsf{PatchGen}$. Furthermore, in Hyb_3, the punctured key $K[2^{\widetilde{\mathrm{pv}}}_G]$ is sampled using the algorithm $\mathsf{OPuncture}$ using the $\mathsf{o.state}$ information generated by the algorithm $\mathsf{OPatchGen}$ that had obliviously generated the patches; on the other hand, in the previous hybrid, $K[2^{\widetilde{\mathrm{pv}}}_G]$ is generated by running $\mathsf{F.KeyGen}$ followed by the algorithm $\mathsf{Puncture}$. This difference

corresponds exactly to the two modes of generating a punctured key and its block-wise patches, that are guaranteed to give identical joint distributions, from the property of *oblivious patchability* of F. Thus, we have that, the view of the adversary in Hyb_3 is identical to its view in Hyb_2. Hence, $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_3$. ■

Hyb_4. This experiment is the same as Hyb_3 except for the following modification in the distribution from which the challenger samples $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$ for all $i \in [N]$. Recall that in Hyb_3, $\widetilde{\mathrm{pv}}_i$ is a commitment to the ciphertext $\mathrm{ct}_i \leftarrow \mathsf{Enc}(\mathrm{pk}, \mathrm{pt}_i)$, where, pt_i is chosen uniformly at random. The modification we introduce in this hybrid, more specifically, is in the way pt_i is sampled: here, we set pt_i to be patch_i. (Observe that in Hyb_3, the challenger computes patch_i before sampling $\widetilde{\mathrm{pv}}_i$. Hence, this modification is well-defined.) Details follow.

The challenger behaves as follows. It begins by choosing $\textsc{expt} \in \{\textsc{real}, \textsc{rand}\}$ uniformly at random, and executes with $\mathcal{A}$ the experiment $\textsc{expt}_{\mathcal{A}}^{\mathsf{Hyb}_4}$ defined as follows.

1. Run $\mathsf{F.params} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$, $\mathrm{crs} \leftarrow \mathsf{HGen}(1^\lambda)$, and $(\mathrm{pk}, \cdot) \leftarrow \mathsf{E.Gen}(1^\lambda)$.
2. Obliviously (of $\widetilde{\mathrm{pv}}_i$) generate the patches: For every $i \in [N]$, compute
$$(\{\mathsf{patch}_i\}_{i\in[N]}, \mathsf{o.state}) \leftarrow \mathsf{OPatchGen}(1^\lambda, \mathsf{F.params}))$$
3. Compute $\widetilde{\mathsf{KD\text{-}Prog}}$ as follows.
   - Then, for every $i \in [N]$, sample $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$ as follows and store $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$: set $\mathrm{pt}_i \leftarrow \mathsf{patch}_i$, encrypt pt_i as $\mathrm{ct}_i \leftarrow \mathsf{Enc}(\mathrm{pk}, \mathrm{pt}_i)$, and commit to ct_i as $\mathrm{com}_i = \mathsf{HCommit}(\mathrm{crs}, \mathrm{ct}_i; \mathrm{open}_i)$. Set $\widetilde{\mathrm{sv}}_i = (\mathrm{ct}_i, \mathrm{open}_i)$ and $\widetilde{\mathrm{pv}}_i = \mathrm{com}_i$.
   - Recall that $\widetilde{\mathrm{pv}}_i \in \{0,1\}^n$. Using $\mathsf{OPuncture}$, consistent with the already generated patches and with all $\widetilde{\mathrm{pv}}_i$, generate a key punctured at $2^{\widetilde{\mathrm{pv}}}_G$, where, $\widetilde{\mathrm{pv}} := (\widetilde{\mathrm{pv}}_1, \ldots, \widetilde{\mathrm{pv}}_N)$:
$$K[2^{\widetilde{\mathrm{pv}}}_G] \leftarrow \mathsf{OPuncture}(1^\lambda, \mathsf{o.state}, \widetilde{\mathrm{pv}})$$
   - For $(K[2^{\widetilde{\mathrm{pv}}}_G], \{\mathsf{patch}_i\}_{i\in[N]})$, generate $\mathsf{KD\text{-}Prog}_4$ exactly the same way as $\mathsf{KD\text{-}Prog}_3$ and compute $\widetilde{\mathsf{KD\text{-}Prog}} \leftarrow i\mathcal{O}(\mathsf{KD\text{-}Prog}_4)$.

Thus, set $\mathsf{params} = (\widetilde{\mathsf{KD\text{-}Prog}}, \mathsf{F.params}, \mathrm{crs}, \mathrm{pk})$ and give it to $\mathcal{A}$. Then respond to the adversary's queries as follows.

4. Upon receiving $\mathsf{Reg}(i \in [N])$, respond via $\mathrm{pv}_i = \widetilde{\mathrm{pv}}_i$. (On the other hand, if the adversary himself registers a (corrupted) party P_i^* by presenting $\mathsf{RegCorr}(i \in [N], \mathrm{pv}_i)$, then simply ignore $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$.)
5. Upon receiving $\mathsf{Ext}(i)$ for a registered honest P_i, respond via $\mathrm{sv}_i = \widetilde{\mathrm{sv}}_i$.
6. Upon receiving $\mathsf{Rev}(\mathcal{S}, j)$, run $\widetilde{\mathsf{KD\text{-}Prog}}$ on input $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, j, \mathrm{sv}_j)$ and respond via the output.
7. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, choose $j \leftarrow \mathsf{ChQ}$, compute $Y_{\textsc{real}} := \widetilde{\mathsf{KD\text{-}Prog}}(\mathsf{ChQ}, (\mathrm{pv}_i)_{i\in\mathsf{ChQ}}, j, \mathrm{sv}_j)$ and sample $Y_{\textsc{rand}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via $Y_{\textsc{expt}}$.
8. Finally, output whatever $\mathcal{A}$ outputs.

Lemma 4. $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_4$.

Proof. Observe that the only difference between Hyb_3 and Hyb_4 is in the distribution from which the challenger samples plaintexts pt_i in the process of computing $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$ for all $i \in [N]$. Intuitively, since these plaintexts are encrypted, and since neither the secret key nor the random coins used in encrypting is used anywhere else in the execution of either hybrid, we will be able to argue that owing to CPA security of the encryption scheme $\Sigma = (\mathsf{E.Gen}, \mathsf{Enc}, \mathsf{Dec})$, the two hybrids are computationally indistinguishable.

Let $\mathcal{A}$ be an adversary that distinguishes Hyb_3 and Hyb_4; that is, $|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_3}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_4}(\lambda)|$ is non-negligible in λ, where, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_i}(\lambda)$ is as defined in Equation (6). Recall that, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_i}(\lambda) := |\Pr[\text{REAL}_{\mathcal{A}}^{\mathsf{Hyb}_i} \to 1] - \Pr[\text{RAND}_{\mathcal{A}}^{\mathsf{Hyb}_i} \to 1]|$. This implies that there exists $\widetilde{\text{EXPT}} \in \{\text{REAL}, \text{RAND}\}$ such that $|\Pr[\widetilde{\text{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_3} \to 1] - \Pr[\widetilde{\text{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_4} \to 1]| = \epsilon$, where, $\epsilon = \epsilon(\lambda)$ is non-negligible in λ. We shall build an adversary $\mathcal{B}$ that emulates either $\widetilde{\text{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_3}$ or $\widetilde{\text{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_4}$ and breaks the CPA security of Σ. Details follow.

For simplicity, we consider a variant of the standard CPA game that is equivalent to the standard CPA game through a simple hybrid argument which loses a factor of $1/N$ in the adversary's advantage. The modified game is as follows. Upon receiving the public key pk, the adversary $\mathcal{B}$ gives to the challenger two N-vectors of plaintext messages and receives an N-vector of ciphertexts that encrypts one of the two plaintext vectors. The objective of the adversary would be to guess which plaintext vector was encrypted. Note that neither the secret key nor the random coins used in encrypting are used anywhere else in the either of the hybrids Hyb_3 and Hyb_4; hence, the description of $\mathcal{B}$ is well-defined.

Description of $\mathcal{B}$. Corresponding respectively to the two plaintext vectors, the adversary $\mathcal{B}$ emulates to $\mathcal{A}$ either $\widetilde{\text{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_3}$ or $\widetilde{\text{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_4}$, respectively, and exploits the success probability of $\mathcal{A}$ in the CPA game. Let the experiment with $\mathcal{A}$ as emulated by $\mathcal{B}$ be denoted by $\widetilde{\text{EXPT}}_{3-4}$. Upon receiving pk from its challenger, $\mathcal{B}$ proceeds as follows.

1. Run $\mathsf{F.params} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$ and $\text{crs} \leftarrow \mathsf{HGen}(1^\lambda)$.
2. Obliviously (of $\widetilde{\text{pv}}_i$) generate the patches: For every $i \in [N]$, compute
$$(\{\mathsf{patch}_i\}_{i\in[N]}, \mathsf{o.state}) \leftarrow \mathsf{OPatchGen}(1^\lambda, \mathsf{F.params}))$$
3. Compute $\widetilde{\mathsf{KD\text{-}Prog}}$ as follows.
   - For every $i \in [N]$, compute $(\widetilde{\text{pv}}_i, \widetilde{\text{sv}}_i)$ as follows and store $(\widetilde{\text{pv}}_i, \widetilde{\text{sv}}_i)$: Firstly, compute two N-vectors of plaintexts $(\text{pt}_1^{(3)}, \ldots, \text{pt}_N^{(3)})$ and $(\text{pt}_1^{(4)}, \ldots, \text{pt}_N^{(4)})$ as follows.
     - For every $i \in [N]$, sample $\text{pt}_i^{(3)} \leftarrow \{0,1\}^\ell$.
     - For every $i \in [N]$, set $\text{pt}_i^{(4)} \leftarrow \mathsf{patch}_i$.

     Present the two N-vectors of plaintexts, $(\text{pt}_1^{(3)}, \ldots, \text{pt}_N^{(3)})$ and $(\text{pt}_1^{(4)}, \ldots, \text{pt}_N^{(4)})$ to the CPA challenger. The CPA challenger flips a coin $b \in \{0,1\}$ and if $b = 0$ encrypts $(\text{pt}_1^{(3)}, \ldots, \text{pt}_N^{(3)})$ and if $b = 1$ encrypts $(\text{pt}_1^{(4)}, \ldots, \text{pt}_N^{(4)})$. He gives the resultant N-vector of ciphertexts $(\text{ct}_1, \ldots, \text{ct}_N)$ to $\mathcal{B}$. $\mathcal{B}$ then commits to these ciphertexts $\text{com}_i = \mathsf{HCommit}(\text{crs}, \text{ct}_i; \text{open}_i)$ and sets $\widetilde{\text{sv}}_i = (\text{ct}_i, \text{open}_i)$ and $\widetilde{\text{pv}}_i = \text{com}_i$.
   - Recall that $\widetilde{\text{pv}}_i \in \{0,1\}^n$. Using $\mathsf{OPuncture}$, consistent with the already generated patches and with all $\widetilde{\text{pv}}_i$, generate a key punctured at $2_G^{\widetilde{\text{pv}}}$, where, $\widetilde{\text{pv}} := (\widetilde{\text{pv}}_1, \ldots, \widetilde{\text{pv}}_N)$:
$$K[2_G^{\widetilde{\text{pv}}}] \leftarrow \mathsf{OPuncture}(1^\lambda, \mathsf{o.state}, \widetilde{\text{pv}})$$
   - For $(K[2_G^{\widetilde{\text{pv}}}], \{\mathsf{patch}_i\}_{i\in[N]})$, generate $\mathsf{KD\text{-}Prog}_{3-4}$ exactly the same way as $\mathsf{KD\text{-}Prog}_3$ and compute $\widetilde{\mathsf{KD\text{-}Prog}} \leftarrow i\mathcal{O}(\mathsf{KD\text{-}Prog}_{3-4})$.

Thus, set $\mathsf{params} = (\widetilde{\mathsf{KD\text{-}Prog}}, \mathsf{F.params}, \text{crs}, \text{pk})$ and give it to $\mathcal{A}$. Then respond to the adversary's queries as follows.

4. Upon receiving $\mathsf{Reg}(i \in [N])$, respond via $\mathrm{pv}_i = \widetilde{\mathrm{pv}}_i$. (On the other hand, if the adversary himself registers a (corrupted) party P_i^* by presenting $\mathsf{RegCorr}(i \in [N], \mathrm{pv}_i)$, then simply ignore $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$.)

5. Upon receiving $\mathsf{Ext}(i)$ for a registered honest P_i, respond via $\mathrm{sv}_i = \widetilde{\mathrm{sv}}_i$.

6. Upon receiving $\mathsf{Rev}(\mathcal{S}, j)$, run $\widetilde{\mathsf{KD\text{-}Prog}}$ on input $(\mathcal{S}, (\mathrm{pv}_i)_{i \in \mathcal{S}}, j, \mathrm{sv}_j)$ and respond via the output.

7. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, choose $j \leftarrow \mathsf{ChQ}$, compute $Y_{\text{REAL}} := \widetilde{\mathsf{KD\text{-}Prog}}(\mathsf{ChQ}, (\mathrm{pv}_i)_{i \in \mathsf{ChQ}}, j, \mathrm{sv}_j)$ and sample $Y_{\text{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via $Y_{\widetilde{\text{EXPT}}}$.

Finally, output whatever $\mathcal{A}$ outputs.

Analysis. Before we proceed, recall that the programs $\mathsf{KD\text{-}Prog}_3$ and $\mathsf{KD\text{-}Prog}_4$ are identical. Hence, the only difference between $\widetilde{\text{EXPT}}$ in Hyb_3 and the experiment $\widetilde{\text{EXPT}}_{3-4}$ emulated by $\widetilde{\text{EXPT}}_{3-4}$ is in the distribution from which pt_i are sampled. Thus, if the CPA challenger encrypts $(\mathrm{pt}_1^{(3)}, \ldots, \mathrm{pt}_N^{(3)})$, then the view of $\mathcal{A}$ in $\widetilde{\text{EXPT}}_{3-4}$ is identical to that in $\widetilde{\text{EXPT}}$ of Hyb_3; on the other hand, if the CPA challenger encrypts $(\mathrm{pt}_1^{(4)}, \ldots, \mathrm{pt}_N^{(4)})$, then the view of $\mathcal{A}$ in $\widetilde{\text{EXPT}}_{3-4}$ is identical to that in $\widetilde{\text{EXPT}}$ of Hyb_4. Therefore, $\mathcal{B}$ distinguishes the ciphertexts of two distinct plaintexts with probability $|\Pr[\widetilde{\text{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_3} \to 1] - \Pr[\widetilde{\text{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_4} \to 1]| = \epsilon$ which by assumption is non-negligible, hence breaking the CPA security of Σ. ■

Hyb_5. This experiment is the same as Hyb_4 except for the following modification in the way the program $\mathsf{KD\text{-}Prog}_5$ obtains patch_j. Recall that in $\mathsf{KD\text{-}Prog}_4$, $\{\mathsf{patch}_i\}_{i\in[N]}$ formed a part of the constants of the program. Furthermore, recall that for an input $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, j, \mathrm{sv}_j)$ where $\mathrm{pv}_j = \widetilde{\mathrm{pv}}_j$ (and where the checks performed in the program $\mathsf{KD\text{-}Prog}_4$ go through), the output of the program is $\mathsf{Eval}((K[2^{\widetilde{\mathrm{pv}}}_G], \mathsf{patch}_j), (\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}))$. Thus, roughly speaking, to compute the output of an input $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, j, \mathrm{sv}_j)$, the program $\mathsf{KD\text{-}Prog}_4$ does not need any of $\{\mathsf{patch}_i\}_{i\neq j}$. Keeping this in mind, the modification we introduce here is that in $\mathsf{KD\text{-}Prog}_5$, we do not provide any of $\{\mathsf{patch}_i\}_{i\in[N]}$ as constants to the program. Instead, we include sk of the encryption scheme as a part of the constants in the program. Then, upon an input $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, j, \mathrm{sv}_j)$, where, $\mathrm{pv}_j = \widetilde{\mathrm{pv}}_j$ and $\mathrm{sv}_j = (\mathrm{ct}_j; \mathrm{open}_j)$ (an opening of the commitment $\mathrm{pv}_j = \mathrm{com}_j$), the program $\mathsf{KD\text{-}Prog}_5$ would decrypt ct_j to get patch_j, and compute and output $\mathsf{Eval}((K[2^{\widetilde{\mathrm{pv}}}_G], \mathsf{patch}_j), (\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}))$. The crucial point which will be necessary in order to establish indistinguishability from the previous hybrid is that for every such input, ct_j is indeed an encryption of the required patch. Details follow.

The challenger behaves as follows. It begins by choosing $\text{EXPT} \in \{\text{REAL}, \text{RAND}\}$ uniformly at random, and executes with $\mathcal{A}$ the experiment $\text{EXPT}^{\mathsf{Hyb}_5}_{\mathcal{A}}$ defined as follows.

1. Run $\mathsf{F.params} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$, $\mathrm{crs} \leftarrow \mathsf{HGen}(1^\lambda)$, and $(\mathrm{pk}, \underline{\mathrm{sk}}) \leftarrow \mathsf{E.Gen}(1^\lambda)$.
2. Obliviously (of $\widetilde{\mathrm{pv}}_i$) generate the patches: For every $i \in [N]$, compute
$$(\{\mathsf{patch}_i\}_{i\in[N]}, \mathsf{o.state}) \leftarrow \mathsf{OPatchGen}(1^\lambda, \mathsf{F.params}))$$
3. Compute $\widetilde{\mathsf{KD\text{-}Prog}}$ as follows.
   - Then, for every $i \in [N]$, sample $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$ as follows and store $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$: set $\mathrm{pt}_i \leftarrow \mathsf{patch}_i$, encrypt pt_i as $\mathrm{ct}_i \leftarrow \mathsf{Enc}(\mathrm{pk}, \mathrm{pt}_i)$, and commit to ct_i as $\mathrm{com}_i = \mathsf{HCommit}(\mathrm{crs}, \mathrm{ct}_i; \mathrm{open}_i)$. Set $\widetilde{\mathrm{sv}}_i = (\mathrm{ct}_i, \mathrm{open}_i)$ and $\widetilde{\mathrm{pv}}_i = \mathrm{com}_i$.
   - Recall that $\widetilde{\mathrm{pv}}_i \in \{0,1\}^n$. Using $\mathsf{OPuncture}$, consistent with the already generated patches and with all $\widetilde{\mathrm{pv}}_i$, generate a key punctured at $2^{\widetilde{\mathrm{pv}}}_G$, where, $\widetilde{\mathrm{pv}} := (\widetilde{\mathrm{pv}}_1, \ldots, \widetilde{\mathrm{pv}}_N)$:
   $$K[2^{\widetilde{\mathrm{pv}}}_G] \leftarrow \mathsf{OPuncture}(1^\lambda, \mathsf{o.state}, \widetilde{\mathrm{pv}})$$
   - For $(K[2^{\widetilde{\mathrm{pv}}}_G], \{\mathsf{patch}_i\}_{i\in[N]})$, generate $\mathsf{KD\text{-}Prog}_5$ exactly the same way as $\mathsf{KD\text{-}Prog}_4$ and compute $\widetilde{\mathsf{KD\text{-}Prog}} \leftarrow \mathsf{i}\mathcal{O}(\mathsf{KD\text{-}Prog}_5)$.

Thus, set $\mathsf{params} = (\widetilde{\mathsf{KD\text{-}Prog}}, \mathsf{F.params}, \mathrm{crs}, \mathrm{pk})$ and give it to $\mathcal{A}$. Then respond to the adversary's queries as follows.

4. Upon receiving $\mathsf{Reg}(i \in [N])$, respond via $\mathrm{pv}_i = \widetilde{\mathrm{pv}}_i$. (On the other hand, if the adversary himself registers a (corrupted) party P^*_i by presenting $\mathsf{RegCorr}(i \in [N], \mathrm{pv}_i)$, then simply ignore $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$.)
5. Upon receiving $\mathsf{Ext}(i)$ for a registered honest P_i, respond via $\mathrm{sv}_i = \widetilde{\mathrm{sv}}_i$.
6. Upon receiving $\mathsf{Rev}(\mathcal{S}, j)$, run $\widetilde{\mathsf{KD\text{-}Prog}}$ on input $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, j, \mathrm{sv}_j)$ and respond via the output.
7. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, choose $j \leftarrow \mathsf{ChQ}$, compute $Y_{\text{REAL}} := \widetilde{\mathsf{KD\text{-}Prog}}(\mathsf{ChQ}, (\mathrm{pv}_i)_{i\in\mathsf{ChQ}}, j, \mathrm{sv}_j)$ and sample $Y_{\text{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
8. Finally, output whatever $\mathcal{A}$ outputs.

$\mathsf{KD\text{-}Prog}_5$

Constants:

- $\mathsf{F.params}, \mathrm{crs}, \mathrm{pk}$
- $\{\widetilde{\mathrm{pv}}_i\}_{i\in[N]}$
- $(K[2^{\widetilde{\mathrm{pv}}}_G],$ ~~$\{\mathsf{patch}_i\}_{i\in[N]}$~~$)$
- sk

Input: $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, j, \mathrm{sv}_j)$, where $\mathcal{S} \subseteq [N]$, $|\mathcal{S}| \leq G$, where, $\mathrm{sv}_j = (\mathrm{ct}_j, \mathrm{open}_j)$.
Procedure:

1. Check whether $j \in \mathcal{S}$; if not, output $\perp$. Then, check whether $(\mathrm{pv}_j, \mathrm{sv}_j) \in \mathrm{R}^{\mathsf{HCOM}}$; if not, output $\perp$. Otherwise, proceed as follows.
2. - If $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}) \notin 2^{\widetilde{\mathrm{pv}}}_G$, then compute and output $\mathsf{Eval}(K[2^{\widetilde{\mathrm{pv}}}_G], (\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}))$.
   - If $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}) \in 2^{\widetilde{\mathrm{pv}}}_G$, then obtain patch_j by decrypting ct_j as $\mathsf{patch}_j \leftarrow \mathsf{Dec}(\mathrm{sk}, \mathrm{ct}_j)$, compute and output $\mathsf{Eval}((K[2^{\widetilde{\mathrm{pv}}}_G], \mathsf{patch}_j), (\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}))$.

Figure 6: Key-derivation Program

Lemma 5. $\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_5$.

Proof. Observe that the only difference between the two hybrids is in the way the programs $\mathsf{KD\text{-}Prog}_4$ and $\mathsf{KD\text{-}Prog}_5$ behave when input $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, j, \mathrm{sv}_j)$ which goes through the check performed in the first step and for which $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}) \in 2^{\widetilde{\mathrm{pv}}}_G$. More specifically, upon such an input, the output is computed as $\mathsf{Eval}((K[2^{\widetilde{\mathrm{pv}}}_G], \mathsf{patch}_j), (\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}))$ in both the programs; however, the difference is in how the programs obtain patch_j. In the former hybrid, this value formed a part of the constants within the program. However, in the latter hybrid, this is no longer a part of the program's constants; the program instead obtains it as follows. Parse $\widetilde{\mathrm{sv}}_i \leftarrow (\mathrm{ct}_i, \mathrm{open}_i)$ and $\widetilde{\mathrm{pv}}_i \leftarrow \mathrm{com}_i$. Then obtain patch_j by decrypting ct_j as $\mathsf{patch}_j \leftarrow \mathsf{Dec}(\mathrm{sk}, \mathrm{ct}_j)$.

Thus, if we show that, despite the disparate ways of obtaining patch_j, the distribution of patch_j is distributed the same way w.r.t. to the rest of the elements in the game, then we can conclude that the programs are functionally equivalent.

In Hyb_5, recall how the challenger samples $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$ for every i. The challenger first computes patch_i for all i (just like in Hyb_4). Then it encrypts patch_i with pk, denoting the plaintext patch_i by pt_i and the resulting ciphertext by ct_i. Then, it commits to ct_i with random coins open_i to get com_i, assigning $\widetilde{\mathrm{sv}}_i \leftarrow (\mathrm{ct}_i, \mathrm{open}_i)$ and $\widetilde{\mathrm{pv}}_i \leftarrow \mathrm{com}_i$. The crucial fact for our current interest is that $\widetilde{\mathrm{sv}}_j = (\mathrm{ct}_j, \cdot)$, where, ct_j is an encryption of patch_j. Next, recall that the program $\mathsf{KD\text{-}Prog}_5$ obtains patch_j by decrypting ct_j with sk. Thus, we will be able to argue function equivalence of the two programs $\mathsf{KD\text{-}Prog}_4$ and $\mathsf{KD\text{-}Prog}_5$, provided, for every input $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, j, \mathrm{sv}_j)$ that passes the check in the program, if $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}) \in 2^{\widetilde{\mathrm{pv}}}_G$, then $\mathrm{sv}_j = (\mathrm{ct}_j, \cdot)$, where ct_j is an encryption of patch_j.

We now prove that, for every $j \in [N]$, for every $\widetilde{\mathrm{pv}}_j$, there exists only one value for sv_j such that for an input $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, j, \mathrm{sv}_j)$ (with with $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}) \in 2^{\widetilde{\mathrm{pv}}}_G$ which implies that $\mathrm{pv}_j = \widetilde{\mathrm{pv}}_j$), the check (namely, $\mathrm{R}^{\mathsf{HCOM}}(\mathrm{pv}_j, \mathrm{sv}_j) = \mathsf{HVer}(\mathrm{crs}, \mathrm{pv}_j, \mathrm{sv}_j) \stackrel{?}{=} 1$, as per Equation (2)) in Step 1 in program $\mathsf{KD\text{-}Prog}_5$ passes. Towards this end, observe that the commitment $\mathrm{pv}_j = \mathrm{com}_j$ is computed using

crs that is sampled as $\mathrm{crs} \leftarrow \mathsf{HGen}(1^\lambda)$. Now recall that, with all but negligible probability, for crs that is sampled using HGen, the resulting commitment scheme is unconditionally binding. Hence, there can exist only one $\mathrm{sv}_j = (\mathrm{ct}_j, \mathrm{open}_j)$ such that $\mathrm{com}_j = \mathsf{HCommit}(\mathrm{crs}, \mathrm{ct}_j; \mathrm{open}_j)$. Thus, on such an input, the program $\mathsf{KD\text{-}Prog}_5$ is assured to obtain patch_j with all but negligible probability, hence maintaining functional equivalence with $\mathsf{KD\text{-}Prog}_4$ even for the case when $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}) \in 2^{\widetilde{\mathrm{pv}}}_G$.

In conclusion, with all but negligible probability, the programs $\mathsf{KD\text{-}Prog}_4$ and $\mathsf{KD\text{-}Prog}_5$ are functionally equivalent, thus allowing us to apply security of $\mathsf{i}\mathcal{O}$ and concluding that $\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_5$. ■

Hyb_6. This experiment is the same as Hyb_5 except for the following modification in the way crs for the commitment scheme is sampled. Recall that the challenger in Hyb_5 sampled crs as $\text{crs} \leftarrow \mathsf{HGen}(1^\lambda)$. Now the modification for Hyb_6 is that the challenger samples crs as $(\text{crs}, \text{aux}) \leftarrow \mathsf{HTGen}(1^\lambda)$. Details follow.

The challenger behaves as follows. It begins by choosing $\text{EXPT} \in \{\text{REAL}, \text{RAND}\}$ uniformly at random, and executes with $\mathcal{A}$ the experiment $\text{EXPT}_{\mathcal{A}}^{\mathsf{Hyb}_6}$ defined as follows.

1. Run $\mathsf{F.params} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$, $(\text{crs}, \text{aux}) \leftarrow \mathsf{HTGen}(1^\lambda)$, and $(\text{pk}, \text{sk}) \leftarrow \mathsf{E.Gen}(1^\lambda)$. (Ignore aux).
2. Obliviously (of $\widetilde{\text{pv}}_i$) generate the patches: For every $i \in [N]$, compute
$$(\{\mathsf{patch}_i\}_{i\in[N]}, \mathsf{o.state}) \leftarrow \mathsf{OPatchGen}(1^\lambda, \mathsf{F.params}))$$
3. Compute $\widetilde{\mathsf{KD\text{-}Prog}}$ as follows.
   - Then, for every $i \in [N]$, sample $(\widetilde{\text{pv}}_i, \widetilde{\text{sv}}_i)$ as follows and store $(\widetilde{\text{pv}}_i, \widetilde{\text{sv}}_i)$: set $\text{pt}_i \leftarrow \mathsf{patch}_i$, encrypt pt_i as $\text{ct}_i \leftarrow \mathsf{Enc}(\text{pk}, \text{pt}_i)$, and commit to ct_i as $\text{com}_i = \mathsf{HCommit}(\text{crs}, \text{ct}_i; \text{open}_i)$. Set $\widetilde{\text{sv}}_i = (\text{ct}_i, \text{open}_i)$ and $\widetilde{\text{pv}}_i = \text{com}_i$.
   - Recall that $\widetilde{\text{pv}}_i \in \{0,1\}^n$. Using $\mathsf{OPuncture}$, consistent with the already generated patches and with all $\widetilde{\text{pv}}_i$, generate a key punctured at $2^{\widetilde{\text{pv}}}_G$, where, $\widetilde{\text{pv}} := (\widetilde{\text{pv}}_1, \ldots, \widetilde{\text{pv}}_N)$:
   $$K[2^{\widetilde{\text{pv}}}_G] \leftarrow \mathsf{OPuncture}(1^\lambda, \mathsf{o.state}, \widetilde{\text{pv}})$$
   - For $(K[2^{\widetilde{\text{pv}}}_G], \{\mathsf{patch}_i\}_{i\in[N]})$, generate $\mathsf{KD\text{-}Prog}_6$ exactly the same way as $\mathsf{KD\text{-}Prog}_5$ and compute $\widetilde{\mathsf{KD\text{-}Prog}} \leftarrow i\mathcal{O}(\mathsf{KD\text{-}Prog}_6)$.

Thus, set $\mathsf{params} = (\widetilde{\mathsf{KD\text{-}Prog}}, \mathsf{F.params}, \text{crs}, \text{pk})$ and give it to $\mathcal{A}$. Then respond to the adversary's queries as follows.

4. Upon receiving $\mathsf{Reg}(i \in [N])$, respond via $\text{pv}_i = \widetilde{\text{pv}}_i$. (On the other hand, if the adversary himself registers a (corrupted) party P_i^* by presenting $\mathsf{RegCorr}(i \in [N], \text{pv}_i)$, then simply ignore $(\widetilde{\text{pv}}_i, \widetilde{\text{sv}}_i)$.)
5. Upon receiving $\mathsf{Ext}(i)$ for a registered honest P_i, respond via $\text{sv}_i = \widetilde{\text{sv}}_i$.
6. Upon receiving $\mathsf{Rev}(\mathcal{S}, j)$, run $\widetilde{\mathsf{KD\text{-}Prog}}$ on input $(\mathcal{S}, (\text{pv}_i)_{i\in\mathcal{S}}, j, \text{sv}_j)$ and respond via the output.
7. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, choose $j \leftarrow \mathsf{ChQ}$, compute $Y_{\text{REAL}} := \widetilde{\mathsf{KD\text{-}Prog}}(\mathsf{ChQ}, (\text{pv}_i)_{i\in\mathsf{ChQ}}, j, \text{sv}_j)$ and sample $Y_{\text{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
8. Finally, output whatever $\mathcal{A}$ outputs.

Lemma 6. $\mathsf{Hyb}_5 \approx_c \mathsf{Hyb}_6$.

Proof. Observe that the only difference in Hyb_5 and Hyb_6 is that while the challenger in Hyb_5 samples the CRS for the commitment scheme using HGen $\text{crs}^{(5)} \leftarrow \mathsf{HGen}(1^\lambda)$, the challenger in Hyb_5 samples it using HTGen as $(\text{crs}^{(6)}, \text{aux}) \leftarrow \mathsf{HTGen}(1^\lambda)$, (while ignoring aux in the rest of the experiment).

We shall show that $\mathsf{Hyb}_5 \approx_c \mathsf{Hyb}_6$ by applying the 'hybrid property' of the hybrid trapdoor commitment scheme HCOM (See Definition 4).

Assume for contradiction that there exists an adversary $\mathcal{A}$ that distinguishes the two hybrids Hyb_5 and Hyb_6; that is, $|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_5}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_6}(\lambda)|$ is non-negligible in λ, where, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_i}(\lambda)$ is as defined in Equation (6). Recall that, for any i, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_i}(\lambda) := |\Pr[\text{REAL}_{\mathcal{A}}^{\mathsf{Hyb}_i} \to 1] - \Pr[\text{RAND}_{\mathcal{A}}^{\mathsf{Hyb}_i} \to 1]|$. This implies that there exists $\widetilde{\text{EXPT}} \in \{\text{REAL}, \text{RAND}\}$ such that $|\Pr[\widetilde{\text{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_5} \to 1] - \Pr[\widetilde{\text{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_6} \to 1]| = \epsilon$, where, $\epsilon = \epsilon(\lambda)$ is non-negligible in λ. We shall build an adversary $\mathcal{B}$ that emulates either $\widetilde{\text{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_5}$ or $\widetilde{\text{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_6}$ and breaks the hybrid property of HCOM. Details follow.

Corresponding respectively to the two distributions of crs, the adversary $\mathcal{B}$ emulates to $\mathcal{A}$ either $\widetilde{\text{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_5}$ or $\widetilde{\text{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_6}$, respectively, and exploits the success probability of $\mathcal{A}$ in its own hybrid security game. Let the experiment with $\mathcal{A}$ as emulated by $\mathcal{B}$ be denoted by $\widetilde{\text{EXPT}}_{5-6}$.

From its challenger, $\mathcal{B}$ receives crs that is sampled using either HGen or HTGen. The objective of $\mathcal{B}$ is to tell the two cases apart. $\mathcal{B}$ proceeds as follows.

Adversary $\mathcal{B}$ runs $\mathcal{A}$ just like the challenger of Hyb_5 runs $\mathcal{A}$ except that, instead of sampling by itself the CRS for the commitment scheme, it uses crs that it received from its challenger. Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Observe that at no point in the execution of the Hyb_5 experiment do we need the random coins used in generating the crs. Hence, the adversary $\mathcal{B}$ is well-defined.

Now observe that when crs given by the challenger of $\mathcal{B}$ is sampled using HGen, then the view of $\mathcal{A}$ generated by $\mathcal{B}$ is identical to that generated by Hyb_5. On the other hand, when crs is sampled using HTGen, then the view of $\mathcal{A}$ generated by $\mathcal{B}$ is identical to that generated by Hyb_6. Hence,

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{B}}^{\mathsf{hyb}}(\lambda) &= |\Pr[1 \leftarrow \mathcal{B}(\text{crs}_0) \ : \ \text{crs}_0 \leftarrow \mathsf{HGen}(1^\lambda)] - \Pr[1 \leftarrow \mathcal{B}(\text{crs}_1) \ : \ \text{crs}_1 \leftarrow \mathsf{HTGen}(1^\lambda)]| \\
&= |\Pr[\widetilde{\text{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_5} \to 1] - \Pr[\widetilde{\text{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_6} \to 1]| \\
&= \epsilon
\end{aligned}$$

thus, arriving at a contradiction.

Hence, we have that $\mathsf{Hyb}_5 \approx_c \mathsf{Hyb}_6$.

■

Hyb_7. This experiment is the same as Hyb_6 except for the following modification in the way in which the challenger computes the public values $\widetilde{\mathrm{pv}}_i = \mathrm{com}_i$ for honest parties. Roughly speaking, instead of computing com_i as a commitment to a specific value (namely, an encryption of patch_i) using $\mathsf{HCommit}$ algorithm, the modification is that the challenger, using $\mathsf{HTCommit}$, first computes com_i, along with a trapdoor. Since the challenger is able to open com_i to any value, it opens it to an encryption of patch_i, later. Details follow.

1. Run $\mathsf{F.params} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$, $(\mathrm{crs}, \mathrm{aux}) \leftarrow \mathsf{HTGen}(1^\lambda)$, and $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathsf{E.Gen}(1^\lambda)$. ~~(Ignore aux).~~
2. Obliviously (of $\widetilde{\mathrm{pv}}_i$) generate the patches: For every $i \in [N]$, compute
$$(\{\mathsf{patch}_i\}_{i\in[N]}, \mathsf{o.state}) \leftarrow \mathsf{OPatchGen}(1^\lambda, \mathsf{F.params}))$$
3. Compute $\widetilde{\mathsf{KD\text{-}Prog}}$ as follows.
   - For every $i \in [N]$, compute $(\mathrm{com}_i, \mathrm{aux}_i) \leftarrow \mathsf{HTCommit}(\mathrm{crs}, \mathrm{aux})$. Set $\widetilde{\mathrm{pv}}_i \leftarrow \mathrm{com}_i$.
   - Recall that $\widetilde{\mathrm{pv}}_i \in \{0,1\}^n$. Using $\mathsf{OPuncture}$, consistent with the already generated patches and with all $\widetilde{\mathrm{pv}}_i$, generate a key punctured at $2^{\widetilde{\mathrm{pv}}}_G$, where, $\widetilde{\mathrm{pv}} := (\widetilde{\mathrm{pv}}_1, \ldots, \widetilde{\mathrm{pv}}_N)$:
   $$K[2^{\widetilde{\mathrm{pv}}}_G] \leftarrow \mathsf{OPuncture}(1^\lambda, \mathsf{o.state}, \widetilde{\mathrm{pv}})$$
   - For $(K[2^{\widetilde{\mathrm{pv}}}_G], \{\mathsf{patch}_i\}_{i\in[N]})$, generate $\mathsf{KD\text{-}Prog}_7$ exactly the same way as $\mathsf{KD\text{-}Prog}_6$ and compute $\widetilde{\mathsf{KD\text{-}Prog}} \leftarrow i\mathcal{O}(\mathsf{KD\text{-}Prog}_7)$.

Thus, set $\mathsf{params} = (\widetilde{\mathsf{KD\text{-}Prog}}, \mathsf{F.params}, \mathrm{crs}, \mathrm{pk})$ and give it to $\mathcal{A}$. Then respond to the adversary's queries as follows.

4. Upon receiving $\mathsf{Reg}(i \in [N])$, respond via $\mathrm{pv}_i = \widetilde{\mathrm{pv}}_i$. (On the other hand, if the adversary himself registers a (corrupted) party P_i^* by presenting $\mathsf{RegCorr}(i \in [N], \mathrm{pv}_i)$, then simply ignore $\widetilde{\mathrm{pv}}_i$.)
5. Upon receiving $\mathsf{Ext}(i)$ for a registered honest P_i, compute an encryption of patch_i as $\mathrm{ct}_i \leftarrow \mathsf{Enc}(\mathrm{pk}, \mathsf{patch}_i)$ as follows. Then, compute the opening of com_i to ct_i as $\mathrm{open}_i \leftarrow \mathsf{HTDec}(\mathrm{aux}_i, \mathrm{ct}_i)$, where aux_i was the auxiliary information created when com_i was generated earlier. With this, set $\mathrm{sv}_i = \widetilde{\mathrm{sv}}_i = (\mathrm{ct}_i, \mathrm{open}_i)$ and then respond to the adversary's query via sv_i.
6. Upon receiving $\mathsf{Rev}(\mathcal{S}, j)$, respond via $\widetilde{\mathsf{KD\text{-}Prog}}(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, k, \mathrm{sv}_k)$, where k is determined as follows:
   - Suppose there exists $k' \in \mathcal{S}$ such that $P_{k'}$ was initially registered as an honest party, but has already been corrupted (via query $\mathsf{Ext}(k')$) by the adversary. Then k is such an arbitrary, say the smallest, k'.
   - Otherwise, choose any arbitrary (say the smallest) $k \in \mathcal{S}$, and compute $\widetilde{\mathrm{sv}}_i$ as above.
7. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, choose $j \leftarrow \mathsf{ChQ}$, compute $\mathrm{sv}_j = \widetilde{\mathrm{sv}}_j$ as before, compute $Y_{\mathrm{REAL}} := \widetilde{\mathsf{KD\text{-}Prog}}(\mathsf{ChQ}, (\mathrm{pv}_i)_{i\in\mathsf{ChQ}}, j, \mathrm{sv}_j)$ and sample $Y_{\mathrm{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
8. Finally, output whatever $\mathcal{A}$ outputs.

Lemma 7. $\mathsf{Hyb}_6 \approx_c \mathsf{Hyb}_7$.

Proof. Observe that the only differences between Hyb_6 and Hyb_7 is in the way the challenger computes $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$. More specifically, recall that, while in both the hybrids, crs for the commitment scheme is sampled using the HTGen algorithm, the difference in the hybrid lies in the way this crs is used in computing the commitments. In detail:

- In Hyb_6, the challenger first computes ct_i, an encryption of patch_i, and then commits to ct_i as $\mathrm{com}_i = \mathsf{HCommit}(\mathrm{crs}, \mathrm{ct}_i; \mathrm{open}_i)$, finally setting $\widetilde{\mathrm{sv}}_i = (\mathrm{ct}_i, \mathrm{open}_i)$ and $\widetilde{\mathrm{pv}}_i = \mathrm{com}_i$.
- On the other hand, in Hyb_7, the challenger computes com_i as an equivocable commitment using $\mathsf{HTCommit}$ as $(\mathrm{com}_i, \mathrm{aux}_i) \leftarrow \mathsf{HTCommit}(\mathrm{crs}, \mathrm{aux})$, and then computes an opening of this commitment com_i to ct_i as $\mathrm{open}_i \leftarrow \mathsf{HTDec}(\mathrm{aux}_i, \mathrm{ct}_i)$, finally setting $\widetilde{\mathrm{sv}}_i = (\mathrm{ct}_i, \mathrm{open}_i)$ and $\widetilde{\mathrm{pv}}_i = \mathrm{com}_i$.

Observe that the above difference in the two hybrids Hyb_6 and Hyb_7 corresponds to the two cases in the security game for trapdoor property of the hybrid trapdoor commitment scheme HCOM. Thus, any adversary who distinguishes the two hybrids can be reduced to one that breaks the trapdoor property of HCOM, as we shall show below.

Let $\mathcal{A}$ be an adversary that distinguishes Hyb_6 and Hyb_7; that is, $|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_6}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_7}(\lambda)|$ is non-negligible in λ, where, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_i}(\lambda)$ is as defined in Equation (6). Recall that, for any i, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_i}(\lambda) := |\Pr[\mathrm{REAL}_{\mathcal{A}}^{\mathsf{Hyb}_i} \to 1] - \Pr[\mathrm{RAND}_{\mathcal{A}}^{\mathsf{Hyb}_i} \to 1]|$. This implies that there exists $\widetilde{\mathrm{EXPT}} \in \{\mathrm{REAL}, \mathrm{RAND}\}$ such that $|\Pr[\widetilde{\mathrm{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_6} \to 1] - \Pr[\widetilde{\mathrm{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_7} \to 1]| = \epsilon$, where, $\epsilon = \epsilon(\lambda)$ is non-negligible in λ. We shall build an adversary $\mathcal{A}_{\mathsf{trap}}$ that emulates either $\widetilde{\mathrm{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_6}$ or $\widetilde{\mathrm{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_7}$ and breaks the trapdoor property of HCOM. Details follow.

Recall from Definition 4 that $\mathcal{A}_{\mathsf{trap}}$ first receives $(\mathrm{crs}, \mathrm{aux})$ from the challenger. Then $\mathcal{A}_{\mathsf{trap}}$ gives to its challenger a message, upon which the challenger computes a commitment either using $\mathsf{HCommit}$ or $\mathsf{HTCommit}$ algorithm and gives $\mathcal{A}_{\mathsf{trap}}$ the resulting commitment and its opening to the message. The objective of the adversary is to tell apart the two cases. $\mathcal{A}_{\mathsf{trap}}$ is described as below.

$\mathcal{A}_{\mathsf{trap}}$ behaves the same way as the challenger in Hyb_6 except for the way it computes $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$ for all i. Namely, instead of computing $\widetilde{\mathrm{pv}}_i = \mathrm{com}_i$ by itself, it uses the commitments received from its challenger. In detail, upon receiving $(\mathrm{crs}, \mathrm{aux})$ from the challenger, $\mathcal{A}_{\mathsf{trap}}$ proceeds as follows.

1. Run $\mathsf{F.params} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$, ~~$(\mathrm{crs}, \mathrm{aux}) \leftarrow \mathsf{HTGen}(1^\lambda)$~~, and $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathsf{E.Gen}(1^\lambda)$. (Ignore aux).
2. Obliviously (of $\widetilde{\mathrm{pv}}_i$) generate the patches: For every $i \in [N]$, compute
$$(\{\mathsf{patch}_i\}_{i\in[N]}, \mathsf{o.state}) \leftarrow \mathsf{OPatchGen}(1^\lambda, \mathsf{F.params}))$$
3. Compute $\widetilde{\mathsf{KD\text{-}Prog}}$ as follows.
   - For every i, encrypt patch_i as $\mathrm{ct}_i \leftarrow \mathsf{Enc}(\mathrm{pk}, \mathsf{patch}_i)$. Present to the challenger $\{\mathrm{ct}_i\}_{i\in[N]}$ to receive N commitments and corresponding openings $\{(\mathrm{com}_i, \mathrm{open}_i)\}_{i\in[N]}$ to $\{\mathrm{ct}_i\}_{i\in[N]}$, resp. Set $\widetilde{\mathrm{sv}}_i = (\mathrm{ct}_i, \mathrm{open}_i)$.
   - Recall that $\widetilde{\mathrm{pv}}_i \in \{0,1\}^n$. Using $\mathsf{OPuncture}$, consistent with the already generated patches and with all $\widetilde{\mathrm{pv}}_i$, generate a key punctured at $2_G^{\widetilde{\mathrm{pv}}}$, where, $\widetilde{\mathrm{pv}} := (\widetilde{\mathrm{pv}}_1, \ldots, \widetilde{\mathrm{pv}}_N)$:
$$K[2_G^{\widetilde{\mathrm{pv}}}] \leftarrow \mathsf{OPuncture}(1^\lambda, \mathsf{o.state}, \widetilde{\mathrm{pv}})$$
   - For $(K[2_G^{\widetilde{\mathrm{pv}}}], \{\mathsf{patch}_i\}_{i\in[N]})$, generate $\mathsf{KD\text{-}Prog}_{6-7}$ exactly the same way as $\mathsf{KD\text{-}Prog}_6$ and compute $\widetilde{\mathsf{KD\text{-}Prog}} \leftarrow i\mathcal{O}(\mathsf{KD\text{-}Prog}_{6-7})$.

Thus, set $\mathsf{params} = (\widetilde{\mathsf{KD\text{-}Prog}}, \mathsf{F.params}, \mathrm{crs}, \mathrm{pk})$ and give it to $\mathcal{A}$. Then respond to the adversary's queries as follows.

4. Upon receiving $\mathsf{Reg}(i \in [N])$, respond via $\mathrm{pv}_i = \widetilde{\mathrm{pv}}_i$. (On the other hand, if the adversary himself registers a (corrupted) party P_i^* by presenting $\mathsf{RegCorr}(i \in [N], \mathrm{pv}_i)$, then simply ignore $\widetilde{\mathrm{pv}}_i$.)

5. Upon receiving $\mathsf{Ext}(i)$ for a registered honest P_i, respond via $\mathrm{sv}_i = \widetilde{\mathrm{sv}}_i$ generated earlier with the help of the challenger.

6. Upon receiving $\mathsf{Rev}(\mathcal{S}, j)$, respond via $\widetilde{\mathsf{KD\text{-}Prog}}(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, k, \mathrm{sv}_k)$, where k is determined as follows:

   - Suppose there exists $k' \in \mathcal{S}$ such that $P_{k'}$ was initially registered as an honest party, but has already been corrupted (via query $\mathsf{Ext}(k')$) by the adversary. Then k is such an arbitrary, say the smallest, k'.
   - Otherwise, choose any arbitrary (say the smallest) $k \in \mathcal{S}$.

7. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, choose $j \leftarrow \mathsf{ChQ}$, compute $\mathrm{sv}_j = \widetilde{\mathrm{sv}}_j$ as before, compute $Y_{\mathrm{REAL}} := \widetilde{\mathsf{KD\text{-}Prog}}(\mathsf{ChQ}, (\mathrm{pv}_i)_{i\in\mathsf{ChQ}}, j, \mathrm{sv}_j)$ and sample $Y_{\mathrm{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.

8. Finally, output whatever $\mathcal{A}$ outputs.

This completes the description of $\mathcal{A}_{\mathsf{trap}}$. Observe that if the challenger computes the commitments com_i using $\mathsf{HCommit}$ as $\mathrm{com}_i = \mathsf{HCommit}(\mathrm{crs}, \mathrm{ct}_i; \mathrm{open}_i)$, then the view of $\mathcal{A}$ as generated by $\mathcal{A}_{\mathsf{trap}}$ is the same as the view of $\mathcal{A}$ in Hyb_6. On the other hand, if the challenger computes the commitments com_i using $\mathsf{HTCommit}$ and HTDec as $(\mathrm{com}_i, \mathrm{aux}_i) \leftarrow \mathsf{HTCommit}(\mathrm{crs}, \mathrm{aux})$, $\mathrm{open}_i \leftarrow \mathsf{HTDec}(\mathrm{aux}_i, \mathrm{ct}_i)$, then the view of $\mathcal{A}$ as generated by $\mathcal{A}_{\mathsf{trap}}$ is the same as the view of $\mathcal{A}$ in Hyb_7.

$$
\begin{aligned}
\mathsf{Adv}^{\mathsf{trap}}_{\mathcal{A}_{\mathsf{trap}}}(\lambda) =& \Big| \Pr[1 \leftarrow \mathcal{A}_{\mathsf{trap}}(\mathrm{crs}, \mathrm{aux}, \{(\mathrm{com}_i, \mathrm{open}_i)\}_{i\in[N]}) \ : \ (\mathrm{com}_i, \mathrm{open}_i) \leftarrow \mathsf{HCommit}(\mathrm{crs}, \mathrm{ct}_i)] \\
& - \Pr\left[1 \leftarrow \mathcal{A}_{\mathsf{trap}}(\mathrm{crs}, \mathrm{aux}, \{(\mathrm{com}_i, \mathrm{open}_i)\}_{i\in[N]}) \ : \ \begin{array}{c}(\mathrm{com}_i, \mathrm{aux}_{\mathrm{com}_i}) \leftarrow \mathsf{HTCommit}(\mathrm{crs}, \mathrm{aux}), \\ \mathrm{open}_i \leftarrow \mathsf{HTDec}(\mathrm{aux}_{\mathrm{com}_i}, \mathrm{ct}_i)\end{array}\right] \Big| \\
=& \big| \Pr[\widetilde{\mathrm{EXPT}}^{\mathsf{Hyb}_6}_{\mathcal{A}} \to 1] - \Pr[\widetilde{\mathrm{EXPT}}^{\mathsf{Hyb}_7}_{\mathcal{A}} \to 1] \big| \\
=& \epsilon
\end{aligned}
$$

thus, arriving at a contradiction.

Hence, $\mathsf{Hyb}_6 \approx_c \mathsf{Hyb}_7$.

■

Hyb_8. This experiment is the same as Hyb_7 except for the following modification. Recall that, earlier, we sampled the patches *obliviously of* $\widetilde{\mathrm{pv}}_i$, since, it was crucial that the patches were sampled before we computed $\widetilde{\mathrm{pv}}_i$. This, to recall again, is because $\widetilde{\mathrm{pv}}_i$ was computed as a function of patch_i (namely, as a commitment to an encryption of patch_i). However, by Hyb_7, we had modified the computation of $\widetilde{\mathrm{pv}}_i$ as just generating an equivocal commitment, that could later be opened to any value, and in particular to an encryption of patch_i. Thus, we do not need to have computed patch_i ahead of time. Thus, in the current hybrid, we switch back to computing the patches using the algorithm $\mathsf{PatchGen}$, consistent with the punctured key. Details follow.

1. Run $\mathsf{F.params} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$, $(\mathrm{crs}, \mathrm{aux}) \leftarrow \mathsf{HTGen}(1^\lambda)$, and $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathsf{E.Gen}(1^\lambda)$.
2. Obliviously (of $\widetilde{\mathrm{pv}}_i$) generate the patches: For every $i \in [N]$, compute
$$\boxed{(\{\mathsf{patch}_i\}_{i\in[N]}, \mathsf{o.state}) \leftarrow \mathsf{OPatchGen}(1^\lambda, \mathsf{F.params})}$$
3. Compute $\widetilde{\mathsf{KD\text{-}Prog}}$ as follows.
   - For every $i \in [N]$, compute $(\mathrm{com}_i, \mathrm{aux}_i) \leftarrow \mathsf{HTCommit}(\mathrm{crs}, \mathrm{aux})$. Set $\widetilde{\mathrm{pv}}_i \leftarrow \mathrm{com}_i$.
   - Recall that $\widetilde{\mathrm{pv}}_i \in \{0,1\}^n$. Let $\widetilde{\mathrm{pv}} := (\widetilde{\mathrm{pv}}_1, \ldots, \widetilde{\mathrm{pv}}_N)$. Sample a key and puncture it at $2^{\widetilde{\mathrm{pv}}}_G$:
     - Sample a key, $K \leftarrow \mathsf{F.KeyGen}(1^\lambda, \mathsf{F.params})$.
     - Puncture K at $2^{\widetilde{\mathrm{pv}}}_G$: $K[2^{\widetilde{\mathrm{pv}}}_G] \leftarrow \mathsf{Puncture}(K, 2^{\widetilde{\mathrm{pv}}}_G)$.
     - Generate patches for $K[2^{\widetilde{\mathrm{pv}}}_G]$ at every ith block: $\mathsf{patch}(K, (i, \widetilde{\mathrm{pv}}_i)) \leftarrow \mathsf{PatchGen}(K, i, \widetilde{\mathrm{pv}}_i)$.
   - For $K[2^{\widetilde{\mathrm{pv}}}_G]$, generate $\mathsf{KD\text{-}Prog}_8$ exactly the same way as $\mathsf{KD\text{-}Prog}_7$ and compute $\widetilde{\mathsf{KD\text{-}Prog}} \leftarrow i\mathcal{O}(\mathsf{KD\text{-}Prog}_8)$.

Thus, set $\mathsf{params} = (\widetilde{\mathsf{KD\text{-}Prog}}, \mathsf{F.params}, \mathrm{crs}, \mathrm{pk})$ and give it to $\mathcal{A}$. Then respond to the adversary's queries as follows.

4. Upon receiving $\mathsf{Reg}(i \in [N])$, respond via $\mathrm{pv}_i = \widetilde{\mathrm{pv}}_i$. (On the other hand, if the adversary himself registers a (corrupted) party P_i^* by presenting $\mathsf{RegCorr}(i \in [N], \mathrm{pv}_i)$, then simply ignore $\widetilde{\mathrm{pv}}_i$.)
5. Upon receiving $\mathsf{Ext}(i)$ for a registered honest P_i, compute an encryption of $\mathsf{patch}(K, (i, \widetilde{\mathrm{pv}}_i))$ as $\mathrm{ct}_i \leftarrow \mathsf{Enc}(\mathrm{pk}, \mathsf{patch}(K, (i, \widetilde{\mathrm{pv}}_i)))$ as follows. Then, compute the opening of com_i to ct_i as $\mathrm{open}_i \leftarrow \mathsf{HTDec}(\mathrm{aux}_i, \mathrm{ct}_i)$, where aux_i was the auxiliary information created when com_i was generated earlier. With this, set $\mathrm{sv}_i = \widetilde{\mathrm{sv}}_i = (\mathrm{ct}_i, \mathrm{open}_i)$ and then respond to the adversary's query via sv_i.
6. Upon receiving $\mathsf{Rev}(\mathcal{S}, j)$, respond via $\widetilde{\mathsf{KD\text{-}Prog}}(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, k, \mathrm{sv}_k)$, where k is determined as follows:
   - Suppose there exists $k' \in \mathcal{S}$ such that $P_{k'}$ was initially registered as an honest party, but has already been corrupted (via query $\mathsf{Ext}(k')$) by the adversary. Then k is such an arbitrary, say the smallest, k'.
   - Otherwise, choose any arbitrary (say the smallest) $k \in \mathcal{S}$.
7. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, choose $j \leftarrow \mathsf{ChQ}$, compute $\mathrm{sv}_j = \widetilde{\mathrm{sv}}_j$ as before, compute $Y_{\mathrm{REAL}} := \widetilde{\mathsf{KD\text{-}Prog}}(\mathsf{ChQ}, (\mathrm{pv}_i)_{i\in\mathsf{ChQ}}, j, \mathrm{sv}_j)$ and sample $Y_{\mathrm{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.

8. Finally, output whatever $\mathcal{A}$ outputs.

Lemma 8. $\mathsf{Hyb}_7 \equiv \mathsf{Hyb}_8$.

Proof. Observe that the only difference between Hyb_7 and Hyb_8 is in the way the punctured key and its patches are computed. While in Hyb_7, they are computed using algorithms $\mathsf{OPatchGen}, \mathsf{OPuncture}$, in Hyb_8, they are computed using $\mathsf{Puncture}, \mathsf{PatchGen}$. Recall that this exactly corresponds to switching back the difference introduced while moving from Hyb_2 to Hyb_3. As noted in Lemma 3, these two different processes of generating the key and its patches result in identical distributions. Hence, the Lemma.

■

Hyb_9. This experiment is the same as Hyb_8 except for the following modification. Note that in Hyb_8, we computed the patches soon after we computed the punctured key. The modification we introduce in this hybrid is that we would compute patch_j (and its encryption ct_j) only if necessary: that is, only if the adversary presents a query that is either $\mathsf{Ext}(j)$ (i.e., query for the secret value $\widetilde{\mathrm{sv}}_j$ of P_j) or $\mathsf{Rev}(\mathcal{S}, j)$ (Recall that the challenger in Hyb_8, for certain $\mathcal{S}, j$, used $\mathrm{sv}_j = \widetilde{\mathrm{sv}}_j$ to compute response to such a query; more specifically, it executed the program $\widetilde{\mathsf{KD\text{-}Prog}}$ on input $(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, j, \mathrm{sv}_j)$). Details follow.

1. Run $\mathsf{F.params} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$, $(\mathrm{crs}, \mathrm{aux}) \leftarrow \mathsf{HTGen}(1^\lambda)$, and $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathsf{E.Gen}(1^\lambda)$.
2. Compute $\widetilde{\mathsf{KD\text{-}Prog}}$ as follows.
   - For every $i \in [N]$, compute $(\mathrm{com}_i, \mathrm{aux}_i) \leftarrow \mathsf{HTCommit}(\mathrm{crs}, \mathrm{aux})$. Set $\widetilde{\mathrm{pv}}_i \leftarrow \mathrm{com}_i$.
   - Recall that $\widetilde{\mathrm{pv}}_i \in \{0,1\}^n$. Let $\widetilde{\mathrm{pv}} := (\widetilde{\mathrm{pv}}_1, \ldots, \widetilde{\mathrm{pv}}_N)$. Sample a key and puncture it at $2^{\widetilde{\mathrm{pv}}}_G$:
     - Sample a key, $K \leftarrow \mathsf{F.KeyGen}(1^\lambda, \mathsf{F.params})$.
     - Puncture K at $2^{\widetilde{\mathrm{pv}}}_G$: $K[2^{\widetilde{\mathrm{pv}}}_G] \leftarrow \mathsf{Puncture}(K, 2^{\widetilde{\mathrm{pv}}}_G)$.
     - Generate patches for $K[2^{\widetilde{\mathrm{pv}}}_G]$ at every ith block: ~~$\mathsf{patch}(K, (i, \widetilde{\mathrm{pv}}_i)) \leftarrow \mathsf{PatchGen}(K, i, \widetilde{\mathrm{pv}}_i)$~~.
   - For $K[2^{\widetilde{\mathrm{pv}}}_G]$, generate $\mathsf{KD\text{-}Prog}_9$ exactly the same way as $\mathsf{KD\text{-}Prog}_8$ and compute $\widetilde{\mathsf{KD\text{-}Prog}} \leftarrow i\mathcal{O}(\mathsf{KD\text{-}Prog}_9)$.

Thus, set $\mathsf{params} = (\widetilde{\mathsf{KD\text{-}Prog}}, \mathsf{F.params}, \mathrm{crs}, \mathrm{pk})$ and give it to $\mathcal{A}$. Then respond to the adversary's queries as follows.

4. Upon receiving $\mathsf{Reg}(i \in [N])$, respond via $\mathrm{pv}_i = \widetilde{\mathrm{pv}}_i$. (On the other hand, if the adversary himself registers a (corrupted) party P_i^* by presenting $\mathsf{RegCorr}(i \in [N], \mathrm{pv}_i)$, then simply ignore $\widetilde{\mathrm{pv}}_i$.)
5. Upon receiving $\mathsf{Ext}(i)$ for a registered honest P_i, firstly compute a patch of $2^{\widetilde{\mathrm{pv}}}_G$ at the ith block as $\mathsf{patch}(K, (i, \widetilde{\mathrm{pv}}_i)) \leftarrow \mathsf{PatchGen}(K, i, \widetilde{\mathrm{pv}}_i)$, compute an encryption of $\mathsf{patch}(K, (i, \widetilde{\mathrm{pv}}_i))$ as $\mathrm{ct}_i \leftarrow \mathsf{Enc}(\mathrm{pk}, \mathsf{patch}(K, (i, \widetilde{\mathrm{pv}}_i)))$ as follows. Then, compute the opening of com_i to ct_i as $\mathrm{open}_i \leftarrow \mathsf{HTDec}(\mathrm{aux}_i, \mathrm{ct}_i)$, where aux_i was the auxiliary information created when com_i was generated earlier. With this, set $\mathrm{sv}_i = \widetilde{\mathrm{sv}}_i = (\mathrm{ct}_i, \mathrm{open}_i)$ and then respond to the adversary's query via sv_i.
6. Upon receiving $\mathsf{Rev}(\mathcal{S}, j)$, respond via $\widetilde{\mathsf{KD\text{-}Prog}}(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, k, \mathrm{sv}_k)$, where k is determined as follows:
   - Suppose there exists $k' \in \mathcal{S}$ such that $P_{k'}$ was initially registered as an honest party, but has already been corrupted (via query $\mathsf{Ext}(k')$) by the adversary. Then k is such an arbitrary, say the smallest, k'.
   - Otherwise, choose any arbitrary (say the smallest) $k \in \mathcal{S}$.
7. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, choose $j \leftarrow \mathsf{ChQ}$, compute $\mathrm{sv}_j = \widetilde{\mathrm{sv}}_j$ as before, compute $Y_{\mathrm{REAL}} := \widetilde{\mathsf{KD\text{-}Prog}}(\mathsf{ChQ}, (\mathrm{pv}_i)_{i\in\mathsf{ChQ}}, j, \mathrm{sv}_j)$ and sample $Y_{\mathrm{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}. Respond to the adversary via Y_{EXPT}.
8. Finally, output whatever $\mathcal{A}$ outputs.

Lemma 9. $\mathsf{Hyb}_8 \equiv \mathsf{Hyb}_9$.

Proof. Observe that the only difference between Hyb_8 and Hyb_9 is the following. At a high level, in Hyb_8, we computed all the patches ahead of time and used them if and when necessary. On the other hand, in Hyb_9, we generate the patches only if and when they are required. This clearly does not introduce any deviation in the view of the adversary. Hence, the Lemma.

■

Hyb_{10}. This experiment is the same as Hyb_9 except for the following modification. Note that responses to certain valid queries can be computed by the adversary himself; for instance, a query $\mathsf{Rev}(\mathcal{S}, j)$ where there exists $j' \in \mathcal{S}$ and $P_{j'}$ has been corrupted by the adversary – the response to this query, namely the common key for the set $\mathcal{S}$, can be computed by the adversary himself, since, this common key is the same as the common key derived by any party in $\mathcal{S}$ and in particular $P_{j'}$. Having recalled this aspect, the modification we introduce in this hybrid is the following. For the queries the responses to which the adversary can himself compute, the challenger behaves the same way as it did in the previous hybrid – namely, it simply runs the program in the CRS. On the other hand, for the rest of the queries, the challenger computes the response by evaluating the PRF F directly. On the same lines, response to the challenge query is also computed by evaluating the PRF F directly. Details follow.

1. Run $\mathsf{F.params} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$, $(\mathrm{crs}, \mathrm{aux}) \leftarrow \mathsf{HTGen}(1^\lambda)$, and $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathsf{E.Gen}(1^\lambda)$.
2. Compute $\widetilde{\mathsf{KD\text{-}Prog}}$ as follows.
   - For every $i \in [N]$, compute $(\mathrm{com}_i, \mathrm{aux}_i) \leftarrow \mathsf{HTCommit}(\mathrm{crs}, \mathrm{aux})$. Set $\widetilde{\mathrm{pv}}_i \leftarrow \mathrm{com}_i$.
   - Recall that $\widetilde{\mathrm{pv}}_i \in \{0,1\}^n$. Let $\widetilde{\mathrm{pv}} = (\widetilde{\mathrm{pv}}_1, \ldots, \widetilde{\mathrm{pv}}_N)$. Sample a key and puncture it at $2^{\widetilde{\mathrm{pv}}}_G$:
     - Sample a key, $K \leftarrow \mathsf{F.KeyGen}(1^\lambda, \mathsf{F.params})$.
     - Puncture K at $2^{\widetilde{\mathrm{pv}}}_G$: $K_{\{2^{\widetilde{\mathrm{pv}}}_G\}} \leftarrow \mathsf{F.Puncture}(K, 2^{\widetilde{\mathrm{pv}}}_G)$.
   - Output $\widetilde{\mathsf{KD\text{-}Prog}}(\mathrm{crs}, \mathrm{aux}, j) := \mathsf{KD\text{-}Prog}(\mathrm{crs}, \mathrm{aux}, K_{\{2^{\widetilde{\mathrm{pv}}}_G\}}, \widetilde{\mathrm{pv}}, \mathrm{aux}_j)$

Lemma 10. $\mathsf{Hyb}_9 \equiv \mathsf{Hyb}_{10}$.

Proof. Observe that the only difference between Hyb_9 and Hyb_{10} is the following. For certain queries, while the challenger in Hyb_9 responded by running the program in the CRS, the challenger in Hyb_{10} responds by computing the PRF directly. Since the program also works by effectively evaluating the PRF, the view of the adversary is identical in both the hybrids. Hence, the Lemma. ∎

Lemma 11. Let F be an obliviously-patchable puncturable PRF. Then the advantage of any PPT algorithm $\mathcal{A}$ in Hyb_{10}, namely $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_{10}}(\cdot)$, is negligible in the security parameter.

Proof. Assume for contradiction that there exists a PPT algorithm $\mathcal{A}$ with non-negligible advantage ϵ in Hyb_{10}. That is, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_{10}}(\lambda) = \epsilon$. Then we construct an adversary $\mathcal{B}$ that breaks the security of F also with a non-negligible probability.

Recall that upon $\mathcal{B}$ presenting $\widetilde{\mathrm{pv}} \in (\{0,1\}^n)^N$, it receives $K[2_G^{\widetilde{\mathrm{pv}}}]$. Then, it is given access to an oracle $\mathcal{O}$ to which it can either make queries for patches or for PRF outputs for any inputs in $2_G^{\widetilde{\mathrm{pv}}}$. Then, along the way, the adversary makes a challenge query $\mathsf{ChQ} \subseteq [N]$ that satisfies certain conditions; it either receives the output of the PRF computed on $(\mathsf{ChQ}, (\widetilde{\mathrm{pv}}_i)_{i \in \mathsf{ChQ}})$ or a random element from the co-domain of F, and its objective is to distinguish them.

We shall show that using the given values and access to the oracle $\mathcal{O}$, $\mathcal{B}$ can simulate to $\mathcal{A}$ the hybrid game Hyb_{10} such that the following holds. If $\mathcal{A}$ has a non-negligible advantage in Hyb_{10}, then $\mathcal{B}$ can break the security of F also with a non-negligible advantage.

At a high level, $\mathcal{B}$ predominantly behaves as the challenger of Hyb_{10} except that it obtains all the values pertaining to the PRF from its interaction with its own challenger and the oracle $\mathcal{O}$. For clarity and ease of reading, these values shall be highlighted in a red underlined font.

1. Let F.params be the parameters received by $\mathcal{B}$ from its challenger. Run $(\mathrm{crs}, \mathrm{aux}) \leftarrow \mathsf{HTGen}(1^\lambda)$, and $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathsf{E.Gen}(1^\lambda)$.
2. Compute $\widetilde{\mathsf{KD\text{-}Prog}}$ as follows.
   - For every $i \in [N]$, compute $(\mathrm{com}_i, \mathrm{aux}_i) \leftarrow \mathsf{HTCommit}(\mathrm{crs}, \mathrm{aux})$. Set $\widetilde{\mathrm{pv}}_i \leftarrow \mathrm{com}_i$. Furthermore, if the adversary requests to register an honest party under an identity i, then set $(\mathrm{pv}_i, \mathrm{sv}_i) = (\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$ and henceforth use this pair. On the other hand, if the adversary himself registers a (corrupted) party P_i^* by presenting $\mathsf{RegCorr}(i \in [N], \mathrm{pv}_i)$, then the challenger simply ignores $(\widetilde{\mathrm{pv}}_i, \widetilde{\mathrm{sv}}_i)$.
   - Recall that $\widetilde{\mathrm{pv}}_i \in \{0,1\}^n$. Let $\widetilde{\mathrm{pv}} := (\widetilde{\mathrm{pv}}_1, \ldots, \widetilde{\mathrm{pv}}_N)$. Query the challenger with $\widetilde{\mathrm{pv}}$ to receive $K[2_G^{\widetilde{\mathrm{pv}}}]$, a key punctured at $2_G^{\widetilde{\mathrm{pv}}}$.
   - For $K[2_G^{\widetilde{\mathrm{pv}}}]$, generate generate $\mathsf{KD\text{-}Prog}_{\mathtt{redu}}$ exactly the same way as $\mathsf{KD\text{-}Prog}_{10}$ and compute $\widetilde{\mathsf{KD\text{-}Prog}} \leftarrow \mathsf{i}\mathcal{O}(\mathsf{KD\text{-}Prog}_{\mathtt{redu}})$.

Thus, set $\mathsf{params} = (\widetilde{\mathsf{KD\text{-}Prog}}, \mathsf{F.params}, \mathrm{crs}, \mathrm{pk})$ and give it to $\mathcal{A}$. Then respond to the adversary's queries as follows.

4. Upon receiving $\mathsf{Ext}(i)$ for a registered honest P_i, query $\mathcal{O}$ with (PATCH-AT i); let $\mathsf{patch}(K, (i, \widetilde{\mathrm{pv}}_i))$, a patch of $K[2_G^{\widetilde{\mathrm{pv}}}]$ at the ith block, be the response received; then compute an encryption of $\mathsf{patch}(K, (i, \widetilde{\mathrm{pv}}_i))$ as $\mathrm{ct}_i \leftarrow \mathsf{Enc}(\mathrm{pk}, \mathsf{patch}(K, (i, \widetilde{\mathrm{pv}}_i)))$ as follows. Then, compute the opening of com_i to ct_i as $\mathrm{open}_i \leftarrow \mathsf{HTDec}(\mathrm{aux}_i, \mathrm{ct}_i)$, where aux_i was the auxiliary information created

when com_i was generated earlier. With this, set $\mathrm{sv}_i = \widetilde{\mathrm{sv}}_i = (\mathrm{ct}_i, \mathrm{open}_i)$ and then respond to the adversary's query via sv_i.

5. Upon receiving $\mathsf{Rev}(\mathcal{S}, j)$, consider the following cases:
   - If there exists $j' \in \mathcal{S}$ such that $P_{j'}$ was registered as an honest party (i.e., via $\mathsf{Reg}(j' \in [N])$) and the adversary has already queried $\mathsf{Ext}(j')$ and received $\mathrm{sv}_{j'} = \widetilde{\mathrm{sv}}_{j'}$, then output $\widetilde{\mathsf{KD\text{-}Prog}}(\mathcal{S}, (\mathrm{pv}_i)_{i\in\mathcal{S}}, j', \mathrm{sv}_{j'})$ and then respond to the adversary's query via the output of the program.
   - Else, query $\mathcal{O}$ with (EVAL-AT $\mathcal{S}$) and respond to $\mathcal{A}$ via the response given by $\mathcal{O}$.
6. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, query $\mathcal{O}$ with (CHAL-AT ChQ) and respond to $\mathcal{A}$ via the response given by $\mathcal{O}$.
7. Finally, output whatever $\mathcal{A}$ outputs.

From the above description of $\mathcal{B}$, we have that if $\mathcal{B}$ interacts with its own challenger in the real game, then the view of $\mathcal{A}$ during its interaction with $\mathcal{B}$ is identical to its view in the real experiment of Hyb_{10}, namely $\textsc{real}_{\mathcal{A}}^{\mathsf{Hyb}_{10}}$. On the other hand, if $\mathcal{B}$ interacts with its challenger in the random experiment, then the view of $\mathcal{A}$ during its interaction with $\mathcal{B}$ is identical to its view in the random experiment of Hyb_{10}, namely $\textsc{rand}_{\mathcal{A}}^{\mathsf{Hyb}_{10}}$. Therefore,

$$\begin{aligned}\mathsf{Adv}_{F,\mathcal{A}}(\lambda) =& |\Pr[\textsc{real}_{F,\mathcal{A}}^{\mathsf{PRF}}(\lambda) \to 1] - \Pr[\textsc{rand}_{F,\mathcal{A}}^{\mathsf{PRF}}(\lambda) \to 1]| \\ =& |\Pr[\textsc{real}_{\mathcal{A}}^{\mathsf{Hyb}_{10}} \to 1] - \Pr[\textsc{rand}_{\mathcal{A}}^{\mathsf{Hyb}_{10}} \to 1]| \\ =& \epsilon,\end{aligned}$$

thus arriving at a contradiction. Hence, the lemma.

■

■

B Security Of Construction Without Any Setup

Theorem 4. If $\mathsf{i}\mathcal{O}$ is indistinguishably secure, Σ is a CPA-secure encryption scheme with pseudorandom ciphertexts, HCOM is a hybrid trapdoor commitment scheme, and F is an obliviously-patchable puncturable PRF, then Construction 2 is an *adaptively* secure non-interactive key exchange scheme *with no setup.*

Proof. Assume towards contradiction that an adversary $\mathcal{A}$ has non-negligible advantage in breaking the adaptive security of our Construction 2 as in Definition 8. We arrive at a contradiction through a carefully designed sequence of several hybrids $\mathsf{Hyb}_0, \mathsf{Hyb}_1, \ldots, \mathsf{Hyb}_{10}$. If the challenger in Hyb_i chooses to play the real experiment (respectively, the random experiment) and $\mathcal{A}$ outputs 1, then we denote the event by $\textsc{real}_{\mathcal{A}}^{\mathsf{Hyb}_i} \to 1$ (respectively, $\textsc{rand}_{\mathcal{A}}^{\mathsf{Hyb}_i} \to 1$). The advantage of $\mathcal{A}$ in Hyb_i is defined as

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_i}(\lambda) := |\Pr[\textsc{real}_{\mathcal{A}}^{\mathsf{Hyb}_i} \to 1] - \Pr[\textsc{rand}_{\mathcal{A}}^{\mathsf{Hyb}_i} \to 1]| \tag{6}$$

Furthermore, we shall use Hyb_i to also denote the view of the adversary in the hybrid whenever the connotation is unambiguous.

To maintain ease of verification for the reader, we present a full description of each hybrid experiment, each one given on a separate page. The modification introduced in the current hybrid in comparison with the previous hybrid will be highlighted in red underlined font. Furthermore, if a value is removed as we move from one hybrid to the next, then in the latter hybrid, we highlight the ~~value~~ within a red frame with a red strike-through. Also, let's say that while switching to the next hybrid, a value is moved from one step of execution to a later step; then in the latter hybrid, ~~value~~ shall appear at its position in the previous hybrid and value shall appear at its new position in the current hybrid.

Hyb_0. This experiment is same as the original experiment in Definition 6. Here, we simply unwrap the original experiment as per our construction. The challenger behaves as follows. It begins by choosing $\text{EXPT} \in \{\text{REAL}, \text{RAND}\}$ uniformly at random, and executes with $\mathcal{A}$ the experiment $\text{EXPT}_{\mathcal{A}}^{\mathsf{Hyb}_0}$ defined as follows.

1. Initiate by running the Setup algorithm. Namely, simply output (λ, N, G) as the output of the Setup algorithm. Next, respond to the adversary's queries as follows.
2. Upon receiving $\mathsf{Reg}(\hat{i} \in [N])$, run $\mathsf{Publish}(\mathsf{params}, \lambda, \hat{i})$ to obtain $(\mathrm{pv}^{(\hat{i})}, \mathrm{sv}^{(\hat{i})})$, where $\mathrm{pv}^{(\hat{i})} = (\widetilde{x}^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$, proceed as follows.

   Computing $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.

   - Run the statistically-binding parameter-generation algorithm of HCOM: $\mathrm{crs}^{(\hat{i})} \leftarrow \mathsf{HGen}(1^\lambda)$. Sample uniformly at random $\mathrm{ct}_1^{(\hat{i})}, \ldots, \mathrm{ct}_N^{(\hat{i})} \leftarrow \{0,1\}^{\ell'}$. Commit to $\mathrm{ct}_j^{(\hat{i})}$: $\mathrm{com}_j^{(\hat{i})} = \mathsf{HCommit}(\mathrm{crs}^{(\hat{i})}, \mathrm{ct}_j^{(\hat{i})}; \mathrm{open}_j^{(\hat{i})})$ using uniformly chosen random coins $\mathrm{open}_j^{(\hat{i})}$.

   Let
   $$\widetilde{x}^{(\hat{i})} := (\mathrm{crs}^{(\hat{i})}, \mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})})$$
   and
   $$\widetilde{\mathrm{sv}}^{(\hat{i})} := ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})})).$$

   Computing the program $\mathsf{ioP}^{(\hat{i})}$.

   - Run the parameter-generation algorithms of the PRF: $\mathsf{F.params}^{(\hat{i})} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$.
   - Choose $K^{(\hat{i})} \leftarrow \mathsf{F.KeyGen}(1^\lambda, \mathsf{F.params}^{(\hat{i})})$.
   - For $K^{(\hat{i})}$, build the key-derivation program $\mathsf{KD\text{-}Prog}_0^{(\hat{i})}$ in Figure 7, padded to the appropriate length. Compute $\mathsf{ioP}^{(\hat{i})} \leftarrow i\mathcal{O}(\mathsf{KD\text{-}Prog}_0^{(\hat{i})})$.

   Let $\mathrm{pv}^{(\hat{i})} = \widetilde{\mathrm{pv}}^{(\hat{i})} = (\widetilde{x}^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$.

3. Upon receiving $\mathsf{Ext}(\hat{i})$ for a registered honest $P_{\hat{i}}$, respond via $\mathrm{sv}^{(\hat{i})}$.
4. Upon receiving $\mathsf{Rev}(\mathcal{S}, \hat{j})$, let $\hat{i}^*$ be the smallest element in $\mathcal{S}$. Run $\mathsf{ioP}^{(\hat{i}^*)}$ on input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k} \in \mathcal{S}}, \hat{j}, \mathrm{sv}_{\hat{i}^*}^{(\hat{j})})$ and respond via the output.
5. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, let $\hat{i}^*$ be the smallest element in ChQ. Choose $\hat{j} \leftarrow \mathsf{ChQ}$ and run $\mathsf{ioP}^{(\hat{i}^*)}$ as $Y_{\text{REAL}} := \mathsf{ioP}^{(\hat{i}^*)}(\mathsf{ChQ}, (x^{(\hat{k})})_{\hat{k} \in \mathsf{ChQ}}, \hat{j}, \mathrm{sv}_{\hat{i}^*}^{(\hat{j})})$ and sample $Y_{\text{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
6. Finally, output whatever $\mathcal{A}$ outputs.

$\mathsf{KD\text{-}Prog}_0^{(\hat{i})}$

Constants:

- $\mathsf{F.params}^{(\hat{i})}$
- $K^{(\hat{i})}$

Input: $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{j}, \mathrm{sv}_{\hat{i}^*}^{(\hat{j})})$, where $\mathcal{S} \subseteq [N]$, $|\mathcal{S}| \leq G$.
Procedure:

1. Check whether $\hat{j} \in \mathcal{S}$; if not, output $\perp$. Then, check whether $(x^{(\hat{j})}, \mathrm{sv}_{\hat{i}^*}^{(\hat{j})}) \in \mathrm{R}_{\mathcal{S}}^{\mathsf{HCOM}}$ (defined in Equation (4)); if not, output $\perp$. Otherwise, proceed as follows.
2. Compute and output $F(K, (\mathcal{S}, \{x^{(\hat{k})}\}_{\hat{k}\in\mathcal{S}}))$.

Figure 7: Key-derivation Program

Hyb_1. This experiment is the same as Hyb_0 except for the following modification. Essentially, this modification corresponds to the sequence in which the challenger computes certain values. Namely, before the adversary begins presenting queries, the challenger samples the public-value secret-value pairs ahead of time – we shall denote these pairs as $(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$. A further modification is that in computing $\{(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})\}_{\hat{i}\in[N]}$, the challenger first samples $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$ for every $\hat{i} \in [N]$ before computing the remaining portion of the public value $\widetilde{\mathrm{pv}}^{(\hat{i})}$, namely, the program $\mathsf{ioP}^{(\hat{i})}$ for any $\hat{i}$. (Recall that $\mathrm{pv}^{(\hat{i})} = (x^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$.) In other words, only after having computed $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$ for every $\hat{i} \in [N]$ does the challenger start computing $\mathsf{ioP}^{(\hat{j})}$ for any $\hat{j}$. This modification is well-defined since in Hyb_0, for every $\hat{i}$, the order of computation is that the challenger first computes $x^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})}$ and then computes $\mathsf{ioP}^{(\hat{i})}$; furthermore, for any $\hat{j}$, the computation of $\mathsf{ioP}^{(\hat{j})}$ does not depend on $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$ for any $\hat{i} \in [N]$. Additionally, in every $\mathsf{ioP}^{(\hat{i})}$, we introduce $\{\widetilde{x}^{(\hat{k})}\}_{\hat{k}\in[N]}$ as constants. Since this does not affect the input/output functionality of the programs, applying the security of indistinguishability obfuscation, we will be able to argue indistinguishability of the current hybrid from Hyb_0. Details follow.

The challenger behaves as follows. It begins by choosing $\textsc{expt} \in \{\textsc{real}, \textsc{rand}\}$ uniformly at random, and executes with $\mathcal{A}$ the experiment $\textsc{expt}_{\mathcal{A}}^{\mathsf{Hyb}_1}$ defined as follows.

1. Initiate by running the Setup algorithm. Namely, simply output (λ, N, G) as the output of the Setup algorithm. Next, respond to the adversary's queries as follows.
2. Then, for every $\hat{i} \in [N]$, firstly compute $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$:

   Computing $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.

   - Run the statistically-binding parameter-generation algorithm of HCOM: $\mathrm{crs}^{(\hat{i})} \leftarrow \mathsf{HGen}(1^\lambda)$. Sample uniformly at random $\mathrm{ct}_1^{(\hat{i})}, \ldots, \mathrm{ct}_N^{(\hat{i})} \leftarrow \{0,1\}^{\ell'}$. Commit to $\mathrm{ct}_j^{(\hat{i})}$: $\mathrm{com}_j^{(\hat{i})} = \mathsf{HCommit}(\mathrm{crs}^{(\hat{i})}, \mathrm{ct}_j^{(\hat{i})}; \mathrm{open}_j^{(\hat{i})})$ using uniformly chosen random coins $\mathrm{open}_j^{(\hat{i})}$.

   Let
   $$\widetilde{x}^{(\hat{i})} := (\mathrm{crs}^{(\hat{i})}, \mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})})$$
   and
   $$\widetilde{\mathrm{sv}}^{(\hat{i})} := ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})})).$$
3. Then, for every $\hat{i} \in [N]$, compute programs $\mathsf{ioP}^{(\hat{i})}$:

Computing the program $\mathsf{ioP}^{(\hat{i})}$.

- Run $\mathsf{F.params}^{(\hat{i})} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$.
- Choose $K^{(\hat{i})} \leftarrow \mathsf{F.KeyGen}(1^\lambda, \mathsf{F.params}^{(\hat{i})})$.
- For $K^{(\hat{i})}$, build the key-derivation program $\mathsf{KD\text{-}Prog}_1^{(\hat{i})}$ in Figure 8, padded to the appropriate length. Compute $\mathsf{ioP}^{(\hat{i})} \leftarrow i\mathcal{O}(\mathsf{KD\text{-}Prog}_1^{(\hat{i})})$.

Let $\mathrm{pv}^{(\hat{i})} = \widetilde{\mathrm{pv}}^{(\hat{i})} = (\widetilde{x}^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$.

4. Upon receiving $\mathsf{Reg}(\hat{i} \in [N])$, then set $(\mathrm{pv}^{(\hat{i})}, \mathrm{sv}^{(\hat{i})}) = (\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$. On the other hand, if the adversary himself registers a (corrupted) party $P_{\hat{i}}^*$ by presenting $\mathsf{RegCorr}(\hat{i} \in [N], \mathrm{pv}^{(\hat{i})})$, then the challenger simply ignores $(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
5. Upon receiving $\mathsf{Ext}(\hat{i})$ for a registered honest $P_{\hat{i}}$, respond via $\mathrm{sv}^{(\hat{i})}$.
6. Upon receiving $\mathsf{Rev}(\mathcal{S}, \hat{j})$, let $\hat{i}^*$ be the smallest element in $\mathcal{S}$. Run $\mathsf{ioP}^{(\hat{i}^*)}$ on input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{j}, \mathrm{sv}_{\hat{i}^*}^{(\hat{j})})$ and respond via the output.
7. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, let $\hat{i}^*$ be the smallest element in ChQ. Choose $\hat{j} \leftarrow \mathsf{ChQ}$ and run $\mathsf{ioP}^{(\hat{i}^*)}$ as $Y_{\mathrm{REAL}} := \mathsf{ioP}^{(\hat{i}^*)}(\mathsf{ChQ}, (x^{(\hat{k})})_{\hat{k}\in\mathsf{ChQ}}, \hat{j}, \mathrm{sv}_{\hat{i}^*}^{(\hat{j})})$ and sample $Y_{\mathrm{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
8. Finally, output whatever $\mathcal{A}$ outputs.

$\mathsf{KD\text{-}Prog}_1^{(\hat{i})}$

Constants:

- $\mathsf{F.params}^{(\hat{i})}$
- $K^{(\hat{i})}$
- $\{\widetilde{x}^{(\hat{k})}\}_{\hat{k}\in[N]}$

Input: $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{j}, \mathrm{sv}_{\hat{i}^*}^{(\hat{j})})$, where $\mathcal{S} \subseteq [N]$, $|\mathcal{S}| \le G$.
Procedure:

1. Check whether $\hat{j} \in \mathcal{S}$; if not, output $\perp$. Then, check whether $(x^{(\hat{j})}, \mathrm{sv}_{\hat{i}^*}^{(\hat{j})}) \in \mathrm{R}_{\mathcal{S}}^{\mathsf{HCOM}}$; if not, output $\perp$. Otherwise, proceed as follows.
2. Compute and output $F(K^{(\hat{i})}, (\mathcal{S}, \{x^{(\hat{k})}\}_{\hat{k}\in\mathcal{S}}))$.

Figure 8: Key-derivation Program

Lemma 12. $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_1$.

Proof. Observe that the only difference between Hyb_0 and Hyb_1 is in the sequence of the values computed and an additional constant, $\{\widetilde{x}^{(\hat{i})}\}_{\hat{i}\in[N]}$, in the programs $\mathsf{KD\text{-}Prog}_1^{(\hat{i})}$ for Hyb_1. As for the first difference, in Hyb_1 the challenger computes $(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$ for every $\hat{i}$ ahead of time and uses them only when the challenger would use them in Hyb_0. Observe that this does not cause any change in the distribution of the values $(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$ for every identity $\hat{i}$ under which the adversary requests to register an honest party. As for the next difference, since for every $\hat{i}$ the input/output relation of both the programs $\mathsf{KD\text{-}Prog}_0^{(\hat{i})}$ and $\mathsf{KD\text{-}Prog}_1^{(\hat{i})}$ are identical, applying the security of $i\mathcal{O}$ for every obfuscated program, through a standard hybrid argument, we have that the hybrids are computationally indistinguishable. Hence, $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_1$.

■

Hyb_2. This experiment is the same as Hyb_1 except for the following modification in the way the key is sampled for constructing program $\mathsf{KD\text{-}Prog}_2^{(\hat{i})}$ for every $\hat{i}$. Details follow.

The challenger behaves as follows. It begins by choosing $\text{EXPT} \in \{\text{REAL}, \text{RAND}\}$ uniformly at random, and executes with $\mathcal{A}$ the experiment $\text{EXPT}_{\mathcal{A}}^{\mathsf{Hyb}_2}$ defined as follows.

1. Initiate by running the Setup algorithm. Namely, simply output (λ, N, G) as the output of the Setup algorithm. Next, respond to the adversary's queries as follows.
2. Then, for every $\hat{i} \in [N]$, firstly compute $(\widetilde{x}^{(\hat{i})}, \widetilde{\text{sv}}^{(\hat{i})})$:

   Computing $(\widetilde{x}^{(\hat{i})}, \widetilde{\text{sv}}^{(\hat{i})})$.

   - Run the statistically-binding parameter-generation algorithm of HCOM: $\text{crs}^{(\hat{i})} \leftarrow \mathsf{HGen}(1^\lambda)$. Sample uniformly at random $\text{ct}_1^{(\hat{i})}, \ldots, \text{ct}_N^{(\hat{i})} \leftarrow \{0,1\}^{\ell'}$. Commit to $\text{ct}_j^{(\hat{i})}$: $\text{com}_j^{(\hat{i})} = \mathsf{HCommit}(\text{crs}^{(\hat{i})}, \text{ct}_j^{(\hat{i})}; \text{open}_j^{(\hat{i})})$ using uniformly chosen random coins $\text{open}_j^{(\hat{i})}$.

   Let
   $$\widetilde{x}^{(\hat{i})} := (\text{crs}^{(\hat{i})}, \text{com}_1^{(\hat{i})}, \ldots, \text{com}_N^{(\hat{i})})$$
   and
   $$\widetilde{\text{sv}}^{(\hat{i})} := ((\text{ct}_1^{(\hat{i})}, \text{open}_1^{(\hat{i})}), \ldots, (\text{ct}_N^{(\hat{i})}, \text{open}_N^{(\hat{i})})).$$
3. Then, for every $\hat{i} \in [N]$, compute programs $\mathsf{ioP}^{(\hat{i})}$:

   Computing the program $\mathsf{ioP}^{(\hat{i})}$.

   - Run $\mathsf{F.params}^{(\hat{i})} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$.
   - Sample a key, $K^{(\hat{i})} \leftarrow \mathsf{F.KeyGen}(1^\lambda, \mathsf{F.params})$.
   - Puncture K at $2_G^{\widetilde{x}}$: $K^{(\hat{i})}[2_G^{\widetilde{x}}] \leftarrow \mathsf{Puncture}(K^{(\hat{i})}, 2_G^{\widetilde{x}})$, where $\widetilde{x} = (\widetilde{x}^{(1)}, \ldots, \widetilde{x}^{(N)})$.
   - Generate patches for $K^{(\hat{i})}[2_G^{\widetilde{x}}]$ at every $\hat{j}$th block: $\mathsf{patch}(K^{(\hat{i})}[2_G^{\widetilde{x}}], (\hat{j}, \widetilde{x}^{(\hat{j})})) \leftarrow \mathsf{PatchGen}(K^{(\hat{i})}, \hat{j}, \widetilde{x}^{(\hat{j})})$.
   - For $(K^{(\hat{i})}[2_G^{\widetilde{x}}], \{\mathsf{patch}(K^{(\hat{i})}[2_G^{\widetilde{x}}], (\hat{k}, \widetilde{x}^{(\hat{k})}))\}_{\hat{k}\in[N]})$, build the key-derivation program $\mathsf{KD\text{-}Prog}_2^{(\hat{i})}$ in Figure 9, padded to the appropriate length. Compute $\mathsf{ioP}^{(\hat{i})} \leftarrow i\mathcal{O}(\mathsf{KD\text{-}Prog}_2^{(\hat{i})})$.

   Let $\text{pv}^{(\hat{i})} = \widetilde{\text{pv}}^{(\hat{i})} = (\widetilde{x}^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$.
4. Upon receiving $\mathsf{Reg}(\hat{i} \in [N])$, then set $(\text{pv}^{(\hat{i})}, \text{sv}^{(\hat{i})}) = (\widetilde{\text{pv}}^{(\hat{i})}, \widetilde{\text{sv}}^{(\hat{i})})$. On the other hand, if the adversary himself registers a (corrupted) party $P_{\hat{i}}^*$ by presenting $\mathsf{RegCorr}(\hat{i} \in [N], \text{pv}^{(\hat{i})})$, then the challenger simply ignores $(\widetilde{\text{pv}}^{(\hat{i})}, \widetilde{\text{sv}}^{(\hat{i})})$.
5. Upon receiving $\mathsf{Ext}(\hat{i})$ for a registered honest $P_{\hat{i}}$, respond via $\text{sv}^{(\hat{i})}$.
6. Upon receiving $\mathsf{Rev}(\mathcal{S}, \hat{j})$, let $\hat{i}^*$ be the smallest element in $\mathcal{S}$. Run $\mathsf{ioP}^{(\hat{i}^*)}$ on input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{j}, \text{sv}_{\hat{i}^*}^{(\hat{j})})$ and respond via the output.
7. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, let $\hat{i}^*$ be the smallest element in ChQ. Choose $\hat{j} \leftarrow \mathsf{ChQ}$ and run $\mathsf{ioP}^{(\hat{i}^*)}$ as $Y_{\text{REAL}} := \mathsf{ioP}^{(\hat{i}^*)}(\mathsf{ChQ}, (x^{(\hat{k})})_{\hat{k}\in\mathsf{ChQ}}, \hat{j}, \text{sv}_{\hat{i}^*}^{(\hat{j})})$ and sample $Y_{\text{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
8. Finally, output whatever $\mathcal{A}$ outputs.

Lemma 13. $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_2$.

Proof. Observe that the only difference between Hyb_1 and Hyb_2 is in the structure of the key and the way it is used in computing the final outcome. Consider any $\hat{i}$. We shall show that, despite the change, the input-output relations of the programs $\mathsf{KD\text{-}Prog}_1^{(\hat{i})}$ and $\mathsf{KD\text{-}Prog}_2^{(\hat{i})}$ are identical. Thence,

$\mathsf{KD\text{-}Prog}_2^{(\hat{i})}$

Constants:

- $\mathsf{F.params}^{(\hat{i})}$
- $(K^{(\hat{i})}[2^{\widetilde{x}}_G], \{\mathsf{patch}(K^{(\hat{i})}[2^{\widetilde{x}}_G], (\hat{k}, \widetilde{x}^{(\hat{k})}))\}_{\hat{k}\in[N]})$
- $\{\widetilde{x}^{(\hat{k})}\}_{\hat{k}\in[N]}$

Input: $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{j}, \mathrm{sv}^{(\hat{j})}_{\hat{i}^*})$, where $\mathcal{S} \subseteq [N]$, $|\mathcal{S}| \leq G$.
Procedure:

1. Check whether $\hat{j} \in \mathcal{S}$; if not, output $\perp$. Then, check whether $(x^{(\hat{j})}, \mathrm{sv}^{(\hat{j})}_{\hat{i}^*}) \in \mathrm{R}^{\mathsf{HCOM}}_{\mathcal{S}}$; if not, output $\perp$. Otherwise, proceed as follows.
2. - If $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}) \notin 2^{\widetilde{x}}_G$, then compute and output $\mathsf{Eval}(K^{(\hat{i})}[2^{\widetilde{x}}_G], (\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}))$.
   - If $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}) \in 2^{\widetilde{x}}_G$, then compute and output $\mathsf{Eval}((K^{(\hat{i})}[2^{\widetilde{x}}_G], \mathsf{patch}(K^{(\hat{i})}[2^{\widetilde{x}}_G], (\hat{j}, \widetilde{x}^{(\hat{j})}))), (\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}))$.

Figure 9: Key-derivation Program

by applying the security property of $\mathsf{i}\mathcal{O}$, we have that $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_2$. Now to prove the functional equivalence of $\mathsf{KD\text{-}Prog}_1^{(\hat{i})}$ and $\mathsf{KD\text{-}Prog}_2^{(\hat{i})}$, we shall show that, there exists a one-to-one relation between all possible keys $K^{(\hat{i})}$ of $\mathsf{KD\text{-}Prog}_1^{(\hat{i})}$ and all possible keys $(K^{(\hat{i})}[2^{\widetilde{x}}_G], \{\mathsf{patch}(K^{(\hat{i})}[2^{\widetilde{x}}_G], (\hat{k}, \widetilde{x}^{(\hat{k})}))\}_{\hat{k}\in[N]})$ of $\mathsf{KD\text{-}Prog}_2^{(\hat{i})}$, and that the input-output relations of the programs $\mathsf{KD\text{-}Prog}_1^{(\hat{i})}$ and $\mathsf{KD\text{-}Prog}_2^{(\hat{i})}$ are identical for the mapped keys. Towards establishing the one-to-one mapping, consider any $\{\widetilde{x}^{(\hat{i})}\}_{\hat{i}\in[N]}$ (which is a part of the constants in both the programs). For any $K^{(\hat{i})}$ for program $\mathsf{KD\text{-}Prog}_1^{(\hat{i})}$, the corresponding key for $\mathsf{KD\text{-}Prog}_2^{(\hat{i})}$ is $(K^{(\hat{i})}[2^{\widetilde{x}}_G], \{\mathsf{patch}(K^{(\hat{i})}[2^{\widetilde{x}}_G], (\hat{k}, \widetilde{x}^{(\hat{k})}))\}_{\hat{k}\in[N]})$. Now consider any input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{j}, \mathrm{sv}^{(\hat{j})}_{\hat{i}^*})$. Let us analyze the outputs for this input as computed by programs $\mathsf{KD\text{-}Prog}_1^{(\hat{i})}$ and $\mathsf{KD\text{-}Prog}_2^{(\hat{i})}$. Observe that the checks performed at Step 1 in both the programs are identical. In the event that this check does not go through, both the programs output the same value, namely, $\perp$. However, if the check goes through, $\mathsf{KD\text{-}Prog}_1^{(\hat{i})}$ computes the function F on $(\mathcal{S}, (x^{(\hat{i})})_{i\in\mathcal{S}})$ with key K; on the other hand, $\mathsf{KD\text{-}Prog}_2^{(\hat{i})}$ works under two cases:

(a). When $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}) \notin 2^{\widetilde{x}}_G$, it outputs $\mathsf{Eval}(K^{(\hat{i})}[2^{\widetilde{x}}_G], (\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}))$. Note that $\mathsf{Eval}(K^{(\hat{i})}[2^{\widetilde{x}}_G], (\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}})) = F(K^{(\hat{i})}, (\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}))$.

(b). When $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}) \in 2^{\widetilde{x}}_G$, it outputs $\mathsf{Eval}((K^{(\hat{i})}[2^{\widetilde{x}}_G], \mathsf{patch}(K^{(\hat{i})}[2^{\widetilde{x}}_G], (\hat{j}, \widetilde{x}^{(\hat{j})}))), (\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}))$. Note that $\mathsf{Eval}((K^{(\hat{i})}[2^{\widetilde{x}}_G], \mathsf{patch}(K^{(\hat{i})}[2^{\widetilde{x}}_G], (\hat{j}, \widetilde{x}^{(\hat{j})}))), (\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}})) = F(K^{(\hat{i})}, (\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}))$.

Hence, the programs $\mathsf{KD\text{-}Prog}_1^{(\hat{i})}$ and $\mathsf{KD\text{-}Prog}_2^{(\hat{i})}$ are functionally equivalent, thus allowing us to apply security of $\mathsf{i}\mathcal{O}$ and concluding that $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_2$.

■

Hyb_3. This experiment is the same as Hyb_2 except for the following modification in the *sequence* of the computations performed by the challenger in computing $(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$ for every $\hat{i}$. Recall that in Hyb_2, towards computing $(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$, the challenger first computed $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$ and then sampled the key $(K^{(\hat{i})}[2^{\widetilde{x}}_G], \{\mathsf{patch}(K^{(\hat{i})}[2^{\widetilde{x}}_G], (\hat{k}, \widetilde{x}^{(\hat{k})}))\}_{\hat{k}\in[N]})$. At a high level, the following is the modification. For every $\hat{i}$, the challenger first obliviously samples all the patches (i.e., oblivious of $\widetilde{x}$); then, as a function of the patches, it computes $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$ for every $\hat{i}$, after which it computes $K^{(\hat{i})}[2^{\widetilde{x}}_G]$ for $\widetilde{x} := (\widetilde{x}^{(1)}, \ldots, \widetilde{x}^{(N)})$. Details follow.

The challenger behaves as follows. It begins by choosing $\text{EXPT} \in \{\text{REAL}, \text{RAND}\}$ uniformly at random, and executes with $\mathcal{A}$ the experiment $\text{EXPT}^{\mathsf{Hyb}_3}_{\mathcal{A}}$ defined as follows.

1. Initiate by running the Setup algorithm. Namely, simply output (λ, N, G) as the output of the Setup algorithm. Next, respond to the adversary's queries as follows.
2. For every $\hat{i} \in [N]$, obliviously compute all patches as follows.
   Obliviously sampling patches.
   - Run $\mathsf{F.params}^{(\hat{i})} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$.
   - Obliviously (of $\widetilde{x}^{(\hat{j})}$) generate the patches: For every $\hat{j} \in [N]$, compute
   $$(\{\mathsf{patch}^{(\hat{i})}_{\hat{j}}\}_{\hat{j}\in[N]}, \mathsf{o.state}^{(\hat{i})}) \leftarrow \mathsf{OPatchGen}(1^\lambda, \mathsf{F.params}^{(\hat{i})}))$$
3. Then, for every $\hat{i} \in [N]$, firstly compute $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$:
   Computing $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
   - Run the statistically-binding parameter-generation algorithm of HCOM: $\mathrm{crs}^{(\hat{i})} \leftarrow \mathsf{HGen}(1^\lambda)$. Sample uniformly at random $\mathrm{ct}^{(\hat{i})}_1, \ldots, \mathrm{ct}^{(\hat{i})}_N \leftarrow \{0,1\}^{\ell'}$. Commit to $\mathrm{ct}^{(\hat{i})}_j$: $\mathrm{com}^{(\hat{i})}_j = \mathsf{HCommit}(\mathrm{crs}^{(\hat{i})}, \mathrm{ct}^{(\hat{i})}_j; \mathrm{open}^{(\hat{i})}_j)$ using uniformly chosen random coins $\mathrm{open}^{(\hat{i})}_j$.

   Let
   $$\widetilde{x}^{(\hat{i})} := (\mathrm{crs}^{(\hat{i})}, \mathrm{com}^{(\hat{i})}_1, \ldots, \mathrm{com}^{(\hat{i})}_N)$$
   and
   $$\widetilde{\mathrm{sv}}^{(\hat{i})} := ((\mathrm{ct}^{(\hat{i})}_1, \mathrm{open}^{(\hat{i})}_1), \ldots, (\mathrm{ct}^{(\hat{i})}_N, \mathrm{open}^{(\hat{i})}_N)).$$
4. Then, for every $\hat{i} \in [N]$, compute the punctured key $K^{(\hat{i})}[2^{\widetilde{x}}_G]$ and $\mathsf{ioP}^{(\hat{i})}$:
   Puncturing a key and computing $\mathsf{ioP}^{(\hat{i})}$.
   - Recall that $\widetilde{x}^{(\hat{j})} \in \{0,1\}^n$. Using $\mathsf{OPuncture}$, consistent with the already generated patches $\{\mathsf{patch}^{(\hat{i})}_{\hat{j}}\}_{\hat{j}\in[N]}$ and with all $\widetilde{x}^{(\hat{j})}$, generate a key punctured at $2^{\widetilde{x}}_G$, where, $\widetilde{x} := (\widetilde{x}^{(1)}, \ldots, \widetilde{x}^{(N)})$:
   $$K^{(\hat{i})}[2^{\widetilde{x}}_G] \leftarrow \mathsf{OPuncture}(1^\lambda, \mathsf{o.state}^{(\hat{i})}, \widetilde{x})$$
   - For $(K^{(\hat{i})}[2^{\widetilde{x}}_G], \{\mathsf{patch}^{(\hat{i})}_{\hat{j}}\}_{\hat{j}\in[N]})$, generate $\mathsf{KD\text{-}Prog}^{(\hat{i})}_3$ exactly the same way as $\mathsf{KD\text{-}Prog}^{(\hat{i})}_2$ and compute $\mathsf{ioP}^{(\hat{i})} \leftarrow i\mathcal{O}(\mathsf{KD\text{-}Prog}^{(\hat{i})}_3)$.

   Let $\mathrm{pv}^{(\hat{i})} = \widetilde{\mathrm{pv}}^{(\hat{i})} = (\widetilde{x}^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$.
5. Upon receiving $\mathsf{Reg}(\hat{i} \in [N])$, then set $(\mathrm{pv}^{(\hat{i})}, \mathrm{sv}^{(\hat{i})}) = (\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$. On the other hand, if the adversary himself registers a (corrupted) party $P^*_{\hat{i}}$ by presenting $\mathsf{RegCorr}(\hat{i} \in [N], \mathrm{pv}^{(\hat{i})})$, then the challenger simply ignores $(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.

6. Upon receiving $\mathsf{Ext}(\hat{i})$ for a registered honest $P_{\hat{i}}$, respond via $\mathrm{sv}^{(\hat{i})}$.
7. Upon receiving $\mathsf{Rev}(\mathcal{S}, \hat{j})$, let $\hat{i^*}$ be the smallest element in $\mathcal{S}$. Run $\mathsf{ioP}^{(\hat{i^*})}$ on input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{j}, \mathrm{sv}_{\hat{i^*}}^{(\hat{j})})$ and respond via the output.
8. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, let $\hat{i^*}$ be the smallest element in ChQ. Choose $\hat{j} \leftarrow \mathsf{ChQ}$ and run $\mathsf{ioP}^{(\hat{i^*})}$ as $Y_{\mathrm{REAL}} := \mathsf{ioP}^{(\hat{i^*})}(\mathsf{ChQ}, (x^{(\hat{k})})_{\hat{k}\in\mathsf{ChQ}}, \hat{j}, \mathrm{sv}_{\hat{i^*}}^{(\hat{j})})$ and sample $Y_{\mathrm{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
9. Finally, output whatever $\mathcal{A}$ outputs.

Lemma 14. $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_3$.

Proof. Observe that the only difference between Hyb_2 and Hyb_3 is in the sequence of the steps of computation in generating $(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$ for all $\hat{i}$. Namely, in Hyb_2, the challenger first computes $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$ for all $\hat{i}$. Then, using algorithms $\mathsf{PatchGen}, \mathsf{Puncture}$, it punctures and patches a key depending on $\widetilde{x} = (\widetilde{x}^{(1)}, \ldots, \widetilde{x}^{(N)})$ to get $(K^{(\hat{i})}[2^{\widetilde{x}}_G], \{\mathsf{patch}(K^{(\hat{i})}[2^{\widetilde{x}}_G], (\hat{j}, \widetilde{x}^{(\hat{j})}))\}_{\hat{j}\in[N]})$ for all $\hat{i}$ and generates the programs the program $\mathsf{ioP}^{(\hat{i})}$ for all $\hat{i}$. On the other hand, in Hyb_3, the challenger first samples the patches obliviously of $\widetilde{x}^{(\hat{j})}$ using the algorithm $\mathsf{OPatchGen}$. Furthermore, the punctured key $K^{(\hat{i})}[2^{\widetilde{x}}_G]$ is sampled using the algorithm $\mathsf{OPuncture}$ using the $\mathsf{o.state}^{(\hat{i})}$ information generated by the algorithm $\mathsf{OPatchGen}$ that had obliviously generated the patches. This difference corresponds exactly to the two modes of generating a punctured key and its block-wise patches, that are guaranteed to give identical joint distributions, from the property of *oblivious patchability* of F. Thus, we have that, the view of the adversary in Hyb_3 is identical to its view in Hyb_2. Hence, $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_3$.

■

Hyb_4. This experiment is the same as Hyb_3 except for the following modification in the distribution from which the challenger samples $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$ for all $\hat{i} \in [N]$. Parse $x^{(\hat{i})} = (\mathrm{crs}^{(\hat{i})}, \mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})})$. Recall that in Hyb_3, $\mathrm{com}_{\hat{j}}^{(\hat{i})}$ is a commitment to the a random value $\mathrm{ct}_{\hat{j}}^{(\hat{i})} \leftarrow \{0,1\}^{\ell'}$, where ℓ' is the length of ciphertexts for the encryption scheme Σ. The modification we introduce in this hybrid, more specifically, is in the way $\mathrm{ct}_{\hat{j}}^{(\hat{i})}$ is sampled: at a high level, $\mathrm{ct}_{\hat{j}}^{(\hat{i})}$ is set as an encryption of a value that can be used by program $\mathsf{ioP}^{(\hat{j})}$ to compute the common key for a set $\mathcal{S}$ for which $\hat{j} \in \mathcal{S}$. More specifically: recall that in Hyb_3, the program $\mathsf{ioP}^{(\hat{j})}$ consists of $K^{(\hat{j})}[2^{\widetilde{x}}_G]$ from which common keys for only some $\mathcal{S}$ can be efficiently computed; namely, if $\mathcal{S}$ is such that for every $\hat{i} \in \mathcal{S}$, $x^{(\hat{i})} = \widetilde{x}^{(\hat{i})}$, then it is not possible to efficiently compute the common key for $\mathcal{S}$ given just $K^{(\hat{j})}[2^{\widetilde{x}}_G]$ using the Eval algorithm. Thus, in Hyb_3, the program $\mathsf{ioP}^{(\hat{j})}$ computes F, which besides $K^{(\hat{j})}[2^{\widetilde{x}}_G]$, gets a patch $\mathsf{patch}_{\hat{i}}^{(\hat{j})}$ (See Figure 9) – this value essentially allows one to still run the Eval algorithm for inputs for which only $K^{(\hat{j})}[2^{\widetilde{x}}_G]$ would not have sufficed. Now the modification in the current hybrid, Hyb_4, is that we encrypt this patch $\mathsf{patch}_{\hat{i}}^{(\hat{j})}$ and commit to the resulting ciphertext to get $\mathrm{com}_{\hat{j}}^{(\hat{i})}$, that is used by the program $\mathsf{ioP}^{(\hat{j})}$. Details follow.

The challenger behaves as follows. It begins by choosing $\textsc{expt} \in \{\textsc{real}, \textsc{rand}\}$ uniformly at random, and executes with $\mathcal{A}$ the experiment $\textsc{expt}_{\mathcal{A}}^{\mathsf{Hyb}_4}$ defined as follows.

1. Initiate by running the Setup algorithm. Namely, simply output (λ, N, G) as the output of the Setup algorithm. Next, respond to the adversary's queries as follows.
2. For every $\hat{i} \in [N]$, obliviously compute all patches as follows.
   Obliviously sampling patches.
   - Run $\mathsf{F.params}^{(\hat{i})} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$.
   - Obliviously (of $\widetilde{x}^{(\hat{j})}$) generate the patches: For every $\hat{j} \in [N]$, compute
   $$(\{\mathsf{patch}_{\hat{j}}^{(\hat{i})}\}_{\hat{j} \in [N]}, \mathsf{o.state}^{(\hat{i})}) \leftarrow \mathsf{OPatchGen}(1^\lambda, \mathsf{F.params}^{(\hat{i})}))$$
3. For every $\hat{i} \in [N]$, sample a public key using the key-generation algorithm of Σ: $(\mathrm{pk}^{(\hat{i})}, \cdot) \leftarrow \mathsf{E.Gen}(1^\lambda)$.
4. Then, for every $\hat{i} \in [N]$, firstly compute $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$:
   Computing $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
   - Run the statistically-binding parameter-generation algorithm of HCOM: $\mathrm{crs}^{(\hat{i})} \leftarrow \mathsf{HGen}(1^\lambda)$. Compute $\mathrm{ct}_1^{(\hat{i})}, \ldots, \mathrm{ct}_N^{(\hat{i})}$ as follows: For every $\hat{j} \in [N]$, set plaintexts $\mathrm{pt}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{patch}_{\hat{i}}^{(\hat{j})}$, and encrypt these plaintexts $\mathrm{ct}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{Enc}(\mathrm{pk}^{(\hat{j})}, \mathrm{pt}_{\hat{j}}^{(\hat{i})})$ (where $\mathsf{patch}_{\hat{i}}^{(\hat{j})}$ is the patch used by $\mathsf{ioP}^{(\hat{j})}$ whenever it needs to compute the common key for any $\mathcal{S}$ (with $\hat{i} \in \mathcal{S}$), for every $\hat{k} \in \mathcal{S}$, $x^{(\hat{k})} = \widetilde{x}^{(\hat{k})}$, and the derived secret key as derived by party $P_{\hat{i}}$). Commit to $\mathrm{ct}_{\hat{j}}^{(\hat{i})}$: $\mathrm{com}_{\hat{j}}^{(\hat{i})} = \mathsf{HCommit}(\mathrm{crs}^{(\hat{i})}, \mathrm{ct}_{\hat{j}}^{(\hat{i})}; \mathrm{open}_{\hat{j}}^{(\hat{i})})$ using uniformly chosen random coins $\mathrm{open}_{\hat{j}}^{(\hat{i})}$.

   Let
   $$\widetilde{x}^{(\hat{i})} := (\mathrm{crs}^{(\hat{i})}, \mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})})$$
   and
   $$\widetilde{\mathrm{sv}}^{(\hat{i})} := ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})})).$$

5. Then, for every $\hat{i} \in [N]$, compute the punctured key $K^{(\hat{i})}[2^{\widetilde{x}}_G]$ and $\mathsf{ioP}^{(\hat{i})}$:

   Puncturing a key and computing $\mathsf{ioP}^{(\hat{i})}$.

   - Recall that $\widetilde{x}^{(\hat{j})} \in \{0,1\}^n$. Using $\mathsf{OPuncture}$, consistent with the already generated patches $\{\mathsf{patch}^{(\hat{i})}_{\hat{j}}\}_{\hat{j}\in[N]}$ and with all $\widetilde{x}^{(\hat{j})}$, generate a key punctured at $2^{\widetilde{x}}_G$, where, $\widetilde{x} := (\widetilde{x}^{(1)}, \ldots, \widetilde{x}^{(N)})$:

     $$K^{(\hat{i})}[2^{\widetilde{x}}_G] \leftarrow \mathsf{OPuncture}(1^\lambda, \mathsf{o.state}^{(\hat{i})}, \widetilde{x})$$

   - For $(K^{(\hat{i})}[2^{\widetilde{x}}_G], \{\mathsf{patch}^{(\hat{i})}_{\hat{j}}\}_{\hat{j}\in[N]})$, generate $\mathsf{KD\text{-}Prog}^{(\hat{i})}_4$ exactly the same way as $\mathsf{KD\text{-}Prog}^{(\hat{i})}_3$ and compute $\mathsf{ioP}^{(\hat{i})} \leftarrow i\mathcal{O}(\mathsf{KD\text{-}Prog}^{(\hat{i})}_4)$.

   Let $\mathrm{pv}^{(\hat{i})} = \widetilde{\mathrm{pv}}^{(\hat{i})} = (\widetilde{x}^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$.
6. Upon receiving $\mathsf{Reg}(\hat{i} \in [N])$, then set $(\mathrm{pv}^{(\hat{i})}, \mathrm{sv}^{(\hat{i})}) = (\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$. On the other hand, if the adversary himself registers a (corrupted) party $P^*_{\hat{i}}$ by presenting $\mathsf{RegCorr}(\hat{i} \in [N], \mathrm{pv}^{(\hat{i})})$, then the challenger simply ignores $(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
7. Upon receiving $\mathsf{Ext}(\hat{i})$ for a registered honest $P_{\hat{i}}$, respond via $\mathrm{sv}^{(\hat{i})}$.
8. Upon receiving $\mathsf{Rev}(\mathcal{S}, \hat{j})$, let $\hat{i^*}$ be the smallest element in $\mathcal{S}$. Run $\mathsf{ioP}^{(\hat{i^*})}$ on input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{j}, \mathrm{sv}^{(\hat{j})}_{\hat{i^*}})$ and respond via the output.
9. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, let $\hat{i^*}$ be the smallest element in ChQ. Choose $\hat{j} \leftarrow \mathsf{ChQ}$ and run $\mathsf{ioP}^{(\hat{i^*})}$ as $Y_{\text{REAL}} := \mathsf{ioP}^{(\hat{i^*})}(\mathsf{ChQ}, (x^{(\hat{k})})_{\hat{k}\in\mathsf{ChQ}}, \hat{j}, \mathrm{sv}^{(\hat{j})}_{\hat{i^*}})$ and sample $Y_{\text{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
10. Finally, output whatever $\mathcal{A}$ outputs.

Lemma 15. $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_4$.

Proof. Observe that the only difference between Hyb_3 and Hyb_4 is in the distribution from which the challenger samples $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$ for all $\hat{i} \in [N]$. More specifically, parse $x^{(\hat{i})} = (\mathrm{crs}^{(\hat{i})}, \mathrm{com}^{(\hat{i})}_1, \ldots, \mathrm{com}^{(\hat{i})}_N)$. On one hand, in Hyb_3, $\mathrm{com}^{(\hat{i})}_{\hat{j}}$ commits to a random element. On the other hand, in Hyb_4, $\mathrm{com}^{(\hat{i})}_{\hat{j}}$ is a commitment of a ciphertext that encrypts $\mathsf{patch}^{(\hat{j})}_{\hat{i}}$.

Observe that in either of the hybrids, none of the programs (generated by the challenger on behalf of the honest parties) contains the secret key for the encryption scheme. We shall thus see that owing to CPA security of the encryption scheme $\Sigma = (\mathsf{E.Gen}, \mathsf{Enc}, \mathsf{Dec})$, the two hybrids are computationally indistinguishable.

Let $\mathcal{A}$ be an adversary that distinguishes Hyb_3 and Hyb_4; that is, $|\mathsf{Adv}^{\mathsf{Hyb}_3}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathsf{Hyb}_4}_{\mathcal{A}}(\lambda)|$ is non-negligible in λ, where, $\mathsf{Adv}^{\mathsf{Hyb}_i}_{\mathcal{A}}(\lambda)$ is as defined in Equation (6). Recall that, for any i, $\mathsf{Adv}^{\mathsf{Hyb}_i}_{\mathcal{A}}(\lambda) := |\Pr[\text{REAL}^{\mathsf{Hyb}_i}_{\mathcal{A}} \to 1] - \Pr[\text{RAND}^{\mathsf{Hyb}_i}_{\mathcal{A}} \to 1]|$. This implies that there exists $\widetilde{\text{EXPT}} \in \{\text{REAL}, \text{RAND}\}$ such that $|\Pr[\widetilde{\text{EXPT}}^{\mathsf{Hyb}_3}_{\mathcal{A}} \to 1] - \Pr[\widetilde{\text{EXPT}}^{\mathsf{Hyb}_4}_{\mathcal{A}} \to 1]| = \epsilon$, where, $\epsilon = \epsilon(\lambda)$ is non-negligible in λ. We shall build an adversary $\mathcal{B}$ that emulates either $\widetilde{\text{EXPT}}^{\mathsf{Hyb}_3}_{\mathcal{A}}$ or $\widetilde{\text{EXPT}}^{\mathsf{Hyb}_4}_{\mathcal{A}}$ and breaks the CPA security of Σ. Details follow.

For simplicity, we consider a variant of the standard CPA game that is equivalent to the standard CPA game through a simple hybrid argument that loses a factor of $1/N^2$ in the adversary's advantage. The modified game is as follows. Upon receiving N-many public keys $\mathrm{pk}^{(\hat{i})}$, the adversary $\mathcal{B}$ gives to the challenger N-many N-vectors of plaintext messages and receives N-many N-vectors of

elements that are either random elements or encryptions of the given vector of plaintexts with the corresponding public key (Recall that the encryption scheme we use enjoys pseudorandom ciphertexts). The objective of the adversary would be to tell apart the two cases. Note that none of the programs generated by the challenger (on behalf of the honest parties), in either $\mathsf{KD\text{-}Prog}_3^{(\hat{i})}$ and $\mathsf{KD\text{-}Prog}_4^{(\hat{i})}$, uses the secret keys of the encryption scheme; hence, the description of $\mathcal{B}$ is well-defined.

Description of $\mathcal{B}$. Corresponding respectively to the two plaintext vectors, the adversary $\mathcal{B}$ emulates to $\mathcal{A}$ either $\widetilde{\mathrm{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_3}$ or $\widetilde{\mathrm{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_4}$, respectively, and exploits the success probability of $\mathcal{A}$ in the CPA game. Let the experiment with $\mathcal{A}$, as emulated by $\mathcal{B}$, be denoted by $\widetilde{\mathrm{EXPT}}_{3-4}$. This experiment is the same as $\widetilde{\mathrm{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_4}$ except that instead of performing the encryptions by himself, the $\mathcal{B}$ presents to its challenger$N$-many N-vectors of patches as plaintexts. We shall highlight this deviation from $\widetilde{\mathrm{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_4}$. In detail, upon receiving pk from its challenger, $\mathcal{B}$ proceeds as follows.

1. Initiate by running the Setup algorithm. Namely, simply output (λ, N, G) as the output of the Setup algorithm. Next, respond to the adversary's queries as follows.
2. For every $\hat{i} \in [N]$, obliviously compute all patches as follows.
   Obliviously sampling patches.
   - Run $\mathsf{F.params}^{(\hat{i})} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$.
   - Obliviously (of $\widetilde{x}^{(\hat{j})}$) generate the patches: For every $\hat{j} \in [N]$, compute
   $$(\{\mathsf{patch}_{\hat{j}}^{(\hat{i})}\}_{\hat{j}\in[N]}, \mathsf{o.state}^{(\hat{i})}) \leftarrow \mathsf{OPatchGen}(1^\lambda, \mathsf{F.params}^{(\hat{i})}))$$
3. For every $\hat{i} \in [N]$, sample a public key using the key-generation algorithm of Σ: ~~$(\mathsf{pk}^{(\hat{i})}, \cdot) \leftarrow \mathsf{E.Gen}(1^\lambda)$~~.
4. Then, for every $\hat{i} \in [N]$, firstly compute $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$:
   Computing $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
   - Run the statistically-binding parameter-generation algorithm of HCOM: $\mathrm{crs}^{(\hat{i})} \leftarrow \mathsf{HGen}(1^\lambda)$. Compute $\mathrm{ct}_1^{(\hat{i})}, \ldots, \mathrm{ct}_N^{(\hat{i})}$ as follows: For every $\hat{j} \in [N]$, set $\mathrm{pt}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{patch}_{\hat{i}}^{(\hat{j})}$, and present $\mathsf{patch}_{\hat{i}}^{(\hat{j})}$ to the challenger; let $\mathrm{ct}_{\hat{j}}^{(\hat{i})}$ be the corresponding value received. Commit to $\mathrm{ct}_{\hat{j}}^{(\hat{i})}$: $\mathrm{com}_{\hat{j}}^{(\hat{i})} = \mathsf{HCommit}(\mathrm{crs}^{(\hat{i})}, \mathrm{ct}_{\hat{j}}^{(\hat{i})}; \mathrm{open}_{\hat{j}}^{(\hat{i})})$ using uniformly chosen random coins $\mathrm{open}_{\hat{j}}^{(\hat{i})}$.

   Let
   $$\widetilde{x}^{(\hat{i})} := (\mathrm{crs}^{(\hat{i})}, \mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})})$$
   and
   $$\widetilde{\mathrm{sv}}^{(\hat{i})} := ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})})).$$
5. Then, for every $\hat{i} \in [N]$, compute the punctured key $K^{(\hat{i})}[2_G^{\widetilde{x}}]$ and $\mathsf{ioP}^{(\hat{i})}$:
   Puncturing a key and computing $\mathsf{ioP}^{(\hat{i})}$.
   - Recall that $\widetilde{x}^{(\hat{j})} \in \{0,1\}^n$. Using $\mathsf{OPuncture}$, consistent with the already generated patches $\{\mathsf{patch}_{\hat{j}}^{(\hat{i})}\}_{\hat{j}\in[N]}$ and with all $\widetilde{x}^{(\hat{j})}$, generate a key punctured at $2_G^{\widetilde{x}}$, where, $\widetilde{x} := (\widetilde{x}^{(1)}, \ldots, \widetilde{x}^{(N)})$:
   $$K^{(\hat{i})}[2_G^{\widetilde{x}}] \leftarrow \mathsf{OPuncture}(1^\lambda, \mathsf{o.state}^{(\hat{i})}, \widetilde{x})$$

- For $(K^{(\hat{i})}[2^{\widetilde{x}}_G], \{\mathsf{patch}^{(\hat{i})}_{\hat{j}}\}_{\hat{j}\in[N]})$, generate $\mathsf{KD\text{-}Prog}^{(\hat{i})}_{3-4}$ exactly the same way as $\mathsf{KD\text{-}Prog}^{(\hat{i})}_{3}$ and compute $\mathsf{ioP}^{(\hat{i})} \leftarrow \mathsf{i}\mathcal{O}(\mathsf{KD\text{-}Prog}^{(\hat{i})}_{3-4})$

Let $\mathrm{pv}^{(\hat{i})} = \widetilde{\mathrm{pv}}^{(\hat{i})} = (\widetilde{x}^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$.

6. Upon receiving $\mathsf{Reg}(\hat{i} \in [N])$, then set $(\mathrm{pv}^{(\hat{i})}, \mathrm{sv}^{(\hat{i})}) = (\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$. On the other hand, if the adversary himself registers a (corrupted) party $P^*_{\hat{i}}$ by presenting $\mathsf{RegCorr}(\hat{i} \in [N], \mathrm{pv}^{(\hat{i})})$, then the challenger simply ignores $(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
7. Upon receiving $\mathsf{Ext}(\hat{i})$ for a registered honest $P_{\hat{i}}$, respond via $\mathrm{sv}^{(\hat{i})}$.
8. Upon receiving $\mathsf{Rev}(\mathcal{S}, \hat{j})$, let $\hat{i}^*$ be the smallest element in $\mathcal{S}$. Run $\mathsf{ioP}^{(\hat{i}^*)}$ on input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{j}, \mathrm{sv}^{(\hat{j})}_{\hat{i}^*})$ and respond via the output.
9. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, let $\hat{i}^*$ be the smallest element in ChQ. Choose $\hat{j} \leftarrow \mathsf{ChQ}$ and run $\mathsf{ioP}^{(\hat{i}^*)}$ as $Y_{\text{REAL}} := \mathsf{ioP}^{(\hat{i}^*)}(\mathsf{ChQ}, (x^{(\hat{k})})_{\hat{k}\in\mathsf{ChQ}}, \hat{j}, \mathrm{sv}^{(\hat{j})}_{\hat{i}^*})$ and sample $Y_{\text{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via $Y_{\widetilde{\text{EXPT}}}$.

Finally, output whatever $\mathcal{A}$ outputs.

Analysis. Observe that if, on one hand, the responses by the CPA-game challenger are randomly distributed elements, then the view of $\mathcal{A}$ in $\widetilde{\text{EXPT}}_{3-4}$ is identical to that in $\widetilde{\text{EXPT}}$ of Hyb_3. On the other hand, if the responses by the challenger are encryptions of the patches, then the view of $\mathcal{A}$ in $\widetilde{\text{EXPT}}_{3-4}$ is identical to that in $\widetilde{\text{EXPT}}$ of Hyb_4. Therefore, $\mathcal{B}$ distinguishes the ciphertexts of two distinct plaintexts with probability $|\Pr[\widetilde{\text{EXPT}}^{\mathsf{Hyb}_3}_{\mathcal{A}} \to 1] - \Pr[\widetilde{\text{EXPT}}^{\mathsf{Hyb}_4}_{\mathcal{A}} \to 1]| = \epsilon$ which is non-negligible, hence breaking the CPA security of Σ. ■

Hyb_5. This experiment is the same as Hyb_4 except for the following modification in the way in which the programs computed by the challenger (on behalf of the honest parties) obtain the patches. Recall that in Hyb_4, in program $\mathsf{ioP}^{(\hat{i})}$, the patches $\{\mathsf{patch}_{\hat{j}}^{(\hat{i})}\}_{\hat{j}\in[N]}$, corresponding to $\{\widetilde{x}^{(\hat{j})}\}_{\hat{j}\in[N]}$, formed a part of the constants of the program. Also, recall that the patch $\mathsf{patch}_{\hat{j}}^{(\hat{i})}$ is utilized by the program when it is invoked with an input of the following form: $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{j}, \mathrm{sv}_{i^*}^{(\hat{j})})$, where $x^{(\hat{k})} = \widetilde{x}^{(\hat{k})}$ for all $\hat{k} \in \mathcal{S}$.

The modification we introduce here in Hyb_5 is that in every program $\mathsf{ioP}^{(\hat{i})}$ generated by the challenger, we *do not include any of* $\{\mathsf{patch}_{\hat{j}}^{(\hat{i})}\}_{\hat{j}\in[N]}$ *as constants to the program*. Instead, we include the secret key sk of the encryption scheme as a part of the constants in the program. Then, upon an input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{j}, \mathrm{sv}_{i^*}^{(\hat{j})})$, recall that $\mathrm{sv}_{i^*}^{(\hat{j})}$, when parsed, contains an opening of a commitment in $x^{(\hat{j})}$ to an encryption of the required patch. The program simply decrypts that ciphertext to obtain the patch. Note that the commitment in question is generated by the challenger using a CRS (of the commitment scheme) that is also sampled by the challenger using the statistically-binding mode of the commitment scheme. Hence, with all but negligible probability, the resulting programs generated by the challenger do obtain the required patches when run on such inputs. Thus, with all but negligible probability, the programs are functionally equivalent to their respective counterparts in Hyb_4. Details follow.

The challenger behaves as follows. It begins by choosing $\textsc{expt} \in \{\textsc{real}, \textsc{rand}\}$ uniformly at random, and executes with $\mathcal{A}$ the experiment $\textsc{expt}_{\mathcal{A}}^{\mathsf{Hyb}_5}$ defined as follows.

1. Initiate by running the Setup algorithm. Namely, simply output (λ, N, G) as the output of the Setup algorithm. Next, respond to the adversary's queries as follows.
2. For every $\hat{i} \in [N]$, obliviously compute all patches as follows.
   Obliviously sampling patches.
   - Run $\mathsf{F.params}^{(\hat{i})} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$.
   - Obliviously (of $\widetilde{x}^{(\hat{j})}$) generate the patches: For every $\hat{j} \in [N]$, compute
   $$(\{\mathsf{patch}_{\hat{j}}^{(\hat{i})}\}_{\hat{j}\in[N]}, \mathsf{o.state}^{(\hat{i})}) \leftarrow \mathsf{OPatchGen}(1^\lambda, \mathsf{F.params}^{(\hat{i})}))$$
3. For every $\hat{i} \in [N]$, sample a public key using the key-generation algorithm of Σ: $(\mathrm{pk}^{(\hat{i})}, \underline{\mathrm{sk}^{(\hat{i})}}) \leftarrow \mathsf{E.Gen}(1^\lambda)$.
4. Then, for every $\hat{i} \in [N]$, firstly compute $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$:
   Computing $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
   - Run the statistically-binding parameter-generation algorithm of HCOM: $\mathrm{crs}^{(\hat{i})} \leftarrow \mathsf{HGen}(1^\lambda)$. Compute $\mathrm{ct}_1^{(\hat{i})}, \ldots, \mathrm{ct}_N^{(\hat{i})}$ as follows: For every $\hat{j} \in [N]$, set plaintexts $\mathrm{pt}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{patch}_{\hat{i}}^{(\hat{j})}$, and encrypt these plaintexts $\mathrm{ct}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{Enc}(\mathrm{pk}^{(\hat{j})}, \mathrm{pt}_{\hat{j}}^{(\hat{i})})$ (where $\mathsf{patch}_{\hat{i}}^{(\hat{j})}$ is the patch used by $\mathsf{ioP}^{(\hat{j})}$ whenever it needs to compute the common key for any $\mathcal{S}$ (with $\hat{i} \in \mathcal{S}$), for every $\hat{k} \in \mathcal{S}$, $x^{(\hat{k})} = \widetilde{x}^{(\hat{k})}$, and the derived secret key as derived by party $P_{\hat{i}}$). Commit to $\mathrm{ct}_{\hat{j}}^{(\hat{i})}$: $\mathrm{com}_{\hat{j}}^{(\hat{i})} = \mathsf{HCommit}(\mathrm{crs}^{(\hat{i})}, \mathrm{ct}_{\hat{j}}^{(\hat{i})}; \mathrm{open}_{\hat{j}}^{(\hat{i})})$ using uniformly chosen random coins $\mathrm{open}_{\hat{j}}^{(\hat{i})}$.

   Let
   $$\widetilde{x}^{(\hat{i})} := (\mathrm{crs}^{(\hat{i})}, \mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})})$$

and

$$\widetilde{\mathrm{sv}}^{(\hat{i})} := ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})})).$$

5. Then, for every $\hat{i} \in [N]$, compute the punctured key $K^{(\hat{i})}[2^{\widetilde{x}}_G]$ and $\mathsf{ioP}^{(\hat{i})}$:

   Puncturing a key and computing $\mathsf{ioP}^{(\hat{i})}$.

   - Recall that $\widetilde{x}^{(\hat{j})} \in \{0,1\}^n$. Using $\mathsf{OPuncture}$, consistent with the already generated patches $\{\mathsf{patch}_{\hat{j}}^{(\hat{i})}\}_{\hat{j}\in[N]}$ and with all $\widetilde{x}^{(\hat{j})}$, generate a key punctured at $2^{\widetilde{x}}_G$, where, $\widetilde{x} := (\widetilde{x}^{(1)}, \ldots, \widetilde{x}^{(N)})$:

     $$K^{(\hat{i})}[2^{\widetilde{x}}_G] \leftarrow \mathsf{OPuncture}(1^\lambda, \mathsf{o.state}^{(\hat{i})}, \widetilde{x})$$

   - For $K^{(\hat{i})}[2^{\widetilde{x}}_G]$, generate $\mathsf{KD\text{-}Prog}_5^{(\hat{i})}$ exactly the same way as $\mathsf{KD\text{-}Prog}_4^{(\hat{i})}$ and compute $\mathsf{ioP}^{(\hat{i})} \leftarrow i\mathcal{O}(\mathsf{KD\text{-}Prog}_5^{(\hat{i})})$.

   Let $\mathrm{pv}^{(\hat{i})} = \widetilde{\mathrm{pv}}^{(\hat{i})} = (\widetilde{x}^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$.
6. Upon receiving $\mathsf{Reg}(\hat{i} \in [N])$, then set $(\mathrm{pv}^{(\hat{i})}, \mathrm{sv}^{(\hat{i})}) = (\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$. On the other hand, if the adversary himself registers a (corrupted) party $P_{\hat{i}}^*$ by presenting $\mathsf{RegCorr}(\hat{i} \in [N], \mathrm{pv}^{(\hat{i})})$, then the challenger simply ignores $(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
7. Upon receiving $\mathsf{Ext}(\hat{i})$ for a registered honest $P_{\hat{i}}$, respond via $\mathrm{sv}^{(\hat{i})}$.
8. Upon receiving $\mathsf{Rev}(\mathcal{S}, \hat{j})$, let $\hat{i^*}$ be the smallest element in $\mathcal{S}$. Run $\mathsf{ioP}^{(\hat{i^*})}$ on input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{j}, \mathrm{sv}_{\hat{i^*}}^{(\hat{j})})$ and respond via the output.
9. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, let $\hat{i^*}$ be the smallest element in ChQ. Choose $\hat{j} \leftarrow \mathsf{ChQ}$ and run $\mathsf{ioP}^{(\hat{i^*})}$ as $Y_{\mathrm{REAL}} := \mathsf{ioP}^{(\hat{i^*})}(\mathsf{ChQ}, (x^{(\hat{k})})_{\hat{k}\in\mathsf{ChQ}}, \hat{j}, \mathrm{sv}_{\hat{i^*}}^{(\hat{j})})$ and sample $Y_{\mathrm{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
10. Finally, output whatever $\mathcal{A}$ outputs.

Lemma 16. $\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_5$.

Proof. Observe that the only difference between the two hybrids is in the way the programs $\mathsf{KD\text{-}Prog}_4^{(\hat{i})}$ and $\mathsf{KD\text{-}Prog}_5^{(\hat{i})}$ behave when input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{j}, \mathrm{sv}_{\hat{i^*}}^{(\hat{j})})$ which goes through the check performed in the Step 1, and for which $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}) \in 2^{\widetilde{x}}_G$. Before we proceed, we note that in both the programs, upon such an input, the output is computed as $\mathsf{Eval}((K^{(\hat{i})}[2^{\widetilde{x}}_G], \mathsf{patch}_{\hat{j}}^{(\hat{i})}), (\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}))$; however, the difference is in how the programs obtain $\mathsf{patch}_{\hat{j}}^{(\hat{i})}$. In the former hybrid, this value formed a part of the constants within the program. However, in the latter hybrid, this is no longer a part of the program's constants; the program instead obtains it as follows. Parse $\mathrm{sv}_{\hat{i^*}}^{(\hat{j})} = (*, (\mathrm{ct}_{\hat{i}}^{(\hat{j})}, *))$. Then obtain $\mathsf{patch}_{\hat{j}}^{(\hat{i})}$ by decrypting $\mathrm{ct}_{\hat{i}}^{(\hat{j})}$ as $\mathsf{patch}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{Dec}(\mathrm{sk}^{(\hat{i})}, \mathrm{ct}_{\hat{i}}^{(\hat{j})})$.

Thus, if we show that, despite the disparate ways of obtaining $\mathsf{patch}_{\hat{j}}^{(\hat{i})}$, $\mathsf{patch}_{\hat{j}}^{(\hat{i})}$ is distributed the same way w.r.t. to the rest of the elements in the game, then we can conclude that the programs are functionally equivalent.

In Hyb_5, recall how the challenger samples $(x^{(\hat{j})}, \mathrm{sv}_{\hat{i^*}}^{(\hat{j})})$ for every $\hat{j}$. The challenger first computes $\mathsf{patch}_{\hat{j}}^{(\hat{i})}$ for all $\hat{i}$ (just like in Hyb_4). Then it encrypts $\mathsf{patch}_{\hat{j}}^{(\hat{i})}$ with pk, denoting the plaintext $\mathsf{patch}_{\hat{j}}^{(\hat{i})}$ by $\mathrm{pt}_{\hat{i}}^{(\hat{j})}$ and the resulting ciphertext by $\mathrm{ct}_{\hat{i}}^{(\hat{j})}$. Then, it commits using $\mathrm{crs}^{(\hat{j})}$ to $\mathrm{ct}_{\hat{i}}^{(\hat{j})}$ with

$\mathsf{KD\text{-}Prog}_5^{(\hat{i})}$

Constants:

- $\mathsf{F.params}^{(\hat{i})}$
- $K^{(\hat{i})}[2^{\widetilde{x}}_G]$, ~~$(\{\mathsf{patch}^{(\hat{i})}_{\hat{k}}\}_{\hat{k}\in[N]})$~~
- $\{\widetilde{x}^{(\hat{k})}\}_{\hat{k}\in[N]}$
- $\mathrm{sk}^{(\hat{i})}$

Input: $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{j}, \mathrm{sv}^{(\hat{j})}_{\hat{i}^*})$, where $\mathcal{S}\subseteq[N]$, $|\mathcal{S}|\leq G$.
Procedure:

1. Check whether $\hat{j}\in\mathcal{S}$; if not, output $\perp$. Then, check whether $(x^{(\hat{j})}, \mathrm{sv}^{(\hat{j})}_{\hat{i}^*})\in \mathrm{R}^{\mathsf{HCOM}}_{\mathcal{S}}$; if not, output $\perp$. Otherwise, proceed as follows.
2. - If $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}})\notin 2^{\widetilde{x}}_G$, then compute and output $\mathsf{Eval}(K^{(\hat{i})}[2^{\widetilde{x}}_G], (\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}))$.
   - If $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}})\in 2^{\widetilde{x}}_G$, then, parse $\mathrm{sv}^{(\hat{j})}_{\hat{i}^*} = (*, (\mathrm{ct}^{(\hat{j})}_{\hat{i}}, *))$ and obtain $\mathsf{patch}^{(\hat{i})}_{\hat{j}}$ by decrypting $\mathrm{ct}^{(\hat{j})}_{\hat{i}}$ as $\mathsf{patch}^{(\hat{i})}_{\hat{j}}\leftarrow\mathsf{Dec}(\mathrm{sk}^{(\hat{i})}, \mathrm{ct}^{(\hat{j})}_{\hat{i}})$, compute and output $\mathsf{Eval}((K^{(\hat{i})}[2^{\widetilde{x}}_G], \mathsf{patch}^{(\hat{i})}_{\hat{j}}), (\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}))$.

Figure 10: Key-derivation Program

random coins $\mathrm{open}^{(\hat{j})}_{\hat{i}}$ to get $\mathrm{com}^{(\hat{j})}_{\hat{i}}$, assigning $\mathrm{sv}^{(\hat{j})} = (*, (\mathrm{ct}^{(\hat{j})}_1, \mathrm{open}^{(\hat{j})}_1), \ldots, (\mathrm{ct}^{(\hat{j})}_N, \mathrm{open}^{(\hat{j})}_N))$ and $x^{(\hat{j})}\leftarrow(\mathrm{crs}^{(\hat{j})}, \mathrm{com}^{(\hat{j})}_1, \ldots, \mathrm{com}^{(\hat{j})}_N)$. Then, w.r.t. a set $\mathcal{S}$, $\mathrm{sv}^{(\hat{j})}$ is transformed into $\mathrm{sv}^{(\hat{j})}_{\hat{i}^*}$ using the function NR. We note from Equation (3) that NR transforms only the first component of $\mathrm{sv}^{(\hat{j})}$ while leaving $(\mathrm{ct}^{(\hat{j})}_{\hat{i}}, \mathrm{open}^{(\hat{j})}_{\hat{i}})$ intact. Thus, $\mathrm{sv}^{(\hat{j})}_{\hat{i}^*} = (\mathrm{ct}^{(\hat{j})}_{\hat{i}}, \mathrm{open}^{(\hat{j})}_{\hat{i}})$.

The crucial fact for our current interest is that $\mathrm{sv}^{(\hat{j})}_{\hat{i}^*}$ contains $\mathrm{ct}^{(\hat{j})}_{\hat{i}}$, where, $\mathrm{ct}^{(\hat{j})}_{\hat{i}}$ is an encryption of $\mathsf{patch}^{(\hat{i})}_{\hat{j}}$. Next, recall that the program $\mathsf{KD\text{-}Prog}^{(\hat{i})}_5$ obtains $\mathsf{patch}^{(\hat{i})}_{\hat{j}}$ by decrypting $\mathrm{ct}^{(\hat{j})}_{\hat{i}}$ with sk. Thus, we will be able to argue function equivalence of the two programs $\mathsf{KD\text{-}Prog}^{(\hat{i})}_4$ and $\mathsf{KD\text{-}Prog}^{(\hat{i})}_5$, provided, for every input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{j}, \mathrm{sv}^{(\hat{j})}_{\hat{i}^*})$ that passes the check in the program, if $x^{(\hat{j})} = \widetilde{x}^{(\hat{j})}$, then $\mathrm{sv}^{(\hat{j})}_{\hat{i}^*}$ contains $\mathrm{ct}^{(\hat{j})}_{\hat{i}}$, where $\mathrm{ct}^{(\hat{j})}_{\hat{i}}$ is an encryption of $\mathsf{patch}^{(\hat{i})}_{\hat{j}}$.

We now prove that, for every $\hat{j}\in[N]$, for every $x^{(\hat{j})}$, there exists only one value for $\mathrm{sv}^{(\hat{j})}_{\hat{i}^*}$ is unique upto containing $\mathrm{ct}^{(\hat{j})}_{\hat{i}}$ such that for an input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{j}, (\mathrm{sv}^{(\hat{j})}_{\hat{i}^*})')$ (with $x^{(\hat{j})} = \widetilde{x}^{(\hat{j})}$), if the check in Step 1 goes through then $(\mathrm{sv}^{(\hat{j})}_{\hat{i}^*})'$ contains $\mathrm{ct}^{(\hat{j})}_{\hat{i}}$.

Now recall that for $\mathrm{crs}^{(\hat{j})}$ sampled using HGen, the resulting commitment scheme unconditionally binding. Hence, $\mathrm{sv}^{(\hat{j})}_{\hat{i}^*}$ for which the check passes is unique upto containing $\mathrm{ct}^{(\hat{j})}_{\hat{i}}$ such that $\mathrm{com}^{(\hat{i})}_{\hat{j}} = \mathsf{HCommit}(\mathrm{crs}^{(\hat{i})}, \mathrm{ct}^{(\hat{i})}_{\hat{j}}; \mathrm{open}^{(\hat{i})}_{\hat{j}})$. Thus, on such an input, the program $\mathsf{KD\text{-}Prog}^{(\hat{i})}_5$ is assured to obtain $\mathsf{patch}^{(\hat{i})}_{\hat{j}}$ with all but negligible probability, hence maintaining functional equivalence with $\mathsf{KD\text{-}Prog}^{(\hat{i})}_4$ even for the case when $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}})\in 2^{\widetilde{x}}_G$.

■

Hyb_6. This experiment is the same as Hyb_5 except for the following modification in the distribution from which the challenger samples $\mathrm{crs}^{(\hat{i})}$ for every $\hat{i} \in [N]$. Recall that the challenger in Hyb_5 sampled $\mathrm{crs}^{(\hat{i})}$ using algorithm HGen ($\mathrm{crs}^{(\hat{i})} \leftarrow \mathsf{HGen}(1^\lambda)$). Now the modification for Hyb_6 is that the challenger samples $\mathrm{crs}^{(\hat{i})}$ as $(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{crs}}^{(\hat{i})}) \leftarrow \mathsf{HTGen}(1^\lambda)$. Details follow.

The challenger behaves as follows. It begins by choosing $\textsc{expt} \in \{\textsc{real}, \textsc{rand}\}$ uniformly at random, and executes with $\mathcal{A}$ the experiment $\textsc{expt}_{\mathcal{A}}^{\mathsf{Hyb}_6}$ defined as follows.

1. Initiate by running the Setup algorithm. Namely, simply output (λ, N, G) as the output of the Setup algorithm. Next, respond to the adversary's queries as follows.
2. For every $\hat{i} \in [N]$, obliviously compute all patches as follows.
   Obliviously sampling patches.
   - Run $\mathsf{F.params}^{(\hat{i})} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$.
   - Obliviously (of $\widetilde{x}^{(\hat{j})}$) generate the patches: For every $\hat{j} \in [N]$, compute
   $$(\{\mathsf{patch}_{\hat{j}}^{(\hat{i})}\}_{\hat{j}\in[N]}, \mathsf{o.state}^{(\hat{i})}) \leftarrow \mathsf{OPatchGen}(1^\lambda, \mathsf{F.params}^{(\hat{i})}))$$
3. For every $\hat{i} \in [N]$, sample a public key using the key-generation algorithm of Σ: $(\mathrm{pk}^{(\hat{i})}, \mathrm{sk}^{(\hat{i})}) \leftarrow \mathsf{E.Gen}(1^\lambda)$.
4. Then, for every $\hat{i} \in [N]$, firstly compute $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$:
   Computing $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
   - Run again the trapdoor parameter-generation algorithm of HCOM: $(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{crs}}^{(\hat{i})}) \leftarrow \mathsf{HTGen}(1^\lambda)$. (Ignore $\mathrm{aux}_{\mathrm{crs}}^{(\hat{i})}$). Compute $\mathrm{ct}_1^{(\hat{i})}, \ldots, \mathrm{ct}_N^{(\hat{i})}$ as follows: For every $\hat{j} \in [N]$, set plaintexts $\mathrm{pt}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{patch}_{\hat{i}}^{(\hat{j})}$, and encrypt these plaintexts $\mathrm{ct}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{Enc}(\mathrm{pk}^{(\hat{j})}, \mathrm{pt}_{\hat{j}}^{(\hat{i})})$ (where $\mathsf{patch}_{\hat{i}}^{(\hat{j})}$ is the patch used by $\mathsf{ioP}^{(\hat{j})}$ whenever it needs to compute the common key for any $\mathcal{S}$ (with $\hat{i} \in \mathcal{S}$), for every $\hat{k} \in \mathcal{S}$, $x^{(\hat{k})} = \widetilde{x}^{(\hat{k})}$, and the derived secret key as derived by party $P_{\hat{i}}$). Commit to $\mathrm{ct}_{\hat{j}}^{(\hat{i})}$: $\mathrm{com}_{\hat{j}}^{(\hat{i})} = \mathsf{HCommit}(\mathrm{crs}^{(\hat{i})}, \mathrm{ct}_{\hat{j}}^{(\hat{i})}; \mathrm{open}_{\hat{j}}^{(\hat{i})})$ using uniformly chosen random coins $\mathrm{open}_{\hat{j}}^{(\hat{i})}$.

   Let
   $$\widetilde{x}^{(\hat{i})} := (\mathrm{crs}^{(\hat{i})}, \mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})})$$
   and
   $$\widetilde{\mathrm{sv}}^{(\hat{i})} := ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})})).$$
5. Then, for every $\hat{i} \in [N]$, compute the punctured key $K^{(\hat{i})}[2_G^{\widetilde{x}}]$ and $\mathsf{ioP}^{(\hat{i})}$:
   Puncturing a key and computing $\mathsf{ioP}^{(\hat{i})}$.
   - Recall that $\widetilde{x}^{(\hat{j})} \in \{0,1\}^n$. Using $\mathsf{OPuncture}$, consistent with the already generated patches $\{\mathsf{patch}_{\hat{j}}^{(\hat{i})}\}_{\hat{j}\in[N]}$ and with all $\widetilde{x}^{(\hat{j})}$, generate a key punctured at $2_G^{\widetilde{x}}$, where, $\widetilde{x} := (\widetilde{x}^{(1)}, \ldots, \widetilde{x}^{(N)})$:
   $$K^{(\hat{i})}[2_G^{\widetilde{x}}] \leftarrow \mathsf{OPuncture}(1^\lambda, \mathsf{o.state}^{(\hat{i})}, \widetilde{x})$$
   - For $K^{(\hat{i})}[2_G^{\widetilde{x}}]$, generate $\mathsf{KD\text{-}Prog}_6^{(\hat{i})}$ exactly the same way as $\mathsf{KD\text{-}Prog}_5^{(\hat{i})}$ and compute $\mathsf{ioP}^{(\hat{i})} \leftarrow i\mathcal{O}(\mathsf{KD\text{-}Prog}_6^{(\hat{i})})$.

Let $\mathrm{pv}^{(\hat{i})} = \widetilde{\mathrm{pv}}^{(\hat{i})} = (\widetilde{x}^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$.

6. Upon receiving $\mathsf{Reg}(\hat{i} \in [N])$, then set $(\mathrm{pv}^{(\hat{i})}, \mathrm{sv}^{(\hat{i})}) = (\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$. On the other hand, if the adversary himself registers a (corrupted) party $P^*_{\hat{i}}$ by presenting $\mathsf{RegCorr}(\hat{i} \in [N], \mathrm{pv}^{(\hat{i})})$, then the challenger simply ignores $(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
7. Upon receiving $\mathsf{Ext}(\hat{i})$ for a registered honest $P_{\hat{i}}$, respond via $\mathrm{sv}^{(\hat{i})}$.
8. Upon receiving $\mathsf{Rev}(\mathcal{S}, \hat{j})$, let $\hat{i^*}$ be the smallest element in $\mathcal{S}$. Run $\mathsf{ioP}^{(\hat{i^*})}$ on input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{j}, \mathrm{sv}^{(\hat{j})}_{\hat{i^*}})$ and respond via the output.
9. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, let $\hat{i^*}$ be the smallest element in ChQ. Choose $\hat{j} \leftarrow \mathsf{ChQ}$ and run $\mathsf{ioP}^{(\hat{i^*})}$ as $Y_{\mathrm{REAL}} := \mathsf{ioP}^{(\hat{i^*})}(\mathsf{ChQ}, (x^{(\hat{k})})_{\hat{k}\in\mathsf{ChQ}}, \hat{j}, \mathrm{sv}^{(\hat{j})}_{\hat{i^*}})$ and sample $Y_{\mathrm{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
10. Finally, output whatever $\mathcal{A}$ outputs.

Lemma 17. $\mathsf{Hyb}_5 \approx_c \mathsf{Hyb}_6$.

Proof. Observe that the only difference in Hyb_5 and Hyb_6 is that while in Hyb_5, for every $\hat{i} \in [N]$, the challenger samples $\mathrm{crs}^{(\hat{i})}$ using the statistically-binding parameter-generation algorithm HGen, in Hyb_6, the for every $\hat{i} \in [N]$, the challenger samples $\mathrm{crs}^{(\hat{i})}$ using the trapdoor parameter-generation algorithm HTGen. With this being the only modification as we switch between the hybrids, we remark in particular that the challenger in Hyb_6 does not use the auxiliary information output by HTGen while generating $\mathrm{crs}^{(\hat{i})}$. This allows us to apply the 'hybrid property' of the hybrid trapdoor commitment scheme HCOM (See Definition 4) to argue indistinguishability between the hybrids. Details follow.

Assume for contradiction that there exists an adversary $\mathcal{A}$ that distinguishes the two hybrids Hyb_5 and Hyb_6; that is, $|\mathsf{Adv}^{\mathsf{Hyb}_5}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathsf{Hyb}_6}_{\mathcal{A}}(\lambda)|$ is non-negligible in λ, where, $\mathsf{Adv}^{\mathsf{Hyb}_i}_{\mathcal{A}}(\lambda)$ is as defined in Equation (6). Recall that, for any i, $\mathsf{Adv}^{\mathsf{Hyb}_i}_{\mathcal{A}}(\lambda) := |\Pr[\mathrm{REAL}^{\mathsf{Hyb}_i}_{\mathcal{A}} \to 1] - \Pr[\mathrm{RAND}^{\mathsf{Hyb}_i}_{\mathcal{A}} \to 1]|$. This implies that there exists $\widetilde{\mathrm{EXPT}} \in \{\mathrm{REAL}, \mathrm{RAND}\}$ such that $|\Pr[\widetilde{\mathrm{EXPT}}^{\mathsf{Hyb}_5}_{\mathcal{A}} \to 1] - \Pr[\widetilde{\mathrm{EXPT}}^{\mathsf{Hyb}_6}_{\mathcal{A}} \to 1]| = \epsilon$, where, $\epsilon = \epsilon(\lambda)$ is non-negligible in λ. We shall build an adversary $\mathcal{B}$ that emulates either $\widetilde{\mathrm{EXPT}}^{\mathsf{Hyb}_5}_{\mathcal{A}}$ or $\widetilde{\mathrm{EXPT}}^{\mathsf{Hyb}_6}_{\mathcal{A}}$ and breaks the hybrid property of HCOM.

Corresponding respectively to the two distributions of crs, the adversary $\mathcal{B}$ emulates to $\mathcal{A}$ either $\widetilde{\mathrm{EXPT}}^{\mathsf{Hyb}_5}_{\mathcal{A}}$ or $\widetilde{\mathrm{EXPT}}^{\mathsf{Hyb}_6}_{\mathcal{A}}$, respectively, and exploits the success probability of $\mathcal{A}$ in its own hybrid security game. Let the experiment with $\mathcal{A}$ as emulated by $\mathcal{B}$ be denoted by $\widetilde{\mathrm{EXPT}}_{5-6}$.

From its challenger, $\mathcal{B}$ receives N-many CRS that are all either sampled either using the algorithm HGen or the algorithm HTGen. The objective of $\mathcal{B}$ is to tell the two cases apart. $\mathcal{B}$ proceeds as follows.

Adversary $\mathcal{B}$ runs $\mathcal{A}$ just like the challenger of Hyb_5 runs $\mathcal{A}$ except that, instead of sampling by itself $\mathrm{crs}^{(\hat{1})}, \ldots, \mathrm{crs}^{(\hat{N})}$, it uses the CRS values that it received from its challenger. Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Observe that at no point in the execution of the Hyb_5 experiment do we need the random coins used in generating the CRS values. Hence, the adversary $\mathcal{B}$ is well-defined.

Now observe that when the CRS values given by the challenger of $\mathcal{B}$ are all sampled using HGen, then the view of $\mathcal{A}$ generated by $\mathcal{B}$ is identical to that generated by Hyb_5. On the other hand, when the CRS values are all sampled using HTGen, then the view of $\mathcal{A}$ generated by $\mathcal{B}$ is identical to that generated by Hyb_6. Hence, with probability $|\Pr[\widetilde{\mathrm{EXPT}}^{\mathsf{Hyb}_5}_{\mathcal{A}} \to 1] - \Pr[\widetilde{\mathrm{EXPT}}^{\mathsf{Hyb}_6}_{\mathcal{A}} \to 1]| = \epsilon$, the adversary $\mathcal{B}$ distinguishes statistically-binding CRS vales from trapdoor CRS values, thus, leading

to a contradiction.

Hence, we have that $\mathsf{Hyb}_5 \approx_c \mathsf{Hyb}_6$.

■

Hyb_7. This experiment is the same as Hyb_6 except for the following modification in the way in which for every $\hat{i} \in [N]$ the challenger computes the commitments $\mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})}$ (which form a part of $\widetilde{x}^{(\hat{i})}$). Roughly speaking, instead of computing $\mathrm{com}_{\hat{j}}^{(\hat{i})}$ as a commitment to a specific value (namely, an encryption of $\mathsf{patch}_{\hat{i}}^{(\hat{j})}$) using $\mathsf{HCommit}$ algorithm, the modification is that the challenger, using $\mathsf{HTCommit}$, first computes $\mathrm{com}_{\hat{j}}^{(\hat{i})}$, along with a trapdoor. Since the challenger is able to open $\mathrm{com}_{\hat{j}}^{(\hat{i})}$ to any value, it opens it to an encryption of $\mathsf{patch}_{\hat{i}}^{(\hat{j})}$, later. Details follow.

1. Initiate by running the Setup algorithm. Namely, simply output (λ, N, G) as the output of the Setup algorithm. Next, respond to the adversary's queries as follows.
2. For every $\hat{i} \in [N]$, obliviously compute all patches as follows.

   Obliviously sampling patches.

   - Run $\mathsf{F.params}^{(\hat{i})} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$.
   - Obliviously (of $\widetilde{x}^{(\hat{j})}$) generate the patches: For every $\hat{j} \in [N]$, compute

   $$(\{\mathsf{patch}_{\hat{j}}^{(\hat{i})}\}_{\hat{j}\in[N]}, \mathsf{o.state}^{(\hat{i})}) \leftarrow \mathsf{OPatchGen}(1^\lambda, \mathsf{F.params}^{(\hat{i})}))$$

3. For every $\hat{i} \in [N]$, sample a public key using the key-generation algorithm of Σ: $(\mathrm{pk}^{(\hat{i})}, \mathrm{sk}^{(\hat{i})}) \leftarrow \mathsf{E.Gen}(1^\lambda)$.
4. Then, for every $\hat{i} \in [N]$, firstly compute $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$:

   Computing $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.

   - Run again the trapdoor parameter-generation algorithm of HCOM: $(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{crs}}^{(\hat{i})}) \leftarrow \mathsf{HTGen}(1^\lambda)$. ~~(Ignore $\mathrm{aux}_{\mathrm{crs}}^{(\hat{i})}$).~~ Compute $\mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})}$ as follows: For every $\hat{j} \in [N]$, $(\mathrm{com}_{\hat{j}}^{(\hat{i})}, \mathrm{aux}_{\mathrm{com}_{\hat{j}}}^{(\hat{i})}) \leftarrow \mathsf{HTCommit}(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{crs}}^{(\hat{i})})$.

   Let

   $$\widetilde{x}^{(\hat{i})} := (\mathrm{crs}^{(\hat{i})}, \mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})})$$

   and

   $$\widetilde{\mathrm{sv}}^{(\hat{i})} := ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})}))$$

5. Then, for every $\hat{i} \in [N]$, compute the punctured key $K^{(\hat{i})}[2_G^{\widetilde{x}}]$ and $\mathsf{ioP}^{(\hat{i})}$:

   Puncturing a key and computing $\mathsf{ioP}^{(\hat{i})}$.

   - Now continue to build the rest of the program $\mathsf{ioP}^{(\hat{i})}$ as follows.
     - Recall that $\widetilde{x}^{(\hat{j})} \in \{0,1\}^n$. Using $\mathsf{OPuncture}$, consistent with the already generated patches $\{\mathsf{patch}_{\hat{j}}^{(\hat{i})}\}_{\hat{j}\in[N]}$ and with all $\widetilde{x}^{(\hat{j})}$, generate a key punctured at $2_G^{\widetilde{x}}$, where, $\widetilde{x} := (\widetilde{x}^{(1)}, \ldots, \widetilde{x}^{(N)})$:

     $$K^{(\hat{i})}[2_G^{\widetilde{x}}] \leftarrow \mathsf{OPuncture}(1^\lambda, \mathsf{o.state}^{(\hat{i})}, \widetilde{x})$$

     - For $K^{(\hat{i})}[2_G^{\widetilde{x}}]$, generate $\mathsf{KD\text{-}Prog}_7^{(\hat{i})}$ exactly the same way as $\mathsf{KD\text{-}Prog}_6^{(\hat{i})}$ and compute $\mathsf{ioP}^{(\hat{i})} \leftarrow i\mathcal{O}(\mathsf{KD\text{-}Prog}_7^{(\hat{i})})$.

   Let $\mathrm{pv}^{(\hat{i})} = \widetilde{\mathrm{pv}}^{(\hat{i})} = (\widetilde{x}^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$.

6. Upon receiving $\mathsf{Reg}(\hat{i} \in [N])$, compute $\widetilde{\mathrm{sv}}^{(\hat{i})}$: compute an encryption of $\mathsf{patch}_{\hat{i}}^{(\hat{j})}$ as $\mathrm{ct}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{Enc}(\mathrm{pk}^{(\hat{j})}, \mathsf{patch}_{\hat{i}}^{(\hat{j})})$. Finally, compute the opening for $\mathrm{com}_{\hat{j}}^{(\hat{i})}$ to $\mathrm{ct}_{\hat{j}}^{(\hat{i})}$ as $\mathrm{open}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{HTDec}(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{com}_{\hat{j}}}^{(\hat{i})}, \mathrm{ct}_{\hat{j}}^{(\hat{i})})$. With this, set
$$\widetilde{\mathrm{sv}}^{(\hat{i})} = ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})}))$$
and respond to the adversary's query via $\widetilde{\mathrm{pv}}^{(\hat{i})}$. On the other hand, if the adversary himself registers a (corrupted) party $P_{\hat{i}}^*$ by presenting $\mathsf{RegCorr}(\hat{i} \in [N], \mathrm{pv}^{(\hat{i})})$, then the challenger simply ignores $(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
7. Upon receiving $\mathsf{Ext}(\hat{i})$ for a registered honest $P_{\hat{i}}$, respond via $\mathrm{sv}^{(\hat{i})}$.
8. Upon receiving $\mathsf{Rev}(\mathcal{S}, \hat{j})$, let $\hat{i}^*$ be the smallest element in $\mathcal{S}$. Run $\mathsf{ioP}^{(\hat{i}^*)}$ on input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k} \in \mathcal{S}}, \hat{j}, \mathrm{sv}_{\hat{i}^*}^{(\hat{j})})$ and respond via the output.
9. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, let $\hat{i}^*$ be the smallest element in ChQ. Choose $\hat{j} \leftarrow \mathsf{ChQ}$ and run $\mathsf{ioP}^{(\hat{i}^*)}$ as $Y_{\mathrm{REAL}} := \mathsf{ioP}^{(\hat{i}^*)}(\mathsf{ChQ}, (x^{(\hat{k})})_{\hat{k} \in \mathsf{ChQ}}, \hat{j}, \mathrm{sv}_{\hat{i}^*}^{(\hat{j})})$ and sample $Y_{\mathrm{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
10. Finally, output whatever $\mathcal{A}$ outputs.

Lemma 18. $\mathsf{Hyb}_6 \approx_c \mathsf{Hyb}_7$.

Proof. Observe that the only differences between Hyb_6 and Hyb_7 is in the way the challenger computes $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$. Also, recall that in both the hybrids, for every $\hat{i} \in [N]$, $\mathrm{crs}^{(\hat{i})}$ is sampled using HTGen algorithm. However, the difference in the hybrid lies in the way this $\mathrm{crs}^{(\hat{i})}$ is used in computing the commitments. In detail, for every $\hat{i} \in [N]$:

- In Hyb_6, the challenger first computes an encryption of $\mathsf{patch}_{\hat{i}}^{(\hat{j})}$ to get $\mathrm{ct}_{\hat{j}}^{(\hat{i})}$, and then commits to $\mathrm{ct}_{\hat{j}}^{(\hat{i})}$ as $\mathrm{com}_{\hat{j}}^{(\hat{i})} = \mathsf{HCommit}(\mathrm{crs}^{(\hat{i})}, \mathrm{ct}_{\hat{j}}^{(\hat{i})}; \mathrm{open}_{\hat{j}}^{(\hat{i})})$, finally setting $\widetilde{\mathrm{sv}}^{(\hat{i})} = (*, ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})}))$ and $\widetilde{x}^{(\hat{i})} = ((*,*), (\mathrm{crs}^{(\hat{i})}, \mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})}))$.
- On the other hand, in Hyb_7, the challenger computes $\mathrm{com}_{\hat{j}}^{(\hat{i})}$ as an equivocable commitment using $\mathsf{HTCommit}$ as $(\mathrm{com}_{\hat{j}}^{(\hat{i})}, \mathrm{aux}_{\mathrm{com}_{\hat{j}}}^{(\hat{i})}) \leftarrow \mathsf{HTCommit}(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{crs}}^{(\hat{i})})$ and then computes an opening of this commitment $\mathrm{com}_{\hat{j}}^{(\hat{i})}$ to $\mathrm{ct}_{\hat{j}}^{(\hat{i})}$ as $\mathrm{open}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{HTDec}(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{com}_{\hat{j}}}^{(\hat{i})}, \mathrm{ct}_{\hat{j}}^{(\hat{i})})$, finally setting $\widetilde{\mathrm{sv}}^{(\hat{i})} = (*, ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})})))$ and $\widetilde{x}^{(\hat{i})} = ((*,*), (\mathrm{crs}^{(\hat{i})}, \mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})}))$.

Observe that the above difference in the two hybrids Hyb_6 and Hyb_7 corresponds to the two cases in the hybrid game of the hybrid trapdoor commitment scheme HCOM. Thus, any adversary who distinguishes the two hybrids can be reduced to one that breaks the trapdoor property of HCOM, as we shall show below.

Let $\mathcal{A}$ be an adversary that distinguishes Hyb_6 and Hyb_7; that is, $|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_6}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_7}(\lambda)|$ is non-negligible in λ, where, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_i}(\lambda)$ is as defined in Equation (6). Recall that, for any i, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_i}(\lambda) := |\Pr[\mathrm{REAL}_{\mathcal{A}}^{\mathsf{Hyb}_i} \to 1] - \Pr[\mathrm{RAND}_{\mathcal{A}}^{\mathsf{Hyb}_i} \to 1]|$. This implies that there exists $\widetilde{\mathrm{EXPT}} \in \{\mathrm{REAL}, \mathrm{RAND}\}$ such that $|\Pr[\widetilde{\mathrm{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_6} \to 1] - \Pr[\widetilde{\mathrm{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_7} \to 1]| = \epsilon$, where, $\epsilon = \epsilon(\lambda)$ is non-negligible in λ. We shall build an adversary $\mathcal{A}_{\mathsf{trap}}$ that emulates either $\widetilde{\mathrm{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_6}$ or $\widetilde{\mathrm{EXPT}}_{\mathcal{A}}^{\mathsf{Hyb}_7}$ and breaks the trapdoor property of HCOM. Details follow.

We shall consider a slight variant of the trapdoor game in Definition 4 that is equivalent upto a polynomial factor by a standard hybrid argument. The adversary $\mathcal{A}_{\mathsf{trap}}$ first receives N-many CRS values $\mathrm{crs}^{(\hat{1})}, \ldots, \mathrm{crs}^{(\hat{N})}$ each sampled using HTGen. Then $\mathcal{A}_{\mathsf{trap}}$ gives to its challenger N-many N-vector of messages, upon which the challenger, either using $\mathsf{HCommit}$ or $\mathsf{HTCommit}$, computes commitments of the $\hat{i}$th N-vector of messages for $\mathrm{crs}^{(\hat{i})}$ and gives $\mathcal{A}_{\mathsf{trap}}$ the resulting commitments and their openings. The objective of the adversary is to tell apart the two cases. $\mathcal{A}_{\mathsf{trap}}$ is described as below.

$\mathcal{A}_{\mathsf{trap}}$ behaves the same way as the challenger in Hyb_7 except for the way it computes $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$ for all $\hat{i}$. It proceeds as follows:

1. Initiate by running the Setup algorithm. Namely, simply output (λ, N, G) as the output of the Setup algorithm. Next, respond to the adversary's queries as follows.
2. For every $\hat{i} \in [N]$, obliviously compute all patches as follows.
   Obliviously sampling patches.
   - Run $\mathsf{F.params}^{(\hat{i})} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$.
   - Obliviously (of $\widetilde{x}^{(\hat{j})}$) generate the patches: For every $\hat{j} \in [N]$, compute
   $$(\{\mathsf{patch}_{\hat{j}}^{(\hat{i})}\}_{\hat{j}\in[N]}, \mathsf{o.state}^{(\hat{i})}) \leftarrow \mathsf{OPatchGen}(1^\lambda, \mathsf{F.params}^{(\hat{i})})$$
3. For every $\hat{i} \in [N]$, sample a public key using the key-generation algorithm of Σ: $(\mathrm{pk}^{(\hat{i})}, \mathrm{sk}^{(\hat{i})}) \leftarrow \mathsf{E.Gen}(1^\lambda)$.
4. Then, for every $\hat{i} \in [N]$, firstly compute $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$:
   Computing $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
   - For every $\hat{j} \in [N]$, $\mathrm{ct}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{Enc}(\mathrm{pk}^{(\hat{j})}, \mathsf{patch}_{\hat{i}}^{(\hat{j})})$. Upon receiving $\mathrm{crs}^{(\hat{i})}$ from the challenger, give $(\mathrm{ct}_1^{(\hat{i})}, \ldots, \mathrm{ct}_N^{(\hat{i})})$ to the challenger. Let $((\mathrm{com}_1^{(\hat{i})}, \mathrm{aux}_{\mathrm{com}_1}^{(\hat{i})}), \ldots, (\mathrm{com}_N^{(\hat{i})}, \mathrm{aux}_{\mathrm{com}_N}^{(\hat{i})}))$ be the commitment-opening pairs received.

   Let
   $$\widetilde{x}^{(\hat{i})} := (\mathrm{crs}^{(\hat{i})}, \mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})})$$
   and
   $$\mathrm{sv}^{(\hat{i})} := ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})}))$$
5. Then, for every $\hat{i} \in [N]$, compute the punctured key $K^{(\hat{i})}[2_G^{\widetilde{x}}]$ and $\mathsf{ioP}^{(\hat{i})}$:
   Puncturing a key and computing $\mathsf{ioP}^{(\hat{i})}$.
   - Now continue to build the rest of the program $\mathsf{ioP}^{(\hat{i})}$ as follows.
     - Recall that $\widetilde{x}^{(\hat{j})} \in \{0,1\}^n$. Using $\mathsf{OPuncture}$, consistent with the already generated patches $\{\mathsf{patch}_{\hat{j}}^{(\hat{i})}\}_{\hat{j}\in[N]}$ and with all $\widetilde{x}^{(\hat{j})}$, generate a key punctured at $2_G^{\widetilde{x}}$, where, $\widetilde{x} := (\widetilde{x}^{(1)}, \ldots, \widetilde{x}^{(N)})$:
     $$K^{(\hat{i})}[2_G^{\widetilde{x}}] \leftarrow \mathsf{OPuncture}(1^\lambda, \mathsf{o.state}^{(\hat{i})}, \widetilde{x})$$
     - For $K^{(\hat{i})}[2_G^{\widetilde{x}}]$, generate $\mathsf{KD\text{-}Prog}_7^{(\hat{i})}$ exactly the same way as $\mathsf{KD\text{-}Prog}_6^{(\hat{i})}$ and compute $\mathsf{ioP}^{(\hat{i})} \leftarrow i\mathcal{O}(\mathsf{KD\text{-}Prog}_7^{(\hat{i})})$.

Let $\mathrm{pv}^{(\hat{i})} = \widetilde{\mathrm{pv}}^{(\hat{i})} = (\widetilde{x}^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$.

6. Upon receiving $\mathsf{Reg}(\hat{i} \in [N])$, compute $\widetilde{\mathrm{sv}}^{(\hat{i})}$: compute an encryption of $\mathsf{patch}_{\hat{i}}^{(\hat{j})}$ as $\mathrm{ct}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{Enc}(\mathrm{pk}^{(\hat{j})}, \mathsf{patch}_{\hat{i}}^{(\hat{j})})$. Finally, compute the opening for $\mathrm{com}_{\hat{j}}^{(\hat{i})}$ to $\mathrm{ct}_{\hat{j}}^{(\hat{i})}$ as $\mathrm{open}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{HTDec}(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{com}_{\hat{j}}}^{(\hat{i})}, \mathrm{ct}_{\hat{j}}^{(\hat{i})})$. With this, set

$$\widetilde{\mathrm{sv}}^{(\hat{i})} = ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})}))$$

and respond to the adversary's query via $\widetilde{\mathrm{pv}}^{(\hat{i})}$. On the other hand, if the adversary himself registers a (corrupted) party $P_{\hat{i}}^*$ by presenting $\mathsf{RegCorr}(\hat{i} \in [N], \mathrm{pv}^{(\hat{i})})$, then the challenger simply ignores $(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.

7. Upon receiving $\mathsf{Ext}(\hat{i})$ for a registered honest $P_{\hat{i}}$, respond via $\mathrm{sv}^{(\hat{i})}$.
8. Upon receiving $\mathsf{Rev}(\mathcal{S}, \hat{j})$, let $\hat{i^*}$ be the smallest element in $\mathcal{S}$. Run $\mathsf{ioP}^{(\hat{i^*})}$ on input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k} \in \mathcal{S}}, \hat{j}, \mathrm{sv}_{\hat{i^*}}^{(\hat{j})})$ and respond via the output.
9. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, let $\hat{i^*}$ be the smallest element in ChQ. Choose $\hat{j} \leftarrow \mathsf{ChQ}$ and run $\mathsf{ioP}^{(\hat{i^*})}$ as $Y_{\mathrm{REAL}} := \mathsf{ioP}^{(\hat{i^*})}(\mathsf{ChQ}, (x^{(\hat{k})})_{\hat{k} \in \mathsf{ChQ}}, \hat{j}, \mathrm{sv}_{\hat{i^*}}^{(\hat{j})})$ and sample $Y_{\mathrm{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
10. Finally, output whatever $\mathcal{A}$ outputs.

This completes the description of $\mathcal{A}_{\mathsf{trap}}$. Observe that if the challenger computes the commitments using $\mathsf{HCommit}$, then the view of $\mathcal{A}$ as generated by $\mathcal{A}_{\mathsf{trap}}$ is the same as the view of $\mathcal{A}$ in Hyb_6. On the other hand, if the challenger computes the commitments using $\mathsf{HTCommit}$ and HTDec, then the view of $\mathcal{A}$ as generated by $\mathcal{A}_{\mathsf{trap}}$ is the same as the view of $\mathcal{A}$ in Hyb_7.

$$\begin{aligned}\mathsf{Adv}^{\mathsf{trap}}_{\mathcal{A}_{\mathsf{trap}}}(\lambda) =& |\Pr[\widetilde{\mathrm{EXPT}}^{\mathsf{Hyb}_6}_{\mathcal{A}} \to 1] - \Pr[\widetilde{\mathrm{EXPT}}^{\mathsf{Hyb}_7}_{\mathcal{A}} \to 1]| \\ =& \epsilon\end{aligned}$$

thus, arriving at a contradiction.

Hence, $\mathsf{Hyb}_6 \approx_c \mathsf{Hyb}_7$.

■

Hyb_8. This experiment is the same as Hyb_7 except for the following modification. Recall that earlier, we sampled the patches *obliviously* of every $\widetilde{x}^{(\hat{i})}$, since, it was crucial that the patches were sampled before we computed $\widetilde{x}^{(\hat{i})}$. This, to recall again, is because $\widetilde{x}^{(\hat{i})}$ was computed as a function of the patches $\mathsf{patch}_{\hat{i}}^{(\hat{j})}$ (namely, commitments to encryptions of these patches formed a part of $\widetilde{x}^{(\hat{i})}$). However, by Hyb_7, we had modified the computation of $\widetilde{x}^{(\hat{i})}$ as follows: in place of the aforementioned commitments, we generate equivocal commitments, that could later be opened to any values, and in particular to encryptions of the corresponding patches. Thus, we do not need to have computed the patches ahead of time. Thus, in the current hybrid, we switch back to computing the patches using the algorithm $\mathsf{PatchGen}$, consistent with the punctured keys. Details follow.

1. Initiate by running the Setup algorithm. Namely, simply output (λ, N, G) as the output of the Setup algorithm. Next, respond to the adversary's queries as follows.
2. For every $\hat{i} \in [N]$, generate parameters for F as follows.
   - Run $\mathsf{F.params}^{(\hat{i})} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$.
   - Obliviously (of $\widetilde{x}^{(\hat{j})}$) generate the patches: For every $\hat{j} \in [N]$, compute

     ~~$(\{\mathsf{patch}_{\hat{j}}^{(\hat{i})}\}_{\hat{j}\in[N]}, \mathsf{o.state}^{(\hat{i})}) \leftarrow \mathsf{OPatchGen}(1^\lambda, \mathsf{F.params}^{(\hat{i})})$~~
3. For every $\hat{i} \in [N]$, sample a public key using the key-generation algorithm of Σ: $(\mathrm{pk}^{(\hat{i})}, \mathrm{sk}^{(\hat{i})}) \leftarrow \mathsf{E.Gen}(1^\lambda)$.
4. Then, for every $\hat{i} \in [N]$, firstly compute $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$:

   Computing $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.

   - Run again the trapdoor parameter-generation algorithm of HCOM: $(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{crs}}^{(\hat{i})}) \leftarrow \mathsf{HTGen}(1^\lambda)$. Compute $\mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})}$ as follows: For every $\hat{j} \in [N]$, $(\mathrm{com}_{\hat{j}}^{(\hat{i})}, \mathrm{aux}_{\mathrm{com}_{\hat{j}}}^{(\hat{i})}) \leftarrow \mathsf{HTCommit}(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{crs}}^{(\hat{i})})$.

   Let
   $$\widetilde{x}^{(\hat{i})} := (\mathrm{crs}^{(\hat{i})}, \mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})})$$
   and
   $$\mathrm{sv}^{(\hat{i})} := ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})}))$$
5. Then, for every $\hat{i} \in [N]$, compute the punctured key $K^{(\hat{i})}[2_G^{\widetilde{x}}]$, its patches $\{\mathsf{patch}(K^{(\hat{i})}[2_G^{\widetilde{x}}], (\hat{j}, \widetilde{x}^{(\hat{j})}))\}_{\hat{j}\in[N]}$, and $\mathsf{ioP}^{(\hat{i})}$:

   Puncturing a key, sampling patches, and computing $\mathsf{ioP}^{(\hat{i})}$.

   - Now continue to build the rest of the program $\mathsf{ioP}^{(\hat{i})}$ as follows.
   - Run $\mathsf{F.params}^{(\hat{i})} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$.
   - Sample a key, $K^{(\hat{i})} \leftarrow \mathsf{F.KeyGen}(1^\lambda, \mathsf{F.params})$.
   - Puncture K at $2_G^{\widetilde{x}}$: $K^{(\hat{i})}[2_G^{\widetilde{x}}] \leftarrow \mathsf{Puncture}(K^{(\hat{i})}, 2_G^{\widetilde{x}})$.
   - Generate patches for $K^{(\hat{i})}[2_G^{\widetilde{x}}]$ at every $\hat{j}$th block: $\mathsf{patch}(K^{(\hat{i})}[2_G^{\widetilde{x}}], (\hat{j}, \widetilde{x}^{(\hat{j})})) \leftarrow \mathsf{PatchGen}(K^{(\hat{i})}, \hat{j}, \widetilde{x}^{(\hat{j})})$.
   - For $K^{(\hat{i})}[2_G^{\widetilde{x}}]$, generate $\mathsf{KD\text{-}Prog}_8^{(\hat{i})}$ the same way as $\mathsf{KD\text{-}Prog}_7^{(\hat{i})}$.

   Let $\mathrm{pv}^{(\hat{i})} = \widetilde{\mathrm{pv}}^{(\hat{i})} = (\widetilde{x}^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$.

6. Upon receiving $\mathsf{Reg}(\hat{i} \in [N])$, compute $\widetilde{\mathrm{sv}}^{(\hat{i})}$: compute an encryption of $\mathsf{patch}_{\hat{i}}^{(\hat{j})}$ as $\mathrm{ct}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{Enc}(\mathrm{pk}^{(\hat{j})}, \mathsf{patch}(K^{(\hat{j})}[2^{\widetilde{x}}_G], (\hat{i}, \widetilde{x}^{(\hat{i})})))$. Finally, compute the opening for $\mathrm{com}_{\hat{j}}^{(\hat{i})}$ to $\mathrm{ct}_{\hat{j}}^{(\hat{i})}$ as $\mathrm{open}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{HTDec}(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{com}_{\hat{j}}}^{(\hat{i})}, \mathrm{ct}_{\hat{j}}^{(\hat{i})})$. With this, set
$$\widetilde{\mathrm{sv}}^{(\hat{i})} = ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})}))$$
and respond to the adversary's query via $\widetilde{\mathrm{pv}}^{(\hat{i})}$. On the other hand, if the adversary himself registers a (corrupted) party $P_{\hat{i}}^*$ by presenting $\mathsf{RegCorr}(\hat{i} \in [N], \mathrm{pv}^{(\hat{i})})$, then the challenger simply ignores $(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
7. Upon receiving $\mathsf{Ext}(\hat{i})$ for a registered honest $P_{\hat{i}}$, respond via $\mathrm{sv}^{(\hat{i})}$.
8. Upon receiving $\mathsf{Rev}(\mathcal{S}, \hat{j})$, let $\hat{i^*}$ be the smallest element in $\mathcal{S}$. Run $\mathsf{ioP}^{(\hat{i^*})}$ on input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k} \in \mathcal{S}}, \hat{j}, \mathrm{sv}_{\hat{i^*}}^{(\hat{j})})$ and respond via the output.
9. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, let $\hat{i^*}$ be the smallest element in ChQ. Choose $\hat{j} \leftarrow \mathsf{ChQ}$ and run $\mathsf{ioP}^{(\hat{i^*})}$ as $Y_{\mathrm{REAL}} := \mathsf{ioP}^{(\hat{i^*})}(\mathsf{ChQ}, (x^{(\hat{k})})_{\hat{k} \in \mathsf{ChQ}}, \hat{j}, \mathrm{sv}_{\hat{i^*}}^{(\hat{j})})$ and sample $Y_{\mathrm{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
10. Finally, output whatever $\mathcal{A}$ outputs.

Lemma 19. $\mathsf{Hyb}_7 \equiv \mathsf{Hyb}_8$.

Proof. Observe that the only difference between Hyb_7 and Hyb_8 is in the way the punctured key and its patches are computed. While in Hyb_7, they are computed using algorithms $\mathsf{OPatchGen}, \mathsf{OPuncture}$, in Hyb_8, they are computed using $\mathsf{Puncture}, \mathsf{PatchGen}$. Recall that this exactly corresponds to switching back the difference introduced in moving from Hyb_2 to Hyb_3. As noted in Lemma 3, these two different processes of generating the key and its patches result in identical distributions. Hence, the Lemma.

■

Hyb_9. This experiment is the same as Hyb_8 except for the following modification. Note that in Hyb_8, we computed the patches soon after we computed the punctured key. The modification we introduce in this hybrid is that we would compute patches (and their encryptions) only if necessary. Details follow.

1. Initiate by running the Setup algorithm. Namely, simply output (λ, N, G) as the output of the Setup algorithm. Next, respond to the adversary's queries as follows.
2. For every $\hat{i} \in [N]$, generate parameters for F as follows:
   - Run $\mathsf{F.params}^{(\hat{i})} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$.
3. For every $\hat{i} \in [N]$, sample a public key using the key-generation algorithm of Σ: $(\mathrm{pk}^{(\hat{i})}, \mathrm{sk}^{(\hat{i})}) \leftarrow \mathsf{E.Gen}(1^\lambda)$.
4. Then, for every $\hat{i} \in [N]$, firstly compute $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$:
   Computing $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
   - Run again the trapdoor parameter-generation algorithm of HCOM: $(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{crs}}^{(\hat{i})}) \leftarrow \mathsf{HTGen}(1^\lambda)$. Compute $\mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})}$ as follows: For every $\hat{j} \in [N]$, $(\mathrm{com}_{\hat{j}}^{(\hat{i})}, \mathrm{aux}_{\mathrm{com}_{\hat{j}}}^{(\hat{i})}) \leftarrow \mathsf{HTCommit}(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{crs}}^{(\hat{i})})$.

   Let
   $$\widetilde{x}^{(\hat{i})} := (\mathrm{crs}^{(\hat{i})}, \mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})})$$
   and
   $$\mathrm{sv}^{(\hat{i})} := ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})}))$$
5. Then, for every $\hat{i} \in [N]$, compute the punctured key $K^{(\hat{i})}[2_G^{\widetilde{x}}]$ and $\mathsf{ioP}^{(\hat{i})}$:
   Puncturing a key and computing $\mathsf{ioP}^{(\hat{i})}$.
   - Now continue to build the rest of the program $\mathsf{ioP}^{(\hat{i})}$ as follows.
   - Run $\mathsf{F.params}^{(\hat{i})} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$.
   - Sample a key, $K^{(\hat{i})} \leftarrow \mathsf{F.KeyGen}(1^\lambda, \mathsf{F.params})$.
   - Puncture K at $2_G^{\widetilde{x}}$: $K^{(\hat{i})}[2_G^{\widetilde{x}}] \leftarrow \mathsf{Puncture}(K^{(\hat{i})}, 2_G^{\widetilde{x}})$.
   - Generate patches for $K^{(\hat{i})}[2_G^{\widetilde{x}}]$ at every $\hat{j}$th block: ~~$\mathsf{patch}(K^{(\hat{i})}[2_G^{\widetilde{x}}], (\hat{j}, \widetilde{x}^{(\hat{j})})) \leftarrow \mathsf{PatchGen}(K^{(\hat{i})}, \hat{j}, \widetilde{x}^{(\hat{j})})$~~.
   - For $K^{(\hat{i})}[2_G^{\widetilde{x}}]$, generate $\mathsf{KD\text{-}Prog}_9^{(\hat{i})}$ the same way as $\mathsf{KD\text{-}Prog}_8^{(\hat{i})}$.

   Let $\mathrm{pv}^{(\hat{i})} = \widetilde{\mathrm{pv}}^{(\hat{i})} = (\widetilde{x}^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$.
6. Upon receiving $\mathsf{Reg}(\hat{i} \in [N])$, ~~compute $\widetilde{\mathrm{sv}}^{(\hat{i})}$~~, respond via $\widetilde{\mathrm{pv}}^{(\hat{i})}$. On the other hand, if the adversary himself registers a (corrupted) party $P_{\hat{i}}^*$ by presenting $\mathsf{RegCorr}(\hat{i} \in [N], \mathrm{pv}^{(\hat{i})})$, then the challenger simply ignores $(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
7. Upon receiving $\mathsf{Ext}(\hat{i})$ for a registered honest $P_{\hat{i}}$, firstly compute the following patches that will need to be used in computing the secret value of $P_{\hat{i}}$: $\{\mathsf{patch}(K^{(\hat{j})}[2_G^{\widetilde{x}}], (\hat{i}, \widetilde{x}^{(\hat{i})}))\}_{\hat{j} \in [N]}$ as $\mathsf{patch}(K^{(\hat{j})}[2_G^{\widetilde{x}}], (\hat{i}, \widetilde{x}^{(\hat{i})})) \leftarrow \mathsf{PatchGen}(K^{(\hat{j})}, \hat{i}, \widetilde{x}^{(\hat{i})})$. Then, compute an encryption of $\mathsf{patch}(K^{(\hat{j})}[2_G^{\widetilde{x}}], (\hat{i}, \widetilde{x}^{(\hat{i})}))$ as $\mathrm{ct}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{Enc}(\mathrm{pk}^{(\hat{j})}, \mathsf{patch}(K^{(\hat{j})}[2_G^{\widetilde{x}}], (\hat{i}, \widetilde{x}^{(\hat{i})})))$. Finally, compute an opening for $\mathrm{com}_{\hat{j}}^{(\hat{i})}$ to $\mathrm{ct}_{\hat{j}}^{(\hat{i})}$ as $\mathrm{open}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{HTDec}(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{com}_{\hat{j}}}^{(\hat{i})}, \mathrm{ct}_{\hat{j}}^{(\hat{i})})$. With this, set

$$\widetilde{\mathrm{sv}}^{(\hat{i})} = ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})}))$$

and respond to the adversary's query via $\widetilde{\mathrm{sv}}^{(\hat{i})}$.

8. Upon receiving $\mathsf{Rev}(\mathcal{S}, \hat{j})$, let $\hat{i^*}$ be the smallest element in $\mathcal{S}$. Then, consider the following three cases:
   1. Consider the case when $P_{\hat{i^*}}$ was registered as an honest party via $\mathsf{Reg}(\hat{i^*} \in [N])$ and there exists $\hat{i} \in \mathcal{S}$ such that $P_{\hat{i}}$ was registered as an honest party and was corrupted by the adversary via $\mathsf{Ext}(\hat{i})$. Under this case, run $\mathsf{ioP}^{(\hat{i^*})}$ on input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k} \in \mathcal{S}}, \hat{i}, \mathrm{sv}_{\hat{i^*}}^{(\hat{i})})$ and respond via the output. (Note here that $\mathrm{sv}_{\hat{i^*}}^{(\hat{i})}$ was already computed while responding to $\mathsf{Ext}(\hat{i})$).
   2. Consider the case when $P_{\hat{i^*}}$ was registered as an honest party via $\mathsf{Reg}(\hat{i^*} \in [N])$ and for every $\hat{i} \in \mathcal{S}$, $P_{\hat{i}}$ was registered as an honest party and was never corrupted. Under this case, choose $\hat{i} \leftarrow \mathcal{S}$, compute $\mathrm{sv}_{\hat{i^*}}^{(\hat{i})}$ (by computing $\mathsf{patch}(K^{(\hat{i^*})}[2\widetilde{x}_G], (\hat{i}, \widetilde{x}^{(\hat{i})}))$, encrypting $\mathrm{ct}_{\hat{i^*}}^{(\hat{i})} \leftarrow \mathsf{Enc}(\mathrm{pk}^{(\hat{i^*})}, \mathsf{patch}(K^{(\hat{i^*})}[2\widetilde{x}_G], (\hat{i}, \widetilde{x}^{(\hat{i})})))$, and finally computing an opening for $\mathrm{com}_{\hat{i^*}}^{(\hat{i})}$ to $\mathrm{ct}_{\hat{i^*}}^{(\hat{i})}$ as $\mathrm{open}_{\hat{i^*}}^{(\hat{i})} \leftarrow \mathsf{HTDec}(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{com}_{\hat{i^*}}}^{(\hat{i})}, \mathrm{ct}_{\hat{i^*}}^{(\hat{i})}))$. Then, run $\mathsf{ioP}^{(\hat{i^*})}$ on input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k} \in \mathcal{S}}, \hat{i}, \mathrm{sv}_{\hat{i^*}}^{(\hat{i})})$ and respond via the output.
   3. Finally, consider the case when $P_{\hat{i^*}}$ was registered as a corrupt party via $\mathsf{RegCorr}(\hat{i} \in [N], \mathrm{pv}^{(\hat{i})})$. Then, compute $\mathrm{sv}_{\hat{i^*}}^{(\hat{i})}$ (by computing $\mathsf{patch}(K^{(\hat{i^*})}[2\widetilde{x}_G], (\hat{i}, \widetilde{x}^{(\hat{i})}))$, encrypting $\mathrm{ct}_{\hat{i^*}}^{(\hat{i})} \leftarrow \mathsf{Enc}(\mathrm{pk}^{(\hat{i^*})}, \mathsf{patch}(K^{(\hat{i^*})}[2\widetilde{x}_G], (\hat{i}, \widetilde{x}^{(\hat{i})})))$, and finally computing an opening for $\mathrm{com}_{\hat{i^*}}^{(\hat{i})}$ to $\mathrm{ct}_{\hat{i^*}}^{(\hat{i})}$ as $\mathrm{open}_{\hat{i^*}}^{(\hat{i})} \leftarrow \mathsf{HTDec}(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{com}_{\hat{i^*}}}^{(\hat{i})}, \mathrm{ct}_{\hat{i^*}}^{(\hat{i})}))$. Then, run $\mathsf{ioP}^{(\hat{i^*})}$ on input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k} \in \mathcal{S}}, \hat{i}, \mathrm{sv}_{\hat{i^*}}^{(\hat{i})})$ and respond via the output. We remark here that the patch $\mathsf{patch}(K^{(\hat{i^*})}[2\widetilde{x}_G], (\hat{i}, \widetilde{x}^{(\hat{i})}))$ was computed for a key $K^{(\hat{i^*})}[2\widetilde{x}_G]$ that is not a part of any obfuscated program – this is so, since, $\mathsf{ioP}^{(\hat{i^*})}$ was registered as a corrupted party. Just to see what to look ahead for, this will be crucial in later hybrids when we move to setting $\mathrm{ct}_{\hat{i^*}}^{(\hat{i})}$ from being a ciphertext to just a random element. The reason we will do so will be clear as we reach those hybrids where we introduce this modification.
4. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, let $\hat{i^*}$ be the smallest element in ChQ. Comparing to the cases we explored while responding to query $\mathsf{Rev}(\mathcal{S}, \hat{j})$, ChQ would always correspond to Case 2. This is because, for for every $\hat{i} \in \mathcal{S}$, $P_{\hat{i}}$ was registered as an honest party and was never corrupted. Y_{REAL} is computed the same way as the response to $\mathsf{Rev}(\mathcal{S}, \hat{j})$ under Case 2. Also, sample $Y_{\mathrm{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
5. Finally, output whatever $\mathcal{A}$ outputs.

Lemma 20. $\mathsf{Hyb}_8 \equiv \mathsf{Hyb}_9$.

Proof. Observe that the only difference between Hyb_8 and Hyb_9 is the following. At a high level, in Hyb_8, we computed all the patches ahead of time and used them if and when necessary. On the other hand, in Hyb_9, we generate the patches only if and when they are required. This clearly does not introduce any deviation in the view of the adversary. Furthermore, as a consequence of computing the patches only when necessary, we computed the responses to queries $\mathsf{Rev}(\mathcal{S}, \hat{j})$ and $\mathsf{Test}(\mathsf{ChQ})$ depending on what patches have already been computed. However, this again does not introduce an deviation in the view of the adversary. Hence, the Lemma. Hence, the Lemma.

■

Hyb_{10}. This experiment is the same as Hyb_9 except for the following modification. Note that responses to certain valid queries can be computed by the adversary himself; for instance, a query $\mathsf{Rev}(\mathcal{S}, \hat{j})$ such that the following holds: for $\hat{i^*}$ that is the smallest element in $\mathcal{S}$, $P_{\hat{i^*}}$ was registered as an honest party via $\mathsf{Reg}(\hat{i^*} \in [N])$ and there exists $\hat{i} \in \mathcal{S}$ such that $P_{\hat{i}}$ was registered as an honest party and was corrupted by the adversary via $\mathsf{Ext}(\hat{i})$. Under such a case, the adversary as the secret value of $P_{\hat{i}}$ using which it can run the program $\mathsf{ioP}^{(\hat{i^*})}$ to generate the response. This argument holds water since $\mathsf{ioP}^{(\hat{i^*})}$ was in fact generated by the challenger himself and the common key derived by any party $P_{\hat{j}}$ belonging to the set $\mathcal{S}$ is the same as the common key derived by any other party (in particular, $P_{\hat{i}}$) in $\mathcal{S}$. Now, having recalled this aspect, the modification we introduce in this hybrid is the following. For the queries the responses to which the adversary can himself compute, the challenger behaves the same way as it did in the previous hybrid – namely, it simply runs the corresponding programs. On the other hand, for the rest of the queries, the challenger computes the response by evaluating the PRF F directly. On the same lines, response to the challenge query is also computed by evaluating the PRF F directly. Details follow.

1. Initiate by running the Setup algorithm. Namely, simply output (λ, N, G) as the output of the Setup algorithm. Next, respond to the adversary's queries as follows.
2. For every $\hat{i} \in [N]$, generate parameters for F as follows:
   - Run $\mathsf{F.params}^{(\hat{i})} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$.
3. For every $\hat{i} \in [N]$, sample a public key using the key-generation algorithm of Σ: $(\mathrm{pk}^{(\hat{i})}, \mathrm{sk}^{(\hat{i})}) \leftarrow \mathsf{E.Gen}(1^\lambda)$.
4. Then, for every $\hat{i} \in [N]$, firstly compute $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$:
   Computing $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
   - Run again the trapdoor parameter-generation algorithm of HCOM: $(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{crs}}^{(\hat{i})}) \leftarrow \mathsf{HTGen}(1^\lambda)$. Compute $\mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})}$ as follows: For every $\hat{j} \in [N]$, $(\mathrm{com}_{\hat{j}}^{(\hat{i})}, \mathrm{aux}_{\mathrm{com}_{\hat{j}}}^{(\hat{i})}) \leftarrow \mathsf{HTCommit}(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{crs}}^{(\hat{i})})$.

   Let
   $$\widetilde{x}^{(\hat{i})} := (\mathrm{crs}^{(\hat{i})}, \mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})})$$
   and
   $$\mathrm{sv}^{(\hat{i})} := ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})}))$$
5. Then, for every $\hat{i} \in [N]$, compute the punctured key $K^{(\hat{i})}[2_G^{\widetilde{x}}]$ and $\mathsf{ioP}^{(\hat{i})}$:
   Puncturing a key and computing $\mathsf{ioP}^{(\hat{i})}$.
   - Now continue to build the rest of the program $\mathsf{ioP}^{(\hat{i})}$ as follows.
   - Run $\mathsf{F.params}^{(\hat{i})} \leftarrow \mathsf{F.ParamGen}(1^\lambda, n, G, N)$.
   - Sample a key, $K^{(\hat{i})} \leftarrow \mathsf{F.KeyGen}(1^\lambda, \mathsf{F.params})$.
   - Puncture K at $2_G^{\widetilde{x}}$: $K^{(\hat{i})}[2_G^{\widetilde{x}}] \leftarrow \mathsf{Puncture}(K^{(\hat{i})}, 2_G^{\widetilde{x}})$.
   - For $K^{(\hat{i})}[2_G^{\widetilde{x}}]$, generate $\mathsf{KD\text{-}Prog}_{10}^{(\hat{i})}$ the same way as $\mathsf{KD\text{-}Prog}_9^{(\hat{i})}$.

   Let $\mathrm{pv}^{(\hat{i})} = \widetilde{\mathrm{pv}}^{(\hat{i})} = (\widetilde{x}^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$.

6. Upon receiving $\mathsf{Reg}(\hat{i} \in [N])$, respond via $\widetilde{\mathrm{pv}}^{(\hat{i})}$. On the other hand, if the adversary himself registers a (corrupted) party $P_{\hat{i}}^*$ by presenting $\mathsf{RegCorr}(\hat{i} \in [N], \mathrm{pv}^{(\hat{i})})$, then the challenger simply ignores $(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
7. Upon receiving $\mathsf{Ext}(\hat{i})$ for a registered honest $P_{\hat{i}}$, firstly compute the following patches that will need to be used in computing the secret value of $P_{\hat{i}}$: $\{\mathsf{patch}(K^{(\hat{j})}[2^{\widetilde{x}}_G], (\hat{i}, \widetilde{x}^{(\hat{i})}))\}_{\hat{j} \in [N]}$, as $\mathsf{patch}(K^{(\hat{j})}[2^{\widetilde{x}}_G], (\hat{i}, \widetilde{x}^{(\hat{i})})) \leftarrow \mathsf{PatchGen}(K^{(\hat{j})}, \hat{i}, \widetilde{x}^{(\hat{i})})$. Then, compute an encryption of $\mathsf{patch}(K^{(\hat{j})}[2^{\widetilde{x}}_G], (\hat{i}, \widetilde{x}^{(\hat{i})}))$ as $\mathrm{ct}^{(\hat{i})}_{\hat{j}} \leftarrow \mathsf{Enc}(\mathrm{pk}^{(\hat{j})}, \mathsf{patch}(K^{(\hat{j})}[2^{\widetilde{x}}_G], (\hat{i}, \widetilde{x}^{(\hat{i})})))$. Finally, compute an opening for $\mathrm{com}^{(\hat{i})}_{\hat{j}}$ to $\mathrm{ct}^{(\hat{i})}_{\hat{j}}$ as $\mathrm{open}^{(\hat{i})}_{\hat{j}} \leftarrow \mathsf{HTDec}(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}^{(\hat{i})}_{\mathrm{com}_{\hat{j}}}, \mathrm{ct}^{(\hat{i})}_{\hat{j}})$. With this, set

$$\widetilde{\mathrm{sv}}^{(\hat{i})} = ((\mathrm{ct}^{(\hat{i})}_1, \mathrm{open}^{(\hat{i})}_1), \ldots, (\mathrm{ct}^{(\hat{i})}_N, \mathrm{open}^{(\hat{i})}_N))$$

and respond to the adversary's query via $\widetilde{\mathrm{sv}}^{(\hat{i})}$.
8. Upon receiving $\mathsf{Rev}(\mathcal{S}, \hat{j})$, let $\hat{i^*}$ be the smallest element in $\mathcal{S}$. Then, consider the following three cases:
   1. Consider the case when $P_{\hat{i^*}}$ was registered as an honest party via $\mathsf{Reg}(\hat{i^*} \in [N])$ and there exists $\hat{i} \in \mathcal{S}$ such that $P_{\hat{i}}$ was registered as an honest party and was corrupted by the adversary via $\mathsf{Ext}(\hat{i})$. Under this case, run $\mathsf{ioP}^{(\hat{i^*})}$ on input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k} \in \mathcal{S}}, \hat{i}, \mathrm{sv}^{(\hat{i})}_{\hat{i^*}})$ and respond via the output. (Note here that $\mathrm{sv}^{(\hat{i})}_{\hat{i^*}}$ was already computed while responding to $\mathsf{Ext}(\hat{i})$).
   2. Consider the case when $P_{\hat{i^*}}$ was registered as an honest party via $\mathsf{Reg}(\hat{i^*} \in [N])$ and for every $\hat{i} \in \mathcal{S}$, $P_{\hat{i}}$ was registered as an honest party and was never corrupted. Under this case, respond via $F(K^{(\hat{i^*})}, (\mathcal{S}, \{x^{(\hat{k})}\}_{\hat{k} \in \mathcal{S}}))$.
   3. Finally, consider the case when $P_{\hat{i^*}}$ was registered as a corrupt party via $\mathsf{RegCorr}(\hat{i} \in [N], \mathrm{pv}^{(\hat{i})})$. Then, compute $\mathrm{sv}^{(\hat{i})}_{\hat{i^*}}$ (by computing $\mathsf{patch}(K^{(\hat{i^*})}[2^{\widetilde{x}}_G], (\hat{i}, \widetilde{x}^{(\hat{i})}))$, encrypting $\mathrm{ct}^{(\hat{i})}_{\hat{i^*}} \leftarrow \mathsf{Enc}(\mathrm{pk}^{(\hat{i^*})}, \mathsf{patch}(K^{(\hat{i^*})}[2^{\widetilde{x}}_G], (\hat{i}, \widetilde{x}^{(\hat{i})})))$, and finally computing an opening for $\mathrm{com}^{(\hat{i})}_{\hat{i^*}}$ to $\mathrm{ct}^{(\hat{i})}_{\hat{i^*}}$ as $\mathrm{open}^{(\hat{i})}_{\hat{i^*}} \leftarrow \mathsf{HTDec}(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}^{(\hat{i})}_{\mathrm{com}_{\hat{i^*}}}, \mathrm{ct}^{(\hat{i})}_{\hat{i^*}})$. Then, run $\mathsf{ioP}^{(\hat{i^*})}$ on input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k} \in \mathcal{S}}, \hat{i}, \mathrm{sv}^{(\hat{i})}_{\hat{i^*}})$ and respond via the output. We remark here that the patch $\mathsf{patch}(K^{(\hat{i^*})}[2^{\widetilde{x}}_G], (\hat{i}, \widetilde{x}^{(\hat{i})}))$ was computed for a key $K^{(\hat{i^*})}[2^{\widetilde{x}}_G]$ that is not a part of any obfuscated program – this is so, since, $\mathsf{ioP}^{(\hat{i^*})}$ was registered as a corrupted party. Just to see what to look ahead for, this will be crucial in later hybrids when we move to setting $\mathrm{ct}^{(\hat{i})}_{\hat{i^*}}$ from being a ciphertext to just a random element. The reason we will do so will be clear as we reach those hybrids where we introduce this modification.
4. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, let $\hat{i^*}$ be the smallest element in ChQ. Compute Y_{REAL} as $Y_{\mathrm{REAL}} := F(K^{(\hat{i^*})}, (\mathsf{ChQ}, \{x^{(\hat{k})}\}_{\hat{k} \in \mathsf{ChQ}}))$. Also, sample $Y_{\mathrm{RAND}} \leftarrow \{0,1\}^m$, where $\{0,1\}^m$ is the co-domain of F. Respond to the adversary via Y_{EXPT}.
5. Finally, output whatever $\mathcal{A}$ outputs.

Lemma 21. $\mathsf{Hyb}_9 \equiv \mathsf{Hyb}_{10}$.

Proof. Observe that the only difference between Hyb_9 and Hyb_{10} is the following. For certain queries, while the challenger in Hyb_9 responded by running the programs, the challenger in Hyb_{10} responds

by computing the PRF directly. Since the programs also work by effectively evaluating the PRF, the view of the adversary is identical in both the hybrids. Hence, the Lemma.

■

Lemma 22. Let F be an obliviously-patchable puncturable PRF. Then the advantage of any PPT algorithm $\mathcal{A}$ in Hyb_{10}, namely $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_{10}}(\cdot)$, is negligible in the security parameter.

Proof. Assume for contradiction that there exists a PPT algorithm $\mathcal{A}$ with non-negligible advantage ϵ in Hyb_{10}. That is, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_{10}}(\lambda) = \epsilon$. Then we construct an adversary $\mathcal{B}$ that breaks the security of F also with a non-negligible probability.

Let N be any polynomial in the security parameter. Consider the following experiment which is a slight modification of the experiment $\text{EXPT}_{F,\mathcal{A}}^{\mathsf{PRF}}$ (cf. Definition 9) that tests pseudorandom at punctured points of the PRF F against adversary $\mathcal{A}$. Consider N such experiments played by a single adversary $\mathcal{A}$. All other parameters of the game are freshly and independently sampled. The adversary can make all kinds of queries that he is allowed to make in the original experiment, with the only difference that he is allowed to make the challenge query only in a single game. His success/failure in the modified game is then is defined to be his success/failure, respectively, in the game where he chooses to make the challenge query. We observe that, through a standard hybrid argument, this modified game is equivalent to the original game by a security loss of $1/N$. We shall work in this modified game in the current proof. However, for simplicity of notation, we shall refer to the modified game also by $\text{EXPT}_{F,\mathcal{A}}^{\mathsf{PRF}}$. However, to distinguish between the multiple PRF keys, we will explicitly specify the key we wish to refer to in a query to the oracle $\mathcal{O}$. We shall outline the modified game in the following. The modified game, like the original game begins by the adversary presenting $\widetilde{x} = (\widetilde{x}^{(1)}, \ldots, \widetilde{x}^{(N)})$; upon this, the challenger of the experiment $\text{EXPT}_{F,\mathcal{A}}^{\mathsf{PRF}}$ samples N keys independently and punctures each of them at $2_G^{\widetilde{x}}$ and gives the punctured keys $\{K^{(\hat{i})}[2_G^{\widetilde{x}}]\}_{\hat{i}\in[N]}$ to the adversary. Then, the adversary is given access to an oracle $\mathcal{O}$, that takes three kinds of queries and responds to them as follows:

- On query ($\text{PATCH } \hat{j} \text{ AT } \hat{i}$), respond via $\mathsf{patch}(K^{(\hat{j})}[2_G^{\widetilde{x}}], (\hat{i}, \widetilde{x}^{(\hat{i})}))$;
- On query ($\text{EVAL } \hat{j} \text{ AT } \mathcal{S}$), where $\mathcal{S} \subseteq [N]$ and $|\mathcal{S}| \leq G$, respond via $F(K^{(\hat{j})}, (\mathcal{S}, \{x^{(\hat{k})}\}_{\hat{k}\in\mathcal{S}}))$;
- On query ($\text{CHAL } \hat{j} \text{ AT } \mathsf{ChQ}$), where $\mathsf{ChQ} \subseteq [N]$ and $|\mathsf{ChQ}| \leq G$, respond via

$$\begin{cases} F(K^{(\hat{j})}, (\mathsf{ChQ}, \{x^{(\hat{k})}\}_{\hat{k}\in\mathsf{ChQ}})) & \text{if EXPT} = \text{REAL} \\ y & \text{if EXPT} = \text{RAND} \end{cases}$$

  where y is a random element from the co-domain, $\{0,1\}^m$, of F.

By the time $\mathcal{A}$ outputs b, let $Q_1^{(\hat{j})}$ be the set of $\hat{i}$ for which $\mathcal{A}$ makes ($\text{PATCH } \hat{j} \text{ AT } \hat{i}$) queries and $Q_2^{(\hat{j})}$ be the set of $\mathcal{S}$ for which $\mathcal{A}$ makes ($\text{EVAL } \hat{j} \text{ AT } \mathcal{S}$) queries. Let the challenger query made by the adversary be ($\text{CHAL } \hat{j} \text{ AT } \mathsf{ChQ}$); then we require that $\mathsf{ChQ} \subseteq [N] \setminus Q_1^{(\hat{j})}$ and $\mathsf{ChQ} \notin Q_2^{(\hat{j})}$.

The objective of the adversary would be to distinguish the two cases.

We shall show that using the given values and access to the oracle $\mathcal{O}$, $\mathcal{B}$ can simulate to $\mathcal{A}$ the hybrid game Hyb_{10} such that the following holds. If $\mathcal{A}$ has a non-negligible advantage in Hyb_{10}, then $\mathcal{B}$ can break the security of F also with a non-negligible advantage.

At a high level, $\mathcal{B}$ predominantly behaves as the challenger of Hyb_{10} except that it obtains all the values pertaining to the PRF from its interaction with its own challenger and the oracle $\mathcal{O}$. For clarity and ease of reading, these values shall be highlighted in a red underlined font.

1. Initiate by running the Setup algorithm. Namely, simply output (λ, N, G) as the output of the Setup algorithm. Next, respond to the adversary's queries as follows.
2. Let $\{\mathsf{F.params}^{(\hat{i})}\}_{\hat{i}\in[N]}$ be the parameters received by $\mathcal{B}$ from its challenger.
3. For every $\hat{i} \in [N]$, sample a public key using the key-generation algorithm of Σ: $(\mathrm{pk}^{(\hat{i})}, \mathrm{sk}^{(\hat{i})}) \leftarrow \mathsf{E.Gen}(1^\lambda)$.
4. Then, for every $\hat{i} \in [N]$, firstly compute $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$:
   Computing $(\widetilde{x}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
   - Run again the trapdoor parameter-generation algorithm of HCOM: $(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{crs}}^{(\hat{i})}) \leftarrow \mathsf{HTGen}(1^\lambda)$. Compute $\mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})}$ as follows: For every $\hat{j} \in [N]$, $(\mathrm{com}_{\hat{j}}^{(\hat{i})}, \mathrm{aux}_{\mathrm{com}_{\hat{j}}}^{(\hat{i})}) \leftarrow \mathsf{HTCommit}(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{crs}}^{(\hat{i})})$.

   Let
   $$\widetilde{x}^{(\hat{i})} := (\mathrm{crs}^{(\hat{i})}, \mathrm{com}_1^{(\hat{i})}, \ldots, \mathrm{com}_N^{(\hat{i})})$$
   and
   $$\mathrm{sv}^{(\hat{i})} := ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})}))$$
5. Then, for every $\hat{i} \in [N]$, obtain the punctured key $K^{(\hat{i})}[2^{\widetilde{x}}_G]$:
   - Query the challenger with $\widetilde{x} := (\widetilde{x}^{(1)}, \ldots, \widetilde{x}^{(N)})$ to receive $\{K^{(\hat{i})}[2^{\widetilde{x}}_G]\}_{\hat{i}\in[N]}$
6. For every $\hat{i} \in [N]$, construct $\mathsf{ioP}^{(\hat{i})}$:
   - For $K^{(\hat{i})}[2^{\widetilde{x}}_G]$, generate $\mathsf{KD\text{-}Prog}_{\mathtt{redu}}^{(\hat{i})}$ the same way as $\mathsf{KD\text{-}Prog}_{10}^{(\hat{i})}$.

   Let $\mathrm{pv}^{(\hat{i})} = \widetilde{\mathrm{pv}}^{(\hat{i})} = (\widetilde{x}^{(\hat{i})}, \mathsf{ioP}^{(\hat{i})})$.
7. Upon receiving $\mathsf{Reg}(\hat{i} \in [N])$, respond via $\widetilde{\mathrm{pv}}^{(\hat{i})}$. On the other hand, if the adversary himself registers a (corrupted) party $P_{\hat{i}}^*$ by presenting $\mathsf{RegCorr}(\hat{i} \in [N], \mathrm{pv}^{(\hat{i})})$, then the challenger simply ignores $(\widetilde{\mathrm{pv}}^{(\hat{i})}, \widetilde{\mathrm{sv}}^{(\hat{i})})$.
8. Upon receiving $\mathsf{Ext}(\hat{i})$ for a registered honest $P_{\hat{i}}$, for every $\hat{j} \in [N]$, proceed as follows: query $\mathcal{O}$ with (PATCH $\hat{j}$ AT $\hat{i}$) to obtain $\mathsf{patch}(K^{(\hat{j})}[2^{\widetilde{x}}_G], (\hat{i}, \widetilde{x}^{(\hat{i})}))$. Then, compute an encryption of $\mathsf{patch}(K^{(\hat{j})}[2^{\widetilde{x}}_G], (\hat{i}, \widetilde{x}^{(\hat{i})}))$ as $\mathrm{ct}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{Enc}(\mathrm{pk}^{(\hat{j})}, \mathsf{patch}(K^{(\hat{j})}[2^{\widetilde{x}}_G], (\hat{i}, \widetilde{x}^{(\hat{i})})))$. Finally, compute an opening for $\mathrm{com}_{\hat{j}}^{(\hat{i})}$ to $\mathrm{ct}_{\hat{j}}^{(\hat{i})}$ as $\mathrm{open}_{\hat{j}}^{(\hat{i})} \leftarrow \mathsf{HTDec}(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{com}_{\hat{j}}}^{(\hat{i})}, \mathrm{ct}_{\hat{j}}^{(\hat{i})})$. With this, set
   $$\widetilde{\mathrm{sv}}^{(\hat{i})} = ((\mathrm{ct}_1^{(\hat{i})}, \mathrm{open}_1^{(\hat{i})}), \ldots, (\mathrm{ct}_N^{(\hat{i})}, \mathrm{open}_N^{(\hat{i})}))$$
   and respond to the adversary's query via $\widetilde{\mathrm{sv}}^{(\hat{i})}$.
9. Upon receiving $\mathsf{Rev}(\mathcal{S}, \hat{j})$, let $\hat{i^*}$ be the smallest element in $\mathcal{S}$. Then, consider the following three cases:
   1. Consider the case when $P_{\hat{i^*}}$ was registered as an honest party via $\mathsf{Reg}(\hat{i^*} \in [N])$ and there exists $\hat{i} \in \mathcal{S}$ such that $P_{\hat{i}}$ was registered as an honest party and was corrupted by the adversary via $\mathsf{Ext}(\hat{i})$. Under this case, run $\mathsf{ioP}^{(\hat{i^*})}$ on input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{i}, \mathrm{sv}_{\hat{i^*}}^{(\hat{i})})$ and

respond via the output. (Note here that $\mathsf{sv}_{\hat{i}^*}^{(\hat{i})}$ was already computed while responding to $\mathsf{Ext}(\hat{i})$).

2. Consider the case when $P_{\hat{i}^*}$ was registered as an honest party via $\mathsf{Reg}(\hat{i}^* \in [N])$ and for every $\hat{i} \in \mathcal{S}$, $P_{\hat{i}}$ was registered as an honest party and was never corrupted. Under this case, query $\mathcal{O}$ with (EVAL $\hat{i}^*$ AT $\mathcal{S}$) and respond to $\mathcal{A}$ via the response given by $\mathcal{O}$.
3. Finally, consider the case when $P_{\hat{i}^*}$ was registered as a corrupt party via $\mathsf{RegCorr}(\hat{i} \in [N], \mathrm{pv}^{(\hat{i})})$. Then, compute $\mathrm{sv}_{\hat{i}^*}^{(\hat{i})}$ (by computing $\mathsf{patch}(K^{(\hat{i}^*)}[2\widetilde{x}_G], (\hat{i}, \widetilde{x}^{(\hat{i})}))$, encrypting $\mathrm{ct}_{\hat{i}^*}^{(\hat{i})} \leftarrow \mathsf{Enc}(\mathrm{pk}^{(\hat{i}^*)}, \mathsf{patch}(K^{(\hat{i}^*)}[2\widetilde{x}_G], (\hat{i}, \widetilde{x}^{(\hat{i})})))$, and finally computing an opening for $\mathrm{com}_{\hat{i}^*}^{(\hat{i})}$ to $\mathrm{ct}_{\hat{i}^*}^{(\hat{i})}$ as $\mathrm{open}_{\hat{i}^*}^{(\hat{i})} \leftarrow \mathsf{HTDec}(\mathrm{crs}^{(\hat{i})}, \mathrm{aux}_{\mathrm{com}_{\hat{i}^*}}^{(\hat{i})}, \mathrm{ct}_{\hat{i}^*}^{(\hat{i})})$. Then, run $\mathsf{ioP}^{(\hat{i}^*)}$ on input $(\mathcal{S}, (x^{(\hat{k})})_{\hat{k}\in\mathcal{S}}, \hat{i}, \mathrm{sv}_{\hat{i}^*}^{(\hat{i})})$ and respond via the output. We remark here that the patch $\mathsf{patch}(K^{(\hat{i}^*)}[2\widetilde{x}_G], (\hat{i}, \widetilde{x}^{(\hat{i})}))$ was computed for a key $K^{(\hat{i}^*)}[2\widetilde{x}_G]$ that is not a part of any obfuscated program – this is so, since, $\mathsf{ioP}^{(\hat{i}^*)}$ was registered as a corrupted party. Just to see what to look ahead for, this will be crucial in later hybrids when we move to setting $\mathrm{ct}_{\hat{i}^*}^{(\hat{i})}$ from being a ciphertext to just a random element. The reason we will do so will be clear as we reach those hybrids where we introduce this modification.
4. Upon receiving $\mathsf{Test}(\mathsf{ChQ})$, let $\hat{i}^*$ be the smallest element in ChQ. query $\mathcal{O}$ with (CHAL $\hat{i}^*$ AT ChQ) and respond to $\mathcal{A}$ via the response given by $\mathcal{O}$.
5. Finally, output whatever $\mathcal{A}$ outputs.

From the above description of $\mathcal{B}$, we have that if $\mathcal{B}$ interacts with its own challenger in the real game, then the view of $\mathcal{A}$ during its interaction with $\mathcal{B}$ is identical to its view in the real experiment of Hyb_{10}, namely $\mathrm{REAL}_{\mathcal{A}}^{\mathsf{Hyb}_{10}}$. On the other hand, if $\mathcal{B}$ interacts with its challenger in the random experiment, then the view of $\mathcal{A}$ during its interaction with $\mathcal{B}$ is identical to its view in the random experiment of Hyb_{10}, namely $\mathrm{RAND}_{\mathcal{A}}^{\mathsf{Hyb}_{10}}$. Therefore,

$$\begin{aligned}\mathsf{Adv}_{F,\mathcal{A}}(\lambda) =& |\Pr[\mathrm{REAL}_{F,\mathcal{A}}^{\mathsf{PRF}}(\lambda) \to 1] - \Pr[\mathrm{RAND}_{F,\mathcal{A}}^{\mathsf{PRF}}(\lambda) \to 1]| \\ =& |\Pr[\mathrm{REAL}_{\mathcal{A}}^{\mathsf{Hyb}_{10}} \to 1] - \Pr[\mathrm{RAND}_{\mathcal{A}}^{\mathsf{Hyb}_{10}} \to 1]| \\ =& \epsilon,\end{aligned}$$

thus arriving at a contradiction. Hence, the lemma.

■

■

C Concrete Constructions of Obliviously-Patchable Puncturable PRFs

In this Section, we give two constructions of obliviously-patchable puncturable PRFs. Our first construction uses multilinear maps. In this construction, we crucially make use of the fact that the size of group elements in multilinear maps can be bounded above by a polynomial independent of the multilinearity (as long as the multilinearity is polynomial in the security parameter). This is crucial to ensure 'succinct patches' (cf. Definition 9). However, succinctness of group elements is not satisfied by existing approximations of multilinear maps [GGH13a, CLT13]. To this end, we propose another construction that is compatible with these approximations, and in particular, does not assume that the group elements can be succinctly represented.

The assumptions on which we base security of our constructions of the PRFs are "one-more" cryptographic hard problems introduced by Bellare, Namprempre, Pointcheval, and Semanko [BNPS03]. Before we proceed with description of our constructions, we briefly review the one-more kind of problems.

Since their introduction, one-more kind of problems have found numerous applications in cryptography [BN05, BP02, BS01, Bol03, BG04, PV05]. In [BNPS03], these problems were studied for one-way functions in general. Later, they were studied as RSA and the Discrete Log variants. While these problems are computational, in the current work, we focus on the decisional variants. Note that the decisional variant of the Discrete Logarithm problem is the Decisional Diffie-Hellman (DDH) problem. In this work, we shall focus on it multilinear version. More specifically, the problem we focus on is the one-more variant of the Multilinear Decisional Diffie-Hellman (MDDH) problem (The MDDH problem was studied by [GGH13a] and this problem has found multiple applications []). Additionally, we also consider the inversion-variant of the MDDH problem, called the Multilinear Decisional Diffie-Hellman Inversion (MDDHI) problem. Given the close relation between DDH and MDDH/MDDHI, let us review the notion of one-more kind of problems with the specific notion of one-more DDH problem [BMV08] ([BMV08] studies the computational variant, the one-more Discrete Logarithm Problem).

To describe the 1-more DDH (1DDH) problem, we shall fix a few useful terminologies and notations. Consider a prime-ordered group $\mathbb{G} = \langle g \rangle$. The problem is described through the following game between a challenger and an adversary $\mathcal{A}$. Together with the description of the group, including the generator g, the adversary is given $n+1$ pairs of random elements $(g^{a_{10}}, g^{a_{11}}), \ldots, (g^{a_{(n+1)0}}, g^{a_{(n+1)1}})$. Then it is given access to an (inefficient) oracle $\mathcal{O}$ to which the adversary can query with *any* pair of elements $(g_0, g_1) = (g^{a_0}, g^{a_1})$; in return, the adversary receives $g^{a_0 a_1}$. Observe, $(g, g^{a_0}, g^{a_1}, g^{a_0 a_1})$ is a DDH tuple. The adversary is allowed to make at most n such queries. Finally, the adversary succeeds if it outputs $g^{a_{i2}}$ such that $a_{i2} = a_{i0} a_{i1}$ for every $i \in [n+1]$.

Note that, generally, in the one-more kind of problems (like in the 1DDH problem described above), the adversary is allowed to query the oracle with any input – in particular, the adversary is not restricted to presenting queries that are just among the $n+1$ challenge queries that the adversary is initially fed with. However, in our case, we shall restrict the adversary to present queries that only belong to the set of queries that it is initially fed with. In this sense, our setting is *weaker* than the usual notion of one-more kind of hard problems. Thus, we christen our problems as "One-more Weak MDDH" and "One-more Weak MDDHI" problems.

Our first construction which uses multilinear maps is based on the One-more Weak MDDH assumption. Our second construction which can work using even the existing approximations of multilinear maps [GGH13a, CLT13] is based on the One-more Weak MDDHI assumption.

This Section is organized as follows. In Section C.1, we review certain preliminaries specific to multilinear maps. In Section C.2, we state the One-more Weak MDDH assumption, present our first construction of obliviously-patchable puncturable PRF, and prove its security based on this assumption. Then, in Section C.3, we state the One-more Weak MDDHI assumption, present our second construction of obliviously-patchable puncturable PRF, and prove its security based on this assumption. Security of these two assumptions in the generic group model is established in the following Section D.

C.1 Preliminaries

Notations.

- We denote a multi-set consisting of m copies of an element g by $\{g\}_{m \text{ times}}$.
- We denote an ordered m-tuple $(g_1, \ldots, g_m)$ by $(g_i)_{i\in[m]}$. Furthermore, to denote an ordered sub-tuple consisting of g_i only at $i \in \mathcal{S}$, we write $(g_i)_{i\in\mathcal{S}}$.
- We shall denote the jth bit of a bit string x_i by $x_i[j]$.

Multilinear Maps. We assume the existence of a multilinear group generation algorithm GrpGen, which takes as input a security parameter λ and a positive integer κ to indicate the number of allowed pairing operations. $\mathsf{GrpGen}(1^\lambda, \kappa)$ outputs a sequence of groups $\overrightarrow{\mathbb{G}} = (\mathbb{G}_1, \ldots, \mathbb{G}_\kappa)$ each of large prime order $p > 2^\lambda$. In addition, we let g_i be a canonical generator of $\mathbb{G}_i$ (and we let the group's description contain g_i). We let $g = g_1$.

We assume the existence of a set of bilinear maps $\{\mathrm{e}_{i,j} : \mathbb{G}_i \times \mathbb{G}_j \leftarrow \mathbb{G}_{i+j} \mid i, j \geq 1; i + j \leq \kappa\}$. The map $\mathrm{e}_{i,j}$ satisfies the following relation for every $a, b \in \mathbb{Z}_p$:

$$\mathrm{e}_{i,j}(g_i^a, g_j^b) = g_{i+j}^{ab}.$$

We observe that one consequence of the above relation is that $\mathrm{e}_{i,j}(g_i, g_j) = g_{i+j}$ for every valid i, j. For simplicity, we denote the output of GrpGen as $\mathsf{grpparams} := (\overrightarrow{\mathbb{G}}, g, p, \mathrm{e}_{i,j}) \leftarrow \mathsf{GrpGen}(1^\lambda, \kappa)$. We shall follow the following simplifying shorthands throughout the paper.

- When the context is obvious, we will sometimes abuse notation and drop the subscripts i, j to specify the mapping; for example, we may simply write:
$$\mathrm{e}(g_i^a, g_j^b) = g_{i+j}^{ab}.$$
- Furthermore, we write $\mathrm{e}(h_1, h_2, \ldots, h_j)$ to denote $\mathrm{e}(h_1, \mathrm{e}(h_2, \ldots, h_j))$.
- Also, for convenience of notation, we sometimes represent the inputs via set notation; for eg., $\mathrm{e}(\{h_i\}_{i\in[j]}) = \mathrm{e}(h_1, h_2, \ldots, h_j)$.
- Let $h \in \mathbb{G}_{m_1}$ and let $m_2 > m_1$. Then we denote by $\mathrm{e}^{(m_2)}(h)$ the representation of h at level m_2. That is, $\mathrm{e}^{(m_2)}(h) = \mathrm{e}(h, g_{m_2-m_1})$. Observe $\mathrm{e}^{(m_2)}(h) \in \mathbb{G}_{m_2}$. The reader is suggested to read $\mathrm{e}^{(m_2)}(h)$ as 'raise h to level m_2'.
- Let $h \in \mathbb{G}_{m_1}$ and let $\tilde{g} \in \mathbb{G}_1$. Then we denote by $\mathrm{e}_{(\tilde{g})}^{(m_2)}(h)$ the representation of h raised to level m_2 by pairing with $\tilde{g}$ (instead of g as in $\mathrm{e}^{(m_2)}(h)$). That is, $\mathrm{e}_{(\tilde{g})}^{(m_2)}(h) = \mathrm{e}(h, \mathrm{e}(\{\tilde{g}\}_{(m_2-m_1) \text{ times}}))$. Observe $\mathrm{e}_{(\tilde{g})}^{(m_2)}(h) \in \mathbb{G}_{m_2}$. The reader is suggested to read $\mathrm{e}_{(\tilde{g})}^{(m_2)}(h)$ as 'raise h to level m_2 with $\tilde{g}$'.

Succinct Multilinear Maps. We consider a class of multilinear maps where, roughly speaking, the size of group elements (number of bits needed to represent them) can be bounded by a polynomial that depends only on the security parameter and not on the multilinearity level. Observe that this property is satisfied by multilinear maps in the generic model; therein, for instance, we consider a random mapping $\Phi : \mathbb{Z}_p \times [\kappa] \leftarrow \{0,1\}^\ell$. The group elements can thus be represented in $\ell + \log(\kappa) \leq \ell + \lambda$ bits. We define such class of multilinear maps, more formally, below.

Definition 14 (Succinct Multilinear Maps)**.** Let GrpGen specify a class of multilinear maps. These maps are said to be succinct maps if there exists a polynomial P, such that for any polynomial $\kappa(\cdot)$, the following holds. For any $((\mathbb{G}_1, \ldots, \mathbb{G}_{\kappa(\lambda)}), g, p, \mathrm{e}_{i,j}) \leftarrow \mathsf{GrpGen}(1^\lambda, \kappa)$, for any $i \in [\kappa]$, any element $h \in \mathbb{G}_i$ can be represented in $P(\lambda)$ bits.

C.2 Construction Based On Succinct Multilinear Maps

Computational Assumption. Now we define the One-more Weak (n, G, N)-Multilinear Decisional Diffie-Hellman ((n, G, N)-1wMDDH) assumption as follows:

Assumption 1 (Weak One-more (n, G, N)-Multilinear Decisional Diffie-Hellman Assumption: (n, G, N)-1wMDDH Assumption)**.** The One-more Weak (n, G, N)-Multilinear Decisional Diffie-Hellman ((n, G, N)-1wMDDH) Assumption is said to hold for GrpGen if, for every PPT adversary $\mathcal{A}$, the following is negligible:

$$\mathsf{Adv}_{\mathcal{A}}^{(n,G,N)\text{-}1\mathsf{wMDDH}}(\lambda) := |\Pr[\text{REAL}_{\mathcal{A}}^{1\mathsf{wMDDH}} \to 1] - \Pr[\text{RAND}_{\mathcal{A}}^{1\mathsf{wMDDH}} \to 1]|$$

where, for $\text{EXPT} \in \{\text{REAL}, \text{RAND}\}$, $\text{EXPT}_{\mathcal{A}}^{1\mathsf{wMDDH}}$ is defined as follows.

$$\left|\begin{array}{c} \textbf{Experiment } \text{EXPT}_{\mathcal{A}}^{1\mathsf{wMDDH}}: \\ \mathsf{grpparams} := (\overrightarrow{\mathbb{G}}, g, p, \mathrm{e}_{i,j}) \leftarrow \mathsf{GrpGen}(1^\lambda, nG) \\ \text{Sample } \ \beta, \eta \leftarrow \mathbb{Z}_p, \ \{d_i[1], \ldots, d_i[n]\}_{i\in[N]} \leftarrow \mathbb{Z}_p \\ \text{Compute } \ (B, E) = (g^\beta, g_n^\eta), \forall i \in [N], \ (D_i[1], \ldots, D_i[n]) = (g^{d_i[1]}, \ldots, g^{d_i[n]}) \\ \text{Output } \ b \leftarrow \mathcal{A}^{\mathcal{O}.1\mathsf{wMDDH}}(\mathsf{grpparams}, B, E, \{D_i[1], \ldots, D_i[n]\}_{i\in[N]}) \end{array}\right|$$

where, the oracle $\mathcal{O}.1\mathsf{wMDDH}$ takes three kinds of queries:

1. On query (ONE-MORE-AT i) for $i \in [N]$, $\mathcal{O}.1\mathsf{wMDDH}$ returns $\widetilde{\mathsf{Prod}}_i = g_n^{\beta(d_i[1]\cdots d_i[n])}$. By the end of the experiment, let Q_1 denote the set of all i for which the adversary queries (ONE-MORE-AT i).
2. On query (COMBINE $\mathcal{S}$) for $\mathcal{S} \in [N]$ and $|\mathcal{S}| \leq G$, $\mathcal{O}.1\mathsf{wMDDH}$ returns $g_{nG}^{\left(\beta \prod_{i\in\mathcal{S}} (d_i[1]\cdots d_i[n])\right)\eta^{(G-|\mathcal{S}|)}}$. By the end of the experiment, let Q_2 denote the set of all $\mathcal{S}$ for which the adversary queries (COMBINE $\mathcal{S}$).
3. On query (CHAL-AT ChQ) for $\mathsf{ChQ} \in [N]$ and $|\mathcal{S}| \leq G$, $\mathcal{O}.1\mathsf{wMDDH}$ returns $\widetilde{\mathsf{Prod}}_{\text{EXPT}}$ that is computed as follows: Let $\widetilde{\mathsf{Prod}}_{\text{REAL}} = g_{nG}^{\left(\beta \prod_{i\in\mathsf{ChQ}} (d_i[1]\cdots d_i[n])\right)\eta^{(G-|\mathsf{ChQ}|)}}$; let $\widetilde{\mathsf{Prod}}_{\text{RAND}} \leftarrow \mathbb{G}_{nG}$. $\mathcal{A}$ is allowed to make only one such query. It is required that $|\mathsf{ChQ}| \leq G$, $\mathsf{ChQ} \subseteq [N] \setminus Q_1$, and $\mathsf{ChQ} \notin Q_2$.

We now present our first concrete construction of obliviously-patchable puncturable PRF F, which is based on succinct multilinear maps.

Let n, G, N be polynomials. We shall construct a function family $F : \mathcal{K} \times [[n, G, N]] \to \mathcal{Z}$ associated with PPT algorithms ($\mathsf{F.ParamGen}$, $\mathsf{F.KeyGen}$, $\mathsf{Puncture}$, $\mathsf{PatchGen}$, Eval, $\mathsf{OPatchGen}$, $\mathsf{OPuncture}$).

Algorithm $\mathsf{F.ParamGen}(1^\lambda, n, G, N)$: $\mathsf{F.ParamGen}$ simply runs GrpGen of the succinct multilinear maps for multilinearity nG: $(\overrightarrow{\mathbb{G}}, g, p, \mathrm{e}_{i,j}) \leftarrow \mathsf{GrpGen}(1^\lambda, nG)$. Return $\mathsf{F.params} := (\overrightarrow{\mathbb{G}}, g, p, \mathrm{e}_{i,j})$.

Algorithm $\mathsf{F.KeyGen}(1^\lambda, \mathsf{F.params})$:

- Sample $\beta, \eta \leftarrow \mathbb{Z}_p$.
- For $i \in [N]$, sample $\begin{pmatrix} d_i[1,0] & \ldots, & d_i[n,0] \\ d_i[1,1] & & d_i[n,1] \end{pmatrix} \leftarrow \mathbb{Z}_p^{2\times n}$. Return
- Return $K := \left(\beta, \eta, \left\{\begin{pmatrix} d_i[1,0] & \ldots, & d_i[n,0] \\ d_i[1,1] & & d_i[n,1] \end{pmatrix}\right\}_{i\in[N]}\right)$.

Function $F(K,x)$: F takes as inputs a key K generated by $\mathsf{F.KeyGen}$ and $x \in [[n,G,N]]$. Parse $K = \left(\beta, \eta, \left\{\begin{pmatrix} d_i[1,0] & \dots, & d_i[n,0] \\ d_i[1,1] & & d_i[n,1] \end{pmatrix}\right\}_{i\in[N]}\right)$ and $x = (\mathcal{S}, (x_i)_{i\in\mathcal{S}})$.

Its output is defined to be $F(K,x) := g_{nG}^{\left(\beta \prod_{i\in\mathcal{S}} (d_i[1,x_i[1]]\cdots d_i[n,x_i[n]])\right)\eta^{(G-|\mathcal{S}|)}}$.

Algorithm $\mathsf{Puncture}(K, 2^{\widetilde{x}}_G)$: $\mathsf{Puncture}$ takes as input a key K and a set $2^{\widetilde{x}}_G$, where, $x \in (\{0,1\}^n)^N$. (Recall that $2^{\widetilde{x}}_G$ is the set of all $x' \in [[n,G,N]]$ where, for every $x'|_i \neq (\perp,\dots,\perp)$, we have $x'|_i = x|_i$.) Parse $K = \left(\beta, \eta, \left\{\begin{pmatrix} d_i[1,0] & \dots, & d_i[n,0] \\ d_i[1,1] & & d_i[n,1] \end{pmatrix}\right\}_{i\in[N]}\right)$. Also, parse $x = (\tilde{x}_1,\dots,\tilde{x}_N)$.

- Compute $(B,E) \leftarrow (g^\beta, g_n^\eta)$;
- For every $i \in [N]$, for the positions specified by $x_i[1],\dots,x_i[n]$, compute $(D_i[1,x_i[1]],\dots,D_i[n,x_i[n]]) \leftarrow (g^{d_i[1,x_i[1]]},\dots,g^{d_i[n,x_i[n]]})$.
- Return[5] $K[2^{\widetilde{x}}_G] := \left(B, E, \left\{\begin{matrix} D_i[1,x_i[1]] & \dots, & D_i[n,x_i[n]] \\ d_i[1,(1-x_i[1])] & & d_i[n,(1-x_i[n])] \end{matrix}\right\}_{i\in[N]}\right)$.

Algorithm $\mathsf{PatchGen}(K, i, x_i, \mathsf{state}_i)$: $\mathsf{PatchGen}$ takes as input a key K, an index $i \in [N]$, and $x_i \in \{0,1\}^n$. Parse $K = \left(\beta, \eta, \left\{\begin{pmatrix} d_i[1,0] & \dots, & d_i[n,0] \\ d_i[1,1] & & d_i[n,1] \end{pmatrix}\right\}_{i\in[N]}\right)$.

- Compute $g_n^{\beta(d_i[1,x_i[1]]\cdots d_i[n,x_i[n]])}$.
- Return $\mathsf{patch}(K,(i,x_i)) := g_n^{\beta(d_i[1,x_i[1]]\cdots d_i[n,x_i[n]])}$.

Algorithm $\mathsf{Eval}(K[2^{\widetilde{x}}_G], x)$ / $\mathsf{Eval}((K[2^{\widetilde{x}}_G], \mathsf{patch}(K,(j,\widetilde{x}_j))), x)$: Eval takes two kinds of queries:

$\mathsf{Eval}(K[2^{\widetilde{x}}_G], x)$**:** One kind corresponds to evaluating the PRF output for inputs *not* in $2^{\widetilde{x}}_G$ using a key punctured at $2^{\widetilde{x}}_G$. More formally: For $\widetilde{x} \in (\{0,1\}^n)^N$, $\widetilde{x} = (\widetilde{x}_1,\dots,\widetilde{x}_N)$, parse $K[2^{\widetilde{x}}_G] = \left(B, E, \left\{\begin{matrix} D_i[1,\widetilde{x}_i[1]] & \dots, & D_i[n,\widetilde{x}_i[n]] \\ d_i[1,(1-\widetilde{x}_i[1])] & & d_i[n,(1-\widetilde{x}_i[n])] \end{matrix}\right\}_{i\in[N]}\right)$. Now consider an input $x \notin 2^{\widetilde{x}}_G$, where, $x = (\mathcal{S},(x_i)_{i\in\mathcal{S}})$; we have that there exists $j \in \mathcal{S}$ such that $x_j \neq \widetilde{x}_j$. On input $(K[2^{\widetilde{x}}_G], x)$, proceed as follows.

- For every $i \in \mathcal{S}\setminus\{j\}$, compute $\mathsf{Prod}_i \in \mathbb{G}_n$ as

$$\begin{aligned}\mathsf{Prod}_i &= \left(\mathrm{e}^{(n)}(\{D_i[k,\widetilde{x}_i[k]]\}_{k\,:\,x_i[k]=\widetilde{x}_i[k]})\right)^{\prod_{k'\,:\,x_i[k']=1-\widetilde{x}_i[k']} d_i[k',(1-\widetilde{x}_i[k'])]} \\ &= g_n^{(d_i[1,x_i[1]]\cdots d_i[n,x_i[n]])}\end{aligned}$$

- Compute $\mathsf{Prod}_j \in \mathbb{G}_n$ as

$$\begin{aligned}\mathsf{Prod}_j &= \left(\mathrm{e}^{(n)}(\underline{B}, \{D_j[k,\widetilde{x}_j[k]]\}_{k\,:\,x_j[k]=\widetilde{x}_j[k]})\right)^{\prod_{k'\,:\,x_j[k']=1-\widetilde{x}_j[k']} d_j[k',(1-\widetilde{x}_j[k'])]} \\ &= g_n^{\beta(d_j[1,x_j[1]]\cdots d_j[n,x_j[n]])}\end{aligned}$$

[5] Observe, we present all the elements that have been raised to Level-1 in the upper row, for simplicity. We shall assume that whether they correspond to 0 or 1 will be an additional information given alongside.

- Return

$$\begin{aligned}\mathsf{Eval}(K[2^{\widetilde{x}}_G], x) &:= \mathrm{e}(\{\mathsf{Prod}_i\}_{i\in[\mathcal{S}]}, \{E\}_{(G-|\mathcal{S}|) \text{ times}}) \\ &= g_{nG}^{\left(\beta \prod_{i\in\mathcal{S}} (d_i[1,x_i[1]]\cdots d_i[n,x_i[n]])\right)\eta^{(G-|\mathcal{S}|)}} \end{aligned} \tag{7}$$

$\mathsf{Eval}((K[2^{\widetilde{x}}_G], \mathsf{patch}(K,(j,\widetilde{x}_j))), x)$**:** The other kind corresponds to evaluating the PRF output for inputs in $2^{\widetilde{x}}_G$ using a key punctured at $2^{\widetilde{x}}_G$ and using a patch. More formally: For $\widetilde{x} \in (\{0,1\}^n)^N$, $\widetilde{x} = (\widetilde{x}_1, \ldots, \widetilde{x}_N)$, parse $K[2^{\widetilde{x}}_G] = \left(B, E, \left\{ \begin{matrix} D_i[1,\widetilde{x}_i[1]] \\ d_i[1,(1-\widetilde{x}_i[1])] \end{matrix} \quad,\ldots,\quad \begin{matrix} D_i[n,\widetilde{x}_i[n]] \\ d_i[n,(1-\widetilde{x}_i[n])] \end{matrix} \right\}_{i\in[N]}\right)$. Now consider an input $x \in 2^{\widetilde{x}}_G$, where, $x = (\mathcal{S}, (x_i)_{i\in\mathcal{S}})$; we have that for every $i \in \mathcal{S}$, $x_i = \widetilde{x}_i$. On input $((K[2^{\widetilde{x}}_G], \mathsf{patch}(K,(j,\widetilde{x}_j))), x)$ for some $j \in \mathcal{S}$ proceed as follows.

- For every $i \in \mathcal{S} \setminus \{j\}$, compute $\mathsf{Prod}_i \in \mathbb{G}_n$ as

$$\begin{aligned}\mathsf{Prod}_i &= \left(\mathrm{e}^{(n)}(\{D_i[k,\widetilde{x}_i[k]]\}_{k\,:\,x_i[k]=\widetilde{x}_i[k]})\right)^{\prod_{k'\,:\,x_i[k']=1-\widetilde{x}_i[k']} d_i[k',(1-\widetilde{x}_i[k'])]} \\ &= g_n^{(d_i[1,x_i[1]]\cdots d_i[n,x_i[n]])}\end{aligned}$$

- Recall that $\mathsf{patch}(K,(j,\widetilde{x}_j)) = g_n^{\beta(d_j[1,\widetilde{x}_j[1]]\cdots d_j[n,\widetilde{x}_j[n]])}$. Define $\mathsf{Prod}_j := \mathsf{patch}(K,(j,\widetilde{x}_j))$.
- Return

$$\begin{aligned}\mathsf{Eval}((K[2^{\widetilde{x}}_G], \mathsf{patch}(K,(i,\widetilde{x}_i))), x) &:= \mathrm{e}(\{\mathsf{Prod}_i\}_{i\in[\mathcal{S}]}, \{E\}_{(G-|\mathcal{S}|) \text{ times}}) \\ &= g_{nG}^{\left(\beta \prod_{i\in\mathcal{S}} (d_i[1,x_i[1]]\cdots d_i[n,x_i[n]])\right)\eta^{(G-|\mathcal{S}|)}}\end{aligned} \tag{8}$$

Algorithm $\mathsf{OPatchGen}(1^\lambda, \mathsf{F.params})$: $\mathsf{OPatchGen}$ takes as input group parameters $\mathsf{F.params} = (\overrightarrow{\mathbb{G}}, g, p, \mathrm{e}_{i,j})$ and outputs a set of patches and some state information as follows.

- Sample $\beta \leftarrow \mathbb{Z}_p$, and compute $B = g^\beta$.
- For every $i \in [N]$, sample $(d_i[1], \ldots, d_i[n]) \leftarrow \mathbb{Z}_p^n$, and compute $(\hat{D}_i[1], \ldots, \hat{D}_i[n]) \leftarrow (g^{d_i[1]}, \ldots, g^{d_i[n]})$.
- Define $\mathsf{o.state} := (B, \{(\hat{D}_i[1], \ldots, \hat{D}_i[n])\}_{i\in[N]})$.
- For every $i \in [N]$, define $\mathsf{patch}_i := g_n^{\beta(d_i[1]\cdots d_i[n])}$.
- Return $(\{\mathsf{patch}_i\}_{i\in[N]}, \mathsf{o.state})$.

Algorithm $\mathsf{OPuncture}(1^\lambda, \mathsf{o.state}, x)$: $\mathsf{OPuncture}$ takes as inputs the security parameter, the state information output by $\mathsf{OPatchGen}$ and $\widetilde{x} \in (\{0,1\}^n)^N$. Parse $\mathsf{o.state} = (B, \{(\hat{D}_i[1], \ldots, \hat{D}_i[n])\}_{i\in[N]})$ and $\widetilde{x} = (\widetilde{x}_1, \ldots, \widetilde{x}_N)$. It outputs a punctured key at $2^{\widetilde{x}}_G$ consistent with $\mathsf{o.state}$ as follows.

- Sample $\eta \leftarrow \mathbb{Z}_p$, and compute $E = g^\eta$.
- For the positions specified by $\widetilde{x}_i[1], \ldots, \widetilde{x}_i[n]$, assign $(D_i[1,\widetilde{x}_i[1]], \ldots, D_i[n,\widetilde{x}_i[n]]) \leftarrow (\hat{D}_i[1], \ldots, \hat{D}_i[n])$.
- Sample $(d_i[1,(1-\widetilde{x}_i[1])], \ldots, d_i[n,(1-\widetilde{x}_i[n])]) \leftarrow \mathbb{Z}_p^n$.

- Return $K[2^{\widetilde{x}}_G] := \left(B, E, \left\{ \begin{matrix} D_i[1, \widetilde{x}_i[1]] \\ d_i[1, (1-\widetilde{x}_i[1])] \end{matrix} \quad , \ldots , \quad \begin{matrix} D_i[n, \widetilde{x}_i[n]] \\ d_i[n, (1-\widetilde{x}_i[n])] \end{matrix} \right\}_{i \in [N]}\right)$.

Theorem 5. Let the One-more Weak (n, G, N)-Multilinear Decisional Diffie-Hellman ((n, G, N)-1wMDDH) Assumption hold for GrpGen. Then F is an obliviously-patchable puncturable PRF.

Proof. We shall show that F satisfies all the properties specified in Definition 9.

By simple analyzing of the description of the associated algorithms, it is clear that F satisfies the following properties : Functionality preserved under puncturing, block-wise patchable, obliviously patchable.

Succinct patches. Observe that each patch is just a single group element. Since the current construction in using succinct multilinear maps, the size of the group elements is just a function of λ and is independent of n (and G, N), as required.

Pseudorandom at punctured points. We shall establish this property under the assumption that (n, G, N)-1wMDDH Assumption hold for GrpGen. Assume for contradiction that there exists an adversary $\mathcal{A}$ for which $\mathsf{Adv}_{F,\mathcal{A}}(\lambda) = \varepsilon(\lambda)$ that is non-negligible. Then we construct an adversary $\mathcal{B}$ that breaks the security of F also with a non-negligible probability.

$\mathcal{B}$, upon receiving $(\mathsf{grpparams}, B, E, \{D_i[1], \ldots, D_i[n]\}_{i \in [N]})$ from its own challenger, proceeds as follows.

- $\mathcal{B}$ invokes $\mathcal{A}$ with grpparams.
- $\mathcal{A}$ then gives $\tilde{x} \in (\{0,1\}^n)^N$ and expects to receive a key punctured at $2^{\tilde{x}}_G$.
- $\mathcal{B}$ then executes the following step: For every $i \in [N]$, for the positions specified by $\tilde{x}_i[1], \ldots, \tilde{x}_i[n]$, set $(D_i[1, \tilde{x}_i[1]], \ldots, D_i[n, \tilde{x}_i[n]]) \leftarrow (D_i[1], \ldots, D_i[n])$.
- Sample $(d_i[1, (1-\tilde{x}_i[1])], \ldots, d_i[n, (1-\tilde{x}_i[n])]) \leftarrow \mathbb{Z}_p^n$.
- Give to $\mathcal{A}$, $K[2^{\widetilde{x}}_G] := \left(B, E, \left\{ \begin{matrix} D_i[1, \widetilde{x}_i[1]] \\ d_i[1, (1-\widetilde{x}_i[1])] \end{matrix} \quad , \ldots , \quad \begin{matrix} D_i[n, \widetilde{x}_i[n]] \\ d_i[n, (1-\widetilde{x}_i[n])] \end{matrix} \right\}_{i \in [N]}\right)$.
- Answer to $\mathcal{A}$'s queries as follows.
  1. On query (PATCH-AT i), where $i \in [N]$, query the 1wMDDH challenger with (ONE-MORE-AT i). Let the response received from by $\mathcal{B}$ be $\widetilde{\mathsf{Prod}}_i = g_n^{\beta(d_i[1]\cdots d_i[n])}$. Respond to $\mathcal{A}$ via $\mathsf{patch}(K, (i, \tilde{x}_i)) := \widetilde{\mathsf{Prod}}_i$.
  2. On query (EVAL-AT $\mathcal{S}$), where $\mathcal{S} \subseteq [N]$ and $|\mathcal{S}| \leq G$, query the 1wMDDH challenger with (COMBINE $\mathcal{S}$). Respond to $\mathcal{A}$ via the response received from by $\mathcal{B}$, namely, $g_{nG}^{\left(\beta \prod_{i \in \mathcal{S}} (d_i[1]\cdots d_i[n])\right)\eta^{(G-|\mathcal{S}|)}}$.
  3. On query (CHAL-AT ChQ), where $\mathsf{ChQ} \subseteq [N]$ and $|\mathsf{ChQ}| \leq G$, query the 1wMDDH challenger with (CHAL-AT ChQ). Let the response received from by $\mathcal{B}$ be $\widetilde{\mathsf{Prod}}_{\text{EXPT}}$. Respond to $\mathcal{A}$ via $\widetilde{\mathsf{Prod}}_{\text{EXPT}}$.

Analysis. We observe that if the experiment played by the 1wMDDH challenger is the real experiment, namely $\text{REAL}_{\mathcal{A}}^{\mathsf{1wMDDH}}$, then the view of $\mathcal{A}$ when run by $\mathcal{B}$ is identical to its view in the real experiment of pseudorandomness of punctured PRFs. On the other hand, if the experiment played by the 1wMDDH challenger is the random experiment, namely $\text{RAND}_{\mathcal{A}}^{\mathsf{1wMDDH}}$, then the view of $\mathcal{A}$ when run by $\mathcal{B}$ is identical to its view in the random experiment of pseudorandomness of punctured PRFs.

Thus, we have that,

$$\begin{aligned}\mathsf{Adv}_{\mathcal{B}}^{(n,G,N)\text{-}\mathsf{1wMDDH}}(\lambda) =& |\Pr[\text{REAL}_{\mathcal{B}}^{\mathsf{1wMDDH}} \to 1] - \Pr[\text{RAND}_{\mathcal{B}}^{\mathsf{1wMDDH}} \to 1]| \\ =& |\Pr[\text{REAL}_{F,\mathcal{A}}^{\mathsf{PRF}}(\lambda) \to 1] - \Pr[\text{RAND}_{F,\mathcal{A}}^{\mathsf{PRF}}(\lambda) \to 1]| \\ =& \varepsilon(\lambda),\end{aligned}$$

thus arriving at a contradiction. Hence, the lemma.

■

C.3 Construction Based On Graded Encoding Schemes

Computational Assumption. Now define the One-more Weak (n, G, N)-Multilinear Decisional Diffie-Hellman Inversion ((n, G, N)-1wMDDHI) assumption as follows:

Assumption 2 (Weak One-more (n, G, N)-Multilinear Decisional Diffie-Hellman Inversion: (n, G, N)-1wMDDHI)**.** GrpGen is said to satisfy the One-more Weak (n, G, N)-Multilinear Decisional Diffie-Hellman ((n, G, N)-1wMDDHI) problem states that for every PPT adversary $\mathcal{A}$, the following is negligible:

$$\mathsf{Adv}_{\mathcal{A}}^{(n,G,N)\text{-}\mathsf{1wMDDHI}}(\lambda) := |\Pr[\text{REAL}_{\mathcal{A}}^{\mathsf{1wMDDHI}} \to 1] - \Pr[\text{RAND}_{\mathcal{A}}^{\mathsf{1wMDDHI}} \to 1]|$$

where, for $\text{EXPT} \in \{\text{REAL}, \text{RAND}\}$, $\text{EXPT}_{\mathcal{A}}^{\mathsf{1wMDDHI}}$ is defined as follows.

$$\boxed{\begin{array}{c}\textbf{Experiment } \text{EXPT}_{\mathcal{A}}^{\mathsf{1wMDDHI}}: \\ \mathsf{grpparams} := (\overrightarrow{\mathbb{G}}, g, p, \mathrm{e}_{i,j}) \leftarrow \mathsf{GrpGen}(1^\lambda, 2G) \\ \beta, \eta, \{\alpha_i\}_{i\in[N]} \leftarrow \mathbb{Z}_p \\ \text{Compute } \ (B, E) = (g^\beta, g_2^\eta),\ \forall i \in [N],\ (A_i[1], \ldots, A_i[n-1]) = (g^{\alpha_i}, \ldots, g^{\alpha_i^{n-1}}) \\ \text{Output } \ b \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{grpparams}, B, E, \{A_i[1], \ldots, A_i[n-1]\}_{i\in[N]})\end{array}}$$

where, the oracle $\mathcal{O}$.1wMDDHI takes three kinds of queries:

1. On query (ONE-MORE-AT i) for $i \in [N]$, $\mathcal{O}$.1wMDDHI returns $\widetilde{\mathsf{Prod}}_i = g_2^{\beta\alpha_i^n}$. By the end of the experiment, let Q_1 denote the set of all i for which the adversary queries (ONE-MORE-AT i).
2. On query (COMBINE $\mathcal{S}$) for $\mathcal{S} \in [N]$ and $|\mathcal{S}| \leq G$, $\mathcal{O}$.1wMDDHI returns $g_{2G}^{(\beta \prod_{i\in\mathcal{S}} \alpha_i^n)\eta^{(G-|\mathcal{S}|)}}$. By the end of the experiment, let Q_2 denote the set of all $\mathcal{S}$ for which the adversary queries (COMBINE $\mathcal{S}$).
3. On query (CHAL-AT ChQ) for $\mathsf{ChQ} \in [N]$ and $|\mathcal{S}| \leq G$, $\mathcal{O}$.1wMDDHI returns $\widetilde{\mathsf{Prod}}_{\text{EXPT}}$ that is computed as follows: Let $\widetilde{\mathsf{Prod}}_{\text{REAL}} = g_{2G}^{(\beta \prod_{i\in\mathsf{ChQ}} \alpha_i^n)\eta^{(G-|\mathsf{ChQ}|)}}$; let $\widetilde{\mathsf{Prod}}_{\text{RAND}} \leftarrow \mathbb{G}_{2G}$. $\mathcal{A}$ is allowed to make only one such query. It is required that $|\mathsf{ChQ}| \leq G$, $\mathsf{ChQ} \subseteq [N] \setminus Q_1$, and $\mathsf{ChQ} \notin Q_2$.

We now present our second concrete construction of obliviously-patchable puncturable PRF F. This can be constructed using the existing candidate constructions of multilinear maps (leveled graded encoding schemes).

Let n, G, N be polynomials. We shall construct a function family $F : \mathcal{K} \times [[n, G, N]] \to \mathcal{Z}$ associated with PPT algorithms ($\mathsf{F.ParamGen}, \mathsf{F.KeyGen}, \mathsf{Puncture}, \mathsf{PatchGen}, \mathsf{Eval}, \mathsf{OPatchGen}, \mathsf{OPuncture}$).

Algorithm $\mathsf{F.ParamGen}(1^\lambda, n, G, N)$: $\mathsf{F.ParamGen}$ simply runs GrpGen of the succinct multilinear maps for multilinearity $2G$: $(\overrightarrow{\mathbb{G}}, g, p, \mathrm{e}_{i,j}) \leftarrow \mathsf{GrpGen}(1^\lambda, 2G)$. Return $\mathsf{F.params} := (\overrightarrow{\mathbb{G}}, g, p, \mathrm{e}_{i,j})$.

Algorithm $\mathsf{F.KeyGen}(1^\lambda, \mathsf{F.params})$:

- Sample $\beta, \eta \leftarrow \mathbb{Z}_p$.
- For $i \in [N]$, sample $\begin{pmatrix} d_i[1,0] & \dots, & d_i[n,0] \\ d_i[1,1] & & d_i[n,1] \end{pmatrix} \leftarrow \mathbb{Z}_p^{2\times n}$. Return
- Return $K := \left(\beta, \eta, \left\{ \begin{pmatrix} d_i[1,0] & \dots, & d_i[n,0] \\ d_i[1,1] & & d_i[n,1] \end{pmatrix} \right\}_{i\in[N]}\right)$.

Function $F(K, x)$: F takes as inputs a key K generated by $\mathsf{F.KeyGen}$ and $x \in [[n, G, N]]$. Parse $K = \left(\beta, \eta, \left\{ \begin{pmatrix} d_i[1,0] & \dots, & d_i[n,0] \\ d_i[1,1] & & d_i[n,1] \end{pmatrix} \right\}_{i\in[N]}\right)$ and $x = (\mathcal{S}, (x_i)_{i\in\mathcal{S}})$.

Its output is defined to be $F(K, x) := g_{2G}^{\left(\beta \prod_{i\in\mathcal{S}} (d_i[1,x_i[1]]\cdots d_i[n,x_i[n]])\right)\eta^{(G-|\mathcal{S}|)}}$.

Algorithm $\mathsf{Puncture}(K, 2_G^{\tilde{x}})$: $\mathsf{Puncture}$ takes as input a key K and a set $2_G^{\tilde{x}}$, where, $\tilde{x} \in (\{0,1\}^n)^N$. (Recall that $2_G^{\tilde{x}}$ is the set of all $x' \in [[n, G, N]]$ where, for every $x'|_i \neq (\perp, \dots, \perp)$, we have $x'|_i = \tilde{x}|_i$.) Parse $K = \left(\beta, \eta, \left\{ \begin{pmatrix} d_i[1,0] & \dots, & d_i[n,0] \\ d_i[1,1] & & d_i[n,1] \end{pmatrix} \right\}_{i\in[N]}\right)$. Also, parse $x = (\tilde{x}_1, \dots, \tilde{x}_N)$.

- Compute $(B, E) \leftarrow (g^\beta, g_2^\eta)$;
- For every $i \in [N]$, sample $\alpha_i \leftarrow \mathbb{Z}_p$ and compute $(A_i[1], A_i[2], \dots, A_i[n-1]) = (g^{\alpha_i}, g^{\alpha_i^2}, \dots, g^{\alpha_i^{n-1}})$.
- Then, for every $i \in [N]$, for the positions specified by $\tilde{x}_i[1], \dots, \tilde{x}_i[n]$, for every $j \in [n]$, compute $c_i[j, \tilde{x}_i[j]]$ such that $d_i[j, \tilde{x}_i[j]] \leftarrow \alpha_i c_i[j, \tilde{x}_i[j]]$. We thus have that $(g^{d_i[1,\tilde{x}_i[1]]}, \dots, g^{d_i[n,\tilde{x}_i[n]]}) = (A_i[1]^{c_i[1,\tilde{x}_i[1]]}, \dots, A_i[1]^{c_i[n,\tilde{x}_i[n]]})$.
- Return $(K[2_G^{\tilde{x}}], \mathsf{state})$, where,
  $K[2_G^{\tilde{x}}] := \left(B, E, \{A_i[1], \dots, A_i[n]\}_{i\in[N]}, \left\{ \begin{matrix} c_i[1,\tilde{x}_i[1]] & \dots, & c_i[n,\tilde{x}_i[n]] \\ d_i[1,(1-\tilde{x}_i[1])] & & d_i[n,(1-\tilde{x}_i[n])] \end{matrix} \right\}_{i\in[N]}\right)$
  and $\mathsf{state} := (\mathsf{state}_1, \dots, \mathsf{state}_N)$, for all i, $\mathsf{state}_i := \alpha_i$.

Algorithm $\mathsf{PatchGen}(K, i, \tilde{x}_i, \mathsf{state}_i)$: $\mathsf{PatchGen}$ takes as input a key K, an index $i \in [N]$, $\tilde{x}_i \in \{0,1\}^n$, and $\mathsf{state}_i = \alpha_i \in \mathbb{Z}_p$. Parse $K = \left(\beta, \eta, \left\{ \begin{pmatrix} d_i[1,0] & \dots, & d_i[n,0] \\ d_i[1,1] & & d_i[n,1] \end{pmatrix} \right\}_{i\in[N]}\right)$.

- Compute $g_2^{\beta\alpha_i^n}$.
- Return $\mathsf{patch}(K, (i, \tilde{x}_i)) := g_2^{\beta\alpha_i^n}$.

Algorithm $\mathsf{Eval}(K[2^{\widetilde{x}}_G], x)$ / $\mathsf{Eval}((K[2^{\widetilde{x}}_G], \mathsf{patch}(K, (j, \widetilde{x}_j))), x)$: Eval takes two kinds of queries:

$\mathsf{Eval}(K[2^{\widetilde{x}}_G], x)$**:** One kind corresponds to evaluating the PRF output for inputs *not* in $2^{\widetilde{x}}_G$ using a key punctured at $2^{\widetilde{x}}_G$. More formally: For $\widetilde{x} \in (\{0,1\}^n)^N$, $\widetilde{x} = (\widetilde{x}_1, \ldots, \widetilde{x}_N)$, parse

$$K[2^{\widetilde{x}}_G] = \left(B, E, \{A_i[1], \ldots, A_i[n-1]\}_{i\in[N]}, \left\{ \begin{matrix} c_i[1,\tilde{x}_i[1]] \\ d_i[1,(1-\tilde{x}_i[1])] \end{matrix} \quad , \ldots , \quad \begin{matrix} c_i[n,\tilde{x}_i[n]] \\ d_i[n,(1-\tilde{x}_i[n])] \end{matrix} \right\}_{i\in[N]} \right).$$

Now consider an input $x \notin 2^{\widetilde{x}}_G$, where, $x = (\mathcal{S}, (x_i)_{i\in\mathcal{S}})$; we have that there exists $j \in \mathcal{S}$ such that $x_j \neq \widetilde{x}_j$. On input $(K[2^{\widetilde{x}}_G], x)$, proceed as follows.

- For every $i \in \mathcal{S} \setminus \{j\}$, compute $\mathsf{Prod}_i \in \mathbb{G}_n$ as follows:
  - If $x_i \neq \widetilde{x}_i$, then $\phi(\widetilde{x}_i, x_i) < n$, $\phi(\widetilde{x}_i, x_i)$ is the number of bit-positions at which strings $\widetilde{x}_i, x_i$ agree. Under this case, compute

$$\begin{aligned} \mathsf{Prod}_i &= (\mathrm{e}(g, A_i[\phi(\widetilde{x}_i, x_i)]))^{\prod_{k \,:\, x_i[k]=\widetilde{x}_i[k]} c_i[k,(\widetilde{x}_i[k])] \prod_{k' \,:\, x_i[k']=1-\widetilde{x}_i[k']} d_i[k',(1-\widetilde{x}_i[k'])]} \\ &= g_2^{(d_i[1,x_i[1]]\cdots d_i[n,x_i[n]])} \end{aligned}$$

  - If $x_i = \widetilde{x}_i$, then $\phi(\widetilde{x}_i, x_i) = n$. Under this case, compute

$$\begin{aligned} \mathsf{Prod}_i &= (\mathrm{e}(A_i[1], A_i[n-1]))^{\prod_k c_i[k,(\widetilde{x}_i[k])]} \\ &= g_2^{(d_i[1,x_i[1]]\cdots d_i[n,x_i[n]])} \end{aligned}$$

- Since $x_j \neq \widetilde{x}_j$, we have that $\phi(\widetilde{x}_j, x_j) < n$. Compute $\mathsf{Prod}_j \in \mathbb{G}_n$ as follows.

$$\begin{aligned} \mathsf{Prod}_j &= (\mathrm{e}(B, A_i[\phi(\widetilde{x}_i, x_i)]))^{\prod_{k \,:\, x_i[k]=\widetilde{x}_i[k]} c_i[k,(\widetilde{x}_i[k])] \prod_{k' \,:\, x_j[k']=1-\widetilde{x}_j[k']} d_j[k',(1-\widetilde{x}_j[k'])]} \\ &= g_2^{\beta(d_j[1,x_j[1]]\cdots d_j[n,x_j[n]])} \end{aligned}$$

- Return

$$\begin{aligned} \mathsf{Eval}(K[2^{\widetilde{x}}_G], x) &:= \mathrm{e}(\{\mathsf{Prod}_i\}_{i\in[\mathcal{S}]}, \{E\}_{(G-|\mathcal{S}|) \text{ times}}) \\ &= g_{2G}^{\left(\beta \prod_{i\in\mathcal{S}} (d_i[1,x_i[1]]\cdots d_i[n,x_i[n]])\right)\eta^{(G-|\mathcal{S}|)}} \end{aligned} \tag{9}$$

$\mathsf{Eval}((K[2^{\widetilde{x}}_G], \mathsf{patch}(K, (j, \widetilde{x}_j))), x)$**:** The other kind corresponds to evaluating the PRF output for inputs in $2^{\widetilde{x}}_G$ using a key punctured at $2^{\widetilde{x}}_G$ and using a patch. More formally: For $\widetilde{x} \in (\{0,1\}^n)^N$, $\widetilde{x} = (\widetilde{x}_1, \ldots, \widetilde{x}_N)$, parse

$$K[2^{\widetilde{x}}_G] = \left(B, E, \{A_i[1], \ldots, A_i[n-1]\}_{i\in[N]}, \left\{ \begin{matrix} c_i[1,\tilde{x}_i[1]] \\ d_i[1,(1-\tilde{x}_i[1])] \end{matrix} \quad , \ldots , \quad \begin{matrix} c_i[n,\tilde{x}_i[n]] \\ d_i[n,(1-\tilde{x}_i[n])] \end{matrix} \right\}_{i\in[N]} \right)$$

and $\mathsf{patch}(K, (j, \tilde{x}_j)) = g_2^{\beta\alpha_j^n}$. Now consider an input $x \in 2^{\widetilde{x}}_G$, where, $x = (\mathcal{S}, (x_i)_{i\in\mathcal{S}})$; we have that for every $i \in \mathcal{S}$, $x_i = \widetilde{x}_i$. On input $((K[2^{\widetilde{x}}_G], \mathsf{patch}(K, (j, \widetilde{x}_j))), x)$ for some $j \in \mathcal{S}$ proceed as follows.

- For every $i \in \mathcal{S} \setminus \{j\}$, compute $\mathsf{Prod}_i \in \mathbb{G}_n$ as follows (Recall that since $x_i = \widetilde{x}_i$, $\phi(\widetilde{x}_i, x_i) = n$):

$$\begin{aligned} \mathsf{Prod}_i &= (\mathrm{e}(A_i[1], A_i[n-1]))^{\prod_k c_i[k,(\widetilde{x}_i[k])]} \\ &= g_2^{(d_i[1,x_i[1]]\cdots d_i[n,x_i[n]])} \end{aligned}$$

- Recall that $\mathsf{patch}(K,(j,\tilde{x}_j)) = g_2^{\beta\alpha_j^n}$. Define $\mathsf{Prod}_j := \mathsf{patch}(K,(j,\widetilde{x}_j))$.
- Return

$$\mathsf{Eval}((K[2^{\widetilde{x}}_G], \mathsf{patch}(K,(j,\widetilde{x}_j))), x) := \mathrm{e}(\{\mathsf{Prod}_i\}_{i\in[\mathcal{S}]}, \{E\}_{(G-|\mathcal{S}|) \text{ times})}) = g_{2G}^{\left(\beta \prod_{i\in\mathcal{S}} (d_i[1,x_i[1]]\cdots d_i[n,x_i[n]])\right)\eta^{(G-|\mathcal{S}|)}} \quad (10)$$

Algorithm $\mathsf{OPatchGen}(1^\lambda, \mathsf{F.params})$: $\mathsf{OPatchGen}$ takes as input group parameters $\mathsf{F.params} = (\overrightarrow{\mathbb{G}}, g, p, \mathrm{e}_{i,j})$ and outputs a set of patches and some state information as follows.

- Sample $\beta \leftarrow \mathbb{Z}_p$, and compute $B = g^\beta$.
- For every $i \in [N]$, sample $\alpha_i \leftarrow \mathbb{Z}_p$ and compute $(A_i[1], A_i[2], \ldots, A_i[n-1]) = (g^{\alpha_i}, g^{\alpha_i^2}, \ldots, g^{\alpha_i^{n-1}})$.
- Define $\mathsf{o.state} := (B, \{A_i[1], \ldots, A_i[n-1]\}_{i\in[N]})$.
- For every $i \in [N]$, define $\mathsf{patch}_i := g_2^{\beta\alpha_i^n}$.
- Return $(\{\mathsf{patch}_i\}_{i\in[N]}, \mathsf{o.state})$.

Algorithm $\mathsf{OPuncture}(1^\lambda, \mathsf{o.state}, x)$: $\mathsf{OPuncture}$ takes as inputs the security parameter, the state information output by $\mathsf{OPatchGen}$ and $\widetilde{x} \in (\{0,1\}^n)^N$. Parse $\mathsf{o.state} = (B, \{A_i[1], \ldots, A_i[n-1]\}_{i\in[N]})$ and $\widetilde{x} = (\widetilde{x}_1, \ldots, \widetilde{x}_N)$. It outputs a punctured key at $2^{\widetilde{x}}_G$ consistent with $\mathsf{o.state}$ as follows.

- Sample $\eta \leftarrow \mathbb{Z}_p$, and compute $E = g_2^\eta$.
- For the positions specified by $\widetilde{x}_i[1], \ldots, \widetilde{x}_i[n]$, sample $(c_i[1,\tilde{x}_i[1]], \ldots, c_n[1,\tilde{x}_i[n]]) \leftarrow \mathbb{Z}_p^n$ and $(d_i[1,(1-\widetilde{x}_i[1])], \ldots, d_i[n,(1-\widetilde{x}_i[n])]) \leftarrow \mathbb{Z}_p^n$.
- Return

$$K[2^{\widetilde{x}}_G] := \left(B, E, \{A_i[1], \ldots, A_i[n-1]\}_{i\in[N]}, \left\{ \begin{matrix} c_i[1,\tilde{x}_i[1]] \\ d_i[1,(1-\tilde{x}_i[1])] \end{matrix} \quad , \ldots , \quad \begin{matrix} c_i[n,\tilde{x}_i[n]] \\ d_i[n,(1-\tilde{x}_i[n])] \end{matrix} \right\}_{i\in[N]}\right).$$

Theorem 6. Let the One-more Weak (n,G,N)-Multilinear Decisional Diffie-Hellman Inversion ((n,G,N)-1wMDDHI) Assumption hold for GrpGen. Then F is an obliviously-patchable puncturable PRF.

Proof. We shall show that F satisfies all the properties specified in Definition 9.

By simple analyzing of the description of the associated algorithms, it is clear that F satisfies the following properties : Functionality preserved under puncturing, block-wise patchable, obliviously patchable.

Succinct patches. Observe that each patch is just a single group element. Since the multilinearity of the graded encoding schemes we work with is $2G$, the size of the group elements is only a function of λ, G. In particular, the size of the group elements is independent of n, as required.

Pseudorandom at punctured points. We shall establish this property under the assumption that (n, G, N)-1wMDDH Assumption hold for GrpGen. Assume for contradiction that there exists an adversary $\mathcal{A}$ for which $\mathsf{Adv}_{F,\mathcal{A}}(\lambda) = \varepsilon(\lambda)$ that is non-negligible. Then we construct an adversary $\mathcal{B}$ that breaks the security of F also with a non-negligible probability.

$\mathcal{B}$, upon receiving $(\mathsf{grpparams}, B, E, \{A_i[1], \ldots, A_i[n-1]\}_{i\in[N]})$ from its own challenger, proceeds as follows.

- $\mathcal{B}$ invokes $\mathcal{A}$ with grpparams.
- $\mathcal{A}$ then gives $\tilde{x} \in (\{0,1\}^n)^N$ and expects to receive a key punctured at $2^{\tilde{x}}_G$.
- $\mathcal{B}$ then executes the following step: For the positions specified by $\widetilde{x}_i[1], \ldots, \widetilde{x}_i[n]$, sample $(c_i[1, \tilde{x}_i[1]], \ldots, c_n[1, \tilde{x}_i[n]]) \leftarrow \mathbb{Z}_p^n$ and $(d_i[1, (1 - \widetilde{x}_i[1])], \ldots, d_i[n, (1 - \widetilde{x}_i[n])]) \leftarrow \mathbb{Z}_p^n$.
- Give to $\mathcal{A}$,

$$K[2^{\widetilde{x}}_G] := \left(B, E, \{A_i[1], \ldots, A_i[n-1]\}_{i\in[N]}, \left\{ \begin{matrix} c_i[1, \tilde{x}_i[1]] \\ d_i[1, (1 - \tilde{x}_i[1])] \end{matrix} \quad , \ldots , \quad \begin{matrix} c_i[n, \tilde{x}_i[n]] \\ d_i[n, (1 - \tilde{x}_i[n])] \end{matrix} \right\}_{i\in[N]} \right).$$

- Answer to $\mathcal{A}$'s queries as follows.
  1. On query (PATCH-AT i), where $i \in [N]$, query the 1wMDDHI challenger with (ONE-MORE-AT i). Let the response received from by $\mathcal{B}$ be $\widetilde{\mathsf{Prod}}_i = g_2^{\beta\alpha_i^n}$. Respond to $\mathcal{A}$ via $\mathsf{patch}(K, (i, \tilde{x}_i)) := \widetilde{\mathsf{Prod}}_i$.
  2. On query (EVAL-AT $\mathcal{S}$), where $\mathcal{S} \subseteq [N]$ and $|\mathcal{S}| \leq G$, query the 1wMDDHI challenger with (COMBINE $\mathcal{S}$). Respond to $\mathcal{A}$ via the response received from by $\mathcal{B}$, namely, $g_{2G}^{(\beta \prod_{i\in\mathcal{S}} \alpha_i^n)\eta^{(G-|\mathcal{S}|)}}$.
  3. On query (CHAL-AT ChQ), where $\mathsf{ChQ} \subseteq [N]$ and $|\mathsf{ChQ}| \leq G$, query the 1wMDDHI challenger with (CHAL-AT ChQ). Let the response received from by $\mathcal{B}$ be $\widetilde{\mathsf{Prod}}_{\text{EXPT}}$. Respond to $\mathcal{A}$ via $\widetilde{\mathsf{Prod}}_{\text{EXPT}}$.

Analysis. We observe that if the experiment played by the 1wMDDHI challenger is the real experiment, namely $\text{REAL}_{\mathcal{A}}^{\mathsf{1wMDDHI}}$, then the view of $\mathcal{A}$ when run by $\mathcal{B}$ is identical to its view in the real experiment of pseudorandomness of punctured PRFs. On the other hand, if the experiment played by the 1wMDDHI challenger is the random experiment, namely $\text{RAND}_{\mathcal{A}}^{\mathsf{1wMDDHI}}$, then the view of $\mathcal{A}$ when run by $\mathcal{B}$ is identical to its view in the random experiment of pseudorandomness of punctured PRFs.

Thus, we have that,

$$\begin{aligned} \mathsf{Adv}_{\mathcal{B}}^{(n,G,N)\text{-1wMDDHI}}(\lambda) =& |\Pr[\text{REAL}_{\mathcal{B}}^{\mathsf{1wMDDHI}} \to 1] - \Pr[\text{RAND}_{\mathcal{B}}^{\mathsf{1wMDDHI}} \to 1]| \\ =& |\Pr[\text{REAL}_{F,\mathcal{A}}^{\mathsf{PRF}}(\lambda) \to 1] - \Pr[\text{RAND}_{F,\mathcal{A}}^{\mathsf{PRF}}(\lambda) \to 1]| \\ =& \varepsilon(\lambda), \end{aligned}$$

thus arriving at a contradiction. Hence, the lemma.

■

We define the One-more Weak (n, G, N)-Multilinear Decisional Diffie-Hellman Inversion ((n, G, N)-1wMDDHI) assumption as follows:

Assumption 3 (Weak One-more (n, G, N)-Multilinear Decisional Diffie-Hellman Inversion: (n, G, N)-1wMDDHI)**.** GrpGen is said to satisfy the One-more Weak (n, G, N)-Multilinear Decisional Diffie-Hellman ((n, G, N)-1wMDDHI) problem states that for every PPT adversary $\mathcal{A}$, the following is negligible:

$$\mathsf{Adv}_{\mathcal{A}}^{(n,G,N)\text{-}\mathsf{1wMDDHI}}(\lambda) := |\Pr[\text{REAL}_{\mathcal{A}}^{\mathsf{1wMDDHI}} \to 1] - \Pr[\text{RAND}_{\mathcal{A}}^{\mathsf{1wMDDHI}} \to 1]|$$

where, for $\text{EXPT} \in \{\text{REAL}, \text{RAND}\}$, $\text{EXPT}_{\mathcal{A}}^{\mathsf{1wMDDHI}}$ is defined as follows.

$$\boxed{\begin{array}{c}\textbf{Experiment } \text{EXPT}_{\mathcal{A}}^{\mathsf{1wMDDHI}}: \\ (\overrightarrow{\mathbb{G}}, g, p, \mathrm{e}_{i,j}) \leftarrow \mathsf{GrpGen}(1^\lambda, 2G) \\ \beta, \eta \leftarrow \mathbb{Z}_p \\ \text{Compute } \ (B, E) = (g^\beta, g_2^\eta) \\ \{\alpha_i\}_{i\in[N]} \leftarrow \mathbb{Z}_p \\ \text{Compute } \ \forall i \in [N],\ (A_i[1], A_i[2], \ldots, A_i[n-1]) = (g^{\alpha_i}, g^{\alpha_i^2}, \ldots, g^{\alpha_i^{n-1}}) \\ \leftarrow \mathcal{A}^{\mathcal{O}}(B, E, \{A_i[1], \ldots, A_i[n-1]\}_{i\in[N]}) \\ \text{Compute } \ \widetilde{\mathsf{Prod}}_{\text{REAL}} = g_{2G}^{(\beta \prod_{i\in\mathsf{ChQ}} \alpha_i^n)\eta^{(G-|\mathsf{ChQ}|)}} \\ \text{Sample } \ \widetilde{\mathsf{Prod}}_{\text{RAND}} \leftarrow \mathbb{G}_{2G} \\ \text{Output } \ \mathcal{A}(, \widetilde{\mathsf{Prod}}_{\text{EXPT}}) \end{array}}$$

where, the oracle $\mathcal{O}$ takes two kinds of inputs:

1. On input $i \in [N]$, $\mathcal{O}$ returns $\widetilde{\mathsf{Prod}}_i = g_2^{\beta\alpha_i^n}$. Let Q_1 denote the set of such queries queried by $\mathcal{A}$ to the oracle.
2. On input $\mathcal{S} \in [N]$ for $|\mathcal{S}| \leq G$, $\mathcal{O}$ returns $g_{2G}^{(\beta \prod_{i\in\mathsf{ChQ}} \alpha_i^n)\eta^{(G-|\mathcal{S}|)}}$. Let Q_2 denote the set of such queries queried by $\mathcal{A}$ to the oracle.

It is required that $|\mathsf{ChQ}| \leq G$, $\mathsf{ChQ} \subseteq [N] \setminus Q_1$, and $\mathsf{ChQ} \notin Q_2$.

D Generic Security of Our Assumptions

D.1 Generic Security of the One-more Weak (n, G, N)-MDDH Assumption

In this section, we discuss the security of our One-more Weak (n, G, N)-MDDH assumption, defined in Assumption 1, in the generic multilinear group model. In particular, we explain why our assumption is secure in the generic model, provided p is sufficiently large.

Generic Multilinear Maps. Generic multilinear maps are a generalization of the generic group model. Roughly speaking, in the generic bilinear group model, elements of the groups appear to be encoded as arbitrary unique strings, so that no property other than equality can be directly tested by the adversary. For instance, like in our proof, the representation uses random-looking strings. Furthermore, the adversary performs operations on group elements by interacting with various oracles to that perform either multiplying two elements of the same group or pairing of two elements. Details follow.

We represent the group elements using a random injective function $\Phi : \mathbb{Z}_p \times [nG] \leftarrow \{0,1\}^\ell$. More precisely, an encoding of an element $x \in \mathbb{Z}_p$ at level i (that is, in group $\mathbb{G}_i$) is given by $\Phi(x, i)$, where, $m = \log(p)$. We are given oracles Mult and Pair to compute the induced multiplication and

pairing[6] operations. More precisely, any algorithm in the generic multilinear map model interacts with the multilinear map using the following queries:

$\mathsf{Encode}(x, i)$**:** For $x \in \mathbb{Z}_p$ and $i \in [nG]$, respond via $\Phi(x, i)$. Note that we can recover the generator g_i for the group $\mathbb{G}_i$ as $\mathsf{Encode}(1, i)$.

$\mathsf{Mult}(\zeta_{x_1}, \zeta_{x_2}, b)$**:** If $\zeta_{x_1} = \Phi(x_1, i_1)$ and $\zeta_{x_2} = \Phi(x_2, i_2)$, where $i_1 = i_2 = i$, then return $\Phi(x_1 + (-1)^b x_2, i)$. Otherwise, return $\perp$.

$\mathsf{Pair}(\zeta_{x_1}, \zeta_{x_2})$**:** If $\zeta_{x_1} = \Phi(x_1, i_1)$ and $\zeta_{x_2} = \Phi(x_2, i_2)$, where $i_1 + i_2 = i \leq nG$, then return $\Phi(x_1 \cdot x_2, i)$. Otherwise, return $\perp$.

Generic security of our assumption.

Theorem 7. For any generic adversary $\mathcal{A}$ whose total number of queries to $\mathsf{Encode}, \mathsf{Mult}, \mathsf{Pair}$ is polynomial in λ, $\mathcal{A}$ has negligible advantage in breaking the One-more Weak (n, G, N)-MDDH assumption, provided $1/p$ is negligible in λ.

Proof. Let $\mathcal{A}$ be a generic attacker. $\mathcal{A}$ plays the following game:

- The challenger flips a coin uniformly at random and chooses whether to play the either the real or the random experiment: $\text{EXPT} \in \{\text{REAL}, \text{RAND}\}$.
- The challenger also samples $\beta, \eta \leftarrow \mathbb{Z}_p$, and for every $i \in [N]$, $\{d_i[1], \ldots, d_i[n]\}_{i \in [N]} \leftarrow \mathbb{Z}_p$.
- $\mathcal{A}$ receives $\Phi(x, 1)$ for every $x \in \{\beta, \{d_i[1], \ldots, d_i[n]\}_{i \in [N]}\}$. $\mathcal{A}$ also receives $\Phi(\eta, n)$.
- $\mathcal{A}$ can adaptively query the One-more wMDDH oracle $\mathcal{O}$ with two kinds of queries: If $\mathcal{A}$ queries a singleton $\{i\}$, then it receives $\Phi(\beta(d_i[1] \cdots d_i[n]), n)$. On the other hand, if $\mathcal{A}$ queries $\mathcal{S} \subseteq [N]$ for $2 \leq |\mathcal{S}| \leq G$, then, it receives $\Phi((\beta \prod_{i \in \mathcal{S}} (d_i[1] \cdots d_i[n]))\eta^{(G - |\mathcal{S}|)}, nG)$. Let Q denote the set of queries made by $\mathcal{A}$ to the One-more wMDDH oracle.
- $\mathcal{A}$ can also adaptively make queries to the oracles, $\mathsf{Encode}, \mathsf{Mult}, \mathsf{Pair}$.
- $\mathcal{A}$ makes a challenge query on a set $\mathsf{ChQ} \subseteq [N]$, subject to the restriction that $|\mathsf{ChQ}| \leq G$, for every $\{i\} \in Q$, $i \notin \mathsf{ChQ}$, and for every $\mathcal{S} \in Q$, $\mathcal{S} \neq \mathsf{ChQ}$. In response, $\mathcal{A}$ receives $\Phi(\widetilde{\mathsf{prod}}_{\text{EXPT}}, nG)$, where, $\widetilde{\mathsf{prod}}_{\text{REAL}} = (\beta \prod_{i \in \mathsf{ChQ}} (d_i[1] \cdots d_i[n]))\eta^{(G - |\mathsf{ChQ}|)}$ and $\widetilde{\mathsf{prod}}_{\text{RAND}} \leftarrow \mathbb{Z}_p$.
- $\mathcal{A}$ can continue making queries to the One-more wMDDH oracle and to $\mathsf{Encode}, \mathsf{Mult}, \mathsf{Pair}$ oracles.
- $\mathcal{A}$ guesses which experiment it is in by outputting a bit b^*.

The advantage of $\mathcal{A}$ in the above game is $|\Pr[\mathcal{A} \to 1 \mid \text{EXPT} = \text{REAL}] - \Pr[\mathcal{A} \to 1 \mid \text{EXPT} = \text{RAND}]|$. Our goal is to show that this is negligible.

Now consider an algorithm $\mathcal{B}$ that plays the above game with $\mathcal{A}$. Rather than choosing values for $\beta, \eta, d_i[1], \ldots, d_i[n], \widetilde{\mathsf{prod}}_{\text{RAND}}$, algorithm $\mathcal{B}$ treats them as formal variables. $\mathcal{B}$ maintains a list

$$L = \{(\text{P}_j, i_j, \zeta_j)\},$$

[6] We allow the adversary to successively pair elements together, rather than only providing the full multilinear map. This reflects the structure of current map candidates.

where, P_j is a polynomial in the variables $\beta, \eta, d_i[1], \ldots, d_i[n]\}_{i\in[N]}$, the integer i_j indexes the groups, and ζ_j is a string in $\{0,1\}^\ell$. The list is initialized with the following tuples:

- $(\mathrm{P}_x, 1, \zeta_x)$, for every $x \in \{\beta, \{d_i[1], \ldots, d_i[n]\}_{i\in[N]}\}$ for randomly generated strings ζ_x,
- (η, n, ζ_η).

Initially, L contains $nG + 2$ entries.

The game starts with $\mathcal{B}$ giving $\mathcal{A}$ the tuple of strings $\{\zeta_x\}_{x\in L}$. Now, $\mathcal{A}$ is allowed to make the following queries:

$\mathsf{Encode}(x, i)$**:** If $x \in \mathbb{Z}_p$ and $i \in [nG]$, then $\mathcal{B}$ looks for a tuple (P, i, ζ), where P is the constant polynomial equal to x. If such a tuple exists, then $\mathcal{B}$ responds via ζ. Otherwise, $\mathcal{B}$ generates a random string $\zeta \in \{0,1\}^\ell$, adds the tuple (P, i, ζ) (again, where P is the constant polynomial equal to x) to L, and responds via ζ.

$\mathsf{Mult}(\zeta_j, \zeta_k, b)$**:** $\mathcal{B}$ looks for tuples $(\mathrm{P}_j, i_j, \zeta_j)$ and $(\mathrm{P}_k, i_k, \zeta_k)$ in L. If either tuple does not exist, then $\mathcal{B}$ responds via $\perp$. If both tuples are found, but $i_j \neq i_k$, then $\mathcal{B}$ responds via $\perp$. Otherwise, $\mathcal{B}$ lets $i := i_j = i_k$, computes the polynomial $\mathrm{P} = \mathrm{P}_j + (-1)^b \mathrm{P}_k$, and looks for a tuple $(\mathrm{P}, i, \zeta) \in L$. If the tuple is found, then $\mathcal{B}$ responds via ζ. Otherwise, $\mathcal{B}$ generates a random string $\zeta \in \{0,1\}^\ell$, adds the tuple (P, i, ζ) to L, and responds via ζ.

$\mathsf{Pair}(\zeta_j, \zeta_k)$**:** $\mathcal{B}$ looks for tuples $(\mathrm{P}_j, i_j, \zeta_j)$ and $(\mathrm{P}_k, i_k, \zeta_k)$ in L. If either tuple does not exist, then $\mathcal{B}$ responds via $\perp$. If both tuples are found, but $i := i_j + i_k > nG$, then $\mathcal{B}$ responds via $\perp$. Otherwise, $\mathcal{B}$ lets $i := i_j + i_k$, computes the polynomial $\mathrm{P} = \mathrm{P}_j \cdot \mathrm{P}_k$, and looks for a tuple $(\mathrm{P}, i, \zeta) \in L$. If the tuple is found, then $\mathcal{B}$ responds via ζ. Otherwise, $\mathcal{B}$ generates a random string $\zeta \in \{0,1\}^\ell$, adds the tuple (P, i, ζ) to L, and responds via ζ.

$\mathcal{O}(\mathcal{S})$**:** If $\mathcal{S} = \{i\}$, $\mathcal{B}$ creates a new formal variable $\beta(d_i[1] \cdots d_i[n])$ and adds the tuple $((\beta(d_i[1] \cdots d_i[n])), n, \zeta)$ to L for a randomly generated $\zeta \in \{0,1\}^\ell$, and gives $\mathcal{A}$ the string ζ. On the other hand, if $2 \leq |\mathcal{S}| \leq G$, then $\mathcal{B}$ creates a new formal variable $((\beta \prod_{i\in\mathcal{S}} (d_i[1] \cdots d_i[n]))\eta^{(G-|\mathcal{S}|)})$. It adds the tuple $((\beta \prod_{i\in\mathcal{S}} (d_i[1] \cdots d_i[n])\eta^{(G-|\mathcal{S}|)}), nG, \zeta)$ to L for a randomly generated $\zeta \in \{0,1\}^\ell$, and gives $\mathcal{A}$ the string ζ.

Challenge: On input a challenge subset $\mathsf{ChQ} \subseteq [N]$, $\mathcal{B}$ creates a new formal variable y^* and adds the tuple (y^*, nG, ζ) to L for a randomly generated $\zeta \in \{0,1\}^\ell$ and gives $\mathcal{A}$ the string ζ.

After a polynomial number of queries, $\mathcal{A}$ ultimately produces a guess b^*. Now, $\mathcal{B}$ chooses uniformly at random $\mathrm{EXPT} \in \{\mathrm{REAL}, \mathrm{RAND}\}$, as well as values for $\beta, \eta, d_i[1], \ldots, d_i[n]$. Furthermore, it also chooses a random value $\widetilde{\mathsf{prod}}_{\mathrm{RAND}} \leftarrow \mathbb{Z}_p$.

$\mathcal{B}$ can increase m arbitrarily, thus making strings ζ hard to guess. Therefore, we can assume without loss of generality that $\mathcal{A}$ only makes Mult and Pair queries on strings obtained from $\mathcal{B}$.

The simulation provided by $\mathcal{B}$ is perfect unless our choices for the variables $\beta, \eta, d_i[1], \ldots, d_i[n], y^*$ results in an equality between values for two values $\mathrm{P}_j, \mathrm{P}_k$ that is not an equality for polynomials. More precisely the simulation is perfect unless for some j, k the following hold:

- $i_j = i_k$,
- $(\mathrm{P}_j - \mathrm{P}_k)(\beta, \eta, d_1[1], \ldots) = 0$, yet the polynomials $\mathrm{P}_j, \mathrm{P}_k$ are not equal.

Let FAIL be the event that these conditions hold for some j, k. We need to bound the probability that FAIL occurs. First, prior to choosing values for all the variables, consider setting

$$y^* = (\beta \prod_{i \in \mathsf{ChQ}} (d_i[1] \cdots d_i[n]))\eta^{(G-|\mathsf{ChQ}|)} \tag{11}$$

as polynomials. We claim that this does not create any new polynomial equalities.

Claim 1. Substituting the formal variable y^* with the polynomial $(\beta \prod_{i \in \mathsf{ChQ}} (d_i[1] \cdots d_i[n]))\eta^{(G-|\mathsf{ChQ}|)}$ does not create any new polynomial equalities. That is, if $\mathrm{P}_j \neq \mathrm{P}_k$ before the substitution, the same is true after the substitution too.

Before proving Claim 2, we use it to complete the proof of Theorem 8. Notice that each of the polynomials has degree at most nG. The Swartz-Zipel lemma then shows that if $\mathrm{P}_j - \mathrm{P}_k \neq 0$, the probability that the polynomial $(\mathrm{P}_j - \mathrm{P}_k)(\beta, \eta, d_1[1], \ldots)$ evaluates to zero is at most $nG/(p-1)$. This means that $\mathrm{P}_j, \mathrm{P}_k$ evaluate to the same value with probability at most $nG/(p-1)$.

Now, let $q_e, q_m, q_p, q_{\mathcal{O}}$ be the total number of $\mathsf{Encode}, \mathsf{Mult}, \mathsf{Pair}, \mathcal{O}$ queries made by $\mathcal{A}$. Then the total length of L is at most

$$|L| \leq q_e + q_m + q_p + q_{\mathcal{O}} + nG + 2.$$

Therefore, the number of pairs is at most

$$\binom{|L|}{2} \leq \frac{(q_e + q_m + q_p + q_{\mathcal{O}} + nG + 2)^2}{2}.$$

Therefore, FAIL happens with probability at most

$$\frac{(q_e + q_m + q_p + q_{\mathcal{O}} + nG + 2)^2 \cdot nG}{2(p-1)}$$

For polynomials in λ, $q_e, q_m, q_p, q_{\mathcal{O}}, G, n, N$, this is negligible in λ, provided $1/p$ is negligible, as desired. It remains to prove Claim 2. Suppose there are two polynomials $\mathrm{P}_j, \mathrm{P}_k$ such that, when we replace the variable y^* with $\widetilde{\mathsf{prod}}_{\mathrm{REAL}} = (\beta \prod_{i \in \mathsf{ChQ}} (d_i[1] \cdots d_i[n]))\eta^{(G-|\mathsf{ChQ}|)}$. This means $\mathrm{P}_j - \mathrm{P}_k = 0$. Consider expanding $\mathrm{P}_j - \mathrm{P}_k$ out into monomials prior to the substitution. First, this expansion must contain a y^* term, and this term cannot have been multiplied by other variables (since polynomials involving y^* can only exist in the target group $\mathbb{G}_{nG}$). Therefore, we can write $\mathrm{P}_j - \mathrm{P}_k$ as

$$\begin{aligned}
\mathrm{P}_j - \mathrm{P}_k =& c y^* && (12)\\
&+ \mathrm{poly}_1\left(\beta, \{d_i[1], \ldots, d_i[n]\}_{i \in [N]}\right) && (13)\\
&+ \sum_{\mathcal{S} \in Q} \mathrm{c}_{2,\mathcal{S}} \left(\left(\beta \prod_{i \in \mathcal{S}} (d_i[1] \cdots d_i[n]) \right) \eta^{(G-|\mathcal{S}|)} \right) && (14)\\
&+ \sum_{\{i\} \in Q} (\beta d_i[1] \cdots d_i[n]) \begin{bmatrix} \mathrm{poly}_{3,i,1}\left(\{(\beta d_{i'}[1] \cdots d_{i'}[n])\}_{\{i'\} \in Q}\right) \\ \mathrm{poly}_{3,i,2}\left(\beta, \{d_i[1], \ldots, d_i[n]\}_{i' \in [N]}\right) \\ \mathrm{poly}_{3,i,3}(\eta) \end{bmatrix} && (15)\\
&+ \mathrm{poly}_4(\eta) \begin{bmatrix} \mathrm{poly}_{4,i,1}\left(\{(\beta d_{i'}[1] \cdots d_{i'}[n])\}_{\{i'\} \in Q}\right) \\ \mathrm{poly}_{4,i,2}\left(\beta, \{d_i[1], \ldots, d_i[n]\}_{i \in [N]}\right) \end{bmatrix} && (16)
\end{aligned}$$

where,

- $\mathrm{c}, \mathrm{c}_{2,\mathcal{S}}$ are constants,
- poly_1 has degree nG,
- $\mathrm{poly}_{3,i,1}, \mathrm{poly}_{3,i,2}, \mathrm{poly}_{3,i,3}$ have degrees $\deg_{3,i,1}, \deg_{3,i,2}, \deg_{3,i,3}$, such that $n\deg_{3,i,1} + \deg_{3,i,2} + n\deg_{3,i,3} = nG - n$,
- poly_4, $\mathrm{poly}_{4,i,1}$, $\mathrm{poly}_{4,i,2}$ have degrees $\deg_4, \deg_{4,i,1}, \deg_{4,i,2}$, respectively, such that $n\deg_4 + n\deg_{4,i,1} + \deg_{4,i,2} = nG$.

$\mathrm{c}, \mathrm{c}_{2,\mathcal{S}}$ are constants, poly_1 has degree nG, $\mathrm{poly}_{3,i,1}$ has degree $0 \leq \deg_{3,i} \leq G-1$ and $\mathrm{poly}_{3,i,2}$ has degree $(G - (\deg_{3,i} + 1))$. If the polynomial $\mathrm{P}_j - \mathrm{P}_k$ is non-zero, but substituting y^* as $(\beta \prod_{i \in \mathsf{ChQ}} (d_i[1] \cdots d_i[n]))\eta^{(G-|\mathsf{ChQ}|)}$ makes it zero, we can conclude the following:

- $\mathrm{c} \neq 0$, and
- $\mathrm{c}' = -\mathrm{c}$, where, c' is the co-efficient of y^* in (Term (13)+Term (14)+Term (15)+Term (16)).

We shall now compute the coefficients contributed by each of Term (13), Term (14), Term (15), and Term (16) to y^*. Firstly, recall that the polynomial $y^* = \left(\beta \prod_{i \in \mathsf{ChQ}} (d_i[1] \cdots d_i[n])\right) \eta^{(G-|\mathsf{ChQ}|)}$ (as per Equation (17)). We begin by analyzing the ways in which this polynomial can be constructed in each of the Terms in question, one by one, below.

- Observe that none of the arguments of poly_1 contains η or $\eta^{(G-|\mathsf{ChQ}|)}$ as a factor. Hence, no monomial in poly_1 can be equal to y^*. Thus, the contribution to the co-efficient of y^* from Term (19) is 0.
- Recall that, for every $\mathcal{S} \in Q$, $\mathcal{S} \neq \mathsf{ChQ}$. Thus, for every $\mathcal{S} \in Q$, $\left(\left(\beta \prod_{i \in \mathcal{S}} (d_i[1] \cdots d_i[n])\right) \eta^{(G-|\mathcal{S}|)}\right) \neq y^* = \left(\beta \prod_{i \in \mathsf{ChQ}} (d_i[1] \cdots d_i[n])\right) \eta^{(G-|\mathsf{ChQ}|)}$, thus implying that the contribution to the co-efficient of y^* from Term (14) also is 0.
- Observe that every term in Term (15) contains $(\beta d_i[1] \cdots d_i[n])$ for some $\{i\} \in Q$. However, for every $\{i\} \in Q$, $i \notin \mathsf{ChQ}$. Thus, the contribution to the co-efficient of y^* from Term (15) also is 0.
- Observe the only way to get $\prod_{i \in \mathsf{ChQ}} (d_i[1] \cdots d_i[n])$ from the arguments of poly_1 is to multiply each of $(\{d_i[1], \ldots, d_i[n]\}_{i \in \mathsf{ChQ}})$ amounting to a degree of $n|\mathsf{ChQ}|$. Furthermore, the only way to get $\eta^{(G-|\mathsf{ChQ}|)}$ from the arguments of poly_1 is to multiply η in the argument $(G - |\mathsf{ChQ}|)$ times itself amounting to a degree of $(G - |\mathsf{ChQ}|)$. Hence, by now, the total degree 'used up' is nG, which is the degree of poly_1. However, β is still not factored in and the only way to factor it in is to multiply it with the current partial product raising the degree to $nG+1$. This is not possible since that would exceed the maximum degree of poly_1. Thus, the contribution to the co-efficient of y^* from Term (13) is 0.
- Let us begin with $\mathrm{poly}_{4,i,1}$. Observe that every argument in $\mathrm{poly}_{4,i,1}$ contains $(\beta d_i[1] \cdots d_i[n])$ for some $\{i'\} \in Q$. However, for every $\{i'\} \in Q$, $i' \notin \mathsf{ChQ}$; in other words, for every $\{i'\} \in Q$,

$d_{i'}[j]$ is not a factor in y^*. Thus, if $\deg_{4,i,1} \neq 0$, then including a factor of such an $d_{i'}[j]$ would entail 'canceling' it out. However, there is no variable with $1/d_{i'}[j]$ as a factor. Hence, we can conclude that $\deg_{4,i,1} = 0$.

Now, observe that in Term (16), only poly_4 contains η as a factor. Since $(G - |\mathsf{ChQ}|)$ is the degree of η in y^*, the only way to get $\eta^{(G-|\mathsf{ChQ}|)}$ is by multiplying η, $(G - |\mathsf{ChQ}|)$ times. Thus, $\deg_4 \geq (G - |\mathsf{ChQ}|)$.

Observe that none of the arguments of $\text{poly}_{4,i,2}$ contains $d_i[j]d_i[j']$ as a factor for any j, j'. Hence, for every $i \in \mathsf{ChQ}$, $(d_i[1] \cdots d_i[n])$ can be obtained only by multiplying the variables $d_i[1], \ldots, d_i[n]$. This amounts to already using up n degrees of $\text{poly}_{4,i,2}$ for every $i \in \mathsf{ChQ}$. That is, $\deg_{4,i,2} \geq n|\mathsf{ChQ}|$. Adding up, in order to obtain $\left(\prod_{i \in \mathsf{ChQ}} (\beta d_i[1] \cdots d_i[n])\right) \eta^{(G-|\mathsf{ChQ}|)}$, we have that $\deg_4 \geq (G - |\mathsf{ChQ}|)$ and $\deg_{4,i,2} \geq |\mathsf{ChQ}|$. However, we need to work with the constraint that $n\deg_4 + n\deg_{4,i,1} + \deg_{4,i,2} = nG$. Thus, we have used all of the available nG degrees. However, we have not yet taken into account the β factor, but there is no room left to take this factor into account. Thus, the contribution to the co-efficient of y^* from Term (22) is 0.

In conclusion, the monomial y^* did not exist in (Term (13)+Term (14)+Term (15)+Term (16)) before the substitution. Thus, the monomial is not existent in the sum even after the substitution. Thus, even after substitution, we are left with the monomial cy^* in the polynomial $\text{P}_j - \text{P}_k$, making it non-zero, thus leading to a contradiction.

■

D.2 Generic Security of the One-more Weak (n, G, N)-MDDHI Assumption

In this section, we discuss the security of our One-more Weak (n, G, N)-MDDHI assumption, defined in Assumption 3, in the generic multilinear group model.

Theorem 8. For any generic adversary $\mathcal{A}$ whose total number of queries to $\mathsf{Encode}, \mathsf{Mult}, \mathsf{Pair}$ is polynomial in λ, $\mathcal{A}$ has negligible advantage in breaking the One-more Weak (n, G, N)-MDDHI assumption, provided $1/p$ is negligible in λ.

Proof. Let $\mathcal{A}$ be a generic attacker. $\mathcal{A}$ plays the following game:

- The challenger flips a coin uniformly at random and chooses whether to play the either the real or the random experiment: $\text{EXPT} \in \{\text{REAL}, \text{RAND}\}$.
- The challenger also samples $\beta, \eta \leftarrow \mathbb{Z}_p$, and for every $i \in [N]$, $\{\alpha_i\}_{i \in [N]} \leftarrow \mathbb{Z}_p$.
- $\mathcal{A}$ receives $\Phi(x, 1)$ for every $x \in \{\beta, \{\alpha_i, \alpha_i^2, \ldots, \alpha_i^{n-1}\}_{i \in [N]}\}$. $\mathcal{A}$ also receives $\Phi(\eta, 2)$.
- $\mathcal{A}$ can adaptively query the One-more wMDDHI oracle $\mathcal{O}$ with two kinds of queries: If $\mathcal{A}$ queries a singleton $\{i\}$, then it receives $\Phi(\beta\alpha_i^n, 2)$. On the other hand, if $\mathcal{A}$ queries $\mathcal{S} \subseteq [N]$ for $2 \leq |\mathcal{S}| \leq G$, then, it receives $\Phi(((\beta \prod_{i \in \mathsf{ChQ}} \alpha_i^n)\eta^{(G-|\mathcal{S}|)}), 2G)$. Let Q denote the set of queries made by $\mathcal{A}$ to the One-more wMDDHI oracle.
- $\mathcal{A}$ can also adaptively make queries to the oracles, $\mathsf{Encode}, \mathsf{Mult}, \mathsf{Pair}$.

- $\mathcal{A}$ makes a challenge query on a set $\mathsf{ChQ} \subseteq [N]$, subject to the restriction that $|\mathsf{ChQ}| \leq G$, for every $\{i\} \in Q$, $i \notin \mathsf{ChQ}$, and for every $\mathcal{S} \in Q$, $\mathcal{S} \neq \mathsf{ChQ}$. In response, $\mathcal{A}$ receives $\Phi(\widetilde{\mathsf{prod}}_{\text{EXPT}}, 2G)$, where, $\widetilde{\mathsf{prod}}_{\text{REAL}} = (\beta \prod_{i \in \mathsf{ChQ}} \alpha_i^n)\eta^{(G-|\mathsf{ChQ}|)})$ and $\widetilde{\mathsf{prod}}_{\text{RAND}} \leftarrow \mathbb{Z}_p$.
- $\mathcal{A}$ can continue making queries to the One-more wMDDHI oracle and to $\mathsf{Encode}, \mathsf{Mult}, \mathsf{Pair}$ oracles.
- $\mathcal{A}$ guesses which experiment it is in by outputting a bit b^*.

The advantage of $\mathcal{A}$ in the above game is $|\Pr[\mathcal{A} \to 1 \mid \text{EXPT} = \text{REAL}] - \Pr[\mathcal{A} \to 1 \mid \text{EXPT} = \text{RAND}]|$. Our goal is to show that this is negligible.

Now consider an algorithm $\mathcal{B}$ that plays the above game with $\mathcal{A}$. Rather than choosing values for $\beta, \eta, \alpha_i, \widetilde{\mathsf{prod}}_{\text{RAND}}$, algorithm $\mathcal{B}$ treats them as formal variables. $\mathcal{B}$ maintains a list

$$L = \{(\text{P}_j, i_j, \zeta_j)\},$$

where, P_j is a polynomial in the variables $\beta, \eta, \alpha_i, \alpha_i^2, \ldots, \alpha_i^{n-1}$, the integer i_j indexes the groups, and ζ_j is a string in $\{0,1\}^\ell$. The list is initialized with the following tuples:

- $(\text{P}_x, 1, \zeta_x)$, for every $x \in \{\beta, \{\alpha_i, \alpha_i^2, \ldots, \alpha_i^{n-1}\}_{i \in [N]}\}$ for randomly generated strings ζ_x,
- $(\eta, 2, \zeta_\eta)$

Initially, L contains $nN + 2$ entries.

The game starts with $\mathcal{B}$ giving $\mathcal{A}$ the tuple of strings $\{\zeta_x\}_{x \in L}$. Now, $\mathcal{A}$ is allowed to make the following queries:

$\mathsf{Encode}(x, i)$**:** If $x \in \mathbb{Z}_p$ and $i \in [2G]$, then $\mathcal{B}$ looks for a tuple (P, i, ζ), where P is the constant polynomial equal to x. If such a tuple exists, then $\mathcal{B}$ responds via ζ. Otherwise, $\mathcal{B}$ generates a random string $\zeta \in \{0,1\}^\ell$, adds the tuple (P, i, ζ) (again, where P is the constant polynomial equal to x) to L, and responds via ζ.

$\mathsf{Mult}(\zeta_j, \zeta_k, b)$**:** $\mathcal{B}$ looks for tuples $(\text{P}_j, i_j, \zeta_j)$ and $(\text{P}_k, i_k, \zeta_k)$ in L. If either tuple does not exist, then $\mathcal{B}$ responds via $\perp$. If both tuples are found, but $i_j \neq i_k$, then $\mathcal{B}$ responds via $\perp$. Otherwise, $\mathcal{B}$ lets $i := i_j = i_k$, computes the polynomial $\text{P} = \text{P}_j + (-1)^b \text{P}_k$, and looks for a tuple $(\text{P}, i, \zeta) \in L$. If the tuple is found, then $\mathcal{B}$ responds via ζ. Otherwise, $\mathcal{B}$ generates a random string $\zeta \in \{0,1\}^\ell$, adds the tuple (P, i, ζ) to L, and responds via ζ.

$\mathsf{Pair}(\zeta_j, \zeta_k)$**:** $\mathcal{B}$ looks for tuples $(\text{P}_j, i_j, \zeta_j)$ and $(\text{P}_k, i_k, \zeta_k)$ in L. If either tuple does not exist, then $\mathcal{B}$ responds via $\perp$. If both tuples are found, but $i := i_j + i_k > nN$, then $\mathcal{B}$ responds via $\perp$. Otherwise, $\mathcal{B}$ lets $i := i_j + i_k$, computes the polynomial $\text{P} = \text{P}_j \cdot \text{P}_k$, and looks for a tuple $(\text{P}, i, \zeta) \in L$. If the tuple is found, then $\mathcal{B}$ responds via ζ. Otherwise, $\mathcal{B}$ generates a random string $\zeta \in \{0,1\}^\ell$, adds the tuple (P, i, ζ) to L, and responds via ζ.

$\mathcal{O}(\mathcal{S})$**:** If $\mathcal{S} = \{i\}$, $\mathcal{B}$ creates a new formal variable $(\beta\alpha_i^n)$ and adds the tuple $((\beta\alpha_i^n), 2, \zeta)$ to L for a randomly generated $\zeta \in \{0,1\}^\ell$, and gives $\mathcal{A}$ the string ζ. On the other hand, if $2 \leq |\mathcal{S}| \leq G$, then $\mathcal{B}$ creates a new formal variable $((\beta \prod_{i \in \mathsf{ChQ}} \alpha_i^n)\eta^{(G-|\mathcal{S}|)})$. It adds the tuple $(((\beta \prod_{i \in \mathsf{ChQ}} \alpha_i^n)\eta^{(G-|\mathcal{S}|)}), 2G, \zeta)$ to L for a randomly generated $\zeta \in \{0,1\}^\ell$, and gives $\mathcal{A}$ the string ζ.

Challenge: On input a challenge subset $\mathsf{ChQ} \subseteq [N]$, $\mathcal{B}$ creates a new formal variable y^* and adds the tuple $(y^*, 2G, \zeta)$ to L for a randomly generated $\zeta \in \{0,1\}^\ell$ and gives $\mathcal{A}$ the string ζ.

After a polynomial number of queries, $\mathcal{A}$ ultimately produces a guess b^*. Now, $\mathcal{B}$ chooses uniformly at random $\text{EXPT} \in \{\text{REAL}, \text{RAND}\}$, as well as values for β, η, α_i. Furthermore, it also chooses a random value $\widetilde{\mathsf{prod}}_{\text{RAND}} \leftarrow \mathbb{Z}_p$.

$\mathcal{B}$ can increase ℓ arbitrarily, thus making strings ζ hard to guess. Therefore, we can assume without loss of generality that $\mathcal{A}$ only makes Mult and Pair queries on strings obtained from $\mathcal{B}$.

The simulation provided by $\mathcal{B}$ is perfect unless our choices for the variables $\beta, \eta, \alpha_i, y^*$ results in an equality between values for two values P_j, P_k that is not an equality for polynomials. More precisely the simulation is perfect unless for some j, k the following hold:

- $i_j = i_k$,
- $(\text{P}_j - \text{P}_k)(\beta, \eta, \alpha_i, \alpha_i^2, \ldots) = 0$, yet the polynomials P_j, P_k are not equal.

Let FAIL be the event that these conditions hold for some j, k. We need to bound the probability that FAIL occurs. First, prior to choosing values for all the variables, consider setting

$$y^* = \left(\beta \prod_{i \in \mathsf{ChQ}} \alpha_i^n\right) \eta^{(G - |\mathsf{ChQ}|)} \tag{17}$$

as polynomials. We claim that this does not create any new polynomial equalities.

Claim 2. Substituting the formal variable y^* with the polynomial $(\beta \prod_{i \in \mathsf{ChQ}} \alpha_i^n)\eta^{(G-|\mathsf{ChQ}|)}$ does not create any new polynomial equalities. That is, if $\text{P}_j \neq \text{P}_k$ before the substitution, the same is true after the substitution too.

Before proving Claim 2, we use it to complete the proof of Theorem 8. Notice that each of the polynomials has degree at most $2G$. The Swartz-Zipel lemma then shows that if $\text{P}_j - \text{P}_k \neq 0$, the probability that the polynomial $(\text{P}_j - \text{P}_k)(\beta, \eta, \alpha_i, \alpha_i^2, \ldots)$ evaluates to zero is at most $2G/(p-1)$. This means that P_j, P_k evaluate to the same value with probability at most $2G/(p-1)$.

Now, let $q_e, q_m, q_p, q_{\mathcal{O}}$ be the total number of $\mathsf{Encode}, \mathsf{Mult}, \mathsf{Pair}, \mathcal{O}$ queries made by $\mathcal{A}$. Then the total length of L is at most

$$|L| \leq q_e + q_m + q_p + q_{\mathcal{O}} + nN + 2.$$

Therefore, the number of pairs is at most

$$\binom{|L|}{2} \leq \frac{(q_e + q_m + q_p + q_{\mathcal{O}} + nN + 2)^2}{2}.$$

Therefore, FAIL happens with probability at most

$$\frac{(q_e + q_m + q_p + q_{\mathcal{O}} + nN + 2)^2 \cdot 2G}{2(p-1)}$$

For polynomials in λ, $q_e, q_m, q_p, q_{\mathcal{O}}, G, n, N$, this is negligible in λ, provided $1/p$ is negligible, as desired. It remains to prove Claim 2. Suppose there are two polynomials P_j, P_k such that, when we replace the variable y^* with $\widetilde{\mathsf{prod}}_{\text{REAL}} = (\beta \prod_{i \in \mathsf{ChQ}} \alpha_i^n)\eta^{(G-|\mathsf{ChQ}|)}$. This means $\text{P}_j - \text{P}_k = 0$. Consider expanding $\text{P}_j - \text{P}_k$ out into monomials prior to the substitution. First, this expansion must contain a

y^* term, and this term cannot have been multiplied by other variables (since polynomials involving y^* can only exist in the target group $\mathbb{G}_{2G}$). Therefore, we can write $\mathrm{P}_j - \mathrm{P}_k$ as

$$\mathrm{P}_j - \mathrm{P}_k = \mathrm{c}y^* \tag{18}$$

$$+ \mathrm{poly}_1\left(\beta, \left\{\alpha_i, \alpha_i^2, \ldots, \alpha_i^{n-1}\right\}_{i\in[N]}\right) \tag{19}$$

$$+ \sum_{\mathcal{S}\in Q} \mathrm{c}_{2,\mathcal{S}} \left(\left(\beta \prod_{i\in\mathcal{S}} \alpha_i^n\right) \eta^{(G-|\mathcal{S}|)}\right) \tag{20}$$

$$+ \sum_{\{i\}\in Q} (\beta\alpha_i^n) \begin{bmatrix} \mathrm{poly}_{3,i,1}\left(\{(\beta\alpha_{i'}^n)\}_{\{i'\}\in Q}\right) \\ \mathrm{poly}_{3,i,2}\left(\beta, \left\{\alpha_{i'}, \alpha_{i'}^2, \ldots, \alpha_{i'}^{n-1}\right\}_{i'\in[N]}\right) \\ \mathrm{poly}_{3,i,3}(\eta) \end{bmatrix} \tag{21}$$

$$+ \mathrm{poly}_4(\eta) \begin{bmatrix} \mathrm{poly}_{4,i,1}\left(\{(\beta\alpha_{i'}^n)\}_{\{i'\}\in Q}\right) \\ \mathrm{poly}_{4,i,2}\left(\beta, \left\{\alpha_{i'}, \alpha_{i'}^2, \ldots, \alpha_{i'}^{n-1}\right\}_{i'\in[N]}\right) \end{bmatrix} \tag{22}$$

where,

- $\mathrm{c}, \mathrm{c}_{2,\mathcal{S}}$ are constants,
- poly_1 has degree $2G$,
- $\mathrm{poly}_{3,i,1}$, $\mathrm{poly}_{3,i,2}$, $\mathrm{poly}_{3,i,3}$ have degrees $\deg_{3,i,1}, \deg_{3,i,2}, \deg_{3,i,3}$, such that $2\deg_{3,i,1} + \deg_{3,i,2} + 2\deg_{3,i,3} = 2G - 2$,
- poly_4, $\mathrm{poly}_{4,i,1}$, $\mathrm{poly}_{4,i,2}$ have degrees $\deg_4, \deg_{4,i,1}, \deg_{4,i,2}$, respectively, such that $2\deg_4 + 2\deg_{4,i,1} + \deg_{4,i,2} = 2G$.

If the polynomial $\mathrm{P}_j - \mathrm{P}_k$ is non-zero, but substituting y^* as $(\beta \prod_{i\in\mathsf{ChQ}} \alpha_i^n)\eta^{(G-|\mathsf{ChQ}|)}$ makes it zero, we can conclude the following:

- $\mathrm{c} \neq 0$, and
- $\mathrm{c}' = -\mathrm{c}$, where, c' is the co-efficient of y^* in (Term (19)+Term (20)+Term (21)+Term (22)).

We shall now compute the coefficients contributed by each of Term (19), Term (20), Term (21), Term (22) to y^*. Firstly, recall that the polynomial $y^* = \left(\beta \prod_{i\in\mathsf{ChQ}} \alpha_i^n\right) \eta^{(G-|\mathsf{ChQ}|)}$ (as per Equation (17)). We begin by analyzing the ways in which this polynomial can be constructed in each of the Terms in question, one by one, below.

- Observe that none of the arguments of poly_1 contains η or $\eta^{(G-|\mathsf{ChQ}|)}$ as a factor. Hence, no monomial in poly_1 can be equal to y^*. Thus, the contribution to the co-efficient of y^* from Term (19) is 0.
- Recall that, for every $\mathcal{S} \in Q$, $\mathcal{S} \neq \mathsf{ChQ}$. Thus, for every $\mathcal{S} \in Q$, $\left(\beta \prod_{i\in\mathcal{S}} \alpha_i^n\right) \eta^{(G-|\mathcal{S}|)} \neq y^* = \left(\beta \prod_{i\in\mathsf{ChQ}} \alpha_i^n\right) \eta^{(G-|\mathsf{ChQ}|)}$, thus implying that the contribution to the co-efficient of y^* from Term (20) also is 0.

- Observe that every term in Term (20) contains $(\beta\alpha_i^n)$ for some $\{i\} \in Q$. However, for every $\{i\} \in Q$, $i \notin \mathsf{ChQ}$. Thus, the contribution to the co-efficient of y^* from Term (21) also is 0.

- Let us begin with $\text{poly}_{4,i,1}$. Observe that every argument in $\text{poly}_{4,i,1}$ contains $(\beta\alpha_{i'}^n)$ for some $\{i'\} \in Q$. However, for every $\{i'\} \in Q$, $i' \notin \mathsf{ChQ}$; in other words, for every $\{i'\} \in Q$, $\alpha_{i'}$ is not a factor in y^*. Thus, if $\deg_{4,i,1} \neq 0$, then including a factor of such an $\alpha_{i'}$ would entail 'canceling' it out. However, there is no variable with $1/\alpha_{i'}$ as a factor. Hence, we can conclude that $\deg_{4,i,1} = 0$.

  Now, observe that in Term (22), only poly_4 contains η as a factor. Since $(G - |\mathsf{ChQ}|)$ is the degree of η in y^*, the only way to get $\eta^{(G-|\mathsf{ChQ}|)}$ is by multiplying η, $(G - |\mathsf{ChQ}|)$ times. Thus, $\deg_4 \geq (G - |\mathsf{ChQ}|)$.

  Observe that none of the arguments of $\text{poly}_{4,i,2}$ contains $\alpha_i\alpha_j$ as a factor for any $i \neq j$. Hence, $\prod_{i\in\mathsf{ChQ}} \alpha_i^n$ can be obtained only by multiplying variables that contain α_i^n as a factor. This amounts to already using up $|\mathsf{ChQ}|$ degree of $\text{poly}_{4,i,2}$. Now consider any $i \in \mathsf{ChQ}$. In order to obtain α_i^n, since no argument contains α_i^n, the only way to obtain it is by multiplying two variables, say α_i and α_i^{n-1}, thus contributing degree 2 per $i \in \mathsf{ChQ}$. Thus, to obtain $\prod_{i\in\mathsf{ChQ}} \alpha_i^n$ itself, one needs to spend the $2|\mathsf{ChQ}|$ of the available degrees. That is, $\deg_{4,i,2} \geq 2|\mathsf{ChQ}|$. Adding up, in order to obtain $\left(\prod_{i\in\mathsf{ChQ}} \alpha_i^n\right)\eta^{(G-|\mathsf{ChQ}|)}$, we have that $\deg_4 \geq (G - |\mathsf{ChQ}|)$ and $\deg_{4,i,2} \geq |\mathsf{ChQ}|$. However, we need to work with the constraint that $2\deg_4 + 2\deg_{4,i,1} + \deg_{4,i,2} = 2G$. Thus, we have used all of the available $2G$ degrees. However, we have not yet taken into account the β factor, but there is no room left to take this factor into account. Thus, the contribution to the co-efficient of y^* from Term (22) is 0.

In conclusion, the monomial y^* did not exist in (Term (19)+Term (20)+Term (21)+Term (22)) before the substitution. Thus, the monomial is not existent in the sum even after the substitution. Thus, even after substitution, we are left with the monomial cy^* in the polynomial $\text{P}_j - \text{P}_k$, making it non-zero, thus leading to a contradiction.

■