Interactive Proofs under Continual Memory Leakage

Prabhanjan Ananth[*]
University of California, Los Angeles, USA
prabhanjan@cs.ucla.edu

Vipul Goyal
Microsoft Research India
vipul@microsoft.com

Omkant Pandey[*]
University of Illinois at Urbana Champaign, USA
omkant@uiuc.edu

Abstract

We consider the task of constructing interactive proofs for NP which can provide meaningful security for a prover even in the presence of continual memory leakage. We imagine a setting where an adversarial verifier participates in multiple sequential interactive proof executions for a fixed NP statement x. In every execution, the adversarial verifier is additionally allowed to leak a fraction of the (secret) memory of the prover. This is in contrast to the recently introduced notion of leakage-resilient zero-knowledge (Garg-Jain-Sahai'11) where there is only a single execution. Under multiple executions, in fact the entire prover witness might end up getting leaked thus leading to a complete compromise of prover security.

Towards that end, we define the notion of non-transferable proofs for all languages in NP. In such proofs, instead of receiving w as input, the prover will receive an "encoding" of the witness w such that the encoding is sufficient to prove the validity of x; further, this encoding can be "updated" to a fresh new encoding for the next execution. We then require that if (x, w) are sampled from a "hard" distribution, then no PPT adversary A^* can gain the ability to prove x (on its own) to an honest verifier, even if A^* has participated in polynomially many interactive proof executions (with leakage) with an honest prover whose input is (x, w). Non-transferability is a strong security guarantee which suffices for many cryptographic applications (and in particular, implies witness hiding).

We show how to construct non-transferable proofs for all languages in NP which can tolerate leaking a constant fraction of prover's secret-state during each execution. Our construction is in the *common reference string* (CRS) model. To obtain our results, we build a witness-encoding scheme which satisfies the following continual-leakage-resilient (CLR) properties:

- The encodings can be randomized to yield a fresh new encoding,
- There does not exist any efficient adversary, who receiving only a constant fraction of leakage on polynomially many fresh encodings of the same witness w, can output a valid encoding provided that the witness w along with its corresponding input instance x were sampled from a hard distribution.

Our encoding schemes are essentially re-randomizable non-interactive zero-knowledge (NIZK) proofs for circuit satisfiability, with the aforementioned CLR properties. We believe that our CLR-encodings, as well as our techniques to build them, may be of independent interest.

[*]Work done at Microsoft Research, India.

Contents

1 Introduction

Traditionally, when defining security of a cryptographic primitive, the adversary is allowed to interact with the underlying cryptographic algorithms only in a black-box manner. Emergence of side channel attacks [QS01, BCS08, BS97, KJJ99, HSH$^+$09, Koc96] has shown that enforcing only black-box access may be difficult in many practical settings. Such attacks exploit the physical characteristics of a cryptographic device, such as the time and electrical power taken by the device, to learn useful information about its secrets. This information is often sufficient to "break" the system completely.

Leakage resilient cryptography [DP08] focuses on the algorithmic aspects of this problem, by developing theoretical paradigms and security notions which can deliver meaningful security under such attacks. In the last few years, we have seen many exciting developments in this direction, resulting in interesting attack models for leakage, as well as cryptographic primitives that guard against leakage [ISW03, DP08, AGV09, NS09, Pie09, DP10, FKPR10, GR10, LRW11, LLW11].

Continual Memory Leakage. To model leakage, one typically allows the adversary to submit leakage queries in the form of efficiently computable functions f, and provide it with $f(st)$, where st is the internal state st of the system during execution. The class of functions f and the length of their output determines the kind and the amount of leakage tolerated by the system. Initial works focused on the *bounded leakage* model which requires that the total amount of leakage throughout the life time of the system is a priori bounded. Construction of several cryptographic primitives such as public-key encryption, identity based encryption, signatures schemes, stream ciphers etc. are known in this model [KV09, BKKV10, ADN$^+$10, CDRW10, ADVW13].

However, enforcing an a priori bound on the total leakage is somewhat unreasonable since usually it is not known how many times the system will be used. The emerging standard for leakage attacks, therefore, seems to be the *continual memory leakage* model. In this attack model, the secret state st of the cryptographic system is "updated" after each time-period *without changing its public parameters* (if any). The adversary is allowed queries f as before—we only require that the amount of leakage between any two *successive* updates is bounded. This a very natural and powerful attack model, allowing the adversary to learn an unbounded overall leakage. Many basic cryptographic primitives such as public-key encryption, signatures schemes, identity-based encryption etc. are now known in this model [DHLAW10, JV10, BKKV10, BGJK12]. With few exceptions, almost all results in the continual leakage model have focused on *non-interactive* tasks such as public-key encryption.

Interactive Protocols under Leakage attacks. Modeling meaningful security for interactive protocols in the presence of leakage is a more delicate task. Very recent works focusing on zero-knowledge [GJS11, Pan14] and multi-party computation [BCH12, BGJK12] have now emerged. While leakage-resilient zero-knowledge (LRZK) protocols of [GJS11, Pan14] do not put a bound on total leakage, the system still does not protect against *continual* leakage. Indeed, the LRZK notion as defined in [GJS11, BCH12] do not model updating a witness; the notion becomes meaningless as soon as the entire witness is leaked. In this work, we will focus on providing meaningful security for the prover in face of *continual* leakage.

Encoding-based Proofs under Continual Leakage. The goal of this paper is to construct interactive protocols in the continual leakage model for arbitrary NP relations. The setting for our interactive protocol, defined for an NP relation, is the following. There exists a prover, who has an instance and a witness that the instance belongs to that relation. The prover executes polynomially (in the security parameter) many times and in each execution he convinces a verifier (which can be different for different executions) that he has a valid witness corresponding to that instance without actually revealing the witness. Now that we have defined the model, we need to come up with a meaningful notion of security that will help us apply this in many practical scenarios especially identification schemes and witness hiding schemes which are of primary interest for us.

1. The first thing to observe here, irrespective of the security notion we consider, is that the prover cannot use the same witness in all the executions because by obtaining leakage on the witness bit-by-bit, the verifier can obtain the entire witness. To this end, we need a refreshing mechanism that updates the witnesses in between executions.

2. The first candidate security definition for our setting is the standard simulation based notion. This says that there exists a PPT simulator who can simulate (entirely on its own) the transcript of conversation between the prover and the verifier as well as simulate the leakage queries made by the verifier. However similar to an argument in [GJS11], we observe that it is unlikely that such a security notion can be satisfied in our setting even if we allow a preprocessing phase [1]. An informal argument to rule out the simulation based notion in the preprocessing model for single execution leakage can be found in [GJS11] (and the same argument applies directly for multiple executions with continual leakage as well). At a high level, the problem occurs when the verifier submits a leakage function that tests whether the memory state of the prover has a valid witness (or an encoding of the witness) or not. In this case, the simulator has no way of answering such a function query since the function query may be "obfuscated" in some sense. Hence, the simulator doesn't know what the right answer to such a query would be (the function may output a pre-encoded value on getting a valid witness which the simulator doesn't know). We refer the reader to [GJS11] for more details.

3. Garg et. al. [GJS11] overcame the above problem by giving more capability to the simulator. Their simulator is given access to a leakage oracle containing the witness. The security requirement is that the simulator, with access to a leakage oracle on the witness, should be able to output a transcript which is indistinguishable from the one obtained by the interaction between the honest prover and the verifier. One can try to adopt such a security notion to our setting as well. In each execution, the simulator is given access to a leakage oracle holding the witness. However note that this becomes meaningless under multiple executions. This is because, under multiple executions, the entire witness can be leaked from the leakage oracle by the simulator!

4. Next option we consider is a variant of the above model where in each execution, the simulator is given access to a leakage oracle containing *an encoding of a witness.* In between executions, there is an update phase which refreshes the encodings. While this is a step towards our final definition, this does not quite give us anything meaningful yet. The reason is that there could be a protocol which satisfies this definition yet an adversarial verifier could obtain a valid encoded witness, by possibly combining leakages on many encodings obtained during its interaction with the prover! Thus, one needs to be more specific about the properties this witness encoding scheme satisfies.

5. From the above observation it follows that any security definition we consider should have the following property – given leakage on many encodings, an efficient adversary should not be able to compute a valid encoding. This is indeed the starting point of our work.

6. We introduce the security notion that we study in the paper. Suppose an efficient adversary after receiving leakage across many executions with the honest prover is able to convince an honest verifier, with non-negligible probability, that he has a valid encoding, then, we should be able to extract the encoding from the adversary with non-negligible probability. Not only does this definition imply that given leakage on many encodings, computing a valid encoding is hard but our definition is in fact stronger. This is because our definition rules out the possibility of obtaining even a partial encoding from the leakage on many encodings, using which membership of the instance in the language can be proven. We term proof systems that satisfy this definition as *non transferable proof systems* (NTP). We consider this as a simplified and a more direct way of obtaining the guarantees that notion in the previous bullet had (where we talk about obtaining leakage on encodings in the ideal world).

[1]In a preprocessing phase, a witness can be preprocessed before the execution of the protocol begins.

Our main result is to achieve the notion outlined above for all of NP in the common reference string (CRS) model. Achieving this notion in the plain model is left as an interesting open problem (and as we discuss later, even in the CRS model, achieving this notion already turns out to be quite intricate and non-trivial).

Note that most of the above discussions can be translated to the CRS setting with the exception of *simulatability of encodings* (see bullet 2). In other words, it is still possible to have a standard simulation based notion where the encodings are completely simulated. Towards that end, consider the following protocol. The prover receives a non interactive zero knowledge proof that the instance belongs to that language and then it sends the same proof to the verifier in every execution. Thus, this protocol is zero-knowledge for any polynomial number of executions even when the entire state of the prover is leaked in every execution. However, it is not clear how meaningful this protocol is: the adversary gets to learn a valid encoding of the witness which it can later use on its own. This is no different from the scenario where the prover gives out the witness to the verifier. Indeed, it cannot be used in the applications like identification schemes that we are interested in.

We believe that requiring the adversary to not be able to compute a valid encoding (which would allow him to prove on his own) is indeed a "right" security notion in the setting of interactive proofs with continuous leakage (indeed, there may be others notions). If the adversary can obtain the same witness (or the witness encoding) as being used by the prover, there is very little left in terms of the security guarantees that can be satisfied. Indeed, our notion of NTP is a formalization of this "intuition": not only it ensures that the adversary cannot get any valid witness encoding, but also, that it cannot compute any "partial" witness encoding which still allows him to complete the proof on his own.

To summarize, the main question that we are concerned with in this work is the following:

Do non-transferable proofs, under (non-trivial) continual memory leakage, exist for NP?

Our Results. In this work, we construct non-transferable CLR proof systems for all languages in NP assuming DDH is hard in pairing-based proofs (called eXternal Diffie Hellman assumption). Our results are in the CRS model. We obtain our result by constructing the so called *CLR-encodings* for all NP languages, and then use them to construct non-transferable (interactive) proofs for all of NP. We work in the CRS model; recall that, as we argued above, non-transferable proofs are non-trivial in the CRS model as well. In our continual leakage model, we do not allow leakage during the update phase. Further, we follow [GJS11, Pan14] and require that the randomness of a particular round of the protocol is not available to the prover until that round begins.[2]

To construct CLR-encodings, we first construct *re-randomizable* NIZK-proofs *for all NP-languages.* Prior to our work, such proofs were known only for specific, number-theoretic, relations. In addition, our re-randomizable NIZK proofs satisfy the stronger property of "controlled malleability": roughly, it means that the only way to construct new valid proofs is to either re-reandomize given proofs or create them from scratch (using the knowledge of an appropriate witness). These are of independent interest.

Finally, we find that the CLR-encodings are a cryptographic primitive of independent interest as well. We show how to use them to build other cryptographic primitives secure under continual memory leakage. In particular, we show how to construct continual leakage resistant public key encryption schemes, identity based encryption schemes as well as attribute based encryption schemes (for polynomial sized attribute space) using CLR-encodings and the recent new primitive of *witness encryption* [GGSW13] (see full version for more details). We hope that this will inspire future work, and our CLR-encoding primitive will be useful in obtaining more primitives resilient against continual leakage attacks. Even though some of these applications already exist under standard assumptions, ours is the first work that describes a generic tool that can be used to construct many continual-leakage-resilient primitives at once.

[2]It might be possible to avoid this model and allow leakage on all randomness *from the start*; however, it is usually difficult to ensure without making further assumptions on the statements. For example, in the context of LRZK, it might be impossible to support such leakage because a cheating verifier can first obtain a "short" commitment to the witness *and* randomness, and later try to check if the transcript is consistent with this state.

A brief overview of our approach. The starting point of our construction is the observation that we can use the "proof of a proof" approach which was used in [DSY90]. That is, instead of receiving a witness w as input, the prover will receive a non-interactive proof π that $x \in L$. If such a proof is sound, then the prover can prove to the verifier that "there exists a convincing non-interactive proof π for $x \in L$." Therefore, π acts as a different kind of witness for x, or, as an "encoding of w." A natural choice for such proofs is NIZK-proofs (which exist for all NP languages). These proofs have strong security properties, e.g., an adversary cannot decode w from a NIZK-proof for x (unless x is easy).

We start by constructing re-randomizable NIZKs for the NP-complete language of circuit satisfiability. However, re-randomization does not guarantee much as it might be possible to use leakage on many such NIZK-proofs and "stitch them together" to obtain a valid proof . To avoid this, we turn to the notion of *non-malleability.* More specifically, we require the following two properties:

- First, we require that each proof must have sufficient min-entropy of its own, say $\ell + \omega(\log n)$, where n is the security parameter. This guarantees that ℓ bits of leakage do not suffice to predict this proof exactly. In turn, given ℓ bits of leakage from polynomially many independent proofs will not suffice to guess *any* of those proofs exactly. This step guarantees some protection against leakage information theoretically.

- However, this leakage might still suffice to compute a *new* proof (that is different from all other proofs so far). To counter this, we require that each proof must be *simulation-extractable* [Sah99, SCO$^+$01]. This would essentially mean that the adversary cannot compute new proofs unless he knows a witness for x. But this is a bit too strong since it means that the proofs cannot even be re-randomized! Therefore, we need a "controlled" form of simulation extractability; one such notion is CM-SSE [CKLM12] but this falls short of what we need.

We formulate an appropriate notion called *decomposable NIZK with controlled malleability.* These are NIZK-proofs which can be re-randomized, and essentially "that is all" an adversary can do with such proofs. We construct such proofs for circuit satisfiability. To construct such proofs, at a very high level, we show how to combine the re-randomizable garbled circuits [GHV10] with malleable proof-systems of Groth and Sahai [GS08, CKLM12].

Prior Work. A series of works [QS01, BCS08, BS97, KJJ99, HSH$^+$09, Koc96], to name a few, studied a class of hardware-based attacks on various cryptosystems, which were later referred to as side channel attacks. Early theoretical works focused on increasingly sophisticated modeling of such attacks and protecting against them[AGV09, DP10, Pie09, ISW03], resulting in constructions for public key encryption schemes [NS09, LRW11], signature schemes [FKPR10], identity based encryption schemes [CDRW10] and so on. Early schemes in the bounded leakage model also appear in [ADN$^+$10, KV09, CDRW10]. Later several works focused on continual leakage [DHLAW10, JV10, BGJK12, GR10]. The works of [GJS11, BCH12, Pan14, BGJK12] focused on interactive protocols in presence of leakage. Dodis et. a. [DHLAW10] were the first to consider identification schemes in presence of continual leakage. To the best of our knowledge, ours is the first work to address the security of *general* interactive proofs in the presence of *continual leakage.*

The concept of non-transferrable proof systems appear in the works of [Jak95, JSI96, OPV08], as well as early works on identification schemes [FFS88, Sch91, GQ90, COSV12]. The idea of "proof of a proof" has appeared, in a very different context, in the work of [DSY90].

2 Notation and Definitions

We will use standard notation and assume familiarity with common cryptographic concepts. We use n as a security parameter. We say that a PPT sampling algorithm $\mathsf{Sampler}$ is a *hard distribution* over an NP relation R if $\mathsf{Sampler}$ outputs pairs $(x, w) \in R$, and for every PPT adversary A^* it holds that the probability, $\Pr[(x, w) \leftarrow \mathsf{Sampler}(1^n); w' \leftarrow A^*(x) \wedge (x, w') \in R]$ is negligible in n. For an NP-language L, we define a witness relation, denoted by R_L, such that $x \in L$ iff there exists w such that $(x, w) \in R_L$. We now define the notion of encoding schemes.

Encoding schemes. An encoding scheme offers a mechanism to not only encode the witnesses of an NP relation, say R_L, but also to refresh these encodings. Further, it can be verified whether x is in the language L or not by using the encodings of the witness(es) of x. Looking ahead, the prover in the continual leakage resistant system would not have the witness corresponding to the original relation. Instead he will possess the encoding of the witness, which he will consequently refresh in between executions. We now formally define the encoding scheme.

An encoding scheme for an NP-relation R_L, in the CRS model, is a tuple of PPT algorithms $\mathbb{E} = (\mathsf{PubGen}, \mathsf{Encode}, \mathsf{Update}, V_L)$ [3]. The PubGen takes as input a security parameter and outputs a string CRS. The algorithm Encode takes as input $(x, w) \in R_L$ as well as CRS and it outputs a fresh "encoded" witness, say $\tilde{w}$. The update algorithm takes as input $(x, \tilde{w})$ as well as CRS and outputs an encoded witness $\tilde{w}'$ which may be different from $\tilde{w}$. The requirement from the update algorithm is that the distributions $\{\mathsf{Encode}(x, w, \mathsf{CRS})\}$ and $\{\mathsf{Update}(x, \mathsf{Encode}(x, w, \mathsf{CRS}))\}$ are computationally indistinguishable. The verifier V_L on input $(\mathsf{CRS}, x, \tilde{w})$ either decides to accept or reject.

Any encoding scheme for a relation R_L needs to satisfy two properties, namely completeness and soundness. These properties are defined the same way they are defined in an interactive proof system [4]. Intuitively, except with negligible error it should happen that the verifier V_L accepts an encoding (generated using either Encode or Update) of w iff w is a valid witness.

Encoding based proofs. Let (P, V) be a pair of PPT interactive Turing machines, L an NP language and $\mathbb{E} = (\mathsf{PubGen}, \mathsf{Encode}, \mathsf{Update}, V_L)$, a refreshable encoding scheme for L. We say that $(P, V, \mathbb{E})$ is an encoding-based proof for L if the following two conditions hold: firstly, (P, V) is an interactive proof (or argument) for L; and second, the prover algorithm P gets as input $x \in L$ along with an encoded witness $\tilde{w}$ such that $\tilde{w} = \mathsf{Encode}(x, w, \mathsf{CRS})$ (where CRS is the output of PubGen algorithm.), where w is a witness such that $(x, w) \in R_L$. Further, the prover can participate in many executions and in between executions he refreshes his encoded witness using the Update procedure by executing Update on an encoded witness $\tilde{w}$ to obtain a new encoding $\tilde{w}'$. In this work, we consider encoding based proof systems which satisfy extractabilty property, which is stronger than soundness.

We can also consider encoding based proofs in the CRS model. In this case, there is a Setup algorithm in addition to the algorithms $(P, V, \mathbb{E})$. Note that $\mathbb{E}$ has its own CRS generation algorithm PubGen. For simplicity we assume that the encoding based proof has a single CRS generation algorithm, denoted by PubGen, that internally runs the CRS generation algorithms of both $\mathbb{E}$ as well as the encoding based proof system (P, V).

Continual leakage attacks on encoding based proofs. Let $l := l(n)$ be a leakage parameter used for bounding the maximum leakage (in number of bits) allowed during a single execution of the proof. Let $(P, V, \mathbb{E})$ be an encoding-based proof for $L \in NP$. From now on, we will denote by R_L a witness relation for L. Let A^* be a PPT algorithm, called the *leakage adversary* and let $(x, w) \in R_L$. An adversary A^*, taking the role of a verifier, on input a statement $x \in L$ (and some advice $z \in \{0, 1\}^*$), interacts with an honest prover P in polynomially many executions (or sessions). At the start of the first execution, the prover receives as auxiliary input an encoding of witness w, denoted by $\tilde{w}_1$. At the start of the i-th

[3] The corresponding scheme in the standard model will not have the CRS generation algorithm. That is, it will consist of $(\mathsf{Encode}, \mathsf{Update}, V_L)$

[4] *Completeness.* Let $(x, w) \in R_L$. And let CRS be the output of PubGen. Let $(x, \tilde{w}_i)$ be such that $\tilde{w}_i$ is either the output of (i) Encode on input (x, w, CRS) if $i = 1$ or (ii) it is the output of $\mathsf{Update}(x, \mathsf{CRS}, \tilde{w}_{i-1})$ for $i > 1$. Then, completeness property says that V_L accepts the input $(\mathsf{CRS}, x, \tilde{w})$ with probability 1.
Soundness. For every $x \notin L$, and every PPT algorithm P^*, and for sufficiently large n and advice $z \in \{0, 1\}^*$ we have that $\Pr[\mathsf{CRS} \leftarrow \mathsf{PubGen}(1^n);\ V_L(\mathsf{CRS}, x, P^*(x, \mathsf{CRS}, z)) = 1]$ is negligible in n.

session, the prover receives as its auxiliary input a "refreshed" encoding of the encoded witness, namely $\tilde{w}_i = \mathsf{Update}(x, \tilde{w}_{i-1})$, and fresh randomness. Following prior work on LRZK [GJS11, Pan14], we will assume that the randomness required by P to execute round j of any given session i is not sampled *until that round begins.*

During the execution of session i, A^* is allowed to make adaptively chosen leakage queries by submitting an efficient leakage function f_j at the beginning of round j of the proof (P, V). In response, the adversary is given the value of $f_j(st_j)$ where st_j denotes the state of the honest prover in round j; st_j consists of the refreshed witness $\tilde{w}_i$ and the randomness of the prover in session i *up to round* j denoted $(r_1^i, \ldots, r_j^i)$ for that session. It is required that the total leakage bits received by A^* in session i – i.e., the sum of lengths of outputs of queries sent *during session* i *only*—is at most l. We say that A^* launches a continual leakage attack on the system $(\mathsf{PubGen}, P, V, \mathbb{E})$.

In this work, we study continual-leakage-resilient encoding based proofs under a specific security notion called *non-transferability*, and we term all the encoding based proofs that satisfy this security definition as non transferable proof systems.

Definition 1 (Continual-Leakage-Resilient Non-transferable Proofs). Let $\Pi :=$ $(\mathsf{PubGen}, P, V, \mathbb{E})$ be an encoding-based proof in the CRS model (as defined above) for an encoding scheme $\mathbb{E} = (\mathsf{PubGen}, \mathsf{Encode}, \mathsf{Update}, V_L)$. We say that Π is a continuous-leakage-resilient *non-transferable proof* for L (CLR-NTP) with leakage-parameter $l(n)$, if for every PPT adversary Adv and every algorithm $\mathsf{Sampler}$ that is a *hard distribution* over R_L, and every advice string $z \in \{0,1\}^*$, the success probability of A^* in the following $\mathsf{NTPGame}$, taken over the randomness of the game, is negligible in n.

$\mathsf{NTPGame}(n, \mathsf{Sampler}, \Pi, \mathsf{Adv}, z)$**.** The game proceeds in the following steps:

1. **Initialize** Run $\mathsf{PubGen}(1^n)$ [5] to obtain a CRS, say ρ; then run $\mathsf{Sampler}(1^n)$ to obtain $(x, w) \in R_L$.
2. **Phase I:** Initiate the adversary algorithm Adv on inputs (x, z), who launches a continual-leakage attack on the encoding-based proof system $(P, V, \mathbb{E})$, as described earlier. The prover P starts by receiving an encoded witness obtained using Encode and this is refreshed (using Update) at the start of every new session. At some point, Adv signals the end of this phase, at which point the next phase begins.
3. **Phase II:** The adversary Adv attempts to prove that $x \in L$ to the honest verifier algorithm V. The verifier V receives fresh random coins in this (stand alone) session. The game ends when this session ends.
4. Adv wins the game if V accepts the proof.

In what follows, the leakage parameter $l(n)$ will often be implicit in the abbreviation CLR-NTP and clear from the context.

3 A Construction from CLR Encodings

In this section we describe a construction of non-transferable proof for all of NP. The construction will use encoding schemes (defined in Section 2) that are secure against continual leakage attacks and we term such encodings as CLR encodings. We first define CLR encodings and then assuming the existence of CLR encodings, we construct non-transferable proof systems. We defer the construction of the CLR encodings to the next section.

[5] Recall that PubGen includes the CRS generation algorithm of the proof system (P, V) as well as the CRS generation algorithm of the encoding scheme $\mathbb{E}$.

3.1 CLR Encodings for NP Languages

Continuous-Leakage-Resilient Encodings. Let $L \in NP$, $\mathbb{E} := (\mathsf{PubGen}, \mathsf{Encode}, \mathsf{Update}, V_L)$ an encoding scheme defined for language L, $l := l(n)$ a leakage parameter, and $\mathsf{Sampler}$ a sampling algorithm for the relation R_L. Informally, we require that for every hard distribution $\mathsf{Sampler}$ for R_L, no PPT adversary Adv receiving at most l bits of leakage from polynomially many valid encodings $\tilde{w}_1, \tilde{w}_2, \ldots, \tilde{w}_{p(n)}$, generated by either Encode or Update, where p is a polynomial chosen by adversary Adv, can output a valid encoding that will be accepted by V_L.

Formally, we consider the following $\mathsf{EncodingGame}$, parameterized by $l(n)$, played between a PPT adversary Adv and a challenger CH. At the start of the game, CH samples (x, w) by executing $\mathsf{Sampler}(1^n)$ and ρ by executing $\mathsf{PubGen}(1^n)$. It sends x to Adv and sets $\mathsf{private}_1 = \mathsf{Encode}(\rho, x, w)$. The game now proceeds in sessions and a session must end before a new fresh session can be started. In each session s, Adv is allowed to make adaptively chosen leakage queries $f_1, f_2, \ldots, f_m$ such that their total output length, that is the sum of the output lengths of $f_1, \ldots, f_m$, is at most l. It receives $f_1(\mathsf{private}_s), f_2(\mathsf{private}_s), \ldots, f_m(\mathsf{private}_s)$ in response. At the beginning of session $s+1$, the challenger sets $\mathsf{private}_{s+1} = \mathsf{Update}(x, \rho, \mathsf{private}_s)$. In the end, Adv halts by outputting an encoding π^*. Adv wins the EncodingGame if $V_L(\rho, x, \pi^*) = 1$.

We say that the scheme $\mathbb{E} := (\mathsf{PubGen}, \mathsf{Encode}, \mathsf{Update}, V_L)$ is a *CLR Encoding* with leakage-parameter $l(n)$ if for every $\mathsf{Sampler}$ that is a hard distribution over R_L, there does not exist any PPT algorithm Adv that can win the $\mathsf{EncodingGame}$ with probability non-negligible in n.

In the next section, we will prove the following theorem.

Theorem 1. *There exists a CLR Encoding scheme for every language $L \in NP$ in the CRS model under the validity of eXternal Diffie Hellman assumption; the encoding scheme supports a constancy fraction of leakage, i.e., it supports $l(n) = \delta \cdot p(n)$ where δ is a constant and $p(n)$ is a polynomial bounding the size of the encodings for statements of length n.*

3.2 Our Construction

We are now ready to describe our construction. Given a CLR encoding, we can use standard tools and paradigms to build a CLR-NTP system for L. We only need to make sure that tools used have an appropriate "leakage resilient" property.

To this end, we use a leakage resilient zero-knowledge proof system (LRZK) for NP denoted by (P_{lr}, V_{lr}). Such proof systems were constructed in [GJS11, Pan14]. Recall that an interactive proof system (P_{lr}, V_{lr}) is said to be *leakage-resilient* if for every cheating verifier V^* there exists a simulator S_{lr} who produces a computationally indistinguishable view on input the statement x and access to a leakage-oracle $\mathcal{L}^n_w(\cdot)$. The leakage oracle takes as input efficient functions f and returns $f(w)$. While, in LRZK, S_{lr} must read no more leakage from $\mathcal{L}^n_w$ than it gives to V^* in the simulated view, for us a weaker requirement suffices as used by GJS. We use a leakage-parameter $l := l(n)$ which we wll fix later.

GJS construct a variant denoted $(1+\epsilon)$-LRZK in n/ϵ rounds for any constant ϵ based on general assumptions. This notion in fact suffices for our purpose. However, since we are willing to use a CRS, we can replace the "preamble" of the GJS protocol (whose sole purpose is to extract a secret string r from the verifier) by $E_{pk}(r)$ where E_{pk} is a binding public-key encryption scheme and pk is a part of the CRS. As a result we get a constant round protocol; the modified simulator now extracts the string r by decrypting the cipher-text and then continue as in GJS. Further, we now get standard LRZK instead of $(1+\epsilon)$-LRZK. Therefore, we let (P_{lr}, V_{lr}) denote this modified constant-round version of the GJS protocol.

The protocol. Let $\mathbb{E} := (\mathsf{PubGen}, \mathsf{Encode}, \mathsf{Update}, V_L)$ be a CLR L-Encoding for the *circuit-satisfiability* problem, and let (P_{lr}, V_{lr}) be the above mentioned LRZK protocol for an NP-complete language.

Our continual-leakage-resilient non-transferable proof system (CLR-NTP). denoted by (P_{ntp}, V_{ntp}), for a language $L \in NP$ is an encoding-based proof system which uses $\mathbb{E}$ as its encoding. It proceeds as follows:

1. A public CRS $\rho \leftarrow \mathsf{PubGen}(1^n)$ is sampled at the beginning.

Let $(x, w) \in R_L$, and let $\mathsf{private}_1 \leftarrow \mathsf{Encode}(\rho, x, w)$ be an encoded witness. The prover P_{ntp} receives $(x, \mathsf{private}_1)$ as input; the verifier V_{ntp} receives x as its only input. *(Note that there is no leakage during this step or during any of the update phases).*

2. The prover P_{lr} proves to the verifier V_{lr}) that there exists an input $\mathsf{private}^*$ such that the (deterministic) algorithm V_L accepts the string $(\rho, x, \mathsf{private}^*)$.

3. At the end of each session, the prover applies the Update algorithm of the encoding scheme $\mathbb{E}$ to its encoded witness $\mathsf{private}$ and receives a fresh encoding which is used for the next session.

Consider the following theorem.

Theorem 2. *The protocol (P_{ntp}, V_{ntp}) is a non-transferable proof system for circuit satisfiability in the continual memory leakage model (CLR-NTP), supporting a constant fraction of leakage under the validity of XDH assumption.*

Proof Sketch. We only provide a proof sketch of this theorem since it uses standard methods and is easy to reconstruct. For now, let us delay calculating the actual amount of leakage tolerated by the system until later. The completeness and the soundness of the system are easy to check.

To prove the theorem, we will show that if a PPT Adv can win the $\mathsf{NTPGame}$ for a hard distribution $\mathsf{Sampler}$, then there exists an efficient machine to win the $\mathsf{EncodingGame}$ against the same $\mathsf{Sampler}$ for the scheme $\mathbb{E}$. This is done by first replacing the honest prover in each of the sessions of Phase I by the simulator S_{lr} which interacts with the adversary in polynomially many executions. Note that S_{lr} needs a leakage oracle to answer the queries made by the adversary and so, it uses the leakage oracle containing the encoding prepared by the challenger of the $\mathsf{EncodingGame}$. Note that, even after replacing the honest prover by S_{lr}, the probability with which the adversary is able to convince the verifier changes by a negligible quantity (this is because of the zero knowledge property of (P_{lr}, V_{lr})). We then replace the honest verifier in Phase II of the $\mathsf{NTPGame}$ by the extractor of the LRZK AoK. Now, since the adversary can convince the verifier with non-negligible probability, the extractor can extract a valid encoded witness from the adversary with non-negligible probability. This shows that there exists an adversary (which internally runs Adv, simulator S_{lr} and the extractor of the LRZK) who given access to leakage on the encodings can output a valid encoding. This contradicts the fact that the encoding scheme satisfied the properties of CLR encodings. This completes the proof.

Amount of Tolerated Leakage. Suppose that the encryption scheme used while generating the CRS (for (P_{lr}, V_{lr})) requires at most a constant factor more bits than the message length (easy using KEM and PRG). Further, suppose that the randomness used in each round is at most a constant-factor more than the size of the witness (ensured using a PRG of sufficiently large polynomial stretch). Then, the total size of prover's internal state at any given point is only a constant factor, say c, more than the length of the encoded witness. Therefore, if $l'(n)$ is the leakage parameter of the encoding scheme $\mathbb{E}$, then our proof system can support $l(n) = l'(n)/c$ bits.

4 Constructing CLR Encodings in the CRS Model

In this section, we will provide a detailed overview of our construction of CLR encodings; however, we will not provide full details due to space constraints. These details are given in relevant sections of the Appendix.

We construct CLR-encodings for the NP-complete language of *circuit-satisfiability* under standard assumptions. The language consists of circuits C as instances and w as witnesses such that $C(w) = 1$. From here on, L denotes the language of circuit-satisfiability and R is the corresponding relation. We will be working in the CRS model.

4.1 Step 1: NIZK proofs as Encodings

As mentioned in the introduction, we start with the idea of "proof of a proof" [DSY90], and encode a given witness as a NIZK-proof. The prover can prove to the verifier that "there exists a convincing non-interactive proof π for $x \in L$." Since we need to update the encodings in future executions, we need to find a way to update the NIZK-proofs. At this point, we have two options:

1. We can take the idea of "proof of a proof" even further. To update the encoding, the update algorithm computes a NIZK proof, say π_1 proving the statement that "there exists a valid NIZK proof π for $x \in L$." This process continues so that π_i uses π_{i-1}, and so on.

   Unfortunately, in this approach, the size of the encodings grows exponentially. We can attempt to reduce the size of π using more advanced techniques such as CS-proofs or SNARKs. However, these systems are usually based on either the random-oracle model or non-standard (knowledge-type) assumptions.

2. Another approach is to use NIZK proof systems which are **re-randomizable**: given π, it is possible to publicly compute a new proof π' of the *same size*; proof π' is computationally indistinguishable from a freshly computed NIZK proof. This is a promising approach, and also the starting point of our solution.

While re-randomization allows us to update the encodings π, it does little to "protect" the encoding under leakage. Consider an encoding which contains two (re-randomizable) proofs (π_1, π_2). Clearly, this new encoding is valid and re-randomizable; yet if one of the proofs is leaked, an adversary can obtain a full encoding from it and use it to prove that $x \in L$. Therefore, in addition to re-randomization, we need the property that it should be hard to obtain a valid encoding even given the leakage from polynomially many refreshed copies of an encoding.

We tackle this problem by considering two more properties of NIZKs:

- **Large min-entropy:** suppose that each proof π has $\ell + \omega(\log n)$ bits of min-entropy. Then, even given ℓ bits of leakage on π, no adversary can predict π *exactly* with more than negligible probability. Consequently, if $\pi, \ldots, \pi_{k=\mathrm{poly}(n)}$ are independently generated proofs, then no adversary receiving at most ℓ bits of leakage from each proof, can predict π_i exactly (for any $i \in [k]$) with more than negligible probability.

  At a high level, this property ensures that if an adversary computes a valid encoding, say π^*, it will be different from all of the encodings prover generates (via the update algorithm).[6]

- **Controlled malleability:** the previous bullet rules out the possibility that an adversary can output a π^* that is one of the valid encodings from $\pi_1, \ldots, \pi_k$; we now want to rule out the possibility that π^* can be a *re-randomization* of one of the encodings, say π_i. This is actually easily guaranteed if the proofs also have "controlled malleability" (CM) property. Let us explain.

  – First, consider the simpler property of *simulation-extractability* (a strengthened version of *simulation soundness*) [Sah99, SCO$^+$01]. It states that if an adversary A, who receives polynomially many simulated proofs (possibly to false statements), outputs a new proof π^* for some statement x^*, and π^* is different from all proofs A has received, then there exists an extractor which extracts a witness for x^* from the proof π^*.[7]

    Clearly, if our proofs satisfy this notion, then A cannot output a new proof π^* (for x) that differs from all previous proofs. This is because if it does, the extractor will extract a

[6]Note that this property by itself still does not guarantee much: most proofs already have this property, and if not, it is trivially satisfied by appending sufficiently many random bits (at the end of the proof) which will be ignored by the verifier. We therefore need the second property, which rules out the proofs of this kind.

[7]The extractor uses an extraction trapdoor corresponding to the system's CRS.

witness w for x. This however, is not possible, since x is sampled from a hard distribution (in the CLR-encoding game). Unfortunately, this notion is too strong for us: it also rules out re-randomization which is essential to update the proofs.

– We therefore turn to the notion of *controlled malleability* which is a generalization simulation-extractability to allow for very limited type of mauling, specifically *re-randomization* [CKLM12].[8] In particular, suppose that the NIZK proof system has the following extraction property. Consider an adversary A receiving several freshly computed proofs $\pi_1, \ldots, \pi_k$ for the statement x. If A outputs a convincing proof π^* for x that is different from all proofs it receives, then there exists an extractor which on input π^* outputs one of the following: (1) a witness for x, *or* (2) one of the previous proofs π_i and randomness r such that π^* is a re-randomization of π_i using randomness r. We will refer to this property as the **controlled malleability (CM)** property.[9]

Note that in our setting, these two properties would indeed suffice: it would be hard to extract one of the previous proofs π_i because A only receives ℓ bits of leakage but the proofs have large min-entropy property; further, tt would also be hard to extract a witness for x since x would be sampled from a hard distribution in the security game for CLR-encodings. Therefore, if we have a NIZK satisfying these properties, we would indeed have CLR encodings.

4.2 Step 2: Decomposable NIZK Proofs

Unfortunately, the CM property defined above is a bit too strong, and we do not achieve it. Instead, we consider a slight variation of both *large min-entropy* and *CM* property, which will also suffice to construct CLR-encodings.

To do this, we consider NIZKs in which the proof Π for a statement x can be decomposed into a pair (y, π). We view y as an *encoding of the statement* x, and π as the actual proof. We require that $\Pi = (y, \pi)$ satisfy all usual properties of a re-randomizable NIZK proof. Let $\mathsf{reRand} = (\mathsf{reRand}_1, \mathsf{reRand}_2)$ be re-rerandomization algorithm where the first and second parts re-randomize y and π respectively and appropriately.

Then, we modify the large min-entropy and CM properties, and require them to hold w.r.t. the first and second parts. More precisely, we require that:

- **Large min-entropy of *first part*:** We require that the first component y of an honestly generated proof $\Pi = (y, \pi)$ have sufficient min-entropy, e.g., $\ell + \omega(\log n)$ to tolerate ℓ bits of leakage. Note that this is a stronger property since if y has large min-entropy then so does Π. As before, it holds that given ℓ bits of leakage from polynomially many independent proof $\Pi_1, \ldots, \Pi_k$, it will be hard to compute y_i for any i where $\Pi = (y_i, \pi_i)$.

- **(Modified) Controlled malleability:** As before we consider an adversary A who receives independently generated proofs $\{\Pi_i = (y_i, \pi_i)\}_{i=1}^k$ and outputs a valid proof $\Pi^* = (y^*, \pi^*)$. All proofs are for the same statement x. Then, there exists an extractor which, on input Π^*, either outputs a witness for x, or it outputs (y_i, r) such that y^* is a re-randomization of y_i using r: i.e., $y^* = \mathsf{reRand}_1(y_i; r)$. Note that this property is strictly weaker than before: the extractor is only required to extract the first part of one of the proofs, namely y_i, but not the second part π_i.

Using the same arguments as before, it is not hard to see that NIZK-proofs satisfying these properties also imply CLR-encodings. We therefore formally define such proofs, and call them *decomposable proofs with controlled malleability (CM) property*, or **decomposable ℓ-CM-NIZK** where ℓ is the parameter for

[8]The CKLM definition is more general: it talks about a general set of *allowable transformations* $\mathcal{T}$ instead of re-randomization.

[9]We remark that this property is inspired by — but different from — the work of [CKLM12]. For simplicity, we are only presenting what is relevant to our context.

large-min-entropy property. The formal definition can be found in Appendix A. The construction of CLR-encodings from these proofs is straightforward: if $\mathbb{P}_{\mathsf{cm}} = (\mathsf{CRSSetup}_{\mathsf{cm}}, P_{\mathsf{cm}}, V_{\mathsf{cm}}, \mathsf{reRand}_{\mathsf{cm}})$ is decomposable ℓ-CM-NIZK then the corresponding CLR encoding scheme $\mathbb{E} = (\mathsf{PubGen}, \mathsf{Encode}, \mathsf{Update}, V)$ is obtained by setting $\mathsf{PubGen} = \mathsf{CRSSetup}_{\mathsf{cm}}$, $\mathsf{Encode} = P_{\mathsf{cm}}$, $\mathsf{Update} = \mathsf{reRand}_{\mathsf{cm}}$, and $V = V_{\mathsf{cm}}$. We have the following theorem.

Theorem 3. *Suppose that* $\mathbb{P}_{\mathsf{cm}} = (\mathsf{CRSSetup}_{\mathsf{cm}}, P_{\mathsf{cm}}, V_{\mathsf{cm}}, \mathsf{reRand}_{\mathsf{cm}})$ *is a decomposable ℓ-CM-NIZK for an NP relation R. Then,* $\mathbb{E} = (\mathsf{PubGen}, \mathsf{Encode}, \mathsf{Update}, V)$ *is a CLR-encoding with leakage parameter ℓ w.r.t. every hard distribution D on the relation R.*

Proof Sketch. Suppose that an adversary A can output a valid encoding Π^* after receiving leakage on previous encodings. Then, by a simple hybrid experiment, it will also output such a Π^* when instead of updating an existing encode, the prover uses a freshly computed encoding from the witness (i.e., a fresh NIZK). This is because updates are indistinguishable from fresh proofs. Now, if we apply the extractor to proof Π^*, due to the CM-property, it will either output a witness for x or it will output the first part, say y_i, of one of the previous encodings. But both of these are not possible because x comes from a hard distribution in the CLR game, and each y_i has independent, large, min-entropy even after leakage. □

The formal proof of the above theorem can be found in Section E.

4.3 Step 3: The sub-proof property

It is clear that in one of the the key challenge in our system is the (modified) controlled-malleability property (we will drop the word "modified" from here on). In this section, we will focus on this property only, and show that it can be achieved from a weaker property call the *sub-proof* property. More specifically, let $\mathbb{P}$ be a NIZK-PoK system which satisfies the first three properties — *decomposability*, *re-randomization*, and *large min-entropy* — and the following sub-proof property:

- **Sub-proof property**: There exist a special language L_{sub} and an (ordinary) NIZK-proof system $\mathbb{P}_{\mathsf{sub}}$ for L_{sub} such that a pair $\Pi = (y, \pi)$ is a valid proof for a statement $x \in L$ *if and only if*:
  1. x is a prefix of y, i.e., $\exists\, \widetilde{y}$ such that $y = x \circ \widetilde{y}$;
  2. π is a valid proof that $y \in L_{\mathsf{sub}}$ (according to the proof system $\mathbb{P}_{\mathsf{sub}}$);

We refer to such a proof system as the *decomposable NIZK with sub-proof (SP) property*, or **decomposable ℓ-SP-NIZK**. The formal definition is given in Appendix A. We now show that the sub-proof property can be "boosted up" to achieve the CM-property using the techniques of Chase et al. [CKLM12].

To do so, we first observe that the re-randomization property of $\mathbb{P}$ imposes the following *malleability* requirement on $\mathbb{P}_{\mathsf{sub}}$:

($\mathbb{P}_{\mathsf{sub}}$ must be malleable): Suppose that $\Pi = (y, \pi)$ is a NIZK-proof that $x \in L$ according to the main system $\mathbb{P}$ (under some CRS ρ). By the sub-proof property, π is a NIZK-proof that $y \in L_{\mathsf{sub}}$ according to the sub-proof system $\mathbb{P}_{\mathsf{sub}}$. By re-randomization property, if we let $\Pi' = \mathsf{reRand}(\Pi) = (\mathsf{reRand}_1(y), \mathsf{reRand}_2(\pi)) := (y', \pi')$, then Π' would also be a valid NIZK-proof for $x \in L$. This means that y' is a statement in L_{sub} and π' is a valid NIZK-proof for y'. Hence, the sub-proof system $\mathbb{P}_{\mathsf{sub}}$ is *malleable* in the following sense: given a proof π for the statement $y \in L_{\mathsf{sub}}$, it is possible to obtain a proof π' for a related statement $y' = \mathsf{reRand}(y)$ (in the same language).

In the terminology of [CKLM12], reRand_1 is called an **allowable transformation** over the statements in L_{sub}, reRand_2 is the corresponding **mauling** algorithm, and $\mathbb{P}_{\mathsf{sub}}$ is a reRand_1-*malleable* proof system. We now recall one of the main results from [CKLM12].

Result from [CKLM12]. Chase et al. formalize the general notion of a $\mathcal{T}$-malleable proof systems where $\mathcal{T}$ is an appropriate set of allowable transformations. They show an automatic compiler which can convert such a $\mathcal{T}$-malleable proof system into a new proof system that achieves *controlled malleability* in the following sense. In the new proof system, given a proof π for a statement y, an adversary can obtain proof for another statement y' *if and only if* $y' = \tau(y)$ where τ is a transformation from the set $\mathcal{T}$. This property is formalized by defining the notion of *controlled-malleable simulation-soundness extractability* (CM-SSE) proofs. In our context, the set $\mathcal{T}$ corresponds to reRand_1, and a specific τ is equivalent to executing reRand_1 with a specific randomness. The notion of CM-SSE then amounts to the following (simpler) experiment:

- **(CM-SSE in our context)**: Let $(\rho, t_1, t_2) \leftarrow \mathsf{FakeCRS}(1^n)$ where ρ is a simulated CRS, and t_1 and t_2 are simulation and extraction trapdoors respectively. A PPT adversary A is allowed to interact with the simulator polynomially many times, say k times, where in i-th interaction A sends an (adaptively chosen) statement y_i and receives a simulated proof π_i (proving that $y_i \in L_{\mathsf{sub}}$). In the end, A outputs a pair (y^*, π^*) such that y^* is different from all previous statements $y_1, \ldots, y_k$ *and* π^* is a valid proof for y^*. We say that the system is CM-SSE if for every A who outputs (y^*, π^*) with noticeable probability, the extraction trapdoor t_2 either extracts a witness w^* for the statement y^*, or it extracts a previous statement (say) y_i and randomness r such that $y^* = \mathsf{reRand}_1(y_i; r)$.

Remark 1. To apply the CKLM transformation, the underlying proof system must satisfy some structural properties. In particular, the NP-relation $\mathcal{R}$ (for which the system works) and the set of transformations $\mathcal{T}$, both must be expressible as a system of bilinear equations over elements in bilinear groups. This property is called *CM-friendliness.* In our context, relation R_{sub} and algorithm reRand_1 must be CM-friendly, i.e., be expressible as a system of bilinear equations. (See Appendix B.4 for formal definitions of CM-friendly relations, $\mathcal{T}$-malleability, CM-SSE proofs, etc.)

This essentially means that by suitably applying the CKLM compiler to the sub-proof system of a given (decomposable) ℓ-SP-NIZK, it should be possible to obtain the desired ℓ-CM-NIZK. This idea indeed works as proven in theorem below. One subtlety is that we need to assume that reRand_1 *does not change the prefix* x of its input. This is because, by definition, CM-SSE only extracts an instance y_i from which y^* has been obtained. To achieve ℓ-CM-NIZK we need extract a witness for x (which is same in all proofs). By requiring that reRand_1 does not change the prefix, we enforce that y_i and y^* will have the same prefix x, and this allows the reduction to go through.

Theorem 4. *Suppose that there exists a decomposable ℓ-SP-NIZK system $\mathbb{P}$ for an NP-relation R. Let* $\mathsf{reRand} = (\mathsf{reRand}_1, \mathsf{reRand}_2)$ *be the re-randomization algorithm,* $\mathbb{P}_{\mathsf{sub}}$ *be the sub-proof system, and* R_{sub} *be the sub-relation associated with* $\mathbb{P}$. *Then there exists a decomposable ℓ-CM-NIZK for R provided that* R_{sub} *and* reRand_1 *are CM-friendly and* reRand_1 *does not change the prefix of its input.*

Proof (sketch): Let $\mathbb{P} = (\mathsf{CRSSetup}, P, V, \mathsf{FakeCRS}, \mathsf{Sim}, \mathsf{Ext}, \mathsf{reRand})$ where $P = (\mathsf{InpGen}, P')$. Let V_{sub} be the verifier of $\mathbb{P}_{\mathsf{sub}}$ and by definition, P' is its prover; in particular, $\mathbb{P}_{\mathsf{sub}} = (\mathsf{CRSSetup}, P', V_{\mathsf{sub}}, \mathsf{FakeCRS}, \mathsf{Sim}, \mathsf{Ext})$, $\mathcal{T} = \mathsf{reRand}_1$ are its allowable set of transformations, and reRand_2 is the corresponding mauling algorithm. We first apply the CKLM transformation to $\mathbb{P}_{\mathsf{sub}}$ to obtain a new proof system, denoted $\mathbb{P}^*_{\mathsf{sub}} = (\mathsf{CRSSetup}^*, P'^*, V^*_{\mathsf{sub}}, \mathsf{FakeCRS}^*, \mathsf{Sim}^*, \mathsf{Ext}^*)$. Note that $\mathbb{P}^*_{\mathsf{sub}}$ satisfies CM-SSE definition.

Now, consider the following system $\mathbb{P}_{\mathsf{cm}} := (\mathsf{CRSSetup}^*, P_{\mathsf{cm}}, V_{\mathsf{cm}}, \mathsf{FakeCRS}^*, \mathsf{Sim}^*, \mathsf{Ext}^*, \mathsf{reRand})$ where:

1. $P_{\mathsf{cm}} := (\mathsf{InpGen}, P'^*)$
2. $V_{\mathsf{cm}}(x, \Pi, \rho)$: parse $\Pi = (y, \pi)$ and reject if x is not a prefix of y; otherwise return the output of $V^*_{\mathsf{sub}}(y, \pi, \rho)$.

It is easy to verify that $\mathbb{P}_{\mathsf{cm}}$ is a decomposable ℓ-CM-NIZK for the same relation R. In particular, if an adversary A outputs a proof $\Pi^* = (y^*, \pi^*)$ for x (in the CM-property game), we can construct an adversary A' against the CM-SSE property in which a A' would output π^* as a proof for y^*. Note that y^* will have the same prefix x as all the previous proofs it receives. Therefore, when we apply the CM-SSE extractor to y^* and we end up extracting a witness w^*, it will also yield a witness w for x. $\square$

4.4 Step 4: Achieving the sub-proof property

We have now seen that in order to construct a CLR L-encoding, it suffices to construct a proof-system $\mathbb{P}$ with the sub-proof property. In the final step, we construct such a proof system for the circuit satisfiability problem. Technically, this is the most involved part of the paper. Due to space constraints, we are only able to present an overview of our construction. The full details appear in relevant sections of the Appendix.

From here on, let L represent the circuit satisfiability language, and R be the corresponding relation. The instances in L are circuits C and witness w is an input such that $C(w) = 1$. To construct $\mathbb{P}$, we will actually start by constructing the sub-proof system in question $\mathbb{P}_{\mathsf{sub}}$. The sub-proof system has to be malleable and must work for a "CM-friendly relation", say R_{sub}.

The starting point of our construction is the observation instead of directly working with the given circuit C, we can actually work with a "garbled version" of C, denoted by GC. If we use re-randomizable garbled circuits, then we can randomize GC to yield a new garbled circuit GC' for the same C. For the proofs, we need to construct an NIZK proof system which proves the correctness of GC and can also be re-randomized. One of the approaches is to use the malleable proof systems of Groth and Sahai [GS08]. However, such proof systems only exist for special languages involving number-theoretic statements in bilinear groups.

Therefore, our next task is to represent GC (and the fact that it is a garbled version of a given C) using a system of bilinear equations such that we can deploy GS-proofs to argue about the solutions to such a system. This is a rather complex step in the proof: we show in several smaller steps how to represent simple relations using bilinear equations, and use them to represent an entire circuit. By doing so, we actually hit two birds with one stone: on one hand, we obtain malleable-proofs for an NP-complete language, and on the other hand, the resulting system is *CM-friendly* (since we already start with equations in bilinear groups). This strategy eventually works out, although there are several other details to be filled in.

To summarize, the main steps in the construction are as follows:

1. The first step is to devise a method to represent a garbled circuit as a system of bilinear equations. We use (a slightly modified version of) the garbled-circuit construction in [GHV10]. This modification is given in Appendix B.2.1. Thereafter, we show how to represent such a garbled circuit using a system of bilinear equations. We require that this representation of GC must re-randomizable (in a manner that is consistent with the randomization of garbled circuits).

2. Next, we define the InpGen algorithm (which will define the first part of the prover algorithm in the main system) as follows. InpGen first generates the garbled circuits and then outputs their representation as a system of bilinear equations. I.e., InpGen, on input (C, w), outputs (y, w') where $y := (C, \mathsf{GC}, \mathsf{w}_{\mathsf{GC}})$, GC is the garbled circuit of C expressed as appropriate bilinear equations, w_{GC} are wire-keys for w expressed as appropriate group elements, and w' is the randomness (and wire-keys) used to generate everything so far. The sub-relation over (y, w') is essentially the garbled circuit relation, and denoted by R_{gc}.

   The re-randomization algorithm of the garbled-circuit construction will act as the algorithm reRand_1. Note that this algorithm will not change (the prefix) C of the input y. We also show that reRand_1 can also be appropriately expressed as a system of bilinear equations, and hence satisfy the condition of CM-friendliness.

3. Next, we show how to use GS-proofs to obtain a NIZK-proof that y is indeed consistent with C and that it is satisfiable. Using previous work [CKLM12], we can ensure that this system is malleable w.r.t. reRand_1. The prover algorithm at this step becomes the second component of the final prover; rest of the algorithms become the corresponding algorithms of the final system.

The resulting system is a decomposable NIZK with the sub-proof property, as stated in the theorem below.

Theorem 5. *Assuming the validity of XDH assumption, there exists a constant $0 < \delta < 1$ and a decomposable ℓ-SP-NIZK $\mathbb{P}$ for the relation R_{gc} such that $\ell = \delta \cdot p(n)$ where $p(n)$ is a fixed polynomial defining the length of proofs (for statements of length n) generated by $\mathbb{P}$. Further, R_{gc} and $\mathbb{P}$ are both CM-friendly.*

The proof of this theorem is obtained by constructing the system $\mathbb{P}$ in Appendix D.1, which in turn relies on the sub-proof system $\mathsf{P}_{\mathsf{sub}}$ constructed in Appendix C.3. The properties of the sub-proof system can be found in Appendix C.4 and the properties of decomposable SP-NIZK can be found in D.

Acknowledgements

We thank the anonymous reviewers of CRYPTO 2014 for valuable suggestions that helped improve the quality of the paper.

References

[ADN$^+$10] J. Alwen, Y. Dodis, M. Naor, G. Segev, S. Walfish, and D. Wichs. Public-key encryption in the bounded-retrieval model. *EUROCRYPT*, pages 113–134, 2010.

[ADVW13] S. Agrawal, Y. Dodis, V. Vaikuntanathan, and D. Wichs. On continual leakage of discrete log representations. In *ASIACRYPT*, pages 401–420. Springer, 2013.

[AGV09] A. Akavia, S. Goldwasser, and V. Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. *TCC*, pages 474–495, 2009.

[BCH12] N. Bitansky, R. Canetti, and S. Halevi. Leakage-tolerant interactive protocols. *TCC*, pages 266–284, 2012.

[BCS08] E. Biham, Y. Carmeli, and A. Shamir. Bug attacks. *CRYPTO*, pages 221–240, 2008.

[BGJK12] E. Boyle, S. Goldwasser, A. Jain, and Y. T. Kalai. Multiparty computation secure against continual memory leakage. In *STOC*, pages 1235–1254. ACM, 2012.

[BHHO08] D. Boneh, S. Halevi, M. Hamburg, and R. Ostrovsky. Circular-secure encryption from decision diffie-hellman. *CRYPTO*, pages 108–125, 2008.

[BKKV10] Z. Brakerski, Y. T. Kalai, J. Katz, and V. Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *FOCS*, pages 501–510. IEEE, 2010.

[BS97] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. *CRYPTO*, pages 513–525, 1997.

[CDH12] J. Camenisch, M. Dubovitskaya, and K. Haralambiev. Efficient structure-preserving signature scheme from standard assumptions. *Security and Cryptography for Networks*, pages 76–94, 2012.

[CDRW10] S. Chow, Y. Dodis, Y. Rouselakis, and B. Waters. Practical leakage-resilient identity-based encryption from simple assumptions. In *ACM CCS*, pages 152–161. ACM, 2010.

[CKLM12] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Malleable proof systems and applications. *EUROCRYPT*, pages 281–300, 2012.

[COSV12] Chongwon Cho, Rafail Ostrovsky, Alessandra Scafuro, and Ivan Visconti. Simultaneously resettable arguments of knowledge. In *TCC*, pages 530–547, 2012.

[DHLAW10] Y. Dodis, K. Haralambiev, A. Lopez-Alt, and D. Wichs. Cryptography against continuous memory attacks. In *FOCS*, pages 511–520. IEEE Computer Society, 2010.

[DP08] S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302. IEEE, 2008.

[DP10] Y. Dodis and K. Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on feistel networks. *CRYPTO*, pages 21–40, 2010.

[DSY90] A. De Santis and M. Yung. Metaproofs (and their cryptographic applications), 1990.

[FFS88] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *J. Cryptology*, 1(2):77–94, 1988.

[FKPR10] S. Faust, E. Kiltz, K. Pietrzak, and G. Rothblum. Leakage-resilient signatures. *TCC*, pages 343–360, 2010.

[GGSW13] S. Garg, C. Gentry, A. Sahai, and B. Waters. Witness encryption and its applications. In *STOC*, pages 467–476. ACM, 2013.

[GHV10] C. Gentry, S. Halevi, and V. Vaikuntanathan. i-hop homomorphic encryption and rerandomizable yao circuits. *CRYPTO*, pages 155–172, 2010.

[GJS11] S. Garg, A. Jain, and A. Sahai. Leakage-resilient zero knowledge. *CRYPTO*, pages 297–315, 2011.

[GQ90] L. Guillou and J.-J. Quisquater. A paradoxical indentity-based signature scheme resulting from zero-knowledge. In *Advances in CryptologyCrypto88*, pages 216–231. Springer, 1990.

[GR10] S. Goldwasser and G. Rothblum. Securing computation against continuous leakage. *CRYPTO*, pages 59–79, 2010.

[GS08] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. *EUROCRYPT*, pages 415–432, 2008.

[HSH$^+$09] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5):91–98, 2009.

[ISW03] Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. *CRYPTO*, pages 463–481, 2003.

[Jak95] M. Jakobsson. Blackmailing using undeniable signatures. In *EUROCRYPT*, pages 425–427. Springer, 1995.

[JSI96] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In *EUROCRYPT*, pages 143–154, 1996.

[JV10] A. Juma and Y. Vahlis. Protecting cryptographic keys against continual leakage. *CRYPTO*, pages 41–58, 2010.

[KJJ99] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO*, pages 789–789. Springer, 1999.

[Koc96] P. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, pages 104–113. Springer, 1996.

[KV09] J. Katz and V. Vaikuntanathan. Signature schemes with bounded leakage resilience. *ASIACRYPT*, pages 703–720, 2009.

[LLW11] A. Lewko, M. Lewko, and B. Waters. How to leak on key updates. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, pages 725–734. ACM, 2011.

[LRW11] A. Lewko, Y. Rouselakis, and B. Waters. Achieving leakage resilience through dual system encryption. *TCC*, pages 70–88, 2011.

[NS09] M. Naor and G. Segev. Public-key cryptosystems resilient to key leakage. *CRYPTO*, pages 18–35, 2009.

[OPV08] Rafail Ostrovsky, Giuseppe Persiano, and Ivan Visconti. Constant-round concurrent non-malleable zero knowledge in the bare public-key model. In *ICALP*, pages 548–559, 2008.

[Pan14] O. Pandey. Achieving constant round leakage-resilient zero-knowledge. In *TCC*, 2014.

[Pie09] K. Pietrzak. A leakage-resilient mode of operation. *EUROCRYPT*, pages 462–482, 2009.

[QS01] J.-J. Quisquater and D. Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. *Smart Card Programming and Security*, pages 200–210, 2001.

[Sah99] A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, pages 543–553, 1999.

[Sch91] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[SCO$^+$01] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, pages 566–598, 2001.

A Definitions for Decomposable Proofs

We now provide definitions for decomposable proofs. Familiarity with NIZK-PoK proof systems is assumed in these definitions. Recall that a NIZK-PoK is defined by two triplets of algorithms: $(CRSSetup, P, V)$ and $(\mathsf{FakeCRS}, \mathsf{Sim}, \mathsf{Ext})$ where algorithms have their usual meanings. Often, the second triplet will be dropped from the mention of the proof system unless required by the context. Sometimes we will also refer to a "simulation trapdoor" t_1 and an "extraction trapdoor" t_2 (output by $\mathsf{FakeCRS}$).

Definition 2 (Decomposable ℓ-CM-NIZK)**.** A decomposable ℓ-CM-NIZK for an NP-language L with corresponding relation R is a collection of algorithms $(CRSSetup, P, V, \mathsf{FakeCRS}, \mathsf{Sim}, \mathsf{Ext}, \mathsf{reRand})$ such that first six algorithms define an ordinary *non-interactive zero-knowledge proof-of-knowledge system* (NIZK-PoK) for relation R. Further, the following properties hold:

- **Decomposability**: every proof Π output by the prover algorithm P can be decomposed into a pair of strings (y, π). That is, the prover algorithm P is actually a pair of algorithms (InpGen, P'); on input (x, w) and a CRS, P first computes $(y, w') \leftarrow \mathsf{InpGen}(x, w; r_1)$ and then $\pi \leftarrow P'(y, w', CRS; r_2)$. P outputs $\Pi = (y, \pi)$ as the proof.

- **Re-randomizability**: algorithm $\mathsf{reRand} = (\mathsf{reRand}_1, \mathsf{reRand}_2)$ is a pair such that on input a proof $\Pi = (y, \pi)$ for an instance x and randomness (r_1, r_2), it outputs a new proof $\Pi' = (y', \pi')$ for x such that $y' \leftarrow \mathsf{reRand}_1(y; r_1)$ and $\pi' \leftarrow (y, \pi, r_1; r_2)$. We require that for every $(x, w) \in R$ following holds for all non-uniform distinguishers:

$$(x, w, \rho, \Pi_1, \Pi_2) \quad \approx_c \quad (x, w, \rho, \Pi, \Pi')$$

where ρ is an honestly sampled CRS, Π_1, Π_2, Π are honestly generated proofs using algorithm $P(x, w, \rho)$, and Π' is a re-randomized version of Π, i.e., $\Pi' \leftarrow \mathsf{reRand}(\Pi)$.

- **Large min-entropy of first component**: we state this property in terms of a leakage game for convenience. Let A be an adversary (possibly unbounded), and let $\mathcal{O}^{\ell}(\cdot)$ be a *leakage oracle* which answers queries of A, denoted as circuits $f_1, f_2, \ldots$ on its argument. Then, for every integer t, every $\{(x_i, w_i)\}_{i=1}^{t} \in R^t$, the probability that adversary $A^{\mathcal{O}^{\ell}(\Pi_1),\ldots,\mathcal{O}^{\ell}(\Pi_t)}(1^n, \rho)$ outputs y^* such that $y^* = y_j$ for some index $j \in [t]$, is negligible. Here ρ is an honestly generated CRS, and every $\Pi_i := (y_i, \pi_i)$ is an honestly generated proof for x_i, i.e., $\Pi_i \leftarrow P(x_i, w_i, \rho)$. The total leakage A receives from any single oracle is at most ℓ.

- **Controlled malleability**: we now formalize the fact that any PPT A can compute proofs "by only re-randomizing the given proofs" unless it knows a witness. In defining this property, the adversary is usually given simulated proofs (using algorithm Sim w.r.t. a "fake" CRS ρ sampled using $\mathsf{FakeCRS}$).

  Formally, let A be a PPT adversary participating in the following game. The game samples $(\rho, t_1, t_2) \leftarrow \mathsf{FakeCRS}(1^n)$ and gives ρ to A; here t_1 and t_2 are simulation and extraction trapdoors respectively. A generates a statement x and receives polynomially many proofs $\Pi_1, \ldots, \Pi_k$ where every Π_i is simulated, i.e., $\Pi_i \leftarrow \mathsf{Sim}(\rho, t_1, x)$ for every $i \in [k]$. Finally, A outputs a string Π^*. We say that A looses the game if Π^* is not a valid proof for x or Π^* is a copy of one of the previous proofs, i.e., $\Pi^* \in \{\Pi_1, \ldots, \Pi_k\}$. Otherwise, let $z \leftarrow \mathsf{Ext}(\rho', t_2, \Pi^*, x)$. We say that A further looses the game if any of the following conditions hold: (1) z is a witness for x for relation R, or (2) $z = (y_i, r)$ for some $i \in [k]$ such that $y^* = \mathsf{reRand}_1(y_i; r)$ where y^* and y_i are the first components of Π^* and Π_i respectively.[10]

  Then, we require that every non-uniform PPT A looses the above game except with negligible probability.

Definition 3 (Decomposable ℓ-SP-NIZK)**.** A decomposable ℓ-SP-NIZK for an NP language L with corresponding relation R is a collection of algorithms $\mathbb{P} := (CRSSetup, P, V, \mathsf{FakeCRS}, \mathsf{Sim}, \mathsf{Ext}, \mathsf{reRand})$ such that the first six algorithms define an ordinary NIZK-PoK for the relation R, and the following properties hold:

- **Decomposability, re-randomizability, large min-entorpy**: as defined in definition 2. Therefore, the prover algorithm is a pair $P = (\mathsf{InpGen}, P')$. Likewise, algorithm $\mathsf{reRand} = (\mathsf{reRand}_1, \mathsf{reRand}_2)$ is also a pair, and ℓ is the parameter for large min-entropy.

- **Sub-proof property**: there exists an NP relation R_{sub} and a PPT algorithm V_{sub} such that:
  1. the tuple $\mathbb{P}_{\mathsf{sub}} := (CRSSetup, P', V_{\mathsf{sub}}, \mathsf{FakeCRS}, \mathsf{Sim}, \mathsf{Ext})$ is a NIZK-PoK for the relation R_{sub};[11]
  2. let ρ be a CRS, x represent a statement in L, and $\Pi = (y, \pi)$ represent a proof; then $V(x, \Pi, \rho)$ accepts *if and only if* x is a prefix of y *and* $V_{\mathsf{sub}}(y, \pi, \rho)$ also accepts.
  3. (i) for every $(x, w) \in R$, and every $(y, w') \leftarrow \mathsf{InpGen}(x, w)$, it holds that $(y, w') \in R_{\mathsf{sub}}$.[12]
     (ii) for every $(y, w') \in R_{\mathsf{sub}}$, there exists a *unique* and efficiently computable w s.t. $(x, w) \in R$ where x is the (appropriate) prefix of y.

B Preliminaries

B.1 BHHO encryption scheme

Consider a group $\mathbb{G}$ of prime order q in which DDH is hard. We describe the BHHO encryption scheme [BHHO08] when the message to be encrypted is a bit.

[10]We note that it also makes sense to consider a definition in which A is allowed to receive proofs for multiple statements instead of same x. For our application however, we require that A to output proof for the same x.

[11]Most algorithms of the sub-proof system are therefore same as the main system. Observe that the prover algorithm P' of the sub-proof is actually the *second* part of the main prover $P = (\mathsf{InpGen}, P')$.

[12]Note that x will always be a prefix of y output by InpGen.

Key Generation: Consider a secret key $s \leftarrow \{0,1\}^l$. Pick a matrix $\Psi \in \mathbb{G}^{(l+1)\times l}$ uniformly at random. Set $\Phi = (\Psi| - \Psi s^T)$. Define

$$\mathsf{sk} \leftarrow s \text{ and } \mathsf{pk} \leftarrow \Phi$$

Encryption: To encrypt a bit m, choose a random vector $r \leftarrow \mathbb{Z}_p^l$ and output the ciphertext

$$c \leftarrow r\Phi + (0^{1\times l}|m.g),$$

where g is an element in $\mathbb{G}$.

Decryption: To decrypt a ciphertext c, first parse the ciphertext as $(c_1||c_2)$ where the first l bits of c is represented as c_1 and the last bit is represented by c_2. Then, the message is obtained as follows.

$$m' \leftarrow c_1 s^T + c_2$$

If $m' = g$, then output the message to be 1 else output the message to be 0. This works because if c is correctly encrypted then c_1 is $r\Psi$ and c_2 is $-r\Psi s^T + m.g$ and so, $c_1 s^T + c_2$ is basically $m.g$ where m is the encrypted message.

Henceforth we will assume that the public key is part of the ciphertext. For example, in the above case we will denote the encryption of m by (c, Φ). The encryption of a vector of bits $\overrightarrow{v} = (v_1, \ldots, v_s)$ is nothing but the encryption of individual elements v_i. We now describe how the plaintext and the secret keys can be homomorphically evaluated with respect to the permutation operation.

Plaintext homomorphism. Since the encryption of the messages are done bit by bit, permuting the bits (or group elements) in the message can be done by permuting their corresponding ciphertexts.

Secret key homomorphism. Let σ be the permutation operation to be performed on the secret key. Let M' be the matrix such that $\mathsf{sk}M' = \sigma(\mathsf{sk})$ [13]. Now, consider the following matrix.

$$M = \begin{bmatrix} M' & 0 \\ 0 & 1 \end{bmatrix}$$

A ciphertext c encrypted with respect to secret key sk can be transformed into a ciphertext c' encrypted decryptable by a secret key $\mathsf{sk}M'$ as follows.

$$c' = cM^{-1}$$

The corresponding public key is $\mathsf{pk}' = \mathsf{pk}M^{-1}$.

B.2 Rerandomizable garbled circuits

We now briefly recall the Gentry et. al. [GHV10] construction of rerandomizable garbled circuits. First, for every wire w in the circuit, two wire keys $(L_{w,0}, L_{w,1})$, which are bit vectors each of length l, are generated which correspond to 0 and 1 respectively. For each gate G in the circuit, we construct a table of ciphertexts as follows. Let the input wires of G be w_1 and w_2 while the output wire key of G be w_3. Let $G(i,j)$ denote the output of gate G on input (i,j). The table of ciphertexts are as defined below.

$$\left\{\left(\mathsf{Enc}_{L_{(w_1,0)}}(\delta_{0,0}), \mathsf{Enc}_{L_{(w_2,0)}}((L_{(w_3,G(0,0))}|0^l) \oplus \delta_{0,0})\right)\right\}$$

$$\vdots$$

$$\left\{\left(\mathsf{Enc}_{L_{(w_1,1)}}(\delta_{1,1}), \mathsf{Enc}_{L_{(w_2,1)}}((L_{(w_3,G(1,1))}|0^l) \oplus \delta_{1,1})\right)\right\},$$

where, the masks $\delta_{i,j} \in \{0,1\}^{2l}$ is picked uniformly at random.

In the next section, we describe how we modify the GHV construction.

[13] $\sigma(\mathsf{sk})$ denotes the vector obtained by permuting the elements of sk using the permutation σ.

B.2.1 Alternate garbled circuit construction

We now modify the rerandomizable garbled circuit construction proposed by Gentry et. al. to suit the purposes of this work. We basically make the following main modifications.

1. The wire keys while constructing the garbled circuit is picked just as in the GHV construction except for the output wire. For the output wire, the wire keys $L_{w_m,0}$ and $L_{w_m,1}$ are v_0 and v_1 respectively where v_0 is the all zero vector of length l while v_1 is also a l length vector containing all zeroes except the first element which is 1.

2. Suppose let GC be the output of the GHV construction. Instead of outputting GC we rerandomize all the ciphertexts in GC along with permuting each table of ciphertexts to obtain GC'. Then, we output GC'. This seems to be a redundant step but will be helpful later when we need to prove that a garbled circuit was correctly computed.

The rest of the steps in this modified construction is same as the GHV construction. It can be verified that all the properties satisfied by the GHV construction continue to be satisfied for the alternate garbled circuit construction. We will denote the modified construction by $\mathsf{reRandGC}$.

We further denote the simulator corresponding to the garbled circuit construction to be SimGC. That is, the simulator takes as input C, $C(x)$ and its output distribution is computationally indistinguishable from the output distribution of $\mathsf{reRandGC}$ on input (C, x).

Choice of parameters. The choice of size for the wire keys depends on the leakage that we tolerate in the system. Denote by ℓ, the number of leakage bits allowed by the prover. We set the length of the wire keys in the garbled circuit construction in (P_{ntp}, V_{ntp}) to be $\max\Big(\Big\lceil\sqrt{\frac{(c^*+1)\ell}{|C|}}\Big\rceil, \Big\lceil\sqrt{\frac{k}{|C|}}\Big\rceil\Big)$, where k is a security parameter, $c^* = \frac{1}{c}$ (c is the fraction of the memory that can be leaked).

B.3 GS proofs

Groth and Sahai in EUROCRYPT 2008, proposed an efficient non interactive witness indistinguishable proof system for a set of restricted relations. These relations, defined over group elements, could be efficiently expressed as a system of bilinear equations. In particular, they consider four types of bilinear equations out of which only two of them are of interest for the purposes of this work. They first propose a generic proof system and then mention how to instantiate their proof system with respect to specific instantiations such as subgroup decision, XDH and DLIN assumptions. Further, they also describe the conditions under which the non interactive witness indistinguishable proof system can be converted to a zero knowledge proof of knowledge system. In this work, we study the DLIN based instantiation of the generic GS proof systems for the two types of bilinear equations mentioned below. And, these proof systems can be converted to a ZKPoK system. We now mention the types of bilinear equations supported by GS proofs which will be useful for our work.

Let $\mathbb{G}$ be a group where DLIN is hard. The setup $(p, \mathbb{G}, \mathbb{G}_T, e, \mathcal{P})$ describes the three $\mathbb{Z}_q$-modules $\mathbb{Z}_q, G$ and G_T where $\mathcal{P}$ is the generator of $\mathbb{G}$.

Multiscalar multiplications in $\mathbb{G}$: Consider the bilinear map $f_1 : \mathbb{Z}_q \times \mathbb{G} \to \mathbb{G}$. We view $\mathbb{Z}_q$ as an additive group here. We define f_1 as follows: $f_1(x, h) = x.h$ [14]

$$\sum_{i=1}^{n} f_1(a_i, y_i) + \sum_{i=1}^{m} f_1(x_i, b_i) + \sum_{i=1}^{n}\sum_{j=1}^{m} \gamma_{i,j} f_1(x_i, y_j) = \tau,$$

[14] $x.h$ denotes the group element obtained by adding h, x number of times. When the context is clear we abuse the notation and use $f_1(h, x)$ instead of $f_1(x, h)$.

where $a_i, \gamma_{i,j} \in \mathbb{Z}_q; b_i, \tau \in \mathbb{G}$ are constants whereas x_i, y_j are variables.

Quadratic equations in $\mathbb{Z}_q$: $f_2 : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{Z}_q$; $\mathbb{Z}_q$ is viewed as an additive group where f_2 is defined as $f_2(x, y) = xy \bmod p$.

$$\sum_{j=1}^{m} f_2(x_i, b_i) + \sum_{i=1}^{m} \sum_{j=1}^{n} \gamma_{i,j} f_2(x_i, y_j) = t,$$

where $b_i, t \in \mathbb{Z}_q$ are constants whereas $x_i, y_i \in \mathbb{Z}_q$ are variables.

In the GS proof system for the above type of equations, the prover tries to convince the verifier that he knows a satisfying assignment to the system of equations. At a high level, the prover proceeds as follows. The prover first computes the commitments to the variables and then tries to plug in the commitments to the equations. The equations will not be satisfied since the commitments satisfy hiding property. And hence, along with commitments the prover also needs to send some additional information in such a way the equations are satisfied. The formal details of the proof can be found in [GS08]. We will denote the GS proof system by a tuple of algorithms $(\mathsf{GSSetup}, \mathsf{GSProver}, \mathsf{GSVerifier})$.

B.4 Malleable proof systems from [CKLM12]

We now introduce different notions of malleable proof systems that will be useful in our construction of CLR encodings. All the definitions in this subsection are taken verbatim from Chase et al. [CKLM12]. Before we do that, we first give the definition of allowable transformations.

Definition 4. An efficient relation R is closed under an n-ary transformation $T = (T_x, T_w)$ if for any n-tuple $\{(x_1, w_1), \ldots, (x_n, w_n)\} \in R^n$, the pair $(T_x(x_1, \ldots, x_n), T_w(w_1, \ldots, w_n)) \in R$. If R is closed under T, then we say that T is admissible for R. Let $\mathcal{T}$ be some set of transformations; if for every $T \in \mathcal{T}$, T is admissible for R, then $\mathcal{T}$ is an allowable set of transformations.

We first start by giving the generic notion of malleable NIZK proof systems. When the context is clear we refer malleable NIZK proof systems as just malleable proof systems.

Malleable proof system. A malleable NIZK proof system is a proof system that satisfies the completeness, soundness and zero-knowledge properties. In addition, it is equipped with a set of transformations that can be applied to instances and proofs. The security property says that a mauled instance-proof pair is computationally indistinguishable from a freshly generated instance-proof pair. The formal definition of malleable proof systems is given below.

Definition 5 ((Malleable non-interactive proof system))**.** Let $(CRSSetup, P, V)$ be a non-interactive NIZK proof system for a relation R. Let $\mathcal{T}$ be an allowable set of transformations for R. Then this proof system is malleable with respect to T if there exists an efficient algorithm $\mathsf{MaulProof}$ that on input $(\sigma_{crs}, T, \{x_i, \pi_i\})$, where $T \in \mathcal{T}$ is an n-ary transformation and $V(\sigma_{crs}, x_i, \pi_i) = 1$ for all i, $1 \leq i \leq n$, outputs a valid proof π for the statement $x = T_x(\{xi\})$ (i.e., a proof π such that $V(\sigma_{crs}, x, \pi) = 1$).

A malleable non-interactive proof system is said to be derivation-private if it satisfies the following definition.

Definition 6. For a non-interactive proof system $(\mathsf{CRSSetup}, P, V, \mathsf{ZKEval})$ for an efficient relation R malleable with respect to $\mathcal{T}$, an adversary A, and a bit b, if the probability that A succeeds in the following game is at most $\frac{1}{2} + \mathsf{negl}(k)$ then we say that the proof system is derivation private:

- *Step 1.* $CRS \leftarrow \mathsf{CRSSetup}(1^k)$
- *Step 2.* $(\mathsf{state}, x_1, w_1, \pi_1, \ldots, x_q, w_q, \pi_q, T = (T_x, T_w)) \leftarrow A(\sigma_{CRS})$.

- *Step 3.* If $V(CRS, x_i, \pi_i) = 0$ for some i, $(x_i, w_i) \notin R$ for some i, or $T \notin \mathcal{T}$, abort and output $\perp$. Otherwise, form $\pi = P(CRS, T_x(x_1, \ldots, x_q), T_w(w_1, \ldots, w_q))$ if $b = 0$, otherwise $\mathsf{ZKEval}(CRS, T, \{x_i, \pi_i\})$ if $b = 1$.
- *Step 4.* $b' \leftarrow \mathcal{A}(\mathsf{state}, \pi)$.

We say that A succeeds in the above game if $b' = b$.

In this work, we will only consider malleable proof systems satisfying the notion of derivation privacy. If $\mathcal{T}$ is the set of transformations associated to a malleable proof system, we refer to the proof system as $\mathcal{T}$-malleable proof system.

Re-randomizable proof system. A rerandomizable proof system is a special case of a malleable proof system. It is a zero knowledge proof system with an additional rerandomization algorithm, denoted by reRand, which satisfies the following game. First the adversary generates an instance along with its witness and a proof and sends it to the challenger. The challenger depending on the challenge bit b decides to either composes a proof on the instance using its witness (when say $b = 0$) or it evaluates reRand on the proof sent by the adversary and sends the resulting proof back to the adversary (when $b = 1$). The proof system is rerandomizable if the output of the adversary is b is negligibly close to $\frac{1}{2}$. We now define rerandomizable proof system more formally.

Definition 7. [CKLM12] (Randomizable non-interactive proof system). For a proof system $(CRSSetup; P; V)$ with an additional randomized algorithm reRand that, on input a proof and a statement outputs a new proof for the same statement, a given adversary Adv, and a bit b, let $p_b^{\mathsf{Adv}}(k)$ be the probability of the event that $b' = 0$ in the following game:

- Step 1. $CRS \leftarrow CRSSetup(1^k)$
- Step 2. $(\mathsf{state}; x; w; \pi) \leftarrow \mathsf{Adv}(CRS)$
- If $V(x, \pi)$ is Reject or $(x, w) \notin R$ then output $\perp$. Otherwise, $\pi' \leftarrow P(x, w, CRS)$ if $b = 0$ else $\pi' \leftarrow \mathsf{reRand}(x, \pi, CRS)$ if $b = 1$.
- $b'' \leftarrow \mathsf{Adv}(\mathsf{state}, \pi')$

We say that the proof system is randomizable if for all PPT algorithms Adv, we have $|p_b^{\mathsf{Adv}}(1^k) - p_{\bar{b}}^{\mathsf{Adv}}(1^k)|$ is negligible in k.

Malleable proof systems satisfying CM-SSE property. In [CKLM12], Chase et. al. proposed the notion of controlled-malleable simulation extractability in proof systems. Controlled malleable simulation extractability essentially allows only a certain set of operations to be performed on the input instances as well as the accompanying proofs. The game proceeds as follows. The adversary receives several simulated proofs from an oracle on input instances of his choice. He then outputs an input instance and an acceptable proof that the instance belongs to the language for which the proof system is defined. The CM-SSE property ensures that there exists an extractor who can extract a transformation as well as a queried instance such that the instance output by the adversary is obtained by applying the transformation on the queried instance or it outputs a valid witness that the instance belongs to that language. We formally define the controlled-malleable simulation sound extractable property below.

Definition 8. [CKLM12] Let $(CRSSetup, P, V)$ be a NIZKPoK system for an efficient relation R with a simulator (S_1, S_2) and an extractor (E_1, E_2). Let $\mathcal{T}$ be an allowable set of transformations for the relation R such that membership in $\mathcal{T}$ is efficiently testable. Let SE_1 be an algorithm that, on input 1^k outputs $(\sigma_{crs}, \tau_s, \tau_e)$ such that (σ_{crs}, τ_s) is distributed identically to the output of S_1. Let Adv be given, and consider the following game:

- $(\sigma_{crs}, \tau_s, \tau_e) \leftarrow SE_1(1^k)$
- $(x, \pi) \leftarrow \mathsf{Adv}^{S_2(\sigma_{crs}, \tau_s, \cdot)}(\sigma_{crs}, \tau_e)$
- $(w, x', T) \leftarrow E_2(CRS, \tau_e, x, \pi)$

We say that the NIZKPoK satisfies controlled malleable simulation sound extractability (CM-SSE) if for all PPT algorithms Adv there exists a negligible function $\nu(\cdot)$ such that the probability (over the choices of SE_1, Adv and S_2) that $V(\sigma_{crs}, x, \pi) = 1$ and $(x, \pi) \notin Q$ (where Q is the set of queried statements and their responses) but either 1. $w \neq \perp$ and $(x, w) \notin R$, 2. $(x', T) \neq (\perp, \perp)$ and either $x' \notin Q_x$ (the set of queried instances), $x \neq T_x(x')$ or $T \notin \mathcal{T}$ or 3. $(w, x', T) = (\perp, \perp, \perp)$ is at most $\nu(k)$.

CM-friendly relations. We will now state a key definition, namely CM-friendly relation, that will be useful to describe the generic construction in the next section.

Definition 9. For a relation R and a class of transformations $\mathcal{T}$, we say $(R, \mathcal{T})$ is CM-friendly if the following six properties hold:

1. Representable statements: any instance and witness of R can be represented as a set of group elements; i.e., there are efficiently computable bijections $F_s : L_R G^{d_s}$ for some d_s and $Fw : W_R \to G^{d_w}$ for some d_w where $W_R := \{w \mid \exists x : (x, w) \in R\}$.
2. Representable transformations: any transformation in $\mathcal{T}$ can be represented as a set of group elements; i.e., there is an efficiently computable bijection $F_t : T \to G^{d_t}$ for some d_t.
3. Provable statements: we can prove the statement $(x, w) \in R$ (using the above representation for x and w) using pairing product equations; i.e., there is a pairing product statement that is satisfied by $F_s(x)$ and $F_w(w)$ iff $(x, w) \in R$.
4. Provable transformations: we can prove the statement "$T_x(x') = x for T \in \mathcal{T}$ (using the above representations for x and T) using a pairing product equation, i.e. there is a pairing product statement that is satisfied by $F_t(T), F_s(x), F_s(x')$ iff $T \in \mathcal{T}$ and $T_x(x') = x$.
5. Transformable statements: for any $T \in \mathcal{T}$, there is a valid transformation $s(T)$ that takes the statement "$(x, w) \in R$" (phrased using pairing products as above) and produces the statement "$(T_x(x), T_w(w)) \in R$."
6. Transformable transformations: for any $T, T' \in \mathcal{T}$ there is a valid transformation $t(T)$ that takes the statement "$T_x(x') = x$ for $T \in \mathcal{T}$" (phrased using pairing products as above) and produces the statement "$T'_x \circ T_x(x') = T'x(x)$ for $T \circ T' \in \mathcal{T}$" and that preserves the variables in x'.

Remark 1. Each of the above transformations operate only on one instance at a time.

B.5 Generic construction to obtain CM-SSE NIZKs

Consider a relation R. Let $\mathcal{T}$ be a set of allowable transformations (Definition 4) defined for the relation R that contains an identity transformation. Let (P, V) be any non interactive witness indistinguishable proof of knowledge system for the relation R and consider a structure preserving group signature scheme. There exists structure preserving group signatures under DLIN assumptions [CDH12]. Chase et. al. proposed a generic construction in [CKLM12] to transform (P, V) into a non interactive zero knowledge proof of knowledge system (P^*, V^*) which satisfied controlled malleable-simulation sound extractability property (see Definition 8). At a high level, this is how the transformation proceeds. The relation is modified to obtain a new relation R' which is described as follows. The tuple $(x, vk; w, x', T, \sigma)$ belongs to R' if $(x, w) \in R$ or $T(x') = x$, $\mathsf{Verify}(x', \sigma, vk) = 1$ and T is an allowable transformation. Here, Verify is the verification algorithm of the signature scheme and vk is the verification key. Denote the proof system defined for this relation R to be (P^*, V^*). It can be shown that (P^*, V^*) satisfies controlled malleable simulation sound extractability property. This can be seen intuitively as follows. Once the adversary outputs an acceptable proof after receiving many simulated proofs, we can use the extractor of (P^*, V^*) to extract either the witness w or an input instance x' along with a transformation T such that $T(x') = x$. The proof of knowledge property ensures that the extractor outputs either of them with non-negligible probability. For a formal proof refer to the proof of Theorem 4.5 in Appendix C in [CKLM12].

C Construction of sub-proof system $\mathbb{P}_{\mathsf{sub}}$

The construction of $\mathbb{P}_{\mathsf{sub}}$ is performed as follows. The main goal is to generate GS proofs that verify whether a garbled circuit is correctly computed or not. As noted before, GS proofs are only defined for specific relations and we need to express a complex relation in terms of GS proofs. We do this by first dividing the complex relation into simpler relations and then expressing the simpler relations as bilinear equations for which GS proofs exist. Once we complete that step, we generate GS proofs corresponding to all these simpler relations. The final proof is the concatenation of all the proofs.

We first describe, in Section C.1, how the problem of proving that a garbled circuit is correctly computed is divided into simpler relations which is then expressed as a system of bilinear equations. In Section C.2, we show how to express how to combine the GS proofs for all these simpler relations. Finally, in Section C.3, the proof system $\mathbb{P}_{\mathsf{sub}}$ is constructed.

C.1 Expressing garbled circuit construction in terms of bilinear equations

To express the garbled circuit construction given in Appendix B.2 as a set of bilinear equations we first describe a set of simple relations by a system of bilinear equations. This will later help us in expressing the garbled circuit construction as a system of equations. In each of the equations below, we classify the variables used in the equations as main variables and temporary variables. The convention followed is that the input variables to the equations are all main variables while all the other variables used in the equations are temporary variables. For example, in a system of equations denoted by $\mathsf{Eq}(x, y, z)$, x, y and z are main variables while all the other variables used in $\mathsf{Eq}(x, y, z)$ other than x, y and z are temporary variables. In some of the equations stated below, the variables are written on the right hand side of the equations. The equation can be modified by a simple trick in such a way that the right hand side of the equation contains a constant as follows. Consider an equation whose right hand side is a variable v. For now, assume that we are dealing with multiscalar multiplication equations (this can be easily extended to quadratic equations also). Let the equation be denoted as follows.

$$Eq = v$$

We can modify this equation to obtain a new equation as follows.

$$Eq - f_1(1, v) = 0$$

It can be checked that the new equation and the original equation have the same set of solutions.

We will introduce one more notation: Let $\mathsf{Eq}(X_1, \ldots, X_n)$ be a system of equations. By $\mathsf{Eq}(X_1 = a_1, \ldots, X_n = a_n) = 1$ we mean that the system of equations Eq is satisfied when the variables $X_1, \ldots, X_n$ are assigned the values $(a_1, \ldots, a_n)$ respectively and appropriately setting the values of the temporary variables and similarly by $\mathsf{Eq}(X_1 = a_1, \ldots, X_n = a_n) = 0$ we mean that the system of equations is not satisfied for that particular assignment of main variables and for any assignment of the temporary variables.

We first give an overview and then describe the technical details.

C.1.1 Overview

We now give a brief overview of how the garbled circuit construction described is expressed in terms of bilinear equations. To show how this proof is generated we first recall the steps of garbled circuit construction and then express each step as a set of bilinear equations (written as italics). We now compose the proof(s) for this step by generating GS proofs corresponding to the set of bilinear equations.

- Generate the wire keys corresponding to all the wires in the circuit as follows. For each wire w, execute KeyGen of the BHHO scheme twice to get $(\mathsf{sk}_1, \mathsf{pk}_1), (\mathsf{sk}_2, \mathsf{pk}_2)$. Set $L_{w,0}$ (resp., $L_{w,1}$) to be sk_1 (resp., sk_2). We denote the public key pk_1 (resp., pk_2) by $\mathsf{pk}_{L_{w,0}}$ (resp., $\mathsf{pk}_{L_{w,1}}$).

  To ensure that the public keys are correctly generated, we make use of the public key consistency equations. For every wire key $L_{w,b}$ associated to wire w corresponding to bit b, we associate the following set of bilinear equations: $\mathsf{PKEq}(X_{\mathsf{pk}_{w,b}}, X_{\mathsf{sk}_{w,b}})$. Further to ensure that every wire key (corresponding to the secret key) is a bit vector we make use of bit equations: Let $L_{w,b}$ be a wire key. To this wire key we associate a set of bilinear equations $\mathsf{VecBitEq}(X_{L_{w,b}})$. We further associate the equations $\mathsf{OutWiresEq}(X_{L_{w_m,0}}, X_{L_{w_m,1}})$, where w_m is the output wire key to ensure the fact that the output wire keys are of the form $L_{w_m,0} = (0^{1\times l})$ and $L_{w_m,1} = (1|0^{1\times(l-1)})$.

- Construct the table of ciphertexts as follows as given in Section B.2. Each entry of the table, denoted by $c^G_{(i,j)}$ corresponding to the gate G consists of two ciphertexts $c^{G,1}_{(i,j)} = \mathsf{Enc}_{L_{(w_1,i)}}(\delta^G_{i,j})$ and $c^{G,2}_{(i,j)} = \mathsf{Enc}_{L_{(w_2,j)}}(\mu^G_{i,j})$ for $i, j \in \{0,1\}$.

  To ensure that the ciphertexts are computed correctly we make use of the ciphertext equations as follows. For every gate G in the circuit with input wires w_1, w_2 and output wire w_3, we associate the ciphertext equations $\mathsf{CTEq}(X_{c^{G,1}_{(i,j)}}, X_{\mathsf{pk}_{L_{w_1,i}}}, X_{r^1_{i,j}}, X_{\delta^G_{i,j}}), \mathsf{CTEq}\Big(X_{c^{G,2}_{i,j}}, X_{\mathsf{pk}_{L_{w_2,j}}}, X_{r^2_{i,j}}, X_{\mu^G_{i,j}}\Big)$ for $i, j \in \{0,1\}$ where $r^1_{i,j}$ denotes the randomness to generate $c^{G,1}_{(i,j)}$ whereas $r^2_{i,j}$ denotes the randomness to generate $c^{G,2}_{(i,j)}$. Further, we use XOR equations to ensure that $\mu^G_{i,j} \oplus \delta^G_{i,j} = (L_{w_3,G(i,j)}|0^l)$ as follows. For every gate G, for every $i, j \in \{0,1\}$ we introduce the equations $\mathsf{VecXOREq}(\mu^G_{i,j}, \delta^G_{i,j}, (L_{w_3,G(i,j)}|0^l))$. Further, we associate bit equations, namely $\mathsf{VecBitEq}(X_{\delta^G_{i,j}})$ and $\mathsf{VecBitEq}(X_{\mu^G_{i,j}})$ for every $i, j \in \{0,1\}$, to ensure that $\mu^G_{i,j}$ and $\delta^G_{i,j}$ are composed of bits.

- For each gate in the circuit, the table of ciphertexts constructed need to be permuted. Consider a table corresponding to gate G. After permuting the table, we denote the entries in the table, in order, by $\mathsf{ct}^G_1, \mathsf{ct}^G_2, \mathsf{ct}^G_3, \mathsf{ct}^G_4$ (where $\mathsf{ct}^G_i = (\mathsf{ct}^{G,1}_i, \mathsf{ct}^{G,2}_i)$ for $i = 1, \ldots, 4$).

  For this step, we associate the following system of bilinear equations for each gate G in the circuit. Let $c^G_{0,0}, c^G_{0,1}, c^G_{1,0}, c^G_{1,1}$ be the table of ciphertexts for gate G as computed in the second step. Then, we associate the following set of equations to gate G, namely $\mathsf{PermEq}(\mathsf{ct}^G_1, \mathsf{ct}^G_2, \mathsf{ct}^G_3, \mathsf{ct}^G_4, (c^G_{0,0}, c^G_{0,1}, c^G_{1,0}, c^G_{1,1}))$. Note that this does not completely ensure that $(\mathsf{ct}^G_1, \mathsf{ct}^G_2, \mathsf{ct}^G_3, \mathsf{ct}^G_4)$ is a permutation of $(c^G_{0,0}, c^G_{0,1}, c^G_{1,0}, c^G_{1,1})$). This just ensures that ct^G_i is one of $(c^G_{0,0}, c^G_{0,1}, c^G_{1,0}, c^G_{1,1})$) for $i = 1, \ldots, 4$. This issue can be taken care of by making sure that all ct^G_i are distinct which can be easily checked by the verifier.

- For each gate in the circuit, we modify the entries of the table of ciphertexts associated to that gate to account for rerandomization and XORing of random values. For every ciphertext $\mathsf{ct}^{G,j}_{i,k}$, we compute $\Delta(R\mathsf{pk}_{L_{w_j,b}} + (0^{1\times(l-1)}|1.g) - ct^{G,j}_i) - (1-\Delta)(R\mathsf{pk}_{L_{w_j,b}} - \mathsf{ct}^{G,j}_i)$ where $\mathsf{pk}_{L_{w_j,b}}$ is the public key part of the ciphertext $\mathsf{ct}^{G,j}_i$ and Δ is set to 0 and R is set to 0. Denote by $\mathsf{ct}^{\mathsf{reRand},G,j}_{i,k}$ the resulting ciphertext.
  To ensure that $\mathsf{ct}^{\mathsf{reRand},G,j}_{i,k}$ is indeed computed from $\mathsf{ct}^{G,j}_{i,k}$ we associate the equations $\mathsf{VecreRandCTEq}(\mathsf{ct}^{G,j}_{i,k}$, $\mathsf{ct}^{\mathsf{reRand},G,j}_{i,k}$, R, $\Delta)$ for every gate G, $j = 1,2$ and $i = 1, \ldots, 4$. [15].

[15] On the hindsight, this might look like an unnecessary operation since $\mathsf{ct}^{\mathsf{reRand},G,j}_{i,k}$ is the same as $\mathsf{ct}^{G,j}_{i,k}$. This is added to account for rerandomization. During the rerandomization process, the commitments to Δ and r are suitably modified depending on what randomness is used to rerandomize the garbled circuit

C.1.2 Expressing simple relations as bilinear equations

Expressing BHHO encryption scheme as a set of bilinear equations.

Consider the bilinear map f_1, defined as in Appendix B.3. We describe the above BHHO encryption scheme as a set of bilinear equations as follows. Before that we introduce the following notation: the $(i,j)^{th}$ entry in the matrix X_Φ is denoted by the element $X_{\Phi_{(i,j)}}$. We first define the public consistency equations and then define equations corresponding to the encryption. Public consistency equations is to ensure that the last column of the public key is correctly computed. Let g be an element in the group $\mathbb{G}$.

Public Key consistency equations: $\mathsf{PKEq}(X_\Phi, X_s)$.

$$\Phi_{(1,l+1)} = f_1(0,g) - (\ f_1(X_{s_1}, X_{\Phi_{1,1}}) + \cdots + f_1(X_{s_l}, X_{\Phi_{1,l}})\ )$$

$$\vdots$$

$$\Phi_{(l,l+1)} = f_1(0,g) - (\ f_1(X_{s_1}, X_{\Phi_{l,1}}) + \cdots + f_1(X_{s_l}, X_{\Phi_{l,l}})\ )$$

Before we state the lemma below, we introduce a terminology. A public-key secret key pair $(\mathsf{pk}, \mathsf{sk})$ pair is a valid public key-secret key pair if $v = -\Psi\mathsf{sk}$, and if pk can be written as $\mathsf{pk} = [\Psi|v]$, where v is the last column of pk. The following follows from the definition of PKEq.

Lemma 1. *A public key-secret key pair* $(\mathsf{pk}, \mathsf{sk})$ *is a valid public key-secret key pair iff* $\mathsf{PKEq}(X_\Phi = \mathsf{pk}, X_s = \mathsf{sk}) = 1$.

Ciphertext equations: We give the description of ciphertext equations. Let X_Φ be defined as before. We will use the following notation: the i^{th} entry in X_c (resp., X_r) is represented by X_{c_i} (resp., X_{r_i}).

$\mathsf{CTEq}(X_c, X_\Phi, X_r, X_m)$.

$$X_{c_1} = f_1(X_{r_1}, X_{\Phi_{1,1}}) + \cdots + f_1(X_{r_l}, X_{\Phi_{l,1}})$$

$$\vdots$$

$$X_{c_l} = f_1(X_{r_1}, X_{\Phi_{1,l}}) + \cdots + f_1(X_{r_l}, X_{\Phi_{l,l}})$$

$$X_{c_{l+1}} = f_1(X_{r_1}, X_{\Phi_{1,l+1}}) + \cdots + f_1(X_{r_l}, X_{\Phi_{l,l+1}}) + e(X_m, g)$$

In the equations described above, there are no constraints placed to ensure that s_i and m are bits. Later, we will introduce the bit equations to deal with this issue. We now state the following lemma which directly follows from the definition of CTEq.

Lemma 2. *A ciphertext c is an encryption of the message m under the randomness r using the public key Φ iff* $\mathsf{CTEq}(X_c = c, X_\Phi = \Phi, X_r = r, X_m = m) = 1$.

We will now describe a system of equations to express the encryption of multiple elements instead of a single element. Consider a vector of elements $(X_{m_1}, \ldots, X_{m_l})$ denoted by X_{m}. The encryption of m using the BHHO encryption scheme can be expressed by the following system of bilinear equations. Let X_R and X_C be matrices of variables with the i^{th} column containing vectors X_{R_i} and X_{C_i} respectively.

$\mathsf{VecCTEq}(X_C, X_\Phi,_R, X_{\mathsf{m}})$:

$$\mathsf{CTEq}(X_{C_1}, X_\Phi, X_{R_1}, X_{m_1}), \ldots, \mathsf{CTEq}(X_{C_l}, X_\Phi, X_{R_l}, X_{m_l})$$

The following lemma follows directly from Lemma 2.

Lemma 3. *A tuple of ciphertexts c is an encryption of a tuple of messages m under a vector of randomness r, where m_i is encrypted to get c_i under the randomness r_i, using the public key Φ iff* $\mathsf{VecCTEq}(X_c, X_{\Phi}, {}_r, X_{\mathsf{m}}) = 1$.

Expressing bitwise OR, bitwise AND and bitwise XOR operations

Consider a tuple of bits $(a_1, \ldots, a_l)$ and c. We will express the statement "$c = a_1$ AND $\cdots$ AND a_l" as a set of bilinear equations as follows. Consider the bilinear mapping $f_2 : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{Z}_q$ as defined in Section B.3.

$\mathsf{ANDEq}(X_{a_1}, \ldots, X_{a_l}, X_c)$:

$$f_2(X_{a_1}, X_{a_2}) = X_{c_1}$$
$$f_2(X_{c_1}, X_{a_3}) = X_{c_2}$$
$$\vdots$$
$$f_2(X_{c_{l-2}}, X_{a_l}) = X_c$$

Consider the following lemma.

Lemma 4. $c = (a_1\ AND\ \ldots\ AND\ a_l)$ *iff* $\mathsf{ANDEq}(X_{a_1} = a_1, \ldots, X_{a_l} = a_l, X_c = c) = 1$.

Proof. Let $\mathsf{ANDEq}(X_{a_1} = a_1, \ldots, X_{a_l} = a_l, X_c = c) = 1$. We claim that c is AND of $a_1, \ldots, a_l$. This is because essentially the above system of equations is calculating $X_c = X_{a_1} \cdots X_{a_l}$ which further means that $c = 0$ if even one $a_i = 0$ and $c = 1$ if all of $a_1, \ldots, a_l$ is 1. In other words, this means that c is AND of $a_1, \ldots, a_l$.

Let c be the AND of $a_1, \ldots, a_l$. We describe how to set the temporary variables of ANDEq such that $\mathsf{ANDEq}(X_{a_1} = a_1, \ldots, X_{a_l} = a_l, X_c = c) = 1$. Set X_{c_1} to be the AND of X_{a_1} and X_{a_2}. Set X_{c_2} to be the AND of X_{c_1} and X_{a_3} and so on. Observe that this will satisfy the system of equations. □

We write down bilinear equations to express the statement "$c = a_1$ OR ... OR a_l".

$\mathsf{OREq}(X_{a_1}, \ldots, X_{a_l}, X_c)$:

$$f_2(1,1) - f_2(X_{a_1}, 1) = X_{\overline{a_1}}$$
$$\vdots$$
$$f_l(1,1) - f_2(X_{a_l}, 1) = X_{\overline{a_l}}$$
$$f_2(1,1) - f_2(c, 1) = X_{\overline{c}}$$
$$\mathsf{ANDEq}(X_{\overline{a_1}}, \ldots, X_{\overline{a_l}}, X_{\overline{c}})$$

We now state the following lemma.

Lemma 5. $c = (a_1\ OR\ \ldots\ OR\ a_l)$ *iff* $\mathsf{OREq}(X_{a_1} = a_1, \ldots, X_{a_l} = a_l, X_c = c) = 1$.

Proof. Let $\mathsf{OREq}(X_{a_1} = a_1, \ldots, X_{a_l} = a_l, X_c = c) = 1$. We claim that c is OR of $a_1, \ldots, a_l$. This is because the above system of equations ensure that $\overline{c} = \overline{a_1}\ OR\ \ldots\ OR\ \overline{a_l}$ where $\overline{c}, \overline{a_i}$ is the complement of c, a_i respectively for $i = 1, \ldots, l$. This further proves that c is indeed the OR of $a_1, \ldots, a_l$.

Let c be the OR of $a_1, \ldots, a_l$. We describe how to set the temporary variables of OREq such that $\mathsf{OREq}(X_{a_1} = a_1, \ldots, X_{a_l} = a_l, X_c = c) = 1$. Set $X_{\overline{a_i}}$ to be 0 if $a_i = 1$ else set $X_{\overline{a_i}}$ to be 1 for all $i = 1, \ldots, l$. Further, set the temporary variables of $\mathsf{ANDEq}(X_{\overline{a_1}}, \ldots, X_{\overline{a_l}}, X_{\overline{c}})$ appropriately (as in Lemma ANDEq). It can be checked that this will ensure that $\mathsf{OREq}(X_{a_1} = a_1, \ldots, X_{a_l} = a_l, X_c = c) = 1$.

□

We will now deal with XOR equations. Unlike the previous two cases, we will only deal with the case when a bit is a XOR of two bits. Similarly, we give bilinear equations for the statement "$c = a$ XOR b" as follows.

$\mathsf{XOREq}(X_a, X_b, X_c)$:

$$f_2(1,1) - f_2(1, X_a) = X_{\overline{a}}$$
$$f_2(1,1) - f_2(1, X_b) = X_{\overline{b}}$$
$$\mathsf{OREq}(X_a, X_b, X_{c_1})$$
$$\mathsf{OREq}(X_{\overline{a}}, X_{\overline{b}}, X_{c_2})$$
$$\mathsf{ANDEq}(X_{c_1}, X_{c_2}, X_c)$$

Lemma 6. *$c = a$ XOR b iff* $\mathsf{XOREq}(X_a = a, X_b = b, X_c = c) = 1$.

Proof. Let $\mathsf{XOREq}(X_a = a, X_b = b, X_c = c) = 1$. From equation $\mathsf{OREq}(X_a, X_b, X_{c_1})$, we have that $c_1 = (a\ OR\ b)$ (where c_1 is the value of the variable X_{c_1}) and from the first two equations and from $\mathsf{OREq}(X_{\overline{a}}, X_{\overline{b}}, X_{c_2})$ we have that $c_2 = (\overline{a}\ OR\ \overline{b})$. Further, from the last equation we have that $c = c_1\ AND\ c_2$ which means that $c = (a\ OR\ b)\ AND\ (\overline{a}\ OR\ \overline{b})$. This proves that $c = (a\ XOR\ b)$.

Let $c = (a\ XOR\ b)$ then we describe how to set the temporary variables appropriately such that$\mathsf{XOREq}(X_a = a, X_b = b, X_c = c) = 1$. Set $X_{\overline{a}}$ to be 1 if $a = 0$ else set $X_{\overline{b}}$ to be 0. Appropriately set the temporary variables of the OREq and ANDEq equations as described in Lemma 5 and Lemma 4. □

We now define XOR equations for vectors of bits. We describe how to set the values of the temporary variables to satisfy the above set of equations provided that $c = a\ XOR\ b$. Set X_a (resp., X_b) to be 1 if a (resp., X_b) is 0 otherwise set X_a (resp., X_b) to be 0. Appropriately set the temporary variables of the OREq and ANDEq equations.

Let $a = (a_1, \ldots, a_m), b = (b_1, \ldots, b_m), c = (c_1, \ldots, c_m)$. Then, we can give bilinear equations for the statement "$c = a$ XOR b" (interpreted as $c_i = a_i$ XOR b_i, for all $i = 1, \ldots, m$) as follows.
$\mathsf{VecXOREq}(a, b, c)$:

$$\mathsf{XOREq}(a_i, b_i, c_i), \ldots, \mathsf{XOREq}(a_i, b_i, c_i)$$

The following lemma follows directly from Lemma 6.

Lemma 7. *A tuple of bit vectors c is a XOR of a tuple of bit vectors a and b (this means that $c_i = a_i$ XOR b_i for all i) iff* $\mathsf{VecXOREq}(X_a = a, X_b = b, X_c = c) = 1$.

Ensuring variables take values either 0 or 1.

To check whether a variable takes the value 0 or 1, we associate the following set of bilinear equations. To do this, we note that we can use OR equations directly here.

$\mathsf{BitEq}(X_b)$:

$$\mathsf{OREq}(0, 1, X_b)$$

The following lemma follows directly from Lemma 5.

Lemma 8. *An integer $b \in \mathbb{Z}_q$ is a bit iff* $\mathsf{BitEq}(X_b = b) = 1$.

To check whether a tuple correspond to bit vectors we consider the following set of equations. Consider a tuple $X_a = (X_{a_1}, \ldots, X_{a_n})$. We associate the following set of bilinear equations to check whether X_{a_i} is a bit for each $i = 1, \ldots, n$.

$\mathsf{VecBitEq}(X_a)$:

$$\mathsf{BitEq}(X_{a_1}), \ldots, \mathsf{BitEq}(X_{a_n})$$

The following lemma follows from Lemma 8.

Lemma 9. *A tuple a with elements in $\mathbb{Z}_q$ is a bit vector iff* $\mathsf{VecBitEq}(X_a = a) = 1$.

Expressing permutations as a set of bilinear equations
Consider two tuples $tuple_1 = (M_1, \dots, M_4)$ and $tuple_2 = (M'_1, \dots, M'_4)$, where M_i and M'_i, for $i = 1, \dots, 4$ are matrices of dimension $r \times s$ containing elements in the group $\mathbb{G}$. We say that $tuple_1$ is a permutation of $tuple_2$ if there exists a permutation σ on $\{1, \dots, 4\}$ such that $M_i = M'_{\sigma(i)}$. We express the statement "whether a tuple $tuple_1$ of matrices (containing elements in $\mathbb{G}$) is a permutation of another tuple $tuple_2$ of matrices" as a system of bilinear equations. We will introduce the following terminology about the variables used in the following system of equations. X_{tuple_k}, for $k = 1, 2$, is a tuple of variables $(X_M^{k,1}, \dots, X_M^{k,4})$. Further, $X_M^{k,1}$ is a matrix of variables with the $(i,j)^{th}$ element in $X_M^{i,1}$ denoted by $X_{M_{i,j}}^{i,1}$.

We will denote the $(i,j)^{th}$ element in a matrix M by $M_{i,j}$.

$\mathsf{PermEq}(X_{tuple_1}, X_{tuple_2})$: For $i \in \{1, \dots, r\}$ and $j \in \{1, \dots, s\}$

$$f_2(X_{v_1^{(1)}}, X_{M_{i,j}}^{1,1}) - f_2(X_{v_1^{(1)}}, X_{M_{i,j}}^{2,1}) = 0$$

$$\vdots$$

$$f_2(X_{v_4^{(1)}}, X_{M_{i,j}}^{1,1}) - f_2(X_{v_4^{(1)}}, X_{M_{i,j}}^{2,4}) = 0$$

$$f_2(1,1) + f_2(1, X_{v_1^{(1)}}) + \cdots + f_2(1, X_{v_4^{(1)}}) = 0$$

$$\vdots$$

$$f_2(X_{v_1^{(4)}}, X_{M_{i,j}}^{1,4}) - f_2(X_{v_1^{(4)}}, X_{M_{i,j}}^{2,1}) = 0$$

$$\vdots$$

$$f_2(X_{v_4^{(4)}}, X_{M_{i,j}}^{1,4}) - f_2(X_{v_4^{(4)}}, X_{M_{i,j}}^{2,4}) = 0$$

$$f_2(1,1) + f_2(1, X_{v_1^{(4)}}) + \cdots + f_2(1, X_{v_4^{(4)}}) = 0$$

$$f_2(1,1) + f_2(1, X_{v_1^{(1)}}) + \cdots + f_2(1, X_{v_1^{(4)}}) = 0$$

$$f_2(1,1) + f_2(1, X_{v_2^{(1)}}) + \cdots + f_2(1, X_{v_2^{(4)}}) = 0$$

$$f_2(1,1) + f_2(1, X_{v_3^{(1)}}) + \cdots + f_2(1, X_{v_3^{(4)}}) = 0$$

$$f_2(1,1) + f_2(1, X_{v_4^{(1)}}) + \cdots + f_2(1, X_{v_4^{(4)}}) = 0$$

Consider the following lemma.

Lemma 10. *A tuple $tuple_1 = (M^{1,1}, \dots, M^{4,1})$ of matrices of group elements is a permutation of another tuple $tuple_2 = (M^{2,1}, \dots, M^{2,4})$, with $M^{2,i} \neq M^{2,j}$ for $i \neq j$, iff* $\mathsf{PermEq}(X_{tuple_1} = tuple_1, X_{tuple_2} = tuple_2) = 1$.

Proof. Let $\mathsf{PermEq}(X_{tuple_1} = tuple_1, X_{tuple_2} = tuple_2) = 1$. We will show that $tuple_1$ is a permutation of $tuple_2$. To show this, we prove that for every i there is a unique j such that $X_{v_j^{(i)}} = 1$. From this we can compute a permutation map σ, with $\sigma(i) = j$ such that $tuple_1$ is a permutation of $tuple_2$. To show that for every i there is a unique j such that $X_{v_j^{(i)}} = 1$, we consider the following cases.
Case i. Let i be an index such that there does not exist any j such that $X_{v_j^{(i)}} \neq 0$:- This is not possible

because the equation $f_2(1,1) + f_2(1, X_{v_1^{(i)}}) + \cdots + f_2(1, X_{v_4^{(i)}}) = 0$ is violated.
Case ii. Let i be an index such that there exists two indices j, j' such that $X_{v_j^{(i)}} \neq 0$ and $X_{v_{j'}^{(i)}} \neq 0$:- This in turn happens only if $M^{2,j} = M^{2,j'}$ which is not possible since the elements are distinct.
Case iii. Let i and i' be two indices such that $X_{v_j^{(i)}} \neq 0$ and $X_{v_j^{(i')}} \neq 0$ for some j:-From the previous two cases, for every i there is only one j such that $X_{v_j^{(i)}} \neq 0$. This means that there exists a j' such that $X_{v_{j'}^{(k)}} = 0$ for all k which violates the equation $f_2(1,1) + f_2(1, X_{v_{j'}^{(1)}}) + \cdots + f_2(1, X_{v_{j'}^{(4)}}) = 0$.
From the above cases, it follows that for every i there is a unique j such that $X_{v_j^{(i)}} = 1$.

Let $tuple_1$ be a permutation of $tuple_2$. We will show that $\mathsf{PermEq}(X_{tuple_1} = tuple_1, X_{tuple_2} = tuple_2) = 1$. We will describe how to set the temporary variables in the above system of equations as follows. Let σ be a permutation on $\{1, \ldots, 4\}$ such that $M^{1,i} = M^{1,\sigma(i)}$. Set $X_{v_i^{\sigma(i)}}$ to be $q-1$ and set $X_{v_i^j}$ to be 0 for $j \neq \sigma(i)$ for all $i = 1, \ldots, 4$. Observe that this satisfies the system of equations. □

Equations for the output wires
We write bilinear equations to check whether "$L^1_{w_m,0} = (0^{1\times l})$ [16] and $L^1_{w_m,1} = (1|0^{1\times(l-1)})$". Represent the vector $(0^{1\times l})$ by $\overrightarrow{v}_1$ and the vector $(1||0^{1\times(l-1)})$ by $\overrightarrow{v}_2$. Further, let $L_{w_m,b} = (L^1_{w_m,b}, \ldots, L^l_{w_m,b})$ for $b = 0, 1$. Before we describe the equations for the output wires, we define the following set of equations.

$$\mathsf{EqualEq}(X_1, X_2) \ : \ f_2(1, X_1) - f_2(1, X_2) = 0$$

It is easy to see that $\mathsf{EqualEq}(X_1, X_2)$ is satisfied with respect to an assignment (a, b) iff $a = b$. We now describe the equations $\mathsf{OutWiresEq}$ which are defined for the output wires.

$\mathsf{OutWiresEq}(X_{L_{w_m,0}}, X_{L_{w_m,1}})$:

$$\mathsf{EqualEq}(X_{L^1_{w_m,0}}, 0), \ldots, \mathsf{EqualEq}(X_{L^l_{w_m,0}}, 0)$$

$$\mathsf{EqualEq}(X_{L^1_{w_m,1}}, 1), \mathsf{EqualEq}(X_{L^1_{w_m,1}}, 0), \ldots, \mathsf{EqualEq}(X_{L^{1,l}_{w_m,1}}, 0)$$

The following lemma easily follows.

Lemma 11. *The wire keys $L_{w_m,0}$ and $L_{w_m,1}$ are vectors $(0^{1\times l})$ and $(1||0^{1\times(l-1)})$ respectively iff* $\mathsf{OutWiresEq}(X_{L_{w_m,0}} = L_{w_m,0},\ X_{L_{w_m,1}} = L_{w_m,1})$.

Equations for rerandomization We write bilinear equations to check whether a set of ciphertexts is a rerandomization of a different set of ciphertexts. We first write bilinear equations for the case when there is a single ciphertext and then we handle the case when there are many ciphertexts. Consider two ciphertexts $c_1 = (c_1^{(1)}, \ldots, c_l^{(l)})$ and $c_2 = (c_2^{(1)}, \ldots, c_2^{(l)})$. Let R be the vector denoting $(r_1, \ldots, r_l)$. Let g be a group element in $\mathbb{G}$.

$\mathsf{reRandCTEq}(X_{c_1}, X_{c_2}, X_\Delta, X_R)$:

$$X_{c_1'} = f_1(X_{r_1}, X_{\Phi_{1,1}}) + \cdots + f_1(X_{r_l}, X_{\Phi_{1,l}}) - X_{c_1^{(1)}}$$

$$\vdots$$

$$X_{c_l'} = f_1(X_{X_{r_1},\Phi_{l,1}}) + \cdots + f_1(X_{X_{r_l},\Phi_{l,l}}) - X_{c_1^{(1)}}$$

$$X_{c_{l+1}'} = f_1(X_{r_1}, X_{\Phi_{l,1}}) + \cdots + f_1(X_{r_l}, X_{\Phi_{l,l}}) + f_1(0, X_g) + X_{c_1^{(l+1)}}$$

[16] $0^{1\times l}$ denotes a row vector of length l.

$$X_{c_1''} = f_1(X_{r_1}, X_{\Phi_{1,1}}) + \cdots + f_1(X_{r_l}, X_{\Phi_{1,l}}) - X_{c_1^{(1)}}$$

$$\vdots$$

$$X_{c_l''} = f_1(X_{r_1}, X_{\Phi_{l,1}}) + \cdots + f_1(X_{r_l}, X_{\Phi_{l,l}}) - X_{c_l^{(l)}}$$

$$X_{c_{l+1}''} = f_1(X_{r_1}, X_{\Phi_{l,1}}) + \cdots + f_1(X_{r_l}, X_{\Phi_{l,l}}) + f_1(1, X_g) - X_{c_1^{(l+1)}}$$

$$X_{\Delta'} = f_2(1,1) - f_2(1, X_\Delta)$$

$$X_{c_2^{(1)}} = f_2(X_\Delta, c_1^{'}) + f_2(X_{\Delta'}, c_1^{''})$$

$$\vdots$$

$$X_{c_2^{(l+1)}} = f_2(X_\Delta, X_{c_{l+1}^{'}}) + f_2(X_{\Delta'}, X_{c_{l+1}^{''}})$$

Before we state the following lemma, we introduce the following terminology: A ciphertext c_1 is a valid rerandomization of c_2 if c_1 can be written in either of the following forms - $R\Phi + c_2$ or $R\Phi - c_2$ for some matrix R of dimension $l \times l$ with elements in $\mathbb{Z}_q$. A ciphertext c_1 is said to be a by valid XORing with b of c_2 if c_2 is obtained by XORing a bit b into the message of c_1. Consider the following lemma.

Lemma 12. *Let c_1 be a ciphertext whose decryption is a bit m. Then, c_2 is a ciphertext whose decryption is $m \oplus \Delta$ iff* $\mathsf{reRandCTEq}(X_{c_1} = c_1, X_{c_2} = c_2, X_\Delta = \Delta, X_r = R) = 1$.

Proof. $\mathsf{reRandCTEq}(X_{c_1} = c_1, X_{c_2} = c_2, X_\Delta = \Delta, X_r = r) = 1$. Let m be the descryption of c_1. Let $c' = (R\Phi - c)$ and let $c'' = R\Phi$. Note that c' and c'' also decrypt to m. Now, consider the following quantity $\Delta(r\Phi + (0 \cdots 0\ 1.g) - c_1) + (1 - \Delta)(r\Phi + (0 \cdots 0\ 0.g) - c_1)$ which is essentially c_2. Further, c_2 is $r\Phi + (0 \cdots 0\ 1.g) - c_1$ if $\Delta = 0$ and $(r\Phi + (0 \cdots 0\ 0.g) - c_1)$ if $\Delta = 1$. In other words, c_2 decrypts to $1 - m$ if $\Delta = 1$ and it decrypts to m if $\Delta = 0$. This further means that c_2 decrypts to $m \oplus \Delta$.

Let the decryption of c_1 be a bit m and let the decryption of c_2 be $m \oplus \Delta$. We show that $\mathsf{reRandCTEq}(X_{c_1} = c_1, X_{c_2} = c_2, X_\Delta = \Delta, X_r = r) = 1$. We describe how to set the temporary variables to satisfy the system of equations. Set $X_{c_1^{'}}$ to be $r\Phi + (0 \ldots 01.g) - c_1$ and set $X_{c_1^{''}}$ to be $r\Phi - c_1$. Set $X_{\Delta'}$ to be $1 - \Delta$. Observe that this will satisfy the system of equations.

□

We now describe the bilinear equations to express the statement when a set of ciphertexts is a rerandomization of a different set of ciphertexts. Let $C = (c_1, \ldots, c_l)$ and $C' = (C_1^{'}, \ldots, C_l^{'})$ be two tuples of ciphertexts, let $\Delta = (\Delta_1, \ldots, \Delta_l)$ be a tuple of bits and let $R = (R_1, \ldots, R_l)$ denote a tuple of random vectors.

$\mathsf{VecreRandCTEq}(C, C', \Delta, R)$:

$$\mathsf{reRandCTEq}(c_1, c_1', \Delta_1), \ldots, \mathsf{reRandCTEq}(c_l, c_l', \Delta_l)$$

Lemma 13. *The tuple of ciphertexts C is obtained from first rerandomizing C' and then XORing the values of Δ to all messages in the ciphertexts in C' iff the above system of equations has a solution.*

C.2 Correctness of Garbled Circuit construction using GS proofs

In this section, we describe an algorithm that computes a proof, which is just composed of Groth-Sahai proofs, to verify whether a given garbled circuit is as computed by the garbled circuit generation algorithm from Section B.2.1.

Building over the tools built in the previous section, we will now express the garbled circuit construction as a system of bilinear equations. Before that, we will describe the main variables used in the equations.

1. For every $i \in \{1, \ldots, m\},\ b \in \{0,1\}$, we have $X_{L_{w_i,b}}$ denote a vector of variables of length l. We denote by $X_{\mathsf{pk}_{w_i,b}}$, a matrix of variables of size $l \times l$.
2. For every gate G, $i,j \in \{0,1\}$ and $k = 1,2$, we have matrices of variables namely $X_{c^{G,k}_{(i,j)}}$ of dimension $(l+1) \times l$, $X_{\mathsf{ct}^{G,k}_{(i,j)}}$ of dimension $(l+1) \times l$, $X_{\Delta^{G,k}_{(i,j)}}$ of dimension $1 \times l$ and $X_{r^{G,k}_{i,j}}$ of dimension $l \times l$.
3. For every gate G, $i,j \in \{0,1\}$ and $k = 1,2$, we have matrices of variables namely $X_{\mathsf{ct}^{\mathsf{reRand},G,k}_{(i,j)}}$ of dimension $(l+1)\times l$ and $X_{r^{\mathsf{reRand},G,k}_{i,j}}$ of dimension $l\times l$ and a vector of variables $X_{\Delta^{\mathsf{reRand},G,k}_{i,j}}$ of dimension $1 \times l$.
4. For every gate G and $i,j \in \{0,1\}$, we have $X_{\delta^G_{i,j}}$ and $X_{\mu^G_{i,j}}$, where $X_{\delta^G_{i,j}}$ and $X_{\mu^G_{i,j}}$ are vectors of variables of length $2l$.

We now describe the system of bilinear equations that captures the garbled circuit construction by Gentry et. al. We denote the system of equations by $\mathsf{GCktEq}(C)$.

$\mathsf{GCktEq}(C)$:
For every $i \in \{1, \ldots, m\},\ b \in \{0,1\}$:

$$\mathsf{PKEq}(X_{L_{\mathsf{pk}_{w_i,b}}}, X_{L_{w_i,b}})$$

$$\mathsf{VecBitEq}(X_{L_{w_i,b}})$$

For every gate G with input wires w_1, w_2 and output wire w_3, for all $i,j \in \{0,1\}$:

$$\mathsf{VecCTEq}(X_{c^{G,1}_{(i,j)}}, X_{L_{w_1,i}}{}^2, X_{r^{G,1}_{i,j}}, X_{\delta^G_{i,j}}), \mathsf{VecCTEq}\Big(X_{c^{G,2}_{i,j}}, X_{L^2_{w_2,j}}, X_{r^{G,2}_{i,j}}, (X_{\mu^G_{i,j}}, X_{L^2_{w_3,G(i,j)}})\Big)$$

$$\mathsf{VecreRandCTEq}(X_{\mathsf{ct}^{G,1}_{i,j}}, X_{\mathsf{ct}^{\mathsf{reRand},G,1}_{i,j}}, X_{\Delta^{\mathsf{reRand},G,1}_{i,j}}, X_{r^{\mathsf{reRand},G,1}_{i,j}})$$

$$\mathsf{VecreRandCTEq}(X_{\mathsf{ct}^{G,2}_{i,j}}, X_{\mathsf{ct}^{\mathsf{reRand},G,2}_{i,j}}, X_{\Delta^{\mathsf{reRand},G,2}_{i,j}}, X_{r^{\mathsf{reRand},G,2}_{i,j}}),$$

$$\mathsf{VecBitEq}(X_{\delta_{i,j}}),$$

$$\mathsf{VecBitEq}(X_{\mu^G_{i,j}})$$

$$\mathsf{VecBitEq}(X_{\Delta^{\mathsf{reRand},G,1}_{i,j}})$$

,

$$\mathsf{VecBitEq}(X_{\Delta^{\mathsf{reRand},G,2}_{i,j}}),$$

For every gate G:

$$\mathsf{PermEq}(X_{\mathsf{ct}^G_1}, X_{\mathsf{ct}^G_2}, X_{\mathsf{ct}^G_3}, X_{\mathsf{ct}^G_4}, (X_{c^G_1}, X_{c^G_2}, X_{c^G_3}, X_{c^G_4})),$$

where $\mathsf{ct}^G_1 = (\mathsf{ct}^{G,1}_{0,0}, \mathsf{ct}^{G,2}_{0,0}), \ldots, \mathsf{ct}^G_4 = (\mathsf{ct}^{G,1}_{1,1}, \mathsf{ct}^{G,2}_{1,1}))$ and $c^G_1 = (c^{G,1}_{0,0}, c^{G,2}_{0,0}), \ldots, \mathsf{ct}^G_4 = (c^{G,1}_{1,1}, c^{G,2}_{1,1}))$. Let m be the index of the output wire, we have

$$\mathsf{OutWiresEq}(L_{w_i,0}, L_{w_i,1})$$

We now describe the procedure $\mathsf{Prover}_{\mathsf{MalGC}}$ which takes as input a garbled circuit along with the wire keys and the randomness (used to generate the ciphertexts) and produces a proof that the garbled circuit is correctly computed. Additionally it also takes as input a CRS produced from the $\mathsf{GSSetup}$ phase, Before we describe $\mathsf{Prover}_{\mathsf{MalGC}}$, we describe the garbled circuit generation phase mainly to fix the notation to be used in $\mathsf{Prover}_{\mathsf{MalGC}}$ algorithm.

The garbled circuit generation phase, which we term as InpGen, takes as input a circuit C, an input to the circuit w and then does the following. It computes a garbled circuit corresponding to C according to the construction given in Section B.2.1. It first picks wire keys $L_{w_i,b}$ for all $i = 1, \ldots, m$ and $b = 0, 1$. Construct the table of ciphertexts as follows as given in Section B.2. Each entry of the table, denoted by $c^G_{(i,j)}$ corresponding to the gate G consists of two ciphertexts $c^{G,1}_{(i,j)} = \mathsf{Enc}_{L_{(w_1,i)}}(\delta^G_{i,j})$ and $c^{G,2}_{(i,j)} = \mathsf{Enc}_{L_{(w_2,j)}}(\mu^G_{i,j})$ for $i, j \in \{0, 1\}$ where $\mu^G_{i,j} = (L_{(w_3,G(i,j))}|0^l) \oplus \delta^G_{i,j}$. Let $c^{G,1}_{(i,j)}$ and $c^{G,2}_{(i,j)}$ be encrypted using randomness $r^{G,1}_{i,j}$ and $r^{G,2}_{i,j}$ respectively. For each gate in the circuit, the table of ciphertexts constructed need to be permuted. Consider a table corresponding to gate G. After permuting the table, we denote the entries in the table, in order, by $\Big((\mathsf{ct}^{G,1}_{0,0}, \mathsf{ct}^{G,2}_{0,0}), (\mathsf{ct}^{G,1}_{0,1}, \mathsf{ct}^{G,2}_{0,1}), (\mathsf{ct}^{G,1}_{1,0}, \mathsf{ct}^{G,2}_{1,0}), (\mathsf{ct}^{G,1}_{1,1}, \mathsf{ct}^{G,2}_{1,1})\Big)$. For $\mathsf{ct}^{G,k}_{i,j}$, for $i, j \in \{0, 1\}$, compute the following. For every a, b^{th} ciphertext in $\mathsf{ct}^{G,j}_{i}$, denoted by $c_{a,b}$, compute $\Delta(c_{a,b}+r^{G,k}_{i,j}\Phi)+(1-\Delta)(r^{G,k}_{i,j}\Phi+c_{a,b})$ to obtain $c'_{a,b}$, which is $(a, b)^{th}$ ciphertext in $\mathsf{ct}^{\mathsf{reRand},G,k}_{i,j}$, by setting $r^G_{i,j,k} = 0$ (here 0 is interpreted as a vector of zeroes of length $2l$) and $\Delta(\mathsf{ct}^{G,k}_{i,j}$ to be 1. For every gate G, we represent the table of ciphertexts, by T_G, the vector $\Big((\mathsf{ct}^{\mathsf{reRand},G,1}_{(0,0)}, \mathsf{ct}^{\mathsf{reRand},G,2}_{(0,0)}), \ldots, (\mathsf{ct}^{\mathsf{reRand},G,1}_{(1,1)}, \mathsf{ct}^{\mathsf{reRand},G,2}_{(1,1)})\Big)$. Now, this completes the execution of the garbled circuit construction. Denote by GC, the set of the tables T_G, for every gate G in the circuit. Further, denote by in_w, the input wires corresponding to w. More precisely, if w is represented as $b_1 \cdots b_n$, where b_i is a bit, then in_w is $(L_{w_1,b_1}, \ldots, L_{w_n,b_n})$. We denote the garbled circuit by GC, with the wire keys in_w embedded into it, and the wire keys along with the randomness used to generate the ciphertexts to be w'_{GC}.

Procedure $\mathsf{Prover}_{\mathsf{MalGC}}((C, \mathsf{GC}), w'_{\mathsf{GC}}, CRS)$:

1. It computes a system of bilinear equations $\mathsf{GCktEq}(C)$. It then assigns values to the variables in the equations so as to satisfy all the equations in $\mathsf{GCktEq}(C)$. Whenever we say that a tuple of elements A is assigned to a vector of variables X_A, we mean that the i^{th} element in A is assigned to the i^{th} variable in X_A [17].

   (a) For every $i \in \{1, \ldots, m\}$, $b \in \{0, 1\}$, we assign $\mathsf{pk}'_{w_i,b}$ to $X_{L_{\mathsf{pk}_{w_i,b}}}$, where $\mathsf{pk}'_{w_i,b}$ is the matrix $\mathsf{pk}_{w_i,b}$, without the last column. Similarly, we assign $L_{w_i,b}$ to the variable $X_{L_{w_i,b}}$.

   (b) For every $i \in \{1, \ldots, m\}$, $b \in \{0, 1\}$, we assign the last $l^2 + l$ bits of $L^2_{w_i,b}$ to the variable $X_{L^2_{w_i,b}}$.

   (c) For every gate G, $i, j \in \{0, 1\}$ and $k = 1, 2$, we assign $c^{G,1}_{(i,j)}, \mathsf{ct}^{G,1}_{(i,j)}$ and $r^k_{i,j}$ to the variables $X_{c^{G,1}_{(i,j)}}, X_{\mathsf{ct}^{G,1}_{(i,j)}}$ and $X_{r^k_{i,j}}$ respectively.

   (d) For every gate G, $i, j \in \{0, 1\}$ and $k = 1, 2$, we assign $\mathsf{ct}^{\mathsf{reRand},G,k}_{(i,j)}, \Delta^{\mathsf{reRand},G,k}_{(i,j)}$ and $r^{\mathsf{reRand},G,k}_{i,j}$ to the variables $X_{\mathsf{ct}^{\mathsf{reRand},G,k}_{(i,j)}}, X_{\Delta^{\mathsf{reRand},G,k}_{(i,j)}}$ and $X_{r^{\mathsf{reRand},G,k}_{i,j}}$ respectively.

   (e) For every gate G and $i, j \in \{0, 1\}$, we assign $\delta^G_{i,j}$ and $\mu^G_{i,j}$ to the variables $X_{\delta^G_{i,j}}$ and $X_{\mu^G_{i,j}}$ respectively.

   $\mathsf{Prover}_{\mathsf{MalGC}}$ then generates a proof Π by executing $\mathsf{GSProver}$ on input $\mathsf{GCktEq}(C), CRS$ and assignment to the variables as described before.

[17]We sometimes abuse the notation and assign a matrix of values to a vector of variables in the following way, where that the size of the matrix and the length of the vector is the same. Let A be a matrix of size $m \times n$ (m is the number of columns and n is the number of rows) and let X_A be a tuple of variables of size N, where $N = m \times n$. When we say A is assigned to X_A we mean that the $(i, j)^{th}$ value in A is assigned to $(ni + j)^{th}$ variable in X_A.

The procedure $\mathsf{Prover}_{\mathsf{MalGC}}$ finally outputs $(\mathsf{comm}, \Pi_{\mathsf{GC}})$.

C.3 Construction of $\mathbb{P}_{\mathsf{sub}}$

We are now ready to present the description of our NIZKPoK proof system $\mathbb{P}_{\mathsf{sub}}$.

CRS generation $\mathsf{CRSSetup}_{\mathsf{sub}}(1^k)$

Execute $\mathsf{GSSetup}(1^k)$ to obtain CRS.
Output CRS.

Prover $P_{\mathsf{sub}}\Big((C, \mathsf{GC}), (w'_{\mathsf{GC}}), CRS\Big)$

$\pi \leftarrow \mathsf{Prover}_{\mathsf{MalGC}}((C, \mathsf{GC}), w'_{\mathsf{GC}}, CRS)$.
Output π.

Verifier $V_{\mathsf{sub}}\Big((C, \mathsf{GC}), \pi, CRS\Big)$

It first parses the garbled circuit as $\mathsf{GC} = (\mathsf{GC}^*, \mathsf{in}_w)$, where in_w is the wire keys corresponding to w. The verifier rejects if any of the following conditions are satisfied. Else, it accepts the proof π.

- If any two ciphertexts in the garbled circuit GC^*.
- If the evaluation of GC^* on in_w results in a vector which is not $(1|0^{1\times(l-1)})$.
- If $\mathsf{GSVerifier}((C, \mathsf{GC})\pi, CRS)$ outputs Reject.

This completes the description of the proof system $\mathbb{P}_{\mathsf{sub}}$.

C.4 Proof of Security of $\mathbb{P}_{\mathsf{sub}}$

It remains to show that the proof system $\mathbb{P}_{\mathsf{sub}}$ is a NIZKPoK. That is, it satisfies the properties of completeness, proof of knowledge and zero knowledge.

Theorem 6. *$\mathbb{P}_{\mathsf{sub}}$ satisfies completeness.*

Proof. Let (C, GC) be the common input of both the prover and the verifier. We show that $(P_{\mathsf{sub}}, V_{\mathsf{sub}})$ satisfies completeness as follows. Firstly, we show that the randomness along with w (such that $C(w) = 1$) used in the garbled circuit computation satisfies all the equations in $\mathsf{GCktEq}(C)$. Then, we show that the completeness property of the GS proofs implies the completeness of $(P_{\mathsf{sub}}, V_{\mathsf{sub}})$.

We show that the assignment of the values for the variables in GCktEq as described in $\mathsf{Prover}_{\mathsf{MalGC}}$ satisfies all the equations in $\mathsf{GCktEq}(C)$.

1. For every $i \in \{1, \ldots, m\}$ and from Lemma 1 and the assignment of the variables $X_{L_{w_i,b}}$ and $X_{\mathsf{pk}_{L_{w_i,b}}}$ it follows that the equations $\mathsf{PKEq}(X_{\mathsf{pk}_{L_{w_i,b}}}, X_{L_{w_i,b}})$ are satisfied. Further, from Lemma 9 and the assignment of the variables $X_{L_{w_i,b}}$ it follows that the equations $\mathsf{VecBitEq}(X_{L^1_{w_i,b}})$ are satisfied.

2. For every gate G with input wires w_1, w_2 and output wire w_3, for all $i, j \in \{0, 1\}$ and $k = 1, 2$ we have the following.
   - From Lemma 3 and the assignment of the variables $X_{c^{G,k}_{i,j}}, X_{\mathsf{pk}_{L_{w_i,j}}}, X_{r^{G,k}_{i,j}}$ and $X_{\delta^{G}_{i,j}}$), the equations $\mathsf{VecCTEq}(X_{c^{G,k}_{(i,j)}}, X_{\mathsf{pk}_{L_{w_1,i}}}, X_{r^{G,k}_{i,j}}, X_{\delta^{G,k}_{i,j}})$ are satisfied.
   - From Lemma 13 and the assignment of the variables $X_{\mathsf{ct}^{G,k}_{i,j}}, X_{\mathsf{ct}^{\mathsf{reRand},G,k}_{i,j}}, X_{\Delta^{\mathsf{reRand},G,k}_{i,j}}$ and $X_{r^{\mathsf{reRand},G,k}_{i,j}}$ the equations $\mathsf{VecreRandCTEq}(X_{\mathsf{ct}^{G,k}_{i,j}}, X_{\mathsf{ct}^{\mathsf{reRand},G,k}_{i,j}}, X_{\Delta^{\mathsf{reRand},G,k}_{i,j}}, X_{r^{\mathsf{reRand},G,k}_{i,j}})$, are satisfied.

- From Lemma 9 and the assignment of the variables $X_{\delta_{i,j}}$ and $X_{\mu^G_{i,j}}$ it follows that the equations $\mathsf{VecBitEq}(X_{\delta_{i,j}})$, and $\mathsf{VecBitEq}(X_{\mu^G_{i,j}})$ are satisfied.
- From Lemma 9 and the assignment of the variables $X_{\Delta^{\mathsf{reRand},G,k}_{i,j}}$ the equations $\mathsf{VecBitEq}(X_{\Delta^{\mathsf{reRand},G,1}_{i,j}}, X_{\Delta^{\mathsf{reRand},G,2}_{i,j}})$, are satisfied.

3. For every gate G we have the following. From Lemma 10 and the assignment of the variables $X_{c^{G,k}_{i,j}}$, $X_{\mathsf{ct}^{G,k}_{i,j}}$, for every $i, j \in \{0, 1\}$ and $k = 1, 2$ the equations $\mathsf{PermEq}(X_{\mathsf{ct}^G_1}, X_{\mathsf{ct}^G_2}, X_{\mathsf{ct}^G_3}, X_{\mathsf{ct}^G_4}, (X_{c^G_{0,0}}, X_{c^G_{0,1}}, X_{c^G_{1,0}}, X_{c^G_{1,1}}))$ are satisfied.
4. From Lemma 11 and the assignment of the variables $X_{L_{w_m,0}}$ and $X_{L_{w_m,0}}$ the equation $\mathsf{OutWiresEq}(X_{L_{w_m,0}}, X_{L_{w_m,1}})$ is satisfied.

   $\mathsf{ct}^G_1, \ldots, \mathsf{ct}^G_4$ is nothing but a permutation of $c^G_{0,0}, \ldots, c^G_{1,1}$.

Now, we have showed that the assignment of the variables in the equations of $\mathsf{Prover}_{\mathsf{MalGC}}$ satisfies all the equations in $\mathsf{GCktEq}(C)$. We have the following from the perfect completeness of the GS proofs. Here, w' denotes the pair (w, R) such that $\mathsf{reRandGC}(C, w; R)$ outputs GC.

$$Pr[\mathsf{GSVerifier}(\mathsf{GCktEq}(C), CRS, \pi) = 1 \;:\; \mathsf{GSSetup}(1^k) \to CRS;\; P_{\mathsf{sub}}((C, \mathsf{GC}), w', CRS) \to \pi] = 1$$

Further, from the garbled circuit construction, the probability that any two ciphertexts are the same is negligible. Also, from the correctness of the garbled circuit construction it follows that the garbled circuit outputs 1 (which is actually the vector $(1|0^{1-l})$) with overwhelming probability. Combining all these facts we have the following.

$$Pr[V_{\mathsf{sub}}((C, \mathsf{GC}), CRS, \pi) = 1 \;:\; \mathsf{CRSSetup}_{\mathsf{sub}}(1^k) \to CRS;\; P_{\mathsf{sub}}((C, \mathsf{GC}), w', CRS) \to \pi] \geq 1 - negl(k)$$

□

Theorem 7. $\mathbb{P}_{\mathsf{sub}}$ *satisfies proof of knowledge.*

Proof. In order to prove the proof of knowledge property, we first construct an extractor that can extract a witness for an instance with non-negligible probability if there exists a PPT adversary who produces a convincing proof for the same instance with non-negligible probability. We construct the extractor as follows. The extractor on input $((C, \mathsf{GC}), \pi, CRS, \tau_{ext})$, where τ_{ext} is an extraction trapdoor of CRS, does the following: It first extracts the values of all the variables in $\mathsf{GCktEq}(C)$ by running the extractor of the GS proof system using the trapdoor τ_{ext}. Denote by values, the values of all the variables obtained during the extraction process. Let $\Big((L_{w_1,0}, L_{w_1,1}),\ \ldots,\ (L_{w_m,0}, L_{w_m,0})\Big)$ be the extracted wire keys. Determine $b_1, \ldots, b_n$ such that $\mathsf{in}_w = (L_{w_1,b_1},\ \ldots,\ L_{w_m,b_m})$. Output $w' = (w, \mathsf{values})$ as the witness.

We claim that with non-negligible probability, the output of the extractor, namely w' is a valid witness for (C, GC). To see this, first note that the proof of extractability property of the GS proofs implies that values satisfies $\mathsf{GCktEq}(C)$ with non-negligible probability. Conditioned on the event that values satisfy the equations we prove that we can extract w with probability negligibly close to 1. Before we show this we introduce some terminology. We say that a garbled circuit GC' is a valid garbled circuit if $\mathsf{GC}' = \mathsf{reRandGC}(C; R)$ for some randomness R. First, we claim that GC is a valid garbled circuit. To see this, consider the following cases.

Case 1. Computation of the wire keys: Consider a wire key w_i for $i = 1, \ldots, m$. We will first consider the case when $i \neq m$. From Lemma 1 and from the equations $\mathsf{PKEq}()$ it follows that the public key $\mathsf{pk}_{L_{w_i,b}}$, for $b = 0, 1$, is computed correctly. That is, the last column of the public key is obtained by multiplying the first l columns of the matrix representing the public key by the secret key. For $i = m$, from Lemma 11 and

from the equations $\mathsf{OutWiresEq}()$ it follows that the output wires $L_{w_m,0}$ and $L_{w_m,1}$ are the vectors $(0^{1\times l})$ and $(1|0^{1\times(l-1)})$ respectively. Further, from Lemma 8 and from the equations $\mathsf{VecBitEq}()$ it follows that all the wire keys are bit vectors of length l.
Case 2. Table of ciphertexts: Consider a gate G in the circuit. Let w_1, w_2 be the input wires of G and let w_3 be the output wire of G. Let T_G be the table of ciphertexts associated with gate G. Consider the following table of ciphertexts.

$$T_G^{orig} = \begin{bmatrix} c_{0,0}^{G,1} & c_{0,0}^{G,2} \\ \vdots & \\ c_{1,1}^{G,1} & c_{1,1}^{G,2} \end{bmatrix}$$

Let $\delta_{i,j}^G$ (resp., $\mu_{i,j}^G$) be the decryption of $c_{i,j}^{G,1}$ (resp., $c_i^{G,2}$) for every $i = 1, \ldots, 4$. Now, from Lemma 8 and from the equations $\mathsf{VecBitEq}()$ it follows that $\delta_{i,j}$ and $\mu_{i,j}$ are bit vectors. Also, from Lemma 7 and the equations $\mathsf{VecXOREq}$ it follows that $\delta_{i,j} \oplus \mu_{i,j}$ is $(L_{w_3,G(i,j)}||0^{1\times l})$.We now show that the table T is correctly computed. Before we show this, we introduce the following terminology. We say that a table T of ciphertexts is a valid table of ciphertexts if T can be obtained from T_G^{orig} by performing any of the following two operations on T_G^{orig} (in no particular order).

1. $(i,j)^{th}$ ciphertext in T_G is a valid rerandomization of the $(i,j)^{th}$ ciphertext in the table T_G^{orig}.

2. Permuting the rows of the table T_G^{orig}.

Observe that in a garbled circuit output by $\mathsf{reRandGC}$ the tables associated to all the gates are valid tables. We claim that the table T_G is a valid table of ciphertexts. From Lemma 13 and the equations $\mathsf{VecreRandCTEq}()$ it follows that T_G is obtained from another table of ciphertexts, namely T'_G, by rerandomizing all the ciphertexts in T'_G. Now, from Lemma 10 and the equations $\mathsf{PermEq}()$ it follows that T'_G is obtained by permuting the rows of T_G^{orig}. These two facts prove that T_G is indeed a valid table of ciphertexts. This shows that in GC the table of ciphertexts associated to all the gates are valid table of ciphertexts.

From the above two cases it follows that values extracted by the extractor corresponds to the randomness used in the computation of GC. We now show how to extract w such that $C(w) = 1$. Now, we use the fact that given $(\mathsf{GC}, \mathsf{in}_w)$ and all the wire keys of GC such that GC is a valid garbled circuit we can extract w such that the output of $C(w)$ is the same as the output of $\mathsf{GC}(\mathsf{in}_w)$ [18]. In this case, since the output of GC corresponds to 1 we can extract w from $(\mathsf{GC}, \mathsf{in}_w)$ such that $C(w) = 1$. This completes the proof. □

Theorem 8. *$\mathbb{P}_{\mathsf{sub}}$ satisfies zero-knowledge.*

Proof. To show that $(P_{\mathsf{sub}}, V_{\mathsf{sub}})$ is a zero knowledge proof system, we construct a simulator $\mathsf{Sim}_{\mathsf{reRand}} = (\mathsf{Sim}^{(1)}_{\mathsf{reRand}}, \mathsf{Sim}^{(2)}_{\mathsf{reRand}})$ such that the output distribution of $\mathsf{Sim}_{\mathsf{reRand}}$ is computationally indistinguishable from the output distribution of the prover P_{sub}. The simulator $\mathsf{Sim}_{\mathsf{reRand}}$ works as follows. $\mathsf{Sim}^{(1)}_{\mathsf{reRand}}$ on input security parameter k, executes $\mathsf{Sim}_{gsproofs}(1^k)$ ($\mathsf{Sim}_{gsproofs}$ is the zero knowledge simulator of the GS proof system) to obtain (CRS, τ_{sim}). $\mathsf{Sim}^{(2)}_{\mathsf{reRand}}$ on input $((C, \mathsf{GC}), CRS, \tau_{sim})$ does the following. Let $\mathsf{YaoGCSim}$ denote the simulator for the alternate garbled circuit construction given in Appendix B.2.

- It first executes $\mathsf{Sim}_{gsproofs}((C, \mathsf{GC}), CRS, \tau_{sim})$ to obtain Π_{Sim}.

- It finally outputs Π_{Sim}.

The computational indistinguishability of the output distribution of $\mathsf{Sim}_{\mathsf{reRand}}$ from the output distribution of P_{sub} directly follows from the zero knowledge property of the GS proof system. This completes the proof. □

[18]This is not entirely true since the output of the garbled circuit is a vector but the output of C is a bit. Hence, by the output of GC and the output of C being the same we mean that if the output of GC is $0\cdots0$ then output of c is 1 and if the output of GC is $1\ 0\cdots0$ then the output of C is 1.

D Construction of $\mathbb{P}$ using $\mathbb{P}_{\mathsf{sub}}$

We use the sub-proof system $\mathbb{P}_{\mathsf{sub}}$ described in the previous section to construct $\mathbb{P}$ which is defined for the circuit satisfiability relation R. The sketch of the construction of $\mathbb{P}_2$ is given below. We denote by InpGen, the algorithm that takes as input (C, w) and generates the garbled circuit for C as given in Appendix B.2.1 along with wire keys for w. The output of InpGen is the garbled circuit along with randomness used in generating the garbled circuit.

D.1 Construction

We describe the proof system $\mathbb{P}_2$ by the algorithms $(\mathsf{CRSSetup}, P, V)$:

CRS generation $\mathsf{CRSSetup}(1^k)$: We execute the CRS generation algorithm of $\mathbb{P}_{\mathsf{sub}}$, denoted by $\mathsf{CRSSetup}_{\mathsf{sub}}$, to obtain CRS. Output CRS.

Prover $P(C, w, CRS)$: First execute InpGen on input (C, w) to obtain (GC, w^*). It then executes the prover of $\mathbb{P}_{\mathsf{sub}}$ on input $((C, \mathsf{GC}), w' = (w||w^*), CRS)$ to obtain proof π. Send $\Pi = ((C, \mathsf{GC}), \pi)$ to the verifier.

Verifier $V(C, \Pi, CRS)$: First, parse Π as $((C', \mathsf{GC}), \pi)$. Accept the proof π only if the following checks are satisfied:

- $C' = C$.
- The verifier of $\mathbb{P}_{\mathsf{sub}}$, namely $V_{\mathsf{sub}}((C, \mathsf{GC}), \pi, CRS)$, outputs accept.

We now prove that the above described system is a decomposable SP-NIZK. We first prove the sub-proof property, then we show the rerandomizability property and in the end we show the large min-entropy property.

D.2 Sub-proof property

We first denote the relation R_{gc} as consisting of all instances of the form (C, GC) whose witness w' consists of two parts: the prefix of w' is w, which is such that $C(w) = 1$ and the second part is the randomness used to generate GC using the garbled circuit generation algorithm given in Section B.2.1 on input (C, w). We prove all the three parts of the sub-proof property below.

1. $\mathbb{P}_{\mathsf{sub}}$ is a NIZKPoK for the relation R_{gc}:- This was already shown in Section C.4.
2. Let CRS be the output of $\mathsf{CRSSetup}$ of $\mathbb{P}_2$. Let Π be a proof produced by the prover of $\mathbb{P}_2$ on input instance x and witness w. We have $V(x, \Pi, CRS)$ accepts if and only if x is a prefix of y and V_{sub} also accepts, where V is the verifier of $\mathbb{P}_2$:- This directly follows from the construction of V.
3. We now prove two claims.
   (i) For every $(C, w) \in R$ and every $(\mathsf{GC}, w') \leftarrow \mathsf{InpGen}(C, w)$, it holds that $(\mathsf{GC}, w') \in R_{gc}$:- This follows from the correctness of the garbled circuit construction.
   (ii) For every $((C, \mathsf{GC}), w') \in R_{gc}$, there exists a unique and efficiently computable w s.t. $(C, w) \in R$:- To see this, notice that the prefix of w' is w such that $C(w) = 1$. Thus given w', we can efficiently compute w.

D.3 Rerandomizability property of $\mathbb{P}$

We first argue about the malleability properties offered by GS proofs. We then use this along with the rerandomizability properties of the garbled circuits to show that $\mathbb{P}$ satisfies rerandomizability property.

D.3.1 Malleability of GS proofs

In [CKLM12], it was shown that GS proofs are malleable with respect to different operations. Here we will list only a few operations, those which are useful to our construction, with respect to which GS proofs are malleable. As mentioned before, we are interested in the DLIN based instantantion of the GS proof system. Please refer to Section 10 in [GS08] for the notation used in this section. The operations of interest for our work are listed below. We first describe how to maul the commitments and then we describe how to maul the proofs.

Commitments: We describe a single operation $\mathsf{MaulComm}$ which actually captures two operations simultaneously, namely that of rerandomizing the commitment and adding value to the committed message.

$\mathsf{MaulComm}(params, \mathsf{Eq}, C, D, R_c, R_d, \Delta_C, \Delta_D, \mathsf{Sign}_C, \mathsf{Sign}_D)$: We will consider the two cases when the system of equations is multiscalar equations and when the system of equations is quadratic equations.

For an equation Eq of the following form do the following.

$$\sum_{i=1}^{m} f_1(a_i, x_i) + \sum_{i=1}^{n} f_1(y_i, b_i) + \sum_{i=1}^{m}\sum_{j=1}^{n} \gamma_{i,j} f_1(x_i, y_j) = t$$

Let $C = (c_1, \ldots, c_m)$ be the commitments of the variables $x_1, \ldots, x_m$ and let $D = (d_1, \ldots, d_n)$ be the commitments of the variables $y_1, \ldots, y_n$. We maul the commitments in C and D in the following way. For every $i = 1, \ldots, m$;

$$c_i' = \iota(\Delta_{C_i}) + \mathsf{Sign}_{C_i}(c_i + R_c^{(i)} \overrightarrow{u})$$

where $R_c^{(i)}$ is the i^{th} element in R_c. Similarly we randomize D as follows. For every $i = 1, \ldots, n$,

$$d_i' = \iota'(\Delta_{D_i}) + \mathsf{Sign}_{D_i}(d_i + R_d^{(i)} \overrightarrow{v})$$

where $R_d^{(i)}$ is the i^{th} element in R_d.

Consider a quadratic equation of the following form.

$$\sum_{i=1}^{m} f_2(a_i, x_i) + \sum_{i=1}^{m}\sum_{j=1}^{n} \gamma_{i,j} f_2(x_i, x_j) = t$$

Let $C = (c_1, \ldots, c_m)$ be the commitments of the variables $x_1, \ldots, x_m$. We maul the commitments in C in the following way. For every $i = 1, \ldots, m$

$$c_i' = \iota'(\Delta_{C_i}) + \mathsf{Sign}_{C_i}(c_i + R_c^{(i)} \overrightarrow{u})$$

where $R_c^{(i)}$ is the i^{th} element in R_c.

We denote by $\mathsf{reRandComm}$ the operation that takes $(params, \mathsf{Eq}, C, D, R_c, R_d)$ as input and output (C', D') as follows. It executes $\mathsf{MaulComm}(params, \mathsf{Eq}, C, D, R_c, R_d, 0, 0, 1, 1)$ and outputs whatever $\mathsf{MaulComm}$ outputs.
We now describe the operations on the proofs as follows.

1. $\mathsf{MaulProof}(params, \mathsf{Eq}, \phi, C, D, R_C, R_D, \Delta_C, \Delta_D, \Delta_t, \mathsf{Sign}_C, \mathsf{Sign}_D, \mathsf{Sign}_t)$ [19]: We will consider the two cases when the system of equations is multiscalar equations and when the system of equations is

[19]The last column of R_D contains just zeroes. Further Sign_C and Sign_D are vector containing elements which are either 1 or -1.

quadratic equations.

For an equation Eq of the following form do the following.

$$\sum_{i=1}^{m} f_1(a_i, x_i) + \sum_{i=1}^{n} f_1(y_i, b_i) + \sum_{i=1}^{m}\sum_{j=1}^{n} \gamma_{i,j} f_1(x_i, y_j) = t$$

Let $C = (c_1, \ldots, c_m)$ be the commitments of the variables $x_1, \ldots, x_m$ and let $D = (d_1, \ldots, d_n)$ be the commitments of the variables $y_1, \ldots, y_n$. We say that the operation is defined on the above equation if the assignment $(x_1, \ldots, x_m) = (z_{x_1}, \ldots, z_{x_m})$ and $(y_1, \ldots, y_n) = (z_{y_1}, \ldots, z_{y_n})$ satisfies the above equation then even the following quantity is satisfied.

$$\sum_{i=1}^{m} f_1(a_i, z_{x_i} + \mathsf{Sign}_C^{(i)}(\Delta_C^{(i)})) + \sum_{i=1}^{n} f_1(z_{y_i} + \mathsf{Sign}_D^{(i)}(\Delta_D^{(i)}), b_i) + \sum_{i=1}^{m}\sum_{j=1}^{n} \gamma_{i,j} f_1(z_{x_i} + \mathsf{Sign}_C^{(i)}(\Delta_C^{(i)}), y_j + \mathsf{Sign}_D^{(j)}(\Delta_D^{(j)}))$$
$$= t + \mathsf{Sign}_t(\Delta_t)$$

We now describe how to modify the proof ϕ as follows. Let r' be picked uniformly at random from $\mathbb{Z}_q$.

$$\phi' = \phi + R_C^T \iota(\overrightarrow{b}) + R_C^T \Gamma \iota(\overrightarrow{a}) + R_D^T \Gamma^T \tau'(\overrightarrow{a}) + R_D^T \Gamma^T \iota'(\overrightarrow{x}) + R^T \Gamma R_D \overrightarrow{u} + r' H_1^{(1)} \overrightarrow{u}$$

Output the proof ϕ', which is verified against the following equation

$$\sum_{i=1}^{m} f_1(a_i, x_i) + \sum_{i=1}^{n} f_1(y_i, b_i) + \sum_{i=1}^{m}\sum_{j=1}^{n} \gamma_{i,j} f_1(x_i, y_j) = t + \mathsf{Sign}_t(\Delta_t)$$

Consider a quadratic equation of the following form.

$$\sum_{i=1}^{m} f_2(a_i, x_i) + \sum_{i=1}^{m}\sum_{j=1}^{n} \gamma_{i,j} f_2(x_i, x_j) = t$$

Let $C = (c_1, \ldots, c_m)$ be the commitments of the variables $x_1, \ldots, x_m$. We say that the operation is defined on the above equation if the assignment $(x_1, \ldots, x_m) = (z_{x_1}, \ldots, z_{x_m})$ satisfies the above equation then even the following equation is satisfied.

$$\sum_{i=1}^{m} f_2(a_i, z_{x_i} + \mathsf{Sign}_C^{(i)}(\Delta_C^{(i)})) + \sum_{i=1}^{m}\sum_{j=1}^{n} \gamma_{i,j} f_2(z_{x_i} + \mathsf{Sign}_C^{(i)}(\Delta_C^{(i)}), z_{x_j} + \mathsf{Sign}_C^{(j)}(\Delta_D^{(j)})) = t + \mathsf{Sign}_t(\Delta_t)$$

We now describe how to modify the proof ϕ as follows. Let r' be picked uniformly at random from $\mathbb{Z}_q$. Then,

$$\phi' = \phi + (R_c^T \iota'(\overrightarrow{b})) + R_c^T(\Gamma + \gamma^T)\tau'(x) + R_c^T \Gamma R \overrightarrow{v} + r' H_1^{(2)} \overrightarrow{u}$$

Output the proof ϕ' which is verified against the following equation,

$$\sum_{i=1}^{m} f_2(a_i, x_i) + \sum_{i=1}^{m}\sum_{j=1}^{n} \gamma_{i,j} f_2(x_i, x_j) = t + \mathsf{Sign}_t(\Delta_t)$$

We will denote by $\mathsf{reRandProof}$ the procedure which takes as input $(\mathsf{Eq}, \phi, C, D, R_c, R_d, CRS)$ and outputs ϕ' in the following way. It executes $\mathsf{MaulProof}(\mathsf{Eq}, \phi, C, D, R_c, R_d, 0, 0, 1, 1)$ to obtain ϕ' for uniformly picked R_c and R_d. $\mathsf{reRandProof}$ outputs ϕ'.

Before we describe the next two operations, we modify the notation we have been using to describe the equations. We rewrite an equation $\mathsf{Eq}(X_1, \ldots, X_s)$, where $X_1, \ldots, X_s$ are the variables used in Eq, as $\mathsf{Eq}(X_1, \ldots, X_s; t)$ where t is the constant appearing on the right hand side of the equation.

2. We define the operation $\mathsf{Exchange}_{\mathfrak{F}}$ as follows. Consider two equations $\mathsf{Eq}_1(X_1, \ldots, X_s; t_1)$ and $\mathsf{Eq}_2(Y_1, \ldots, Y_s; t_2)$ where X_i (resp., Y_i) are the only variables used in Eq_1 (resp., Eq_2). Let $\mathfrak{F}$ be a one-one mapping from the set $\{X_1, \ldots, X_s\}$ to the set $\{Y_1, \ldots, Y_s\}$. The operation $\mathsf{Exchange}_{\mathfrak{F}}$ is said to be *defined* on Eq_1 and Eq_2 only if $\mathsf{Eq}_1(\mathfrak{F}(X_1), \ldots, \mathfrak{F}(X_s); t_2) = \mathsf{Eq}_2(Y_1, \ldots, Y_s; t_2)$. Intuitively, $\mathsf{Exchange}$ is a well defined operation on two equations if by suitably replacing the variables of one equation with another set of variables you get another equation. $\mathsf{Exchange}_{\mathfrak{F}}$ takes as input $(\mathsf{Eq}_1, \mathsf{Eq}_2, C, D, \pi_1, \pi_2, \mathfrak{F})$ and outputs (C', D', π_2, π_1) where C' and D' are defined as follows. Let $C = (c_1, \ldots, c_s)$ and $D = (d_1, \ldots, d_s)$ be the commitments to the variables $(X_1, \ldots, X_s)$ and $(Y_1, \ldots, Y_s)$ respectively. Then, $C' = (c'_1, \ldots, c'_s)$ and $D' = (d'_1, \ldots, d'_s)$ such that $d'_i = c_{\sigma(i)}$ and $c'_i = d_{\sigma^{-1}(i)}$ where σ is a permutation on $\{1, \ldots, s\}$ such that $\mathfrak{F}(X_i) = Y_{\sigma(i)}$ (such a σ will exist since $\mathfrak{F}$ is a one-one mapping). Further, π_2 is verified against the equation $\mathsf{Eq}_1(X_1, \ldots, X_s; t_2)$ and π_1 is verified against the equation $\mathsf{Eq}_2(Y_1, \ldots, Y_s; t_2)$.

3. We define the operation $\mathsf{SymmPerm}$ as follows. Consider a system of equations $\mathsf{Eq}(X_1, \ldots, X_s)$ containing the equations $\mathsf{Eq}_1(X_1, \ldots, X_s; t_1), \ldots, \mathsf{Eq}_m(X_1, \ldots, X_s; t_m)$. Let $\mathfrak{F}_1$ and $\mathfrak{F}_2$ be one-one mappings on the sets $\{1, \ldots, m\}$ and $\{1, \ldots, s\}$ respectively. We say that the operation $\mathsf{SymmPerm}_{\mathfrak{F}_1, \mathfrak{F}_2}$ is said to be well defined on the system of equations Eq only if $\mathsf{Eq}_{\mathfrak{F}_1(i)}(X_1, \ldots, X_s; t_{\mathfrak{F}_1(i)}) =$ $\mathsf{Eq}_i(X_{\mathfrak{F}_2(1)}, \ldots, X_{\mathfrak{F}_2(s)}; t_{\mathfrak{F}_1(i)})$. The operation $\mathsf{SymmPerm}_{\mathfrak{F}_1, \mathfrak{F}_2}$ takes as input (Eq, C, Π) and outputs (C', Π') where $C = (c_1, \ldots, c_s)$ are commitments to the variables $X_1, \ldots, X_s$ respectively and (C', Π') is defined as follows. The set of commitments C' is set to $(c'_1, \ldots, c'_s)$, where $c'_i = c_{\mathfrak{F}_2(i)}$. Let $\Pi = (\pi_1, \ldots, \pi_t)$ be the proofs for the equations $\mathsf{Eq}_1, \ldots, \mathsf{Eq}_m$. Then, Π' is set to $(\pi_{\mathfrak{F}_1(1)}, \ldots, \pi_{\mathfrak{F}_1(t)})$ which are proofs for the equations $\mathsf{Eq}_1(X_1, \ldots, X_s; t_{\mathfrak{F}_1(1)}), \ldots, \mathsf{Eq}_m(X_1, \ldots, X_s; t_{\mathfrak{F}_1(m)})$. This completes the $\mathsf{SymmPerm}$ operation. Observe that $\mathsf{SymmPerm}$ can be defined as a sequence of $\mathsf{Exchange}$ operations.

Theorem 9. *Consider a distribution D on the systems of multiscalar and quadratic equations in the variables $X_1, \ldots, X_s$. Let CRS be the output of $\mathsf{Setup}_{GSproofs}$. Consider the following two experiments.*

- *Let Eq be a system of equations sampled from the distribution D. Let w be an assignment to the set of variables in Eq such that Eq is satisfied. Execute $P_{\mathsf{sub}}(\mathsf{Eq}, w, CRS)$ to obtain Π. Denote the output distribution by D_1.*
- *Let Eq be a system of equations sampled from the distribution D. Let w be an assignment to the set of variables in Eq such that Eq is satisfied. Execute $P_{\mathsf{sub}}(\mathsf{Eq}, w, CRS)$ to obtain π_1. Execute $\mathsf{reRandProof}(\mathsf{Eq}, \pi_1, CRS)$ to obtain π_2. Execute $\mathsf{Oper}(\pi_2)$ [20] to obtain π' where $\mathsf{Oper} \in \{\mathsf{Exchange}, \mathsf{MaulProof}\}$. Output π'. Denote the output distribution by D_2.*

There does not exist any PPT adversary A distinguishing the two output distributions D_1 and D_2.

Proof. We will consider the different cases for which Oper is defined on the system of equations.

Case 1. $\mathsf{Oper} = \mathsf{Exchange}$: Let Eq be written as $(\mathsf{Eq}_1(X_1, \ldots, X_s), \mathsf{Eq}_2(Y_1, \ldots, Y_s))$. Without loss of generality, let the witness w be represented as $(x_1, \ldots, x_s, y_1, \ldots, y_s)$. Let $\mathfrak{F}$ be the function used by $\mathsf{Exchange}$ to map the variables from $X_1, \ldots, X_s$ to $Y_1, \ldots, Y_s$. Consider $w' = (x'_1, \ldots, x'_s, y'_1, \ldots, y'_s)$ such that

[20] We assume here that the exact transformation is part of Oper. For example, if $\mathsf{Oper} = \mathsf{Exchange}$ and $\mathfrak{F}$ is the one-one mapping on the set of variables, we assume that $\mathfrak{F}$ is implicit in the description of Oper.

$y'_i = x_j$ and $x'_j = y_i$ where $Y_i = \mathfrak{F}(X_j)$. Observe that w' satisfies the set of equations Eq if $\mathsf{Exchange}$ is defined on the system of equations. Also, the output distribution of π_2 (as defined in the experiment) is the same as the output distribution of $P_{\mathsf{sub}}(\mathsf{Eq}, w', CRS)$. This combined by the fact that the output distributions of $P_{\mathsf{sub}}(\mathsf{Eq}, w', CRS)$ and $P_{\mathsf{sub}}(\mathsf{Eq}, w, CRS)$ are computationally indistinguishable (from the zero knowledge property) proves this case.

Case 2. $\mathsf{Oper} = \mathsf{MaulProof}$: Let $X_1, \ldots, X_s$ be the set of variables in Eq. Let $w = (x_1, \ldots, x_s)$ be the assignment to the variables $X_1, \ldots, X_S$ such that Eq is satisfied. Let $\mathsf{MaulProof}$ transform $(x_1, \ldots, x_s)$ to obtain $w' = (x'_1, \ldots, x'_s)$ such that w' satisfies the system of equations Eq if $\mathsf{MaulProof}$ is defined on the system of equations. The output distribution of π_2 (as defined in the experiment) is the same as the output distribution of $P_{\mathsf{sub}}(\mathsf{Eq}, w', CRS)$. Further, the output distributions of $P_{\mathsf{sub}}(\mathsf{Eq}, w', CRS)$ and $P_{\mathsf{sub}}(\mathsf{Eq}, w, CRS)$ are computationally indistinguishable. This completes the proof.

□

Theorem 10. *Consider a distribution D on the systems of multiscalar and quadratic equations in the variables $X_1, \ldots, X_s$. Let CRS be the output of $\mathsf{Setup}_{GSproofs}$. Consider the following two experiments.*

- Expt_0*: Sample from D to obtain the equation Eq. Let w be an assignment to the set of variables in Eq such that Eq is satisfied. Execute $P_{\mathsf{sub}}(\mathsf{Eq}, w, CRS)$ to obtain Π.*
- Expt_1*: Sample from D to obtain the equation Eq. Let w be an assignment to the set of variables in Eq such that Eq is satisfied. Execute $P_{\mathsf{sub}}(\mathsf{Eq}, w, CRS)$ to obtain π_1. Execute $\mathsf{reRandProof}(\pi_1)$ to obtain π_2. For $i = 1, \ldots, t$, where $t = poly(k)$, execute $\mathsf{Oper}_i(\pi_2)$ to obtain π' where Oper_i is well defined on the system of equations Eq with $oper_i \in \{\mathsf{Exchange}, \mathsf{MaulProof}\}$ not necessarily different from Oper_j for $i \neq j$. Further, the operations Oper_i and Oper_j, for $i \neq j$ are commutative, i.e., whether $\mathsf{Oper}_i(\mathsf{Oper}_j(\pi))$ is the same as $\mathsf{Oper}_j(\mathsf{Oper}_i(\pi))$.*

There does not exist any PPT adversary A distinguishing the output distributions of Expt_0 and Expt_1.

Proof. We prove this by defining the following hybrid experiments. Hybrid $\mathsf{Hybrid}_i^{(1)}$ is defined as follows - Execute $P_{\mathsf{sub}}(\mathsf{Eq}, w, CRS)$ to obtain π_1. Execute $\mathsf{reRandProof}(\pi_1)$ to obtain π_2. For $j = 1, \ldots, i$ execute $\mathsf{Oper}_j(\pi_2)$ to obtain π'_j. Using Theorem 9 we can show that the hybrids $\mathsf{Hybrid}_i^{(1)}$ and $\mathsf{Hybrid}_{i+1}^{(1)}$ for $i = 1, \ldots, t-1$ are computationally indistinguishable. To see this, lets first assume that the hybrids are distinguishable by an adversary $\mathsf{Adv}_{i,i+1}$. We then construct an adversary Adv to violate the Theorem 9, with respect to operation Oper_{i+1}, as follows. Adv takes as input a proof π and then executes $\mathsf{Oper}_1(\cdots \mathsf{Oper}_i(\pi) \cdots)$ to obtain π'. Input π' to $\mathsf{Adv}_{i,i+1}$ and output whatever $\mathsf{Adv}_{i,i+1}$ outputs. It can be verified that the success probability of the adversary $\mathsf{Adv}_{i,i+1}$ is the same as the success probability of Adv. In other words, with non-negligible probability Adv can distinguish whether the proof input to it is the output of an honest prover or obtained by applying Oper_{i+1} on a proof output by the honest prover. This shows that the hybrids Hybrid_i and Hybrid_{i+1} are computationally indistinguishable. Further, Theorem 9 can be directly applied to show that $\mathsf{Hybrid}_1^{(1)}$ and Expt_0 are computationally indistinguishable. These two facts prove that $\mathsf{Hybrid}_t^{(1)}$, which is the same as Expt_1, is computationally indistinguishable from Expt_0. □

The following corollary follows directly from Theorem 9 and Theorem 10 and from the fact that $\mathsf{SymmPerm}$ is basically a combination of many $\mathsf{Exchange}$ operations.

Corollary 1. *Let Eq be a system of multiscalar/ quadratic bilinear equations. Let w be an assignment to the set of variables in Eq such that Eq is satisfied. Let CRS be the output of $\mathsf{Setup}_{GSproofs}$. Consider the following two experiments.*

- Expt_0*: Execute $P_{\mathsf{sub}}(\mathsf{Eq}, w, CRS)$ to obtain Π.*
- Expt_1*: Execute $P_{\mathsf{sub}}(\mathsf{Eq}, w, CRS)$ to obtain π_1. Execute $\mathsf{reRandProof}(\mathsf{Eq}, \pi_1, CRS)$ to obtain π_2. Execute $\mathsf{SymmPerm}(\pi_2)$ to obtain π'. Output π'.*

There does not exist any PPT adversary which distinguishes the output distributions of Expt_0 and Expt_1.

D.3.2 Rerandomization operation

Overview. We first give an overview of how the proofs are rerandomized. Note that our proof consists of a garbled circuit and a set of GS proofs that the garbled circuit was correctly computed. To rerandomize the proof we rerandomize the garbled circuit and then rerandomize the GS proofs. We recall how the Garbled circuit is rerandomized. To rerandomize a garbled circuit, the following four main operations are performed.

1. The wire keys are homomorphically permuted.
2. The table of ciphertexts are permuted.
3. All the ciphertexts are rerandomized.
4. The masks in the ciphertexts are XORed with random values.

The GS proofs basically can be rerandomized by performing the following steps. In the first step, all the commitments and the proofs are rerandomized. Then, (corresponding to the first and the second step) all the commitments and the proofs are suitably permuted. Further, (corresponding to the third and the fourth step) commitments associated to $\mathsf{ct}_{i,j}^{\mathsf{reRand},G}$, $r_{i,j}^{\mathsf{reRand},G}$ and $\Delta_{i,j}^{\mathsf{reRand},G}$ are suitably modified. Also, the proofs using these commitments are also appropriately modified.

We now describe the rerandomization operation in full detail.

Technical details. We describe the procedure reRand as follows. reRand takes as input Π and outputs another proof Π' such that any polynomial time adversary cannot distinguish whether the proof Π' was a product of rerandomization of Π or was it obtained by executing P_{sub} directly. reRand parses Π as $(\mathsf{GC}, \Pi_{\mathsf{GC}})$. It first rerandomizes the garbled circuit and then mauls the proof Π_{GC}. We first describe the rerandomization of the garbled circuit in a detailed manner. A random permutation on $\{1, \ldots, l\}$ is picked with respect to every wire. More precisely, for every wire w, we denote the random permutation picked with respect to w to be σ_w. Then, perform the following for every gate G in the circuit. Let the table of ciphertexts associated with G, represented by $\mathsf{ct}^{\mathsf{reRand},G}$, be described as follows. Let w_1 and w_2 be the input wires of G and let w_3 be the output wire of G.

$$\begin{bmatrix} \mathsf{ct}_1^{\mathsf{reRand},G,1} & \mathsf{ct}_1^{\mathsf{reRand},G,2} \\ \mathsf{ct}_2^{\mathsf{reRand},G,1} & \mathsf{ct}_2^{\mathsf{reRand},G,2} \\ \mathsf{ct}_3^{\mathsf{reRand},G,1} & \mathsf{ct}_3^{\mathsf{reRand},G,2} \\ \mathsf{ct}_4^{\mathsf{reRand},G,1} & \mathsf{ct}_4^{\mathsf{reRand},G,2} \end{bmatrix}$$

For all gates other than the output gate, do the following. Every ciphertext of the form $\mathsf{ct}_i^{\mathsf{reRand},G,1}$, for $i \in \{1, \ldots, 4\}$, which is a encryption of $\delta_{i',j'}$ under the key $L_{w_1,i'}$ is homomorphically modified to obtain a ciphertext $\mathsf{ct}_i'^{\mathsf{reRand},G,1}$ which is an encryption of $\sigma'_{w_3}(\delta_{i,j})$ under the key $\sigma_{w_1}(L_{w_1,i'})$. Here, the permutation σ'_{w_3} is defined on the set $\{1, \ldots, 2l\}$ and it means the following. On the set $\{1, \ldots, l\}$, σ'_{w_3} is the same as σ_{w_3} while it is an identity function on the set $\{l+1, \ldots, 2l\}$. Similarly $\mathsf{ct}_i^{\mathsf{reRand},G,2}$, which is an encryption of $\mu_{i',j'}$ is modified for $i = 1, \ldots, 4$ to obtain a ciphertext which is an encryption of $\sigma'_{w_3}(\mu_{i',j'})$ under the key $\sigma_{w_2}(L_{w_2,j'})$. Further, for every $i = 1, \ldots, 4$, pick a bit string Δ_i of length $2l$ and homomorphically XOR Δ_i into $\mathsf{ct}_{i,1}^{\mathsf{reRand},G}$ and $\mathsf{ct}_{i,2}^{\mathsf{reRand},G}$.

Further, rerandomize the public keys associated to the ciphertexts (including those in the output gate) by doing the following. Multiply each public key by a matrix of size $l \times l$ containing elements uniformly at random from $\mathbb{Z}_q$. The last step is to permute the table of ciphertexts. Denote the permutation used to permute the ciphertexts by σ_G. This completes the process of rerandomization of the garbled circuit.

We will now describe how to modify the commitments and proofs associated with the garbled circuit. Before we do that, we first parse the proof Π_{GC} to be rerandomized as $(\mathsf{comm}, \pi_{\mathsf{GC}})$. The different components of the proof Π_{GC} is represented below. We use the same notation as was used in the construction in Section B.2.1. For example, $L_{w,i}$ represents a wire key.

We define the commitment matrices for every gate G as follows. Let the input wires of G be w_1, w_2 and let the output wire be w_3.

1. We define the matrix $\mathsf{WireComm}_G$ containing the commitments of the variables representing the wire keys corresponding to gate G. $\mathsf{WireComm}_G$ is a $l \times 6$ matrix represented as follows.

$$\mathsf{WireComm}_G = \begin{bmatrix}\mathsf{comm}(X_{L_{w_1,0}}) & \mathsf{comm}(X_{L_{w_1,1}}) & \cdots & \mathsf{comm}(X_{L_{w_3,0}}) & \mathsf{comm}(X_{L_{w_3,1}})\end{bmatrix}$$

where, $\mathsf{comm}(X_{L_{w_i,b}})$ be a column vector of commitments, of length l, representing the commitment of the variable $X_{L_{w_i,b}}$.

2. We define the matrix $\mathsf{BefPermComm}_G$ containing the commitments to the variables representing the ciphertexts $c_{i,j}^{G,k}$. $\mathsf{BefPermComm}_G$ is a matrix represented as follows.

$$\mathsf{BefPermComm}_G = \begin{bmatrix}\mathsf{comm}(X_{c_{(0,0)}^{G,1}}) & \mathsf{comm}(X_{c_{(0,0)}^{G,2}}) & \cdots & \mathsf{comm}(X_{c_{(1,1)}^{G,1}}) & \mathsf{comm}(X_{c_{(1,1)}^{G,2}})\end{bmatrix}$$

where $\mathsf{comm}(X_{c_{(i,j)}^{G,k}})$ is a matrix of dimension $(l+1) \times l$ with the i^{th} column vector representing the commitment to the i^{th} column vector of the variable $X_{c_{(i,j)}^{G,k}}$.

3. We define the matrix $\mathsf{AftPermComm}_G$, which essentially contains the commitments to the variables representing the ciphertexts $\mathsf{ct}_{i,j}^{G,k}$. $\mathsf{AftPermComm}_G$ is a matrix represented as follows.

$$\mathsf{AftPermComm}_G = \begin{bmatrix}\mathsf{comm}(X_{\mathsf{ct}_{(0,0)}^{G,1}}) & \mathsf{comm}(X_{\mathsf{ct}_{(0,0)}^{G,2}}) & \cdots & \mathsf{comm}(X_{\mathsf{ct}_{(1,1)}^{G,1}}) & \mathsf{comm}(X_{\mathsf{ct}_{(1,1)}^{G,2}})\end{bmatrix}$$

where $\mathsf{comm}(X_{\mathsf{ct}_{(i,j)}^{G,k}})$ is a matrix of dimension $(l+1) \times l$ with the i^{th} column vector representing the commitment to the i^{th} column vector of variable $X_{\mathsf{ct}_{(i,j)}^{G,k}}$.

4. We define the matrix $\mathsf{RandComm}_G$ which contains the commitments to the variables representing the random elements $r_{i,j}^{G,k}$. $\mathsf{RandComm}_G$ is a $l \times 8$ matrix represented as follows.

$$\mathsf{RandComm}_G = \begin{bmatrix}\mathsf{comm}(X_{r_{(0,0)}^{G,1}}) & \mathsf{comm}(X_{r_{(0,0)}^{G,2}}) & \cdots & \mathsf{comm}(X_{r_{(1,1)}^{G,1}}) & \mathsf{comm}(X_{r_{(1,1)}^{G,2}})\end{bmatrix}$$

where $\mathsf{comm}(X_{r_{(i,j)}^{G,k}})$ is a column vector of length l, is a commitment to the vector of variables $X_{r_{(i,j)}^{G,k}}$.

5. We define the matrix $\mathsf{deltaComm}_G$ which contains the commitments to the variables representing the masks $\delta_{i,j}^G$. $\mathsf{deltaComm}_G$ is a $4 \times 2l$ matrix represented as follows.

$$\mathsf{deltaComm}_G = \begin{bmatrix}\mathsf{comm}(X_{\delta_{0,0}^G}) & \mathsf{comm}(X_{\delta_{0,1}^G}) & \mathsf{comm}(X_{\delta_{1,0}^G}) & \mathsf{comm}(X_{\delta_{1,1}^G})\end{bmatrix}$$

where $\mathsf{comm}(\delta_{i,j}^G)$ is a column vector of length $2l$, representing the commitment to the vector of variables $X_{\delta_{i,j}^G}$.

6. We define the matrix muComm_G which contains the commitments to the variables representing the masks $\mu_{i,j}^G$. muComm_G is a $4 \times 2l$ matrix represented as follows.

$$\mathsf{muComm}_G = \begin{bmatrix}\mathsf{comm}(X_{\mu_{0,0}^G}) & \mathsf{comm}(X_{\mu_{0,1}^G}) & \mathsf{comm}(X_{\mu_{1,0}^G}) & \mathsf{comm}(X_{\mu_{1,1}^G})\end{bmatrix}$$

where $\mathsf{comm}(\mu_{i,j}^G)$ is a column vector of length l, representing the commitment to the vector of variables, namely $X_{\mu_{i,j}^G}$.

7. We define the matrix $\mathsf{reRandComm}_G$ which contains the commitments to the variables representing $r_{i,j,k}^{\mathsf{reRand},G}$. $\mathsf{reRandComm}_G$ is a matrix represented as follows.

$$\mathsf{reRandComm}_G = \begin{bmatrix} \mathsf{comm}(X_{r_{0,0}^{\mathsf{reRand},G,1}}) & \mathsf{comm}(X_{r_{0,0}^{\mathsf{reRand},G,2}}) & \cdots & \mathsf{comm}(X_{r_{1,1}^{\mathsf{reRand},G,1}}) & \mathsf{comm}(X_{r_{1,1}^{\mathsf{reRand},G,2}}) \end{bmatrix}$$

where $\mathsf{comm}(X_{r_{i,j}^{\mathsf{reRand},G,k}})$ is a matrix of dimension $l \times l$ with the i^{th} column vector representing the commitment to the a^{th} column vector of the variable $X_{r_{i,j}^{\mathsf{reRand},G,k}}$.

8. We define the matrix $\mathsf{reRandCTComm}_G$ which contains the commitments to the variables representing $\mathsf{ct}_{i,j}^{\mathsf{reRand},G,k}$. $\mathsf{reRandCTComm}_G$ is a matrix represented as follows.

$$\mathsf{reRandCTComm}_G = \begin{bmatrix} \mathsf{comm}(X_{\mathsf{ct}_{0,0}^{\mathsf{reRand},G,1}}) & \mathsf{comm}(X_{\mathsf{ct}_{0,0}^{\mathsf{reRand},G,2}}) & \cdots \mathsf{comm}(X_{\mathsf{ct}_{1,1}^{\mathsf{reRand},G,1}}) & \mathsf{comm}(X_{\mathsf{ct}_{1,1}^{\mathsf{reRand},G,2}}) \end{bmatrix}$$

where $\mathsf{comm}(X_{\mathsf{ct}_{i,j}^{\mathsf{reRand},G,k}})$ denotes the commitment to the variable $\mathsf{comm}(X_{\mathsf{ct}_{i,j}^{\mathsf{reRand},G,k}}$.

9. We define the matrix $\mathsf{reRandDeltaComm}_G$ which contains the commitments to the variables representing $\Delta_{i,j,k}^{G}$. $\mathsf{reRandDeltaComm}$ is a $l \times 4$ matrix represented as follows.

$$\mathsf{reRandCTComm}_G = \begin{bmatrix} \mathsf{comm}(X_{\mathsf{ct}_{0,0}^{\mathsf{reRand},G,1}}) & \mathsf{comm}(X_{\mathsf{ct}_{0,0}^{\mathsf{reRand},G,2}}) & \cdots & \mathsf{comm}(X_{\mathsf{ct}_{1,1}^{\mathsf{reRand},G,1}}) & \mathsf{comm}(X_{\mathsf{ct}_{1,1}^{\mathsf{reRand},G,2}}) \end{bmatrix}$$

where $\mathsf{comm}(X_{\mathsf{ct}_{1,1}^{\mathsf{reRand},G,1}})$ is a matrix of size $(l+1) \times l$ with the i^{th} column vector denoting the commitment to the i^{th} column vector of the variable $X_{\mathsf{ct}_{1,1}^{\mathsf{reRand},G,1}}$.

We now describe the GS proofs constructed by $\mathsf{Prover}_{\mathsf{MalGC}}$ for the equations in $\mathsf{GCktEq}(C)$. For every gate G we define the following matrices. Let G be a gate with input wires w_1, w_2 and output wire w_3.

1. The matrix $\mathsf{PublicEqProof}_G$ is represented as follows.

$$\mathsf{PublicEqProof}_G = \begin{bmatrix} \pi_{1,0}^{publiceq} & \pi_{1,1}^{publiceq} & \cdots & \pi_{3,0}^{publiceq} & \pi_{3,1}^{publiceq} \end{bmatrix}$$

where $\pi_{i,j}^{publiceq}$ represents a vector of proofs, of length l, for the system of equations $\mathsf{PKEq}(X_{L^2_{w_i,b}}, X_{L^1_{w_i,b}})$.

2. The matrix $\mathsf{WireBitProof}_G$ is represented as follows.

$$\mathsf{WireBitProof}_G = \begin{bmatrix} \pi_{1,0}^{wirebiteq} & \pi_{1,1}^{wirebiteq} & \cdots & \pi_{3,0}^{wirebiteq} & \pi_{3,1}^{wirebiteq} \end{bmatrix}$$

where $\pi_{i,j}^{wirebiteq}$ represents a vector of proofs, of length l, for the system of equations $\mathsf{VecBitEq}(X^1_{L_{w_i,b}})$.

3. The matrix $\mathsf{CTEqProof}_G$ is represented as follows.

$$\mathsf{CTEqProof}_G = \begin{bmatrix} \pi_{0,0}^{cteq,1} & \pi_{0,0}^{cteq,2} & \cdots & \pi_{1,1}^{cteq,1} & \pi_{1,1}^{cteq,2} \end{bmatrix}$$

where $\pi_{i,j}^{cteqpf,k}$ represents a matrix of proofs, of dimension $(l+1) \times l$, for the system of equations $\mathsf{VecCTEq}(X_{c_{(i,j)}^{G,k}}, X_{L_{w_1,i}}, X_{r_{i,j}^{G,k}}, X_{\delta_{i,j}^{G}})$.

4. The matrix $\mathsf{reRandCTProof}_G$ is represented as follows.

$$\mathsf{reRandCTProof}_G = \begin{bmatrix}\pi_{1,1}^{reRandCT} & \pi_{1,2}^{reRandCT} & \cdots & \pi_{4,1}^{reRandCT} & \pi_{4,2}^{reRandCT}\end{bmatrix}$$

where $\pi_{i,j}^{reRandCT}$ represents a matrix of proofs of size $l+1 \times l$, for the system of equations $\mathsf{VecreRandCTEq}(X_{\mathsf{ct}_{i,j}^{G,1}}, X_{\mathsf{ct}_{i,j}^{\mathsf{reRand},G,1}}, X_{\Delta_{i,j}^{\mathsf{reRand},G,1}}, X_{r_{i,j}^{\mathsf{reRand},G,1}})$ and $\mathsf{VecreRandCTEq}(X_{\mathsf{ct}_{i,j}^{G,2}}, X_{\mathsf{ct}_{i,j}^{\mathsf{reRand},G,2}}, X_{\Delta_{i,j}^{\mathsf{reRand},G,2}}, X_{r_{i,j}^{\mathsf{reRand},G,2}})$,.

5. The matrix $\mathsf{deltaBitProof}_G$ is represented as follows.

$$\mathsf{deltaBitProof}_G = \begin{bmatrix}\pi_{0,0}^{deltabiteq} & \pi_{0,1}^{deltabiteq} & \pi_{1,0}^{deltabiteq} & \pi_{1,1}^{deltabiteq}\end{bmatrix}$$

where $\pi_{i,j}^{deltabiteq}$ represents a vector of proofs, of length $2l$, for the system of equations $\mathsf{VecBitEq}(X_{\delta_{i,j}})$.

6. The matrix $\mathsf{muBitProof}_G$ of dimension $2l \times 4$ is represented as follows.

$$\mathsf{muBitProof}_G = \begin{bmatrix}\pi_{0,0}^{mubiteq} & \pi_{0,1}^{mubiteq} & \pi_{1,0}^{mubiteq} & \pi_{1,1}^{mubiteq}\end{bmatrix}$$

where $\pi_{i,j}^{mubiteq}$ represents a vector of proofs, of length $2l$, for the system of equations $\mathsf{VecBitEq}(X_{\mu_{i,j}^G})$.

7. The matrix $\mathsf{DeltaBitProof}_G$ of dimension $2l\times$ is represented as follows.

$$\mathsf{muBitProof}_G = \begin{bmatrix}\pi_{0,0}^{Deltabiteq,1} & \pi_{0,0}^{Deltabiteq,2} & \cdots \pi_{1,1}^{Deltabiteq,1} & \pi_{1,1}^{Deltabiteq,2}\end{bmatrix}$$

where $\pi_{i,j}^{Deltabiteq,k}$ represents a vector of proofs, of length $2l$, for the system of equations $\mathsf{VecBitEq}(X_{\Delta_{i,j}^{\mathsf{reRand},G}})$.

8. The matrix $\mathsf{PermEqProof}_G$ is essentially π^{permeq} which is the matrix of proofs for the system of equations $\mathsf{PermEq}(X_{\mathsf{ct}_1^G}, X_{\mathsf{ct}_2^G}, X_{\mathsf{ct}_3^G}, X_{\mathsf{ct}_4^G}, (X_{c_1^G}, X_{c_2^G}, X_{c_3^G}, X_{c_4^G}))$.

9. The matrix $\mathsf{OutWireProof}(L_{w_m,0}, L_{w_m,1})$ is essentially $\pi^{outwires}$ which is the matrix of proofs for the system of equations $\mathsf{OutWiresEq}(L_{w_i,0}, L_{w_i,1})$.

Now that we have defined the data structure to represent the proof, we now show to rerandomize each component of the proof.

1. Rerandomize the proofs: For every commitment c in comm, pick a random string $R_c \in \mathbb{Z}_q^3$ and then execute $\mathsf{reRandComm}(c, R_c)$ to obtain c'. For every proof π in π_{GC} associated to the equation Eq do the following. Let $X_1, \ldots, X_s$ be the variables associated to Eq and let $c_1, \ldots, c_s$ be the commitments to the variables $X_1, \ldots, X_s$. Then, execute $\mathsf{reRandProof}(\pi, c_1, \ldots, c_s, R_{c_1}, \ldots, R_{c_s}, CRS)$ to obtain π'.

2. We first describe how the commitment vectors are modified. We then describe how the proof matrices are modified. We first introduce the following notation. Let $v = (v_1, \ldots, v_s)$ be a vector of elements. By $\sigma(v)$, we mean the vector $v' = (v_{\sigma(1)}, \ldots, v_{\sigma(s)})$. Let M denote a matrix with the $(i,j)^{th}$ element in M represented by $m_{i,j}$. By $(\sigma_1 \times \sigma_2)(M)$ we denote the matrix M' with the $(i,j)^{th}$ element in M' denoted by $m_{\sigma_1(i),\sigma_2(j)}$. Further, for a wire key w_i, we define σ''_{w_i} as follows. Let $M_{\sigma_{w_i}}$ be the matrix representation of σ_{w_i} of size $l \times l$. Then, $M'_{\sigma_{w_i}}$ is a matrix with the first l rows and columns same as that of $M_{\sigma_{w_i}}$ while the last row and the last column contain zeroes except the last entry which contains 1. In other words, the $(j,k)^{th}$ entry in $M_{\sigma_{w_i}}$ is 0 if $k = l+1$ and $i \neq l+1$ and when $j = l+1$ and $k \neq l+1$ and it is 1 if $j = k = l+1$. Now, σ''_{w_3} is nothing but $M'^{-1}_{\sigma_{w_i}}$. This basically says that if

σ_{w_i} is the permutation to permute the secret key then the public key needs to be permuted by σ''_{w_3}.

We now describe how the commitment matrices are modified.

(a) We modify the matrix $\mathsf{WireComm}_G$ to obtain the matrix $\mathsf{WireComm}_G^{(1)}$ which is represented as follows.

$$\begin{bmatrix}\sigma_{w_1}(\mathsf{comm}(X_{L_{w_1,0}})) & \sigma_{w_1}(\mathsf{comm}(X_{L_{w_1,1}})) & \cdots & \sigma_{w_3}(\mathsf{comm}(X_{L_{w_3,0}})) & \sigma_{w_3}(\mathsf{comm}(X_{L_{w_3,1}}))\end{bmatrix}$$

(b) We modify the matrix $\mathsf{BefPermComm}_G$ to obtain the matrix $\mathsf{BefPermComm}_G^{(1)}$ which is the following matrix. For convenience sake, we will represent the transpose of the matrix.

$$\begin{bmatrix}(\sigma''_{w_1} \times \sigma_{w_3})(\mathsf{comm}(X_{c^{G,1}_{(0,0)}})) \\ (\sigma''_{w_2} \times \sigma'_{w_3})(\mathsf{comm}(X_{c^{G,2}_{(0,0)}})) \\ \cdots \\ (\sigma''_{w_1} \times \sigma_{w_3})(\mathsf{comm}(X_{c^{G,1}_{(1,1)}})) \\ (\sigma''_{w_2} \times \sigma'_{w_3})(\mathsf{comm}(X_{c^{G,2}_{(1,1)}}))\end{bmatrix}$$

(c) We modify the matrix $\mathsf{AftPermComm}_G$ to obtain the matrix $\mathsf{AftPermComm}_G^{(1)}$ which is represented as follows. For convenience sake, we will represent the transpose of the matrix

$$\begin{bmatrix}(\sigma''_{w_1} \times \sigma_{w_3})(\mathsf{comm}(X_{\mathsf{ct}^{G,1}_{(0,0)}})) \\ (\sigma''_{w_2} \times \sigma'_{w_3})(\mathsf{comm}(X_{\mathsf{ct}^{G,2}_{(0,0)}})) \\ \cdots \\ (\sigma''_{w_1} \times \sigma_{w_3})(\mathsf{comm}(X_{\mathsf{ct}^{G,1}_{(1,1)}})) \\ (\sigma''_{w_2} \times \sigma'_{w_3})(\mathsf{comm}(X_{\mathsf{ct}^{G,2}_{(1,1)}}))\end{bmatrix}$$

(d) We modify the matrix $\mathsf{RandComm}_G$ to obtain the matrix $\mathsf{RandComm}_G^{(1)}$ which is represented as follows.

$$\begin{bmatrix}(\mathcal{I} \times \sigma_{w_3})(\mathsf{comm}(X_{r^{G,1}_{(0,0)}})) \\ (\mathcal{I} \times \sigma_{w_3})(\mathsf{comm}(X_{r^{G,2}_{(0,0)}})) \\ \cdots \\ (\mathcal{I} \times \sigma_{w_3})(\mathsf{comm}(X_{r^{G,1}_{(1,1)}})) \\ (\mathcal{I} \times \sigma_{w_3})(\mathsf{comm}(X_{r^{G,2}_{(1,1)}}))\end{bmatrix}$$

where $\mathcal{I}$ is the identity permutation.

(e) We modify the matrix $\mathsf{deltaComm}_G$ to obtain the matrix $\mathsf{deltaComm}_G^{(1)}$ represented as follows.

$$\begin{bmatrix}\sigma_{w_3}(\mathsf{comm}(X_{\delta^G_{0,0}})) & \sigma_{w_3}(\mathsf{comm}(X_{\delta^G_{0,1}})) & \sigma_{w_3}(\mathsf{comm}(X_{\delta^G_{1,0}})) & \sigma_{w_3}(\mathsf{comm}(X_{\delta^G_{1,1}}))\end{bmatrix}$$

(f) We modify the matrix muComm_G to obtain the matrix $\mathsf{muComm}_G^{(1)}$ which is represented as follows.

$$\begin{bmatrix}\sigma_{w_3}(\mathsf{comm}(X_{\mu^G_{0,0}})) & \sigma_{w_3}(\mathsf{comm}(X_{\mu^G_{0,1}})) & \sigma_{w_3}(\mathsf{comm}(X_{\mu^G_{1,0}})) & \sigma_{w_3}(\mathsf{comm}(X_{\mu^G_{1,1}}))\end{bmatrix}$$

(g) We modify the matrix $\mathsf{reRandComm}_G$ to obtain $\mathsf{reRandComm}_G^{(1)}$ which is represented as follows. For convenience sake, we will represent the transpose of the matrix below.

$$\begin{bmatrix} (\mathcal{I} \times \sigma_{w_3})(\mathsf{comm}(X_{r_{0,0}^{\mathsf{reRand},G,1}})) \\ (\mathcal{I} \times \sigma_{w_3})(\mathsf{comm}(X_{r_{0,0}^{\mathsf{reRand},G,2}})) \\ \cdots \\ (\mathcal{I} \times \sigma_{w_3})(\mathsf{comm}(X_{r_{1,1}^{\mathsf{reRand},G,1}})) \\ (\mathcal{I} \times \sigma_{w_3})(\mathsf{comm}(X_{r_{1,1}^{\mathsf{reRand},G,2}})) \end{bmatrix}$$

where $\mathcal{I}$ denotes the identity permutation.

(h) We modify the matrix $\mathsf{reRandCTComm}_G$ to obtain $\mathsf{reRandCTComm}_G^{(1)}$ as follows.

$$\begin{bmatrix} (\sigma''_{w_1} \times \sigma_{w_3})(\mathsf{comm}(X_{\mathsf{ct}_{0,0}^{\mathsf{reRand},G,1}})) \\ (\sigma''_{w_2} \times \sigma_{w_3})(\mathsf{comm}(X_{\mathsf{ct}_{0,0}^{\mathsf{reRand},G,2}})) \\ \vdots (\sigma''_{w_1} \times \sigma_{w_3})(\mathsf{comm}(X_{\mathsf{ct}_{1,1}^{\mathsf{reRand},G,1}})) \\ (\sigma''_{w_2} \times \sigma_{w_3})(\mathsf{comm}(X_{\mathsf{ct}_{1,1}^{\mathsf{reRand},G,2}})) \end{bmatrix}$$

(i) We modify the matrix $\mathsf{DeltaComm}_G$ to obtain $\mathsf{DeltaComm}_G^{(1)}$ as follows. For convenience sake, we will represent the transpose of the matrix below.

$$\begin{bmatrix} \sigma_{w_3}(\mathsf{comm}(X_{\Delta_{0,0}^{\mathsf{reRand},G,1}})) \\ \sigma_{w_3}(\mathsf{comm}(X_{\Delta_{0,0}^{\mathsf{reRand},G,2}}))) \\ \cdots \\ \sigma_{w_3}(\mathsf{comm}(X_{\Delta_{1,1}^{\mathsf{reRand},G,1}})) \\ \sigma_{w_3}(\mathsf{comm}(X_{\Delta_{1,1}^{\mathsf{reRand},G,2}})) \end{bmatrix}$$

We now describe how to modify the proof matrices.

(a) The matrix $\mathsf{PublicEqProof}_G$ is modified to obtain $\mathsf{PublicEqProof}_G^{(1)}$ as follows.

$$\begin{bmatrix} \sigma''_{w_1}(\pi_{1,0}^{publiceq}) & \sigma''_{w_1}(\pi_{1,1}^{publiceq}) & \cdots & \sigma''_{w_3}(\pi_{3,0}^{publiceq}) & \sigma''_{w_3}(\pi_{3,1}^{publiceq}) \end{bmatrix}$$

(b) The matrix $\mathsf{WireBitProof}_G$ is modified to obtain $\mathsf{WireBitProof}_G^{(1)}$ as follows.

$$\begin{bmatrix} \sigma_{w_1}(\pi_{1,0}^{wirebiteq}) & \sigma_{w_1}(\pi_{1,1}^{wirebiteq}) & \cdots & \sigma_{w_3}(\pi_{3,0}^{wirebiteq}) & \sigma_{w_3}(\pi_{3,1}^{wirebiteq}) \end{bmatrix}$$

(c) The matrix $\mathsf{CTEqProof}_G$ is modified to obtain $\mathsf{CTEqProof}_G^{(1)}$ as follows.

$$\begin{bmatrix} \sigma''_{w_1} \times \sigma_{w_3}(\pi_{1,1}^{cteq}) & \sigma''_{w_2} \times \sigma_{w_3}(\pi_{1,2}^{cteq}) & \cdots & \sigma''_{w_1} \times \sigma_{w_3}(\pi_{4,1}^{cteq}) & \sigma''_{w_2} \times \sigma_{w_3}(\pi_{4,2}^{cteq}) \end{bmatrix}$$

(d) The matrix $\mathsf{reRandCTProof}_G$ is modified to obtain $\mathsf{reRandCTProof}_G^{(1)}$ as follows.

$$\begin{bmatrix} \sigma''_{w_1} \times \sigma_{w_3}(\pi_{1,1}^{reRandCT}) & \sigma''_{w_2} \times \sigma_{w_3}(\pi_{1,2}^{reRandCT}) & \cdots & \sigma''_{w_1} \times \sigma_{w_3}(\pi_{4,1}^{reRandCT}) & \sigma''_{w_1} \times \sigma_{w_3}(\pi_{4,2}^{reRandCT}) \end{bmatrix}$$

(e) The matrix $\mathsf{deltaBitProof}_G$ is modified to obtain $\mathsf{deltaBitProof}_G^{(1)}$ as follows.

$$\begin{bmatrix}\sigma_{w_3}(\pi_{0,0}^{deltabiteq}) & \sigma_{w_3}(\pi_{0,1}^{deltabiteq}) & \sigma_{w_3}(\pi_{1,0}^{deltabiteq}) & \sigma_{w_3}(\pi_{1,1}^{deltabiteq})\end{bmatrix}$$

(f) The matrix $\mathsf{muBitProof}_G$ is modified to obtain $\mathsf{muBitProof}_G^{(1)}$ as follows.

$$\begin{bmatrix}\sigma_{w_3}(\pi_{0,0}^{mubiteq}) & \sigma_{w_3}(\pi_{0,1}^{mubiteq}) & \sigma_{w_3}(\pi_{1,0}^{mubiteq}) & \sigma_{w_3}(\pi_{1,1}^{mubiteq})\end{bmatrix}$$

(g) The matrix $\mathsf{DeltaBitProof}_G$ is modified to obtain $\mathsf{DeltaBitProof}_G^{(1)}$ as follows.

$$\begin{bmatrix}\sigma_{w_3}(\pi_{0,0}^{Deltabiteq,1}) & \sigma_{w_3}(\pi_{0,0}^{Deltabiteq,2}) & \cdots & \sigma_{w_3}(\pi_{1,1}^{Deltabiteq,1}) & \sigma_{w_3}(\pi_{1,1}^{Deltabiteq,2})\end{bmatrix}$$

(h) The matrix $\mathsf{PermEqProof}_G$ is modified to obtain $\mathsf{PermEqProof}_G^{(1)}$ as follows. Denote by $\mathsf{Eq}_{\mathsf{PermEq}}$, the system of polynomial equations $\mathsf{PermEq}((X_{\mathsf{ct}_1^G}, \ldots, X_{\mathsf{ct}_4^G}), (X_{c_{0,0}^G}, \ldots, X_{c_{1,1}^G}))$. Execute $\mathsf{SymmPerm}(\mathsf{Eq}_{permeq}, \mathsf{PermEqProof}_G)$ to obtain $\mathsf{PermEqProof}'_G$ where $\mathsf{SymmPerm}$ is defined as follows.

   i. Consider the matrix of variables Var with the $(i,j)^{th}$ element containing $X_{v_i^{(j)}}$. Permute the elements in this matrix to obtain Var' as follows. The $(i,j)^{th}$ element in Var' is $X_{v_{\sigma_G(i)}^{\sigma_G(j)}}$.

   ii. The vector of variables $(X_{\mathsf{ct}_1^G}, \ldots, X_{\mathsf{ct}_1^G})$ and $(X_{c_1^G}, \ldots, X_{c_1^G})$ are permuted with respect to σ_G.

3. We will now describe how the commitments and the proofs are mauled with respect to the third and the fourth operations.

   We will first describe as to how the commitments are mauled.

   (a) The commitment matrices that are going to be mauled are $\mathsf{RandComm}_G^{(1)}$, $\mathsf{AftPermComm}_G^{(1)}$, $\mathsf{BefPermComm}_G^{(1)}$, $\mathsf{reRandComm}_G^{(1)}$, $\mathsf{reRandCTComm}_G^{(1)}$ and $\mathsf{DeltaComm}_G^{(1)}$. For every commitment c, committed to the value x, in these matrices we do the following. Let x be mauled to obtain $\Delta_x + \mathsf{Sign}(x)$ during the rerandomization operation. We pick a random string R_c uniformly at random from $\mathbb{Z}_q^3$ and then execute the following to obtain c'.

   $$\mathsf{MaulComm}(params, c, R_c, \Delta_x, \mathsf{Sign})$$

   We denote the resulting commitment matrices by $\mathsf{RandComm}_G^{(2)}$, $\mathsf{AftPermComm}_G^{(2)}$, $\mathsf{BefPermComm}_G^{(2)}$, $\mathsf{reRandComm}_G^{(2)}$, $\mathsf{reRandCTComm}_G^{(2)}$ and $\mathsf{DeltaComm}_G^{(2)}$.

   (b) Rest of the commitment matrices remain the same.

   We will now describe how the proof matrices are modified.

   (a) The matrices that are mauled with respect to third and the fourth operation are $\mathsf{CTEqProof}_G^{(1)}$, $\mathsf{reRandCTProof}_G^{(1)}$, $\mathsf{deltaBitProof}_G^{(1)}$, $\mathsf{muBitProof}_G^{(1)}$, $\mathsf{DeltaBitProof}_G^{(1)}$, $\mathsf{PermEqProof}_G^{(1)}$ and $\mathsf{OutWireProof}_G^{(1)}$. For every proof π in these proof matrices we do the following. Without loss of generality let π be associated to the equation, denoted by $\mathsf{Eq}(X_1, \ldots, X_r)$. Let C denote the commitments to the variables $(X_1, \ldots, X_r)$. Let Δ_i and Sign_i be such that during the rerandomization operation each X_i is mauled to obtain $\Delta_i + \mathsf{Sign}_i(X_i)$. We will denote the vector $(\Delta_1, \ldots, \Delta_r)$ and $(\mathsf{Sign}_1, \ldots, \mathsf{Sign}_r)$ by Δ_C and Sign_C respectively. We modify π to obtain π' by executing the following.

   $$\mathsf{MaulProof}(\mathsf{Eq}, params, C, R_C, \Delta_C, \mathsf{Sign}_C)$$

The proof matrices, denoted by $\mathsf{CTEqProof}_G^{(2)}$, $\mathsf{reRandCTProof}_G^{(2)}$, $\mathsf{deltaBitProof}_G^{(2)}$,$\mathsf{muBitProof}_G^{(2)}$, $\mathsf{DeltaBitProof}_G^{(2)}$, $\mathsf{PermEqProof}_G^{(2)}$ and $\mathsf{OutWireProof}_G^{(2)}$, is basically the matrices obtained by modifying every proof π to obtain π' in the above way.

(b) Rest of the proof matrices remain the same.

This completes the rerandomization operation.

We now show that the proof system $(P_{\mathsf{sub}}, V_{\mathsf{sub}})$ is rerandomizable according to the Definition 7. We have already described the rerandomization operation. The following theorem shows that the output distribution of the rerandomization operation is computationally indistinguishable from the output distribution of the prover P_{sub}.

Theorem 11. *$\mathbb{P}$ is a rerandomizable proof system.*

Proof. To show that the proof system $(P_{\mathsf{sub}}, V_{\mathsf{sub}})$ is a rerandomizable proof system, we need to prove that the following two distributions are computationally indistinguishable.

- $D_0^{C,w,CRS} = \left\{\Big((\mathsf{GC}, \Pi), (\mathsf{GC}_{fresh}, \Pi_{fresh})\Big)\right\}$,
  where $P_{\mathsf{sub}}(C, w, CRS)$ outputs (GC, Π) and $(\mathsf{GC}_{fresh}, \Pi_{fresh})$ is also an output of $P_{\mathsf{sub}}(C, w, CRS)$ [21]
- $D_1^{C,w,CRS} = \left\{\Big((\mathsf{GC}, \Pi), (\mathsf{GC}_{\mathsf{reRand}}, \Pi_{\mathsf{reRand}})\Big)\right\}$,
  where $P_{\mathsf{sub}}(C, w, CRS)$ outputs (GC, Π) and $\mathsf{reRand}(\mathsf{GC})$ outputs $(\mathsf{GC}_{\mathsf{reRand}}, \Pi_{\mathsf{reRand}})$.

To prove this, consider the following two lemmas. Before this, we introduce the following terminology. Let GC be a garbled circuit. By $wirekeys_{\mathsf{GC}}$ we mean the wire keys associated to the garbled circuit GC.

Lemma 14. *The following two distributions are computationally indistinguishable.*

- $Hybrid_1^{C,w,CRS} = \{(\mathsf{GC}, \Pi, \mathsf{GC}_{\mathsf{reRand}}, \Pi_{fresh})\}$,
  where (GC, Π) *is the output of* $P_{\mathsf{sub}}(C, w, CRS)$ *and* $\mathsf{GC}_{\mathsf{reRand}}$ *is the output of* $\mathsf{reRand}(\mathsf{GC}; R)$*, where* R *is the randomness used for rerandomization. Further,* Π_{fresh} *is the output of* $P_{gc}(\mathsf{GC}, wirekeys_{\mathsf{GC}}, T_R, CRS)$*, where* T_R *is the transformation corresponding to the randomness* R*.*
- $Hybrid_2 = \{(\mathsf{GC}, \Pi, \mathsf{GC}_{\mathsf{reRand}}, \Pi_{\mathsf{reRand}})\}$,
  where (GC, Π) *is the output of* $P_{\mathsf{sub}}(C, w, CRS)$ *and* $\mathsf{GC}_{\mathsf{reRand}}$ *is the output of* $\mathsf{reRand}(\mathsf{GC}; R)$*, where* R *is the randomness used for rerandomization. Further,* Π_{fresh} *is the output of* $\mathsf{reRand}_{gc}(\mathsf{GC}, T_R, \Pi, CRS)$*, where* T_R *is the transformation corresponding to the randomness* R*.*

  Further, Π_{reRand} *is the output of* $\mathsf{reRand}(\mathsf{GC}, \mathsf{GC}_{\mathsf{reRand}}, \Pi, T_R)$*.*

Proof. This lemma follows from Theorem 10 and Corollary 1 since the operations performed on the proof are essentially $\mathsf{SymmPerm}$ and $\mathsf{MaulProof}$. □

Lemma 15. *The following two distributions are computationally indistinguishable.*

- $Hybrid_3 = \{(\mathsf{GC}, \Pi, \mathsf{GC}_{\mathsf{reRand}}, \Pi_{fresh})\}$,
  where (GC, Π) *is the output of* $P_{\mathsf{sub}}(C, w, CRS)$ *and* $\mathsf{GC}_{\mathsf{reRand}}$ *is the output of* $\mathsf{reRand}(\mathsf{GC})$*. Further,* Π_{fresh} *is the output of* $P_{gc}(\mathsf{GC}, wirekeys_{\mathsf{GC}}, CRS)$*.*
- $Hybrid_4 = \{(\mathsf{GC}, \Pi, \mathsf{GC}_{fresh}, \Pi_{fresh})\}$,
  where (GC, Π) *is the output of* $P_{\mathsf{sub}}(C, w, CRS)$ *and* $(\mathsf{GC}_{fresh}, \Pi_{fresh})$ *is also the output of* $P_{\mathsf{sub}}(C, w, CRS)$*. Further,* Π_{fresh} *is the output of* $P_{gc}(\mathsf{GC}, wirekeys_{\mathsf{GC}}, CRS)$*.*

[21]Note that (GC, Π) and $(\mathsf{GC}_{fresh}, \Pi_{fresh})$ need not be the same since the randomness to compute both of them might be different.

Proof. To prove this lemma we consider the following two hybrid distributions.

$\mathsf{Hybrid}_5 = \{(\mathsf{GC}, \Pi, \mathsf{GC}_{\mathsf{reRand}}, \Pi_{sim})\}$,
where (GC, Π) is the output of $P_{\mathsf{sub}}(C, w, CRS)$ and $\mathsf{GC}_{\mathsf{reRand}}$ is the output of $\mathsf{reRand}(\mathsf{GC})$. Further, Π_{sim} is the output of $\mathsf{Sim}_{gc}(\mathsf{GC}, CRS)$.

$\mathsf{Hybrid}_6 = \{(\mathsf{GC}, \Pi, \mathsf{GC}_{fresh}, \Pi_{sim})\}$,
where (GC, Π) is the output of $P_{\mathsf{sub}}(C, w, CRS)$ and $(\mathsf{GC}_{fresh}, \Pi_{fresh})$ is also the output of $P_{\mathsf{sub}}(C, w, CRS)$. Further, Π_{fresh} is the output of $\mathsf{Sim}_{gc}(\mathsf{GC}, CRS)$.

We first show that hybrids Hybrid_5 and Hybrid_6 are computationally indistinguishable. We also show that hybrids Hybrid_3 and Hybrid_5 are computationally indistinguishable and then we show that Hybrid_4 and Hybrid_6 are computationally indistinguishable.

Claim 1. Hybrids Hybrid_5 and Hybrid_6 are computationally indistinguishable.
Proof. This follows from the rerandomizability property of the garbled circuits. Let A be a PPT adversary distinguishing these two distributions with non-negligible probability then we construct another PPT adversary A_{reRand} who can distinguish a real garbled circuit from a rerandomized garbled circuit with non-negligible probability. A_{reRand} takes a garbled circuit as input and using a simulation trapdoor composes a proof Π_{Sim} with respect to GC. It then inputs $(\mathsf{GC}, \Pi_{\mathsf{Sim}})$ and outputs whatever A_{reRand} outputs. Since we know that there does not exist any adversary distinguishing real garbled circuits from rerandomized garbled circuits, it follows that there does not exist any PPT adversary distinguishing the hybrids Hybrid_5 and Hybrid_6.

The following two claims follow directly from the zero knowledge property of the proof system (P_{gc}, V_{gc}).
Claim 2. Hybrids Hybrid_3 and $hybrid_5$ are computationally indistinguishable.

Claim 3. Hybrids Hybrid_4 and Hybrid_6 are computationally indistinguishable.

From the above three claims it follows that the hybrid distributions Hybrid_3 and Hybrid_6 are computationally indistinguishable.

□

From Lemmas 1 and Lemma 2 it follows that the hybrids Hybrid_2 and hybrid Hybrid_4 are computationally indistinguishable. But, the distributions Hybrid_2 and Hybrid_4 are exactly the distributions D_0 and D_1. This proves the theorem.

□

D.4 Large Min-Entropy property: Leakage Lemma

We now argue that $\mathbb{P}$ satisfies large min-entropy property. First, we recall that the size of each wire key is $\max\left(\left\lceil\sqrt{\frac{(c^*+1)\ell}{|C|}}\right\rceil, \left\lceil\sqrt{\frac{k}{|C|}}\right\rceil\right)$, where k is a security parameter, $c^* = \frac{1}{c}$ [22] and ℓ is the number of bits of leakage allowed. Without loss of generality, assume that $(c^*+1)\ell \geq k$.

To argue the large min-entropy property, we first start with the observation that the number of possibilities of garbled circuits computed corresponding to circuit C is at least $2^{\lambda^2|C|}$, where λ is the length of the wire key. This is in turn is at least $2^{(c^*+1)\ell}$ [23]. To see this, consider the following. Each ciphertext in an entry in the table of ciphertexts, corresponding to a gate in the circuit, consists of (at least) λ ciphertexts. Further, there are 2^{λ} possibilities for each ciphertext (because of the randomness involved during the encryption – we assume that the size of the randomness required for each encryption is also λ

[22] Recall that c is the fraction of the memory that can be leaked.
[23] If $(c^*+1)\ell < k$ then we would argue that the number of garbled circuits is at least 2^k.

bits). Hence, totally there are 2^{λ^2} possibilites for each entry in the table corresponding to each gate in the circuit. And, since there are of the order of $|C|$ gates in the circuit, we have the fact that there are at least $2^{\lambda^2|C|}$ possible garbled circuits.

We now focus on showing that our proof system $\mathbb{P}$ satisfies the large min-entropy property. The following leakage lemma is a key ingredient in proving the large min-entropy property. It says that given ℓ bits of leakage on a freshly generated garbled circuit GC, where $\ell <<$ size of the garbled circuit, there does not exist any PPT adversary who can output GC. At a high level, this follows from the fact that even after knowing ℓ bits of leakage, there is still a large amount of residual entropy left in the garbled circuit and hence an adversary cannot determine the entire garbled circuit. Note that this lemma holds even *if the adversary is unbounded.*

Lemma 16 (**Leakage Lemma**). *Consider a PPT adversary A. For any input z, auxiliary information* $\mathsf{aux}, \mathsf{aux}^*$*,*

$$Pr[\mathsf{GC}' = \mathsf{GC} \ : \ \mathsf{GC} \leftarrow \mathsf{reRandGC}(1^k, C, z), \ \mathsf{GC}' \leftarrow A^{\mathcal{O}^{\ell}(\mathsf{GC},\mathsf{aux})}(C, z, \mathsf{aux}^*)] \leq negl(k)$$

Proof. We denote f to be the function submitted by the adversary to the leakage oracle. The output length of f is at most ℓ. Let X_A^L be the random variable that denotes the output of the adversary after receiving the leakage L. Let X_{oracle}^L be the random variable denoting the garbled circuit GC produced in the oracle such that $f(\mathsf{GC}) = L$.

We define the event Succ_A as follows.

$$\mathsf{Succ}_A = 1 \text{ if } X_A^L = a \text{ and } X_{oracle}^L = a, \text{ else } \mathsf{Succ}_A = 0$$

Now, we prove that $Pr[\mathsf{Succ}_A = 1]$ is negligible. Consider the following sets,

- $\mathsf{Range}(f) = \{a : f(\mathsf{GC}) = a$ for some garbled circuit GC produced in the oracle$\}$. We order the elements in $\mathsf{Range}(f)$ such that $\mathsf{Range}(f) = \{a_1, \ldots, a_{|\mathsf{Range}(f)|}\}$.The length of the output of f is at most ℓ bits and so the size of the range of f is at most 2^{ℓ}.
- For every $i = 1, \ldots, |\mathsf{Range}(f)|$ we define the set,

$$S_i^f = \{\mathsf{GC} \ : \ \mathsf{GC} \text{ is the output of } \mathsf{reRandGC} \text{ and } f(\mathsf{GC}) = a_i\}$$

We now estimate the success probability of the adversary A.

$$\begin{aligned}
Pr[\mathsf{Succ}_A = 1] &= Pr\Big[\bigcup_{i=1}^{|\mathsf{Range}(f)|} X_A^{a_i} = \mathsf{GC} \text{ and } X_{oracle}^{a_i} = \mathsf{GC}\Big] \\
&\leq \sum_{i=1}^{|\mathsf{Range}(f)|} Pr[X_A^{a_i} = \mathsf{GC} \text{ and } X_{oracle}^{a_i} = \mathsf{GC}] \quad \text{(by union bound)} \\
&\leq \sum_{i=1}^{|\mathsf{Range}(f)|} Pr[X_A^{a_i} = \mathsf{GC}].Pr[X_{oracle}^{a_i} = \mathsf{GC}] \\
&\leq \sum_{i=1}^{|\mathsf{Range}(f)|} \frac{1}{|S_i^f|}.\frac{|S_i^f|}{2^{(c^*+1)\ell}} \qquad (\because \text{there are at least } 2^{(c^*+1)\ell} \text{ garbled circuits}) \\
&\leq \sum_{i=1}^{|\mathsf{Range}(f)|} \frac{1}{2^{(c^*+1)\ell}} \\
&\leq 2^{\ell}.\frac{1}{2^{(c^*+1)\ell}} \\
&\leq \frac{1}{2^{c^*\ell}} = negl(k)
\end{aligned}$$

□

We have the following corollary from the fact that the output distribution of $\mathsf{SimGC}(C, C(x))$ [24] is computationally from $\mathsf{reRandGC}(C, x)$ for any circuit C, input x.

Corollary 2. *Consider a PPT adversary A. For any input z, auxiliary information* $\mathsf{aux}, \mathsf{aux}^*$*,*

$$Pr[\mathsf{GC}' = \mathsf{GC} \;:\; \mathsf{GC} \leftarrow \mathsf{SimGC}(1^k, C, C(z)),\; \mathsf{GC}' \leftarrow A^{\mathcal{O}^l(\mathsf{GC},\mathsf{aux})}(C, z, \mathsf{aux}^*)] \leq negl(k)$$

We now extend the above lemma and show that even if the adversary is given access to many leakage oracles with each containing a garbled circuit then the probability that the adversary outputs a garbled circuit which is the same as one of the garbled circuits in the leakage oracles is negligible.

Lemma 17 (Extended Leakage Lemma). *Consider a PPT adversary A. Let $t \leftarrow A(1^k)$. Let $\mathsf{GC}_1 \leftarrow \mathsf{reRandGC}(1^k, C, z_1)$, $\ldots$, $\mathsf{GC}_t \leftarrow \mathsf{reRandGC}(1^k, C, z_t)$ for any set of inputs $z_1, \ldots, z_t$. For any auxillary inputs $\mathsf{aux}_1, \ldots, \mathsf{aux}_t, \mathsf{aux}^*$, we have,*

$$Pr[\mathsf{GC}' = \mathsf{GC}_i \textit{ for any } i = 1, \ldots, t \;:\; \mathsf{GC}' \leftarrow A^{\mathcal{O}^l(\mathsf{GC}_1,\mathsf{aux}_1),\ldots,\mathcal{O}^l(\mathsf{GC}_t,\mathsf{aux}_t)}(C, \{z_i\}, \mathsf{aux}^*)] \leq negl(k)$$

Proof. Consider a PPT adversary A such that the probability that A outputs $\mathsf{GC}' = \mathsf{GC}_i$, for some $i = 1, \ldots, t$, is at least $\frac{1}{p(k)}$, where p is a polynomial. Let $z_1, \ldots, z_t$ be the inputs for which this holds. We define the following two quantities:

- $p = Pr[\mathsf{GC}' = \mathsf{GC}_i \text{ for any } i = 1, \ldots, t \;:\; \mathsf{GC}' \leftarrow A^{\mathcal{O}^l(\mathsf{GC}_1,\mathsf{aux}_1),\ldots,\mathcal{O}^l(\mathsf{GC}_t,\mathsf{aux}_t)}(C, \mathsf{aux}^*)]$
- $p_i = Pr[\mathsf{GC}' = \mathsf{GC}_i \;:\; \mathsf{GC}' \leftarrow A^{\mathcal{O}^l(\mathsf{GC}_1,\mathsf{aux}_1),\ldots,\mathcal{O}^l(\mathsf{GC}_t,\mathsf{aux}_t)}(C, \mathsf{aux}^*)]$.

Consider the following claim.

Claim 1. *There exists $i \in \{1, \ldots, t\}$ such that $p_i \geq \frac{1}{tp(k)}$.*

Proof. Suppose for every i, we have $p_i < \frac{1}{tp(k)}$. We have,

$$\begin{aligned} p &\leq \sum_{i=1}^{t} p_i \text{ (by union bound)} \\ &< \sum_{i=1}^{t} \frac{1}{p(k)} \\ &< t \cdot \frac{1}{tp(k)} \\ &< \frac{1}{p(k)} \text{(, a contradiction)} \end{aligned}$$

□

Let i be an index such that $p_i \geq \frac{1}{tp(k)}$. We then demonstrate an adversary A' which violates Lemma 16. We construct A' as follows. It first generates garbled circuits GC_j, for $j \in [t]$ and $j \neq i$ honestly. It simulates the leakage oracles to A with the j^{th} oracle containing GC_j, for $j \in [t]$ and $j \neq i$. Any call made by the adversary A to the i^{th} oracle is forwarded by A' to $\mathcal{O}(\mathsf{GC})$. Finally, A' outputs whatever A outputs. In

[24] SimGC is the simulator used in the proof of security of garbled circuits. It just takes as input a circuit C and output $C(w)$ and produces a garbled circuit that is computationally indistinguishable from $\mathsf{reRandGC}(C, w)$.

more detail, we describe A' as follows.

```
Procedure $A'^{\mathcal{O}^l(\mathsf{GC})}(C, aux, aux_1, \ldots, aux_t, z_1, \ldots, z_t, t)$:
$\mathsf{GC}_1 \leftarrow \mathsf{reRandGC}(C, z_1)$
        $\vdots$
$\mathsf{GC}_{i-1} \leftarrow \mathsf{reRandGC}(C, z_{i-1})$
$\mathsf{GC}_{i+1} \leftarrow \mathsf{reRandGC}(C, z_{i+1})$
        $\vdots$
$\mathsf{GC}_t \leftarrow \mathsf{reRandGC}(C, z_t)$
$\mathsf{GC}' \leftarrow A^{\mathcal{O}^l(\mathsf{GC}_1, aux),\ldots,\mathcal{O}^l(\mathsf{GC}_{i-1}, aux_i-1),\mathcal{O}^l(\mathsf{GC}, aux_i),\mathcal{O}^l(\mathsf{GC}_{i+1}, aux_{i+1}),\ldots,\mathcal{O}^l(\mathsf{GC}_t, aux_t)}(aux, \{z_i\}_{i\in[t]})$
Output $\mathsf{GC}'$
```

By our assumption, the probability that A outputs GC' such that $\mathsf{GC}' = \mathsf{GC}_i$ (here GC_i is the same as GC) is non-negligible. This means that the probability that A' outputs GC' such that $\mathsf{GC}' = \mathsf{GC}$ is non-negligible and this contradicts Lemma 16. This completes the proof. □

Similar to Corollary 2, we have a corresponding corollary for the above lemma.

Corollary 3. *Consider a PPT adversary A. Let $t \leftarrow A(1^k)$. Let $\mathsf{GC}_1 \leftarrow \mathsf{SimGC}(1^k, C, C(z_1))$, $\ldots$, $\mathsf{GC}_t \leftarrow \mathsf{SimGC}(1^k, C, C(z_t))$ for any set of inputs $z_1, \ldots, z_t$. For any auxillary inputs $\mathsf{aux}_1, \ldots, \mathsf{aux}_t, \mathsf{aux}^*$, we have,*

$$Pr[\mathsf{GC}' = \mathsf{GC}_i \text{ for any } i = 1, \ldots, t \;:\; \mathsf{GC}' \leftarrow A^{\mathcal{O}^l(\mathsf{GC}_1, \mathsf{aux}_1),\ldots,\mathcal{O}^l(\mathsf{GC}_t, \mathsf{aux}_t)}(C, \{z_i\}, \mathsf{aux}^*)] \le negl(k)$$

E Obtaining CLR encodings from $\mathbb{P}_{\mathsf{cm}}$: Proof of Theorem 3

We first represent $\mathbb{P}_{\mathsf{cm}}$ as $(\mathsf{CRSSetup}, P, V, \mathsf{FakeCRS}, \mathsf{Sim}, \mathsf{Ext}, \mathsf{reRand})$ defined for a relation R. We construct CLR-encodings for the same relation R. Consider the following game defined with respect to an adversary A. Suppose D be a hard distribution on the instances in a relation R. That is, there does exist any PPT adversary who on input C can output w' such that $(C, w') \in R$ with non-negligible probability, where (C, w) is the output of $D(1^k)$ and $(C, w) \in R$. We now prove that the construction of CLR-encodings from $\mathbb{P}_{\mathsf{cm}}$ as sketched in Section 4 is secure.

Proof of Theorem 3. We prove by contradiction. Consider an adversary A that breaks the security of the CLR-encodings scheme then we construct an adversary A^* that breaks the controlled malleability property of $\mathbb{P}_{\mathsf{cm}}$. Let us revisit the security game of the CLR-encoding scheme defined with respect to A.

```
$\mathsf{Game}_A(1^k)$:
$(t, \mathsf{state}) \leftarrow A(1^k)$
$(C, w) \leftarrow D(1^k)$
$CRS \leftarrow \mathsf{CRSSetup}(1^k)$
$\Pi_1 \leftarrow P(C, w, CRS)$
$\Pi_2 \leftarrow \mathsf{reRand}(C, \Pi_1, CRS)$
        $\vdots$
$\Pi_t \leftarrow \mathsf{reRand}(C, \Pi_{t-1}, CRS)$
$\Pi \leftarrow A^{\mathcal{O}^l(\Pi_1),\ldots,\mathcal{O}^l(\Pi_t)}(C, \mathsf{state})$
Output $\Pi$.
```

An adversary A wins the above game if it outputs $\Pi \neq \Pi_i$, for all $i = 1, \ldots, t$, and $V(C, CRS, \Pi) = 1$. We assume that A wins the above game with non-negligible probability and then arrive at a contradiction. We first prove the following claim.

Claim 2. *Consider (C, w) be an instance-witness pair in a relation R. Let $(CRS, \tau_{sim}, \tau_{ext}) \leftarrow \mathsf{FakeCRS}_1(1^k)$. Let $\Pi \leftarrow P(C, w, CRS)$. Then, we have the following:*

$$\{\mathsf{reRand}(CRS, C, \Pi)\} \cong \{\mathsf{Sim}(C, CRS, \tau_{sim})\}$$

Proof. This follows from the fact that $\{\mathsf{reRand}(CRS, C, \Pi)\}$ is computationally indistinguishable from $\{P(C, w, CRS)\}$. Further, $\{P(C, w, CRS)\}$ is computationally indistinguishable from $\{\mathsf{Sim}(C, CRS, \tau_{sim})\}$. The claim follows.

We now consider a sequence of hybrids such that every consecutive pair of hybrids is computationally indistinguishable. The final hybrid is then used to contradict the controlled malleability of the proof system $\mathbb{P}_{\mathsf{cm}}$. But first, we divide the game Game_A into two parts, namely $\mathsf{Game}_A^{(1)}$ and $\mathsf{Game}_A^{(2)}$.

```
$\underline{\mathsf{Game}_A^{(1)}(1^k)}$:
$(t, \mathsf{state}) \leftarrow A(1^k)$

$\underline{\mathsf{Game}_A^{(2)}(1^k, t, \mathsf{state})}$:
$CRS \leftarrow \mathsf{CRSSetup}(1^k)$
$(C, w) \leftarrow D(1^k)$
$\Pi_1 \leftarrow P(C, w, CRS)$
$\Pi_2 \leftarrow \mathsf{reRand}(C, \Pi_1, CRS)$
        $\vdots$
$\Pi_t \leftarrow \mathsf{reRand}(C, \Pi_{t-1}, CRS)$
$\Pi \leftarrow A^{\mathcal{O}^l(\Pi_1), \ldots, \mathcal{O}^l(\Pi_t)}(C, \mathsf{state})$
Output $\Pi$.
```

In the hybrids given below, we change $\mathsf{Game}_A^{(2)}$ one step at a time. Let (t, state) be the output of $\mathsf{Game}_A^{(1)}$.

$\underline{\mathsf{Hybrid}_0}$: This corresponds to the real game. That is, the output of this game is the output of $\mathsf{Game}_A^{(2)}$ on input (t, state). By our assumption, A wins in this hybrid with non-negligible probability.

$\underline{\mathsf{Hybrid}_1}$: In this hybrid, we modify the way CRS is generated in the previous hybrid. Instead of generating the CRS using $\mathsf{CRSSetup}$, it is generated using the algorithm $\mathsf{FakeCRS}$. That is, $\mathsf{FakeCRS}$ is executed to obtain $(CRS, \tau_{sim}, \tau_{ext})$. The rest of the hybrid remains the same.

Claim 3. *A wins in* Hybrid_1 *with non-negligible probability.*

Proof. Suppose A wins in Hybrid_1 with negligible probability then we can use this to construct a distinguisher to contradict the *zero knowledge property* of $\mathbb{P}_{\mathsf{cm}}$. More specifically, we show that this contradicts the fact that there does not exist any PPT distinguisher that can distinguish, with non-negligible probability, a honestly generated CRS from a simulated CRS. The distinguisher gets a CRS from the external challenger (of the zero-knowledge game). The distinguisher then generates $\Pi_1, \ldots, \Pi_t$ on instance C by himself using the witness w and the CRS it obtained externally. It then simulates the leakage oracles to the adversary A. If the adversary A succeeds in the game (that is, it outputs a valid proof which is different from the proofs in the leakage oracles) then the distinguisher outputs 1 else it outputs 0.

If the CRS obtained from the external challenger was honestly generated then the distinguisher outputs 1 with non-negligible probability. If the CRS obtained was simulated then the distinguisher outputs 1 with negligible probability. This contradicts the zero-knowledge property of $\mathbb{P}_{\mathsf{cm}}$.

$\underline{\mathsf{Hybrid}_{2,i}\text{, for } i = 0, \ldots, t}$: We set $\mathsf{Hybrid}_{2,0}$ to be the same as Hybrid_1. In $\mathsf{Hybrid}_{2,i}$, for $i = 1, \ldots, t$, do the following. We modify Game_A such that $\Pi_1, \ldots, \Pi_{t-i}$ are generated as in Game_A and the proofs $\Pi_{t-i+1}, \ldots, \Pi_t$ are generated by executing Sim on input (C, CRS, τ_{sim}), where (CRS, τ_{sim}) is produced by $\mathsf{FakeCRS}$. The rest of the hybrid is the same as hybrid $\mathsf{Hybrid}_{2,i-1}$.

Claim 4. *A wins in $\mathsf{Hybrid}_{2,i}$, for $i = 1, \ldots, t$ with non-negligible probability.*

Proof. Suppose that A wins in the hybrid $\mathsf{Hybrid}_{2,i+1}$ only with negligible probability. As in Claim 3, we show here that this contradicts the *zero-knowledge property* of $\mathbb{P}_{\mathsf{cm}}$. To do this, we demonstrate a distinguisher that can distinguish honestly generated proofs from simulated proofs for valid instances. The distinguisher receives an instance C, a witness w, a proof Π and finally (CRS, τ_{sim}) as produced by $\mathsf{FakeCRS}$ [25] (note that the extraction trapdoor τ_{ext} is *not* given to the distinguisher.). Note that the proof is generated with respect to CRS. The distinguisher generates the first $t - i$ proofs, denoted by $\Pi_1, \ldots, \Pi_{t-i}$ honestly by using C and w. It then sets the $(t - i + 1)^{\text{th}}$ proof to be Π. Finally, the rest of the proofs (if there are any) from $\Pi_{t-i+2}, \ldots, \Pi_t$ are generated using the simulator Sim on input (CRS, τ_{sim}). Finally, adversary A is executed with the distinguisher simulating the leakage oracles for A. If the adversary wins, the distinguisher outputs 1 else it outputs 0.

If the proof obtained by the distinguisher from the external challenger was honestly generated then it outputs 1 with non-negligible probability ($\mathsf{Hybrid}_{2,i}$). If the distinguisher obtains a simulated proof from the external challenger then it outputs 1 with negligible probability. This contradicts the zero-knowledge property of $\mathbb{P}_{\mathsf{cm}}$. This completes the proof.

We use $\mathsf{Hybrid}_{2,t}$ to construct an adversary A^* that breaks the controlled malleability property of the proof system (P, V). Further using the fact that A succeeds in $\mathsf{Hybrid}_{2,t}$ with non-negligible probability we show that A^* succeeds in breaking the controlled malleability property of (P, V) with non-negligible probability, contradicting the fact that $\mathbb{P}_{\mathsf{cm}}$ indeed satisfies the controlled malleability property.

Consider the following adversary A^* who participates in the controlled malleability game of the proof system $\mathbb{P}_{\mathsf{cm}}$. The challenger first executes $\mathsf{FakeCRS}$ to obtain $(CRS, \tau_{sim}, \tau_{ext})$. The adversary A^* gets as input an instance C, where (C, w) is the output of $D(1^k)$ (note that w is *not* given to the adversary). It then internally executes A to get t. It then requests for t proofs $\Pi_1, \ldots, \Pi_t$ from the challenger. It then simulates t leakage oracles to A with the i^{th} oracle containing Π_i and each of them tolerates a leakage of ℓ bits. Finally, A outputs Π which is also output by A^*. This completes the game.

Recall that we assumed that A succeeds in breaking the security property of the CLR-encoding scheme. This in turn means that with non-negligible probability, Π is a valid proof for C. Consider the following two claims.

Claim 5. $\Pi = \Pi_i$*, for $i = 1, \ldots, t$ with negligible probability.*

Proof. This follows from the fact that $\mathbb{P}_{\mathsf{cm}}$ satisfies large min-entropy property (Corollary 3).

Claim 6. *The probability that either $(C, w) \in R$ or $T(\Pi_i) = \Pi$ for an allowable transformation $T =$ $\mathsf{reRand}(\cdot, \cdot, \cdot; R)$ [26], where (w, Π_i, T) is the output of the extractor $\mathsf{Ext}(C, \Pi, CRS, \tau_{ext})$.*

Proof. Suppose $(C, w) \in R$, then this would mean that A^* has output a valid witness for C given just C which was drawn from a hard distribution. This contradicts the property of the hardness of the distribution. Further, the extractor cannot output Π_i for any $i = 1, \ldots, t$ because this would again contradict

[25] In general, we cannot guarantee indistinguishability of honestly generated proofs and simulated proofs even when a simulation trapdoor is given. However, it is safe for us to assume this since we use Groth-Sahai proofs which satisfy the stronger notion of composable zero-knowledge – even given a simulation trapdoor, we cannot distinguish honestly generated proofs from simulated proofs.

[26] This means that reRand algorithm is executed with R as its randomness

Claim 5. Thus, $(C, w) \in R$ or $T(\Pi_i) = \Pi$ for an already queried Π_i and allowable transformation T can happen only with negligible probability.

Summarising the above two claims, we have that the adversary A (and hence, A^*) outputs a valid proof for C with non-negligible probability and yet the extractor of $\mathbb{P}_{\mathsf{cm}}$ succeeds with only negligible probability. This contradicts the controlled malleability property of $\mathbb{P}_{\mathsf{cm}}$. This completes the proof. $\square$

F Applications of CLR encodings

The tool of CLR encodings along with witness encryption, as we see next, is a powerful tool to obtain many continual leakage resistant cryptographic primitives. The construction of witness encryption was first given in the work of Garg et. al. [GGSW13]. The general idea is to consider any cryptographic primitive that is built over witness encryption in such a way that the decryption algorithm of the cryptographic primitive is the same as the decryption algorithm of the witness encryption. Now, to make this continual leakage resistant we compute an encoding of the descryption key using our CLR encoding scheme. This works because if the original secret key was a valid witness then the encoding is a also a valid witness, and hence decryption can be done using the encoding (note that the relation for which the witness encryption is defined changes for the new scheme corresponding to the encoding scheme we are using).

Using the above intuition, we can obtain continual leakage resistant public key encryption, identity based encryption as well as attribute based encryption using the fact that all these applications were constructed using witness encryption in Garg et. al [GGSW13]. Since constructing these applications is (almost) a direct implementation of the above intuition, we just show how to obtain continual leakage resistant PKE and the other applications (IBE and ABE) can be constructed in a similar way. Due to the way CLR encodings are defined, we will not be able to obtain continual leakage resistant ABE for any attribute space but instead we obtain only for polynomial sized attribute space. See the remark in the next subsection for more details. In this section, we describe some of the applications stated in Section 1.

F.1 Example: CLR public key encryption schemes

We consider the public key encryption scheme constructed from witness encryption scheme by Garg et al. [GGSW13]. We modify their encryption scheme in such a way that the decryption key is a CLR encoding of the decryption key of the witness encryption scheme. Here, we assume a stronger property from the CLR encoding scheme in the CRS model – namely, simulatability of encodings. That is, there exists a simulator that can generate encodings for any x (including false statements) which are acceptable by the verifier algorithm of the encoding scheme. Also, for a true statement, any PPT adversary cannot distinguish between valid encodings and simulated encodings. Given any encoding scheme, we can transform it into a simulated encoding scheme using a NIZK proof system [27]. For completeness sake, we present the scheme verbatim from Garg et al. [GGSW13] with a straightforward modification using our CLR encoding.

$\mathsf{SetupPKE}(1^n)$: The setup algorithm chooses a random PRG seed $s \in \{0,1\}^n$. Next, it uses the PRG $G : \{0,1\}^n \rightarrow \{0,1\}^{2n}$ to compute $G(s) \rightarrow t \in \{0,1\}^{2n}$. The public key $PK = (t; n)$, is the output of the PRG and the security parameter. The decryption key is obtained by applying Encode to the seed s to obtain Π_s. The relation for which the encoding scheme is defined will be evident when we describe the encrypt operation.

$\mathsf{EncryptPKE}(PK = (t; n); M)$: To encrypt the algorithm prepares an instance x such that $x \in L$ if and only if t is in the range of G. It uses the Karp-Levin reduction to the NP-complete language L to do this. Next, it computes $EncryptWE(1^n; x; M) \rightarrow C$ to encrypt the message M for the instance x. The output ciphertext is $\mathsf{ct} = (x; C)$.

[27]Observe that the encoding scheme we construct is itself a simulatable encoding scheme since it is a NIZK proof.

$\mathsf{DecryptPKE}(\mathsf{SK} = s; \mathsf{ct} = (x; C))$: The decryption algorithm is given an instance x and witness encryption ciphertext C. If the ciphertext was formed properly, the algorithm can use s to obtain a witness w that $x \in L$. It then executes $\mathsf{WE.Decrypt}(C; w)$ to recover the encrypted message M.

This completes the description of the scheme. The decryption key is updated between executions using the Update algorithm.

We sketch the security argument, which follows the same template as in Garg et al. [GGSW13], that the above scheme is resistant against continual leakage. We follow the definition of continual leakage resilient PKE from Brakerski et al. [BKKV10], except that we do not allow leakage during the key generation phase and the update phase. At a high level, the security game proceeds as follows. The adversary receives a public key produced by a leak-free setup phase. After this, the adversary is given access to a leakage oracle containing the secret key. The adversary can submit two types of queries – leakage queries and update queries. The leakage queries comprises of queries where the adversary can leak on the secret key. Note that the amount of leakage in between two update queries is bounded by a leakage parameter $l(n)$. The adversary can also submit the update query after which the secret key is updated. As mentioned before, until the completion of the update phase the adversary does not have access to the leakage oracle. The adversary then choses the challenge messages m_0, m_1 and sends it to the challenger. The challenger encrypts m_b, with b picked uniformly at random and then sends the ciphertext back to the adversary. Note that, once the adversary gets the ciphertext it no longer has access to the leakage oracle. Now, the adversary outputs b'. The advantage of the adversary is defined to be $|Pr[b' = b] - \frac{1}{2}|$.

We first describe the hybrids. The first hybrid corresponds to the security game when message m_0 is encrypted and the last hybrid corresponds to the security game when m_1 is encrypted. We move from the first hybrid to the last hybrid as follows. In the second hybrid, we switch the valid encoding in the leakage oracle to simulated encodings. The computational indistinguishability of the hybrids follows from the fact that valid encodings and simulated encodings are computationally indistinguishable. In the third hybrid, we switch from encryption using a valid public key to an encryption using a fake public key (See Garg et al. [GGSW13] for more details). The indistinguishability of the second and the third hybrid follows from the security of the PRG scheme. In the fourth hybrid, we switch from encryption of m_0 (still using the fake public key) to encryption of m_1. The indistinguishability follows from the security of the witness encryption scheme. Now, we switch back from the fake public key to a valid public key and in this hybrid, still m_1 is encrypted. The indistinguishability of the third hybrid and the fourth hybrid follows from the security of the PRG scheme. In the fifth hybrid, which is the last hybrid, we use a valid encoding instead of a simulated encoding and the indistinguishability of this hybrid with the previous one follows from the fact that valid encodings and simulated encodings are computationally indistinguishable.

As mentioned before, in a similar way we can construct continual leakage resistant IBE and ABE. There is a caveat in the construction of ABE because of which we will only be able to handle polynomial (in the security parameter) sized attribute space as mention in the following remark.

Remark. In the construction of CLR ABE, during the key generation phase corresponding to a function f, we generate encodings for all attributes x such that $f(x) = 1$. This is possible only if the size of the attributes is polynomial in the security parameter. The reason we need to do this is because of an inherent limitation in the definition of CLR encodings. In the CLR encoding definition, we define the algorithm Encode such that it takes both input instance x as well as the witness w as input. Now, if there was an input instance x' whose witness was also w, using our encoding it is not possible to check whether x' is in the language or not. And so, even if two attributes share the same witness (by this, we mean that the proof that the function on the attribute gives 1), an encoding of the witness created corresponding to one attribute cannot be used to verify the other.