Adaptively Secure Garbled Circuits from One-Way Functions [*]

Brett Hemenway[†] Zahra Jafargholi[‡] Rafail Ostrovsky[§] Alessandra Scafuro[¶]
Daniel Wichs [‖]

Abstract

A garbling scheme is used to garble a circuit C and an input x in a way that reveals the output $C(x)$ but hides everything else. In many settings, the circuit can be garbled *off-line* without strict efficiency constraints, but the input must be garbled very efficiently *on-line*, with much lower complexity than evaluating the circuit. Yao's scheme has essentially optimal on-line complexity, but only achieves *selective security*, where the adversary must choose the input x prior to seeing the garbled circuit. It has remained an open problem to achieve *adaptive security*, where the adversary can choose x after seeing the garbled circuit, while preserving on-line efficiency.

In this work, we modify Yao's scheme in a way that allows us to prove adaptive security under one-way functions. As our main instantiation, we get a scheme where the on-line complexity is only proportional to the width w of the circuit, which corresponds to the space complexity of the computation, but is independent of the circuit's depth d. Alternately, we can also get an instantiation where the on-line complexity is only proportional to the input/output size and the depth d of the circuit but independent of its width w, albeit in this case we incur a $2^{O(d)}$ security loss in our reduction. More broadly, we relate the on-line complexity of adaptively secure garbling schemes in our framework to a certain type of *pebble* complexity of the circuit.

As our main tool, of independent interest, we develop a new notion of *somewhere equivocal* encryption, which allows us to efficiently equivocate on a small subset of the message bits.

[*]This work was done in part while some of the authors were visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant CNS-1523467.

[†]University of Pennsylvania. Department of Computer Science. fbrett@cis.upenn.edu

[‡]Northeastern University. Department of Computer Science. zahra@ccs.neu.edu

[§]University of California, Los Angeles. Department of Computer Science and Mathematics. rafail@cs.ucla.edu Research supported in part by NSF grants 09165174, 1065276, 1118126 and 1136174, US-Israel BSF grant 2008411, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. This material is based upon work supported in part by DARPA Safeware program. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

[¶]Boston University and Northeastern University. Department of Computer Science. scafuro@bu.edu Research supported by NSF grants 1012798.

[‖]Northeastern University. Department of Computer Science. wichs@ccs.neu.edu. Research supported by NSF grants CNS-1347350, CNS-1314722, CNS-1413964.

1 Introduction

Garbled Circuits. A *garbling scheme* (also referred to as a randomized encoding) can be used to garble a circuit C and an input x to derive a garbled circuit $\widetilde{C}$ and a garbled input $\widetilde{x}$. It's possible to evaluate $\widetilde{C}$ on $\widetilde{x}$ and get the correct output $C(x)$. However, the garbled values $\widetilde{C}, \widetilde{x}$ should not reveal anything else beyond this. In particular, there is a simulator that can simulate $\widetilde{C}, \widetilde{x}$ given only $C(x)$.

The notion of garbled circuits was introduced by Yao in (oral presentations of) [Yao82, Yao86], and can be instantiated based on one-way functions. Garbled circuits have since found countless applications in diverse areas of cryptography, most notably to secure function evaluation (SFE) starting with Yao's work, but also in parallel cryptography [AIK04, AIK05], verifiable computation [GGP10, AIK10], software protection [GKR08, GIS$^+$10], functional encryption [SS10, GVW12, GKP$^+$13], key-dependent message security [BHHI10, App11], obfuscation [App14] and many others. These applications rely on various efficiency/functionality properties of garbled circuits and a comprehensive study of this primitive is explored in the work of Bellare, Hoang and Rogaway [BHR12b].

On-line Complexity. In many applications, the garbled circuit $\widetilde{C}$ can be computed in an *off-line* pre-processing phase before the input is known, and therefore the efficiency of this procedure may not be of paramount importance. On the other hand, once the input x becomes available in the *on-line* phase, creating the garbled input $\widetilde{x}$ should be extremely efficient. Therefore, the main efficiency measure that we consider here is the *on-line complexity*, which is the time it takes to garble an input x, and hence also a bound on the size of $\widetilde{x}$. Ideally, the on-line complexity should only be linear in the input size $|x|$ and independent of the potentially much larger circuit size $|C|$.[1]

Yao's Scheme. Yao's garbling scheme already achieves essentially optimal on-line complexity, where the time to garble an input x and the size of $\widetilde{x}$ are only linear in the input size $|x|$, independent of the circuit size.[2] However, it only realizes a weak notion of security called *selective security*, which corresponds to a setting where adversary must choose the input x before seeing the garbled circuit $\widetilde{C}$. In particular, the adversary first chooses both C and x and then gets the garbled values $\widetilde{C}, \widetilde{x}$ which are either correctly computed using the "real" garbling scheme or "simulated" using only $C(x)$. The adversary should not be able to distinguish between the real world and the simulated world.

Selective vs. Adaptive Security. Selective security is often unsatisfactory in precisely the scenarios envisioned in the off-line/on-line setting, where the garbled circuit $\widetilde{C}$ is given out first and the garbled input $\widetilde{x}$ is only given out later. In such settings, the adversary may be able to (partially) influence the choice of the input x after seeing the garbled circuit $\widetilde{C}$. Therefore, we need a stronger notion called *adaptive security*, defined via the following two stage game:

1. The adversary chooses a circuit C and gets the garbled circuit $\widetilde{C}$.
2. After seeing $\widetilde{C}$ the adversary adaptively chooses an input x and gets the garbled input $\widetilde{x}$.

In the real world $\widetilde{C}, \widetilde{x}$ are computed correctly using the garbling scheme, while in the simulated world they are created by a simulator who only gets the output $C(x)$ in step (2) of the game but does not get the input x. The adversary should not be able to distinguish these two worlds.

The work of Bellare, Hoang and Rogaway [BHR12a] gave the first thorough treatment of adaptively secure garbling schemes and showed that this notion is crucial in many applications. They point out that it remains unknown whether Yao's garbling scheme or any of its many variants can satisfy adaptive security, and the proof techniques that work in the selective security setting do not extend to the adaptive setting.

[1] Note that, without any other restrictions on the structure of the garbling scheme, there is a trivial scheme where $\widetilde{C}$ is empty and $\widetilde{x} = C(x)$, whose on-line complexity is proportional to $|C|$.

[2] More precisely, in Yao's garbled circuits, the garbled input is of size $|x| \cdot \mathsf{poly}(\lambda)$ where λ is the security parameter. The work of [AIKW13] shows how to reduce this to $|x| + \mathsf{poly}(\lambda)$ assuming stronger assumptions such as DDH, RSA or LWE.

They left it as the main open problem to construct adaptively secure garbling schemes where the on-line complexity is smaller than the circuit size.[3],[4]

1.1 Prior Approaches to Adaptive Security

Lower Bound and Yao's scheme. The work of Applebaum et al. [AIKW13] (see also [HW15]) gives a lower bound on the on-line complexity of circuit garbling in the adaptive setting, showing that the size of the garbled input $\widetilde{x}$ must exceed the *output size* of the circuit. This is in contrast to the selective security setting, where Yao's garbling scheme achieves on-line complexity that depends only on the input size and not the output size. In particular, this shows that Yao's garbling scheme cannot directly be adaptively secure.

Complexity Leveraging. It turns out that there is a simple and natural modification of Yao's garbling scheme (i.e., by withholding the mapping of output-wire keys to output bits until the on-line phase) that would match the above lower bound and could plausibly be conjectured to provide adaptive security. In fact, one can prove that the above variant of Yao's scheme is secure in the adaptive setting using *complexity leveraging*, but only at a 2^n security loss in the reduction, where n is the input size. There is no known proof of security that avoids this loss.[5]

One-Time Pad and Random-Oracles. An alternate approach, suggested by [BHR12a], is to use one-time pad encryption to encrypt a Yao garbled circuit in the off-line phase and then provide the decryption key with the garbled input in the on-line phase. Intuitively, since a one-time pad encryption is "non-committing" and the ciphertext can be *equivocated* to any possible message by providing a corresponding key, the adversary does not gain any advantage in seeing such a ciphertext in the off-line phase. Unfortunately, this solution blows up the on-line complexity to be at least as large as the circuit size.

The work of [BHR12a] also noted that one can replace the one-time pad encryption in the above solution with a random-oracle based encryption scheme, which can be equivocated by programming random oracle outputs. This gives an adaptively secure garbled circuit construction with optimal parameters in the random oracle model. In fact, this approach can even be used to prove security in parameter regimes that beat the lower bound of [AIKW13], and therefore we should be suspicious about it's implications in the standard model, when the random oracle is replaced by a hash function. In particular, the construction is using the random oracle for equivocation in ways that we know to be uninstantiable in the standard model [Nie02].

Heavy Hammers. Lastly, we mention two approaches that get adaptively secure garbled circuits with good on-line complexity under significantly stronger assumptions than one-way functions. The work of Boneh et al. [BGG+14] implicitly provides such schemes where the on-line complexity is proportional to the input/output size and the depth d of the circuit, under the *learning with errors* assumption with a modulus-to-noise ratio of $2^{\mathsf{poly}(d)}$. This translates to assuming the hardness of lattice problems with $2^{\mathsf{poly}(d)}$ approximation factors. The work of Ananth and Sahai [AS15] shows how to get an essentially optimal schemes, where the on-line complexity is only proportional to the input/output size of the circuit, assuming *indistinguishability obfuscation*. In terms of both assumptions and practical efficiency, these schemes are a far cry from Yao's original scheme.

1.2 Our results

In this work, we construct the first adaptively secure garbling scheme whose on-line complexity is smaller than the circuit size and which only relies on the existence of one-way functions. Our construction is an adaptation of Yao' scheme that maintains essentially all of its desirable properties, such as having highly

[3] The adaptive security notion we described, is denoted $\mathsf{prv1}$ by [BHR12a]. They also consider a stronger variant called $\mathsf{prv2}$, where the adversary adaptively chooses bits of the input x one at a time and gets the corresponding bits of the garbled input $\widetilde{x}$. They show that there is an efficiency preserving transformation from $\mathsf{prv1}$ to $\mathsf{prv2}$ following the ideas from [GKR08]. Therefore, in this work we can focus solely on achieving $\mathsf{prv1}$.

[4] The problem of achieving adaptively secure garbled circuits is different from the problem of achieving adaptively secure two-party computation (with constant rounds) using an approach based on garbled circuits. We do not address the latter.

[5] Even if we're willing to assume exponentially secure primitives, the use of complexity leveraging blows up parameter sizes so that the garbled input must be of size at least $n^2 \cdot \mathsf{poly}(\lambda)$ where λ is the security parameter to get any meaningful security.

parallelizable circuit garbling and projective/decomposable input garbling.[6] In particular, our construction simply encrypts a Yao garbled circuit with a *somewhere equivocal* symmetric-key encryption scheme, which is a new primitive that we define and construct from one-way functions. The encrypted Yao garbled circuit is sent in the off-line phase, and the Yao garbled input along with the decryption key is sent in the on-line phase. We get various provably secure instantiations of the above approach depending on how we set the parameters of the encryption scheme.

As our main instantiation, we get a garbling scheme whose on-line complexity is $w \cdot \mathsf{poly}(\lambda)$ where w is the *width* of the circuit and λ is the security parameter, but is otherwise independent of the depth d of the circuit.[7] Note that, if we think of the circuit as representing a Turing Machine or RAM computation, then the width w of the circuit corresponds to the maximum of the input size n, output size m, and space complexity s of the computation, meaning that our on-line complexity is $(n+m+s)\cdot\mathsf{poly}(\lambda)$, but otherwise independent of the run-time of the computation.

Alternately, we also get a different instantiation where the on-line complexity is only $(n+m+d)\cdot\mathsf{poly}(\lambda)$, where n is the input size, m is the output size, and d is the depth of the circuit, but is otherwise independent of the circuit's width w. In this case, we also incur a $2^{O(d)}$ security loss in our reduction, but this can be a significant improvement over the naive complexity-leveraging approach which incurs a 2^n security loss, where n is the input size. In particular, in the case of NC^1 circuits where $d = O(\log n)$, we get a polynomial reduction and achieve optimal on-line complexity of $(n+m)\cdot\mathsf{poly}(\lambda)$.

More broadly, we develop a connection between constructing adaptively secure schemes in our framework and a certain type of *pebble complexity* of the given circuit. The size of the garbled input is proportional to the maximal number of pebbles and the number of hybrids in our reduction is proportional to the number of moves needed to pebble the circuit.

1.3 Applications of Our Results

We briefly mention how our results can be used to get concrete improvements in several applications of garbled circuits in prior works.

On-line/Off-line Two-Party Computation. One of the main uses of garbled circuits is in two-party secure computation protocols. In this setting, Alice holds an input x_A, Bob holds an input x_B and they wish to compute $f(x_A, x_B)$. To do so, Alice creates a garbled circuit $\widetilde{C}_f$ for the function f and sends $\widetilde{C}_f$ along with her portion of the garbled input $\widetilde{x}_A$ to Bob. Bob runs an oblivious transfer (OT) protocol to get the garbled version of his input $\widetilde{x}_B$ without revealing x_B to Alice. This can be done if the garbling scheme is projective/decomposable (see footnote 6) so that each bit of the garbled input only depends on one bit of the original input. Security against fully malicious parties can be obtained via zero-knowledge proofs or cut-and-choose techniques. It is possible to instantiate the above construction with selectively secure garbled circuits, by having Bob commit to x_B before he gets the garbled circuit $\widetilde{C}_f$. This ensures that the choice of the input cannot depend on the garbled circuit.

However, in many cases, creating the garbled circuit $\widetilde{C}_f$ for the function f is expensive and we would like to do this off-line before the inputs x_A, x_B are known to Alice and Bob. Once the inputs become known, the on-line phase should be extremely efficient, and ideally much smaller than the size of the circuit computing f. This setting was recently explored in the work of Lindell and Ben Riva [LR14] who showed how to solve this problem very efficiently using cut-and-choose techniques, given an adaptively secure garbling scheme with low on-line complexity. To instantiate the latter primitive, they relied on the random oracle model. Using our construction of adaptively secure garbled circuit, we can instantiate the scheme of [LR14] in the standard model, where the on-line complexity of the two-party computation protocol would match that of our garbling schemes.

One-Time Programs and Verifiable Computation. As noted by [BHR12a], two prior works from the literature on one-time programs [GKR08] and verifiable computation [GGP10] implicitly require adaptively

[6] Each bit of the garbled input only depends on one bit of the original input.

[7] We consider circuits made up of fan-in 2 gates with arbitrary fan-out. The circuit is composed of levels and wires can only connect gates in level i with those at the next level $i+1$. The width of the circuit is the maximal number of gates in any level and the depth is the number of levels.

secure garbling.[8] In both cases, we can plug in our construction of adaptively secure garbling to these constructions.

In the case of one-time programs, the on-line complexity of the garbling scheme translates to the number of hardware tokens needed to create the one-time program. In the case of verifiable computation, the on-line complexity of the garbling scheme translates to the complexity of the verification protocol – it is essential that this is smaller than the circuit size to make the verification protocol non-trivial.

Compact Functional Encryption. The recent work of [ABSV15] shows how to convert any selectively secure functional encryption (FE) scheme into an adaptively secure FE. However, their transformation is not compact and the ciphertext size is as large as the maximum circuit size of the allowed functions. This is true even if the selectively secure FE that they start with is compact. Implicitly, the main bottleneck in the transformation is having adaptively secure garbled circuits with low on-line complexity. The work of [AS15] gives an alternate and modular transformation from a selectively secure compact FE to an adaptively secure one using adaptively secure garbled circuits (actually, their main construction is for Turing Machines and relies on garbling TMs which require heavier machinery – however, it can be scaled down to work for circuits to get the above result). This transformation applies to both bounded-collusion schemes and unbounded-collusion schemes. By plugging in our construction of adaptively secure garbled circuits into the above result we get a transformation from compact selectively secure FE to adaptive FE where the ciphertext size is only proportional to the on-line complexity of our garbling scheme.

1.4 Our Techniques

In order to explain our techniques, we must first explain the difficulties in proving the adaptive security of Yao's garbling schemes. Since these difficulties are subtle, we begin with a description of the scheme and the proof of selective security, following Lindell and Pinkas [LP09]. This allows us to fix a precise notation and terminology which will be needed to also explain our new construction and proof. We expect that the reader is already familiar with the basics of Yao circuits and refer to [LP09] for further details.

1.4.1 Yao's Scheme and The Challenge of Adaptive Security

Yao's Scheme. For each wire w in the circuit, we pick two keys k_w^0, k_w^1 for a symmetric-key encryption scheme. For each gate in the circuit computing a function $g : \{0,1\}^2 \to \{0,1\}$ and having input wires a, b and output wire c we create a *garbled gate* consisting of 4 randomly ordered ciphertexts created as:

$$\begin{array}{ll} c_{0,0} = \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^0}(k_c^{g(0,0)})) & c_{1,0} = \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^0}(k_c^{g(1,0)})), \\ c_{0,1} = \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^1}(k_c^{g(0,1)})) & c_{1,1} = \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^1}(k_c^{g(1,1)})) \end{array} \tag{1}$$

where $(\mathsf{Enc}, \mathsf{Dec})$ is a CPA-secure encryption scheme. The garbled circuit $\widetilde{C}$ consists of all of the gabled gates, along with an *output mapping* $\{k_w^0 \to 0, k_w^1 \to 1\}$ which gives the correspondence between the keys and the bits they represent for each output wire w . To garble an n-bit value $x = x_1 x_2 \cdots x_n$, the garbled input $\widetilde{x}$ consists of the keys $k_{w_i}^{x_i}$ for the n input wires w_i.

To evaluate the garbled circuit on the garbled input, it's possible to decrypt (exactly) one ciphertext in each garbled gate and get the key $k_w^{v(w)}$ corresponding to the bit $v(w)$ going over the wire w during the computation $C(x)$. Once the keys for the output wires are computed, it's possible to recover the actual output bits by looking them up in the output mapping.

Selective Security Simulator. To prove the selective security of Yao's scheme, we need to define a simulator that gets the output $y = y_1 y_2 \cdots y_m = C(x)$ and must produce $\widetilde{C}, \widetilde{x}$. The simulator picks random keys k_1^0, k_w^1 for each wire w just like the real scheme, but it creates the garbled gates as follows:

$$\begin{array}{ll} c_{0,0} = \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^0}(k_c^0)) & c_{1,0} = \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^0}(k_c^0)), \\ c_{0,1} = \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^1}(k_c^0)) & c_{1,1} = \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^1}(k_c^0)) \end{array} \tag{2}$$

[8]The work of [GKR08] requires an even stronger notion of adaptivity called $\mathsf{prv2}$ but this can be generically achieved given an adaptively secure scheme in our sense. See footnote 3.

where all four ciphertext encrypt the same key k_c^0. It creates the output mapping $\{k_w^0 \to y_w, k_w^1 \to 1 - y_w\}$ by "programming it" so that the key k_w^0 corresponds to the correct output bit y_w for each output wire w. This defines the simulated garbled circuit $\widetilde{C}$. To create the simulated garbled input $\widetilde{x}$ the simulator simply gives out the keys k_w^0 for each input wire w. Note that, when evaluating the simulated garbled circuit on the simulated garbled input, the adversary only sees the keys k_w^0 for every wire w.

Selective Security Hybrids. To prove indistinguishability between the real world and the simulation, there is a series of carefully defined hybrid games that switch the distribution of one garbled gate at a time, starting with the input level and proceeding up the circuit level by level. In each step we switch the distribution of the ciphertexts in the targeted gate to:

$$\begin{array}{ll} c_{0,0} = \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^0}(k_c^{v(c)})) & c_{1,0} = \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^0}(k_c^{v(c)})), \\ c_{0,1} = \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^1}(k_c^{v(c)})) & c_{1,1} = \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^1}(k_c^{v(c)})) \end{array} \quad (3)$$

where $v(c)$ is the correct *value* of the bit going over the wire c during the computation of $C(x)$.

Let us give names to the three modes for creating garbled gates that we defined above: (1) is called $\mathsf{RealGate}$ mode, (2) is called $\mathsf{SimGate}$ mode, and (3) is called $\mathsf{InputDepSimGate}$ mode, since the way that it is defined depends adaptively on the choice of the input x.

We can switch a gate from $\mathsf{RealGate}$ to $\mathsf{InputDepSimGate}$ mode if the gates in the previous level are in $\mathsf{InputDepSimGate}$ mode (or we are in the input level) by CPA security of ecryption. In particular, we are *not* changing the value contained in ciphertext $c_{v(a),v(b)}$ encrypted under the keys $k_a^{v(a)}, k_b^{v(b)}$ that the adversary obtains during evaluation, but we *can* change the values contained in all of the other ciphertexts since the keys $k^{1-v(a)}, k^{1-v(b)}$ do not appear anywhere inside the garbled gates in the previous level.

At the end of the above sequence of hybrid games, all gates are switched from $\mathsf{RealGate}$ to $\mathsf{InputDepSimGate}$ mode and the output mapping is computed as in the real world. The resulting distribution is *statistically identical* to the simulation where all the gates are in $\mathsf{SimGate}$ mode and the output mapping is programmed. This is because, at any level that's not the output, the keys k_c^0, k_c^1 are used completely identically in the subsequent level so there is no difference between always encrypting $k_c^{v(c)}$ ($\mathsf{InputDepSimGate}$) and k_c^0 ($\mathsf{SimGate}$). At the output level there is no difference between encrypting $k_c^{v(c)}$ and giving the real mapping $k_c^{v(c)} \to y_c$ or encrypting k_c^0 and giving the programmed mapping $k_c^0 \to y_c$ where y_c is the output bit on wire c.

Challenges in Achieving adaptive security. There are two issues in using the above strategy in the adaptive setting: an immediate but easy to fix problem and a more subtle but difficult to overcome problem.

The first immediate issue is that the selective simulator needs to know the output $y = C(x)$ to create the garbled circuit $\widetilde{C}$ and in particular to program the output mapping $\{k_w^0 \to y_w, k_w^1 \to 1 - y_w\}$ for the output wires w. However, the adaptive simulator does not get the output y until *after* it creates the garbled circuit $\widetilde{C}$. Therefore, we cannot (even syntactically) use the selective security simulator in the adaptive setting. This issue turns out to be easy to fix by modifying the construction to send the output-mapping as part of the garbled input $\widetilde{x}$ in the on-line phase, rather than as part of the garbled circuit $\widetilde{C}$ in the off-line phase. This modification raises on-line complexity to also being linear in the output size of the circuit, which we know to be necessary by the lower bound of [AIKW13]. With this modification, the adaptive simulator can program the output mapping after it learns the output $y = C(x)$ in the on-line phase and therefore we get a syntactically meaningful simulation strategy in the adaptive setting.

The second problem is where the true difficulty lies. Although we have a syntactically meaningful simulation strategy, the previous proof of indistinguishability of the real world and the simulation completely breaks down in the adaptive setting. Recall that the proof consisted of a sequence of hybrids where we changed one garbled gate at a time (starting from the input level) from $\mathsf{RealGate}$ mode to the $\mathsf{InputDepSimGate}$ mode. In the latter mode, the gate is created in a way that depends on the input x, but in the adaptive setting the input x is chosen adaptively after the garbled circuit is created, leading to a circularity. In other words, the distribution of $\mathsf{InputDepSimGate}$ as specified in equation (3) doesn't even syntactically make sense in the adaptive setting. Therefore, *although we have a syntactically meaningful simulation strategy for the adaptive setting, we do not have any syntactically meaningful sequence of intermediate hybrids to prove indistinguishability between the real world and the simulated world.*

(One could hope to bypass InputDepSimGate mode altogether and define the hybrids by changing a gate directly from RealGate mode to SimGate mode. Unfortunately, this change is easily distinguishable already for the very first gate we would hope to change at the input level – the output value on the gate would no longer be $v(w)$ but 0 which may result in an overall incorrect output since we have not programmed the output map yet. On the other hand, we cannot immediately jump to a hybrid where we program the output map since all of the keys and their semantics are contained under encryption in prior levels of the circuit and we haven't argued about the security of the ciphertexts in these levels yet.)

1.4.2 Our Solution

Outer Encryption Layer. Our construction starts with the approach of [BHR12a] which is to encrypt the entire garbled circuit with an additional outer encryption layer in the off-line phase (this is unrelated to the encryption used to construct the garbled gates). Then, in the on-line phase, we give out the secret key for this outer encryption scheme. The approach of [BHR12a] required a symmetric-key, one-time encryption scheme which is *equivocal*, meaning that the ciphertext doesn't determine the message and it is possible to come up with a secret key that can open the ciphertext to any possible message. Unfortunately, any fully equivocal encryption scheme where a ciphertext can be opened to any message (e.g., the one-time pad) must necessarily have a secret key size which is as large as the message size. In our case, this is the entire garbled circuit and therefore this ruins the on-line efficiency of the scheme. Our main idea is to use a new type of *partially* equivocal encryption scheme, which we call *somewhere equivocal*.

Somewhere Equivocal Encryption. Intuitively, a somewhere equivocal encryption scheme allows us to create a simulated ciphertext which contains "holes" in some small subset of the message bit positions I chosen by the simulator, but all other message bits are fixed. The simulator can later equivocate this ciphertext and "plug the holes" with any bits it wants by deriving a corresponding secret key. An adversary cannot distinguish between seeing a real encryption of some message $m = m_1 m_2 \cdots m_n$ and the real secret key, from seeing a simulated encryption created using only $(m_i)_{i \notin I}$ with "holes" in positions I and an equivocated secret key that later plugs the holes to the correct bits $(m_i)_{i \in I}$. We show how to construct somewhere equivocal encryption using one-way functions. The size of the secret key is only proportional to the maximum number of holes $t = |I|$ that we allow, which we call the "equivocation parameter", but can be much smaller than the message size.[9]

Our proof of security departs significantly from that of [BHR12a]. In particular, our simulator does *not* take advantage of the equivocation property of the encryption scheme at all, and in fact, our simulation strategy is identical to the adaptive simulator we outlined above for the variant of Yao's garbling where the output map is sent in the on-line phase. However, we crucially rely on the equivocation property to carefully define a meaningful sequence of hybrids that allows us to prove the indistinguishability of the real and simulated worlds.

Hybrids for adaptive security. We define hybrid distributions where various garbled gates will be created in one of three modes discussed above: RealGate, SimGate and InputDepSimGate. However, to make the last option meaningful (even syntactically) in the adaptive setting, we rely on the somewhere equivocal encryption scheme. For these hybrids, when we create the encrypted garbled circuit in the off-line phase, we will simulate the outer encryption layer with a ciphertext that contains "holes" in place of all gates that are in InputDepSimGate mode. Only when we open the outer encryption in the on-line phase after the input x is chosen, we will "plug the holes" by sampling these gates correctly in InputDepSimGate mode in a way that depends on the input x. Our equivocation parameter t for the somewhere equivocal encryption scheme therefore needs to be large enough to support the maximum number of gates in InputDepSimGate mode that we will have in any hybrid.

Sequence of hybrids. For our main result, we use the following sequence of hybrids to prove indisitnguishability of real and simulated worlds. We start by switching the first two levels of gates (starting with

[9] A different notion of partially equivocal encryption, called *somewhat non-committing* encryption, was introduced in [GWZ09]. The latter notion allows a ciphertext to be opened to some small, polynomial size, set of messages which can be chosen arbitrarily by the simulator at encryption time. The two notions are incomparable.

the input level) to InputDepSimGate mode. We then switch the first level of gates to SimGate mode and switch the third level InputDepSimGate mode. We continue this process, where in each step i we maintain level i in InputDepSimGate mode but switch the previous level $i-1$ from InputDepSimGate to SimGate and then switch the next level $i+1$ from RealGate to InputDepSimGate. Eventually all gates will be in SimGate mode as we wanted. We can switch a level $i-1$ from InputDepSimGate to SimGate mode when the subsequent level i is in InputDepSimGate mode since the keys k_c^0, k_c^1 for wires c crossing from level $i-1$ to i are used identically in level i and therefore there is statistically no difference between encrypting the key $k_c^{v(c)}$ (InputDepSimGate) and k_c^0 (SimGate). We can also switch a level $i+1$ from RealGate to InputDepSimGate when the previous level i is InputDepSimGate (or $i+1$ is the input level) by CPA security following the same argument as in the selective setting. With this strategy, at any point in time we have at most two levels in InputDepSimGate mode and therefore our equivocation parameter only needs to be proportional to the circuit width w.

Connection to pebbling. We can generalize the above idea and get other meaningful sequences of hybrids with different parameters and implications. We can think of the process of switching between RealGate, SimGate and InputDepSimGate modes as a new kind of *graph pebbling game*, where pebbles can be placed on the graph representing the circuit according to certain rules. Initially, all gates are in RealGate mode, which we associate with *not having any pebble* on them. We associate InputDepSimGate mode with having a *black pebble* and SimGate mode with having a *gray pebble*. The rules of the game go as follows:

- We can place or remove a black pebble on a gate as long as both predecessors of that gate have black pebbles on them (or the gate is an input gate).
- We can replace a black pebble with a gray pebble on a gate as long as all successors of that gate have black or gray pebbles on them (or the gate is an output gate).

The goal of the game is to end up with a gray pebble on every gate. Any such pebbling strategy leads to a sequence of hybrids that shows the indistinguishability between the real world and the simulation. The number of moves needed to complete the pebbling corresponds to the number of hybrids in our proof, and therefore the security loss of our reduction. The maximum number of black pebbles that are in play at any given time corresponds to the equivocation parameter needed for our somewhere equivocal encryption scheme.

For example, the sequence of hybrids discussed above corresponds to a pebbling strategy where the number of black pebbles used is linear in the circuit width w (but independent of the depth) and the number of moves is linear in the circuit size. We give an alternate recursive pebbling strategy where the number of black pebbles used is linear in the circuit depth d (but independent of the width) and the number of moves is $2^{O(d)}$ times the circuit size.

Constructing somewhere equivocal encryption. Lastly, we briefly discuss our construction of somewhere equivocal encryption from one-way functions, which may be of independent interest. Recall that a somewhere equivocal encryption provides a method for equivocating some small number (t out of n) of bits of the message.

Our construction is based on the techniques developed in recent constructions of *distributed point functions* [GI14, BGI15]. These techniques give us a way to construct a pseudorandom function (PRF) family f_k with the following equivocation property: for any input x, we can create two PRF keys k_0, k_1 that each individually look uniformly random but such that $f_{k_0}(x') = f_{k_1}(x')$ for all $x' \neq x$ and $f_{k_0}(x) \neq f_{k_1}(x)$. The construction is based on a clever adaptation of the Goldreich-Goldwasser-Micali (GGM) PRF [GGM84].

Using distributed point functions, we can immediately create a somewhere equivocal encryption with equivocation parameter $t = 1$. We rely on a PRF family f_k with the above equivocation property and with one-bit output. To encrypt a message $m = m_1 m_2 \cdots m_n \in \{0,1\}^n$ we create a ciphertext $c = f_k(1) \oplus m_1 || f_k(2) \oplus m_2 || \cdots || f_k(n) \oplus m_n$ using the PRF outputs as a one-time pad. To create a simulated encryption with a hole in position i, the simulator samples two PRF keys k_0, k_1 that only differ on input $x = i$. The simulator encrypts the n-bit message by setting the unknown value in position i to $m_i := 0$ and using k_0. If it later wants to open this value to 0, it sets the decryption key to k_0 else k_1.

We can extend the above approach to an arbitrarily large equivocation parameter t, by using the XOR of t independently chosen PRFs with the above equivocation property. The key size will be $t \cdot \mathsf{poly}(\lambda)$.

2 Preliminaries

General Notation. For a positive integer n, we define the set $[n] := \{1, \dots, n\}$. We use the notation $x \leftarrow X$ for the process of sampling a value x according to the distribution X. For a vector $\overline{m} = (m_1, m_2, \cdots, m_n)$, and a subset $P \subset [n]$, we use $(m_i)_{i \in P}$ to denote a vector containing only the values m_i in positions $i \in P$ and $\perp$ symbols in all other positions. We use $(m_i)_{i \notin P}$ as shorthand for $(m_i)_{i \in [n] \setminus P}$.

Circuit Notation. A boolean circuit C consists of gates $\mathsf{gate}_1, \dots, \mathsf{gate}_q$ and wires $w_1, w_2, \dots, w_p$. A gate is defined by the tuple $\mathsf{gate}_i = (g, w_a, w_b, w_c)$ where $g : \{0,1\}^2 \to \{0,1\}$ is the function computed by the gate, w_a, w_b are the incoming wires, and w_c is the outgoing wire. Although each gate has a unique outgoing wire w_c, this wire can be used as an incoming wire to several different gates and therefore this models a circuit with fan-in 2 and unbounded fan-out. We let q denote the number of gates in the circuit, n denotes the number of input wires and m denote the number of output wires. The total number of wires is $p = n + q$ (since each wire can either be input wire or an outgoing wire of some gate). For convenience, we denote the n input wires by $\mathsf{in}_1, \dots, \mathsf{in}_n$ and the m output wires by $\mathsf{out}_1, \dots, \mathsf{out}_m$. For $x \in \{0,1\}^n$ we write $C(x)$ to denote the output of evaluating the circuit C on input x.

We say C is leveled, if each gate has an associated level and any gate at level l has incoming wires only from gates at level $l-1$ and outgoing wires only to gates at level $l+1$. We let the *depth* d denote the number of levels and the *width* w denote the maximum number of gates in any level.

A circuit C is fully specified by a list of gate tuples $\mathsf{gate}_i = (g, w_a, w_b, w_c)$. We use $\Phi_{\mathsf{topo}}(C)$ to refer to the topology of a circuit– which indicates how gates are connected, without specifying the function implement by each gate. In other words, $\Phi_{\mathsf{topo}}(C)$ is the list of *sanitized gate tuples* $\widehat{\mathsf{gate}}_i = (\perp, w_a, w_b, w_c)$ where the function g that the gate implements is removed from the tuple.

3 Garbling Scheme

We now give a formal definition of a garbling scheme. There are many variants of such definitions in the literature, and we refer the reader to [BHR12b] for a comprehensive treatment.

Definition 1. *A Garbling Scheme is a tuple of PPT algorithms* $\mathsf{GC} = (\mathsf{GCircuit}, \mathsf{GInput}, \mathsf{Eval})$ *such that:*

- $(\widetilde{C}, k) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, C)$*: takes as input a security parameter λ, a circuit $C : \{0,1\}^n \to \{0,1\}^m$, and outputs the* garbled circuit *$\widetilde{C}$, and* key *k.*
- $\tilde{x} \leftarrow \mathsf{GInput}(k, x)$*: takes as input, a string $x \in \{0,1\}^n$, and key k and outputs $\tilde{x}$.*
- $y = \mathsf{Eval}(\widetilde{C}, \tilde{x})$*: given a garbled circuit $\widetilde{C}$ and a garbled input $\tilde{x}$ output $y \in \{0,1\}^m$.*

Correctness *There is a negligible function ν such that for any $\lambda \in \mathbb{N}$, any circuit C and input x it holds that* $\Pr[C(x) = \mathsf{Eval}(\widetilde{C}, \tilde{x})] = 1 - \nu(\lambda)$*, where* $(\widetilde{C}, k) \leftarrow \mathsf{GCircuit}(1^\lambda, C)$*,* $\tilde{x} \leftarrow \mathsf{GInput}(k, x)$*.*

Adaptive Security. *There exists a PPT simulator* $\mathsf{Sim} = (\mathsf{SimC}, \mathsf{SimIn})$ *such that, for any PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\Pr[\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(1^\lambda, 1) = 1] \le \nu(\lambda)$$

where the experiment $\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(1^\lambda, b)$ *is defined as follows:*

1. *The adversary $\mathcal{A}$ specifies C and gets $\widetilde{C}$ where $\widetilde{C}$ is created as follows:*
    - *if $b = 0$:* $(\widetilde{C}, k) \leftarrow \mathsf{GCircuit}(1^\lambda, C)$,

- *if* $b = 1$*:* $(\widetilde{C}, \mathsf{state}) \leftarrow \mathsf{SimC}(1^\lambda, \Phi_{\mathsf{topo}}(C))$*, where* $\Phi_{\mathsf{topo}}(C)$ *reveals the topology of* C*.*

2. *The adversary* $\mathcal{A}$ *specifies* x *and gets* $\tilde{x}$ *created as follows:*

   - *if* $b = 0$*,* $\tilde{x} \leftarrow \mathsf{GInput}(k, x)$*,*
   - *if* $b = 1$*,* $\tilde{x} \leftarrow \mathsf{SimIn}(C(x), \mathsf{state})$*.*

3. *Finally, the adversary outputs a bit* b'*, which is the output of the experiment.*

On-line Complexity. The time it takes to garble an input x, (i.e., time complexity of $\mathsf{GInput}(\cdot,\cdot)$) is the *on-line complexity* of the scheme. Clearly the on-line complexity of the scheme gives a bound on the size of the garbled input $\widetilde{x}$. Ideally, the on-line complexity should be much smaller than the circuit size $|C|$.

Projective Scheme. We say a garbling scheme is *projective* if each bit of the garbled input $\widetilde{x}$ only depends on one bit of the actual input x. In other words, each bit of the input, is garbled independently of other bits of the input. Projective schemes are essential for two-party computation where the garbled input is transmitted using an oblivious transfer (OT) protocol. Our constructions will be projective.

Hiding Topology. A garbling scheme that satisfies the above security definition may reveal the topology of the circuit C. However, there is a way to transform any such garbling scheme into one that hides everything, including the topology of the circuit, without a significant asymptotic efficiency loss. More precisely, we rely on the fact that there is a function $\mathsf{HideTopo}(\cdot)$that takes a circuit C as input and outputs a functionally equivalent circuit C', such that for any two circuits C_1, C_2 of equal size, if $C'_1 = \mathsf{HideTopo}(C_1)$ and $C'_2 = \mathsf{HideTopo}(C_2)$, then $\Phi_{\mathsf{topo}}(C'_1) = \Phi_{\mathsf{topo}}(C'_2)$. An easy way to construct such function $\mathsf{HideTopo}$ is by setting C' to be a universal circuit, with a hard-coded description of the actual circuit C. Therefore, to get a topology-hiding garbling scheme, we can simply use a topology-revealing scheme but instead of garbling the circuit C directly, we garble the circuit $\mathsf{HideTopo}(C)$.

4 Somewhere Equivocal Symmetric-Key Encryption

We introduce a new cryptographic primitive called *somewhere* equivocal encryption scheme. Intuitively, a somewhere equivocal encryption scheme allows one to create a simulated ciphertext which contain "holes" in some small subset of the messages in positions I chosen by the simulator, but all other messages are fixed. The simulator can later equivocate this ciphertext and "plug the holes" with any message it wants by deriving a corresponding secret key.

In more detail, encryptions can be computed in two modes: real mode and simulated mode. In the real mode, a key $\mathsf{key} \leftarrow \mathsf{KeyGen}(1^\lambda)$ is generated using the honest key generation procedure and a vector of n messages $\overline{m} = m_1, \ldots, m_n$ is encrypted using the honest encryption procedure $\overline{c} \leftarrow \mathsf{Enc}(\mathsf{key}, \overline{m})$.

In the simulated mode, there is an encryption procedure SimEnc that gets in input a set I (set of holes) and only a subset of messages $(m_i)_{i \notin I}$ and outputs simulated ciphertext $\overline{c}$ that is equivocal in positions I. In a later stage, upon learning the remaining messages $(m_i)_{i \in I}$, there exists a procedure SimKey that plugs the holes by generating a key key' that will decrypt $\overline{c}$ correctly according to $\overline{m}$.

The security property that we require is that the distributions of $\{\overline{c}, \mathsf{key}\}$ generated in the two modes are indistinguishable. To capture this property, one could envision a non-adaptive security game where and adversary $\mathcal{A}$ first selects the full vector $\overline{m}$ and the set I, then it receives the tuple $(\overline{c}, \mathsf{key})$ and needs to distinguish which distribution it belongs to. However, such security definition is not sufficient for our indistinguishability proof where instead we need an adversary to decide on the missing messages *after* she receives the ciphertex $\overline{c}$. Therefore, we consider an adaptive security definition where the security game is defined in two stages: in the first stage, the adversary chooses I, an *incomplete* vector of messages $(m_i)_{i \notin I}$, and a *challenge* index $j \notin I$ and receives the ciphertex $\overline{c}$. In the second stage, the adversary sends the remaining messages $(m_i)_{i \in I}$ and gets key. The adversary knows that all positions in I are equivocal and are plugged to the values $(m_i)_{i \in I}$ chosen in the second stage. The challenge is to distinguish whether the position j is also equivocal or not. Note that this two-stage (adaptive) security definition is stronger than the non-adaptive security definition sketched above. For completeness, we give the simpler non-adaptive definition and prove the above implication in Appendix C.

Definition 2. *A* somewhere equivocal encryption scheme *with* block-length s, message-length n *(in blocks), and* equivocation-parameter t *(all polynomials in the security parameter) is a tuple of probabilistic polynomial algorithms* $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{SimEnc}, \mathsf{SimKey})$ *such that:*

- *The key generation algorithm* KeyGen *takes as input the security parameter* 1^λ *and outputs a key:* $\mathsf{key} \leftarrow \mathsf{KeyGen}(1^\lambda)$.
- *The encryption algorithm* Enc *takes as input a vector of* n *messages* $\overline{m} = m_1, \ldots, m_n$*, with* $m_i \in \{0,1\}^s$*, and a key* key*, and outputs ciphertext* $\overline{c} \leftarrow \mathsf{Enc}(\mathsf{key}, \overline{m})$.
- *The decryption algorithm* Dec *takes as input ciphertext* $\overline{c}$ *and a key* key *and outputs a vector of messages* $\overline{m} = m_1, \ldots, m_n$*. Namely,* $\overline{m} \leftarrow \mathsf{Dec}(\mathsf{key}, \overline{c})$.
- *The simulated encryption algorithm* SimEnc *takes as input a set of indexes* $I \subset [n]$*, such that* $|I| \leq t$*, and a vector of* $n - |I|$ *messages* $(m_i)_{i \notin I}$ *and outputs ciphertext* $\overline{c}$*, and a state* state*. Namely,* $(\mathsf{state}, \overline{c}) \leftarrow \mathsf{SimEnc}((m_i)_{i \notin I}, I)$.
- *The simulated key algorithm* SimKey*, takes as input the variable* state *and messages* $(m_i)_{i \in I}$ *and outputs a key* key'*. Namely,* $\mathsf{key}' \leftarrow \mathsf{SimKey}(\mathsf{state}, (m_i)_{i \in I})$.

and satisfies the following properties:

Correctness. *For every* $\mathsf{key} \leftarrow \mathsf{KeyGen}(1^\lambda)$*, for every* $\overline{m} \in \{0,1\}^{s \times n}$ *it holds that:*

$$\mathsf{Dec}(\mathsf{key}, (\mathsf{Enc}(\mathsf{key}, \overline{m})) = \overline{m}$$

Simulation with No Holes. *We require that the distribution of* $(\overline{c}, \mathsf{key})$ *computed via* $(\overline{c}, \mathsf{state}) \leftarrow \mathsf{SimEnc}(\overline{m}, \emptyset)$ *and* $\mathsf{key} \leftarrow \mathsf{SimKey}(\mathsf{state}, \emptyset)$ *to be identical to* $\mathsf{key} \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and* $\overline{c} \leftarrow \mathsf{Enc}(\mathsf{key}, \overline{m})$*. In other words, simulation when there are no holes (i.e.,* $I = \emptyset$*) is identical to honest key generation and encryption.*

Security. *For any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\nu = \nu(\lambda)$ *such that:*

$$\Pr[\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{A},\Pi}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{A},\Pi}(1^\lambda, 1) = 1] \leq \nu(\lambda)$$

where the experiment $\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{A},\Pi}$ *is defined as follows:*

Experiment $\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{A},\Pi}(1^\lambda, b)$

1. *The adversary* $\mathcal{A}$ *on input* 1^λ *outputs a set* $I \subseteq [n]$ *s.t.* $|I| < t$*, vector* $(m_i)_{i \notin I}$*, and a challenge index* $j \in [n] \setminus I$*. Let* $I' = I \cup j$.
2. – *If* $b = 0$*, compute* $\overline{c}$ *as follows:* $(\mathsf{state}, \overline{c}) \leftarrow \mathsf{SimEnc}((m_i)_{i \notin I}, I)$.
   – *If* $b = 1$*, compute* $\overline{c}$ *as follows:* $(\mathsf{state}, \overline{c}) \leftarrow \mathsf{SimEnc}((m_i)_{i \notin I'}, I')$.
3. *Send* $\overline{c}$ *to the adversary* $\mathcal{A}$.
4. *The adversary* $\mathcal{A}$ *outputs the set of remaining messages* $(m_i)_{i \in I}$.
   – *If* $b = 0$*, compute* key *as follows:* $\mathsf{key} \leftarrow \mathsf{SimKey}(\mathsf{state}, (m_i)_{i \in I})$.
   – *If* $b = 1$*, compute* key *as follows:* $\mathsf{key} \leftarrow \mathsf{SimKey}(\mathsf{state}, (m_i)_{i \in I'})$.
5. *Send* key *to the adversary* $\mathcal{A}$.
6. $\mathcal{A}$ *outputs* b' *which is the output of the experiment.*

In Appendix B, we construct somewhere equivocal encryption from one-way functions, proving the following theorem.

Theorem 1. *Assuming the existence of one-way functions, there exists a somewhere equivocal encryption scheme for any polynomial message-length* n*, block-length* s*, and equivocation parameter* t*, having key size* $t \cdot s \cdot \mathsf{poly}(\lambda)$ *and ciphertext of size* $n \cdot s$ *bits.*

5 Adaptively Secure Garbling Scheme and Simulator

In this section we describe our garbling scheme and the simulation strategy.

5.1 Construction

Our adaptively-secure garbling scheme consists in two simple steps: (1) garble the circuit using Yao's garbling scheme; (2) hide the garbled circuit (without the output tables) under an **outer** layer of encryption instantiated with a *somewhere-equivocal* encryption scheme. In the on-line phase, the garbled input consists of Yao's garbled input plus the output tables. Next we provide the formal description of our scheme that contains the details of Yao's garbling scheme.

Let C be a leveled boolean circuit with fan-in 2 and unbounded fan-out, with inputs size n, output size m, depth d and width w. Let q denote the number of gates in C. Recall that wires are uniquely identified with labels $w_1, w_2, \ldots, w_p$, and a circuit C is specified by a list of gate tuples $\mathsf{gate} = (g, w_a, w_b, w_c)$. The topology of the circuit $\Phi_{\mathsf{topo}}(C)$ consists of the sanitized gate tuples $\widehat{\mathsf{gate}}_i = (\bot, w_a, w_b, w_c)$. For simplicity, we implicitly assume that $\Phi_{\mathsf{topo}}(C)$ is public and known to the circuit evaluator without explicitly including it as part of the garbled circuit $\widetilde{C}$. To simplify the description of our construction, we first describe the procedure for garbling a single gate, that we denote by $\mathsf{GarbleGate}$.

Let $\Gamma = (G, E, D)$ be a CPA-secure symmetric-key encryption scheme satisfying the special correctness property defined in Appendix A. $\mathsf{GarbleGate}$ is defined as follows.

– $\widetilde{g} \leftarrow \mathsf{GarbleGate}(g, \{k_a^\sigma, k_b^\sigma, k_c^\sigma\}_{\sigma \in \{0,1\}})$: This function computes 4 ciphertexts $c_{\sigma_0,\sigma_1} : \sigma_0, \sigma_1 \in \{0,1\}$ as defined below and outputs them in a random order as $\widetilde{g} = [c_1, c_2, c_3, c_4]$.

$$c_{0,0} \leftarrow E_{k_a^0}(E_{k_b^0}(k_c^{g(0,0)})) \quad c_{0,1} \leftarrow E_{k_a^0}(E_{k_b^1}(k_c^{g(0,1)}))$$
$$c_{1,0} \leftarrow E_{k_a^1}(E_{k_b^0}(k_c^{g(1,0)})) \quad c_{1,1} \leftarrow E_{k_a^1}(E_{k_b^0}(k_c^{g(1,1)}))$$

Let $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{SimEnc}, \mathsf{SimKey})$ be a somewhere-equivocal symmetric-encryption scheme as defined in Sec. 4. Recall that in this primitive the plaintext is a vector of n blocks, each of which has s bits. In our construction we use the following parameters: the vector size $n = q$ is the number of gates and the block size $s = |\widetilde{g}|$ is the size of a single garbled gate. The equivocation parameter t is defined by the strategy used in the security proof and will be specified later. The garbling scheme is formally described in Fig. 1.

5.2 Adaptive Simulator

The adaptive security simulator for our garbling scheme is essentially the same as the static security simulator for Yao's scheme (as in [LP09]), with the only difference that the output table is sent in the on-line phase, and is computed adaptively to map to the correct output. Note that the garbled circuit simulator does not rely on the simulation properties of the somewhere equivocal encryption scheme - these are only used in the proof of indistinguishability.

More specifically, the adaptive simulator $(\mathsf{SimC}, \mathsf{SimIn})$ works as follows. In the off-line phase, SimC computes the garbled gates using procedure $\mathsf{GarbleSimGate}$, that generates 4 ciphertexts that encrypt the same output key. More precisely,

– $\mathsf{GarbleSimGate}(\{k_{w_a}^\sigma, k_{w_b}^\sigma\}_{\sigma \in \{0,1\}}, k'_{w_c})$ takes both keys for input wires w_a, w_b and a single key for the output wire w_c, that we denote by k'_{w_c}. It then output $\widetilde{g}_c = [c_1, c_2, c_3, c_4]$ where the ciphertexts, arranged in random order, are computed as follows.

$$c_{0,0} \leftarrow E_{k_a^0}(E_{k_b^0}(k'_c)) \quad c_{1,0} \leftarrow E_{k_a^1}(E_{k_b^0}(k'_c))$$
$$c_{0,1} \leftarrow E_{k_a^0}(E_{k_b^1}(k'_c)) \quad c_{1,1} \leftarrow E_{k_a^1}(E_{k_b^0}(k'_c))$$

The simulator invokes $\mathsf{GarbleSimGate}$ on input $k'_c = k_c^0$. It then encrypts the garbled gates so obtained by using the honest procedure for the somewhere equivocal encryption.

In the on-line phase, SimIn, on input $y = C(x)$ adaptively computes the output tables so that the evaluator obtains the correct output. This is easily achieved by associating each bit of the output, y_j, to the

```
GCircuit(1^λ, C)
1. Garble Circuit (Yao's scheme)
  – (Wires) $k^{\sigma}_{w_i} \leftarrow G(1^\lambda)$ for $i \in [p]$, $\sigma \in \{0,1\}$.
    (Input wires) $K = (k^0_{\mathsf{in}_i}, k^1_{\mathsf{in}_i})_{i\in[n]}$.
  – (Gates) For each $\mathsf{gate}_i = (g, w_a, w_b, w_c)$ in $C$:
    $\widetilde{g}_i \leftarrow \mathsf{GarbleGate}(g, \{k^{\sigma}_{w_a}, k^{\sigma}_{w_b}, k^{\sigma}_{w_c}\}_{\sigma\in\{0,1\}})$.
  – (Output tables) For each output $j \in [m]$:
    $\widetilde{d}_j := [(k^0_{\mathsf{out}_j} \to 0), (k^1_{\mathsf{out}_j} \to 1)]$.
2. Outer Encryption
  – $\mathsf{key} \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$.
  – $\widetilde{C} \leftarrow \mathsf{Enc}(\mathsf{key}, (\widetilde{g}_1, \ldots, \widetilde{g}_q))$.
Output $\widetilde{C}$, $k = (K, \mathsf{key}, (\widetilde{d}_j)_{j\in[m]})$.

GInput(x, k)
  – (Select input keys) $K^x = (k^{x_1}_{\mathsf{in}_1}, \ldots, k^{x_n}_{\mathsf{in}_n})$.
  – Output $\tilde{x} = (K^x, \mathsf{key}, (\widetilde{d}_j)_{j\in[m]})$.
```

```
Eval($\widetilde{C}, \tilde{x}$)
1. Parse $\tilde{x} = (K, \mathsf{key}, (\widetilde{d}_j)_{j\in[m]})$.
2. Decrypt Outer Encryption
   $(\widetilde{g}_i)_{i\in q} \leftarrow \mathsf{Dec}(\mathsf{key}, \widetilde{C})$.
3. Evaluate Circuit.
   Parse $K = (k_{\mathsf{in}_1}, \ldots, k_{\mathsf{in}_n})$.
   For each level $j = 1, \ldots, d$,
   For each $\widehat{\mathsf{gate}}_i = (\perp, w_a, w_b, w_c)$ at level $j$:
 – Let $\widetilde{g}_i = [c_1, c_2, c_3, c_4]$;
 – For $\delta \in [4]$ let $k'_{w_c} \leftarrow D_{k_{w_a}}(D_{k_{w_b}}(c_\delta))$
   If $k'_{w_c} \neq \perp$ then set $k_{w_c} := k'_{w_c}$.
4. Decrypt output.
   For $j \in [m]$:
     • Parse $\widetilde{d}_j = [(k^0_{\mathsf{out}_j} \to 0), (k^1_{\mathsf{out}_j} \to 1)]$.
     • Set $y_j = b$ iff $k_{\mathsf{out}_j} = k^b_{\mathsf{out}_j}$.
   Output $y_1, \ldots, y_m$.
```

Figure 1: Adaptively-secure Garbling Scheme.

only key encrypted in the output gate g_{out_j}, which is $k^0_{\mathsf{out}_j}$. For the input keys, SimIn just sends keys $k^0_{\mathsf{in}_i}$ for each $i \in [n]$. The detailed definition of $(\mathsf{SimC}, \mathsf{SimIn})$ is provided in Fig. 2.

5 Hybrid Games

We now need to prove the indistinguishability of our garbling scheme and the simulation. We devise a modular approach for proving indistinguishability using different strategies that result in different parameters. We first provide a template for defining hybrid games, where each such hybrid game is parametrized by a *circuit configuration*, that is, a vector indicating the way the gates are garbled and encrypted. Then we define the rules that allow us to indistinguishably move from one configuration to another. With this framework in place, an indistinguishability proof consists of a strategy to move from the gate configuration of the real game to the gate configuration of the simulated game, using the allowed rules.

5.1 Template for Defining Hybrid Games

Gate/Circuit Configuration. We start by defining a *gate configuration.* A gate configuration is a pair $(\mathsf{outer\ mode}, \mathsf{garbling\ mode})$ indicating the way a gate is computed. The outer encryption mode can be $\{\mathsf{EquivEnc}, \mathsf{BindEnc}\}$ depending on whether the outer encryption contains a "hole" in place of that gate or whether it is binding on that gate. The garbling mode can be $\{\mathsf{RealGate}, \mathsf{SimGate}, \mathsf{InputDepSimGate}\}$ which corresponds to the distributions outlined in Figure 3. We stress that, if the garbling mode of a gate is $\mathsf{InputDepSimGate}$ then we require that the outer encryption mode is $\mathsf{EquivEnc}$. This means that there are 5 valid gate configurations for each gate.

A *circuit configuration* simply consists of the gate configuration for each gate in the circuit. More specifically, we represent a circuit configuration by a tuple $(I, (\mathsf{mode}_i)_{i\in[q]})$ where

Simulator

$\underline{\mathsf{SimC}(1^\lambda, \Phi_{\mathsf{topo}}(C))}$

- (Wires) $k^\sigma_{w_i} \leftarrow G(1^\lambda)$ for $i \in [p]$, $\sigma \in \{0,1\}$.
- (Garbled gates) For each gate $\widetilde{\mathsf{gate}}_i = (\perp, w_a, w_b, w_c)$ in $\Phi_{\mathsf{topo}}(C)$:
  $\widetilde{g}_i \leftarrow \mathsf{GarbleSimGate}\ (\{k^\sigma_{w_a}, k^\sigma_{w_b}\}_{\sigma\in\{0,1\}}, k^0_{w_c})$.
- (Outer Encryption): $\mathsf{key} \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$, $\widetilde{C} \leftarrow \mathsf{Enc}(\mathsf{key}, \widetilde{g}_1, \ldots, \widetilde{g}_q)$.
- **Output** $\widetilde{C}$, $\mathsf{state} = (\{k^\sigma_{w_i}\}, \mathsf{key})$.

$\underline{\mathsf{SimIn}(y, \mathsf{state})}$

- Generate output table: $\widetilde{sd}_j \leftarrow [(k^{y_j}_{\mathsf{out}_j} \to 0), (k^{1-y_j}_{\mathsf{out}_j} \to 1)]_{j\in[m]}$. // ensures $k^0_{\mathsf{out}_j} \to y_j$
- **Output** $\widetilde{x} = ((k^0_{\mathsf{in}_i})_{i\in[n]}, \mathsf{key}, (\widetilde{sd}_j)_{j\in[m]})$.

Figure 2: Simulator for Adaptive Security.

RealGate	SimGate	InputDepSimGate
$c_{0,0} \leftarrow E_{k^0_a}(E_{k^0_b}(k_c^{g(0,0)}))$	$c_{0,0} \leftarrow E_{k^0_a}(E_{k^0_b}(k_c^0))$	$c_{0,0} \leftarrow E_{k^0_a}(E_{k^0_b}(k_c^{v(c)}))$
$c_{0,1} \leftarrow E_{k^0_a}(E_{k^1_b}(k_c^{g(0,1)}))$	$c_{0,1} \leftarrow E_{k^0_a}(E_{k^1_b}(k_c^0))$	$c_{0,1} \leftarrow E_{k^0_a}(E_{k^1_b}(k_c^{v(c)}))$
$c_{1,0} \leftarrow E_{k^1_a}(E_{k^0_b}(k_c^{g(1,0)}))$	$c_{1,0} \leftarrow E_{k^1_a}(E_{k^0_b}(k_c^0))$	$c_{1,0} \leftarrow E_{k^1_a}(E_{k^0_b}(k_c^{v(c)}))$
$c_{1,1} \leftarrow E_{k^1_a}(E_{k^1_b}(k_c^{g(1,1)}))$	$c_{1,1} \leftarrow E_{k^1_a}(E_{k^1_b}(k_c^0))$	$c_{1,1} \leftarrow E_{k^1_a}(E_{k^1_b}(k_c^{v(c)}))$

Figure 3: Garbling Gate modes: $\mathsf{RealGate}$ (left), $\mathsf{SimGate}$ (center), $\mathsf{InputDepSimGate}$ (right). The value $v(c)$ depends on the input x and corresponds to the bit going over the wire c in the computation $C(x)$.

- The set $I \subseteq [q]$ contains the indexes of the gates i whose outer mode is $\mathsf{EquivEnc}$.
- The value $\mathsf{mode}_i \in \{\mathsf{RealGate}, \mathsf{SimGate}, \mathsf{InputDepSimGate}\}$ describes the garbling mode of gate i.

A *valid circuit configuration* is one where all indexes i such that $\mathsf{mode}_i = \mathsf{InputDepSimGate}$ satisfy $i \in I$.

The Hybrid Game $\mathsf{Hyb}(I, (\mathsf{mode}_i)_{i\in[q]})$. Every valid circuit configuration $I, (\mathsf{mode}_i)_{i\in[q]}$ defines a hybrid game $\mathsf{Hyb}(I, (\mathsf{mode}_i)_{i\in[q]})$ as specified formally Figure 4 and described informally below. The hybrid game consists of two procedures: $\mathsf{GCircuit}'$ for creating the garbled circuit $\widetilde{C}$ and GInput' for creating the garbled input $\widetilde{x}$ respectively. The garbled circuit it created by picking random keys $k^\sigma_{w_j}$ for each wire w_j. For each gate i, such that $\mathsf{mode}_i \in \{\mathsf{RealGate}, \mathsf{SimGate}\}$ it creates a garbled gate $\widetilde{g}_i$ using the corresponding distribution as described in Figure 3. The garbled circuit $\widetilde{C}$ is then created by simulating the outer encryption using the values $\widetilde{g}_i$ in locations $i \notin I$ and "holes" in the locations I. The garbled input is created by first sampling the garbled gates $\widetilde{g}_i$ for each i such that $\mathsf{mode}_i = \mathsf{InputDepSimGate}$ using the corresponding distribution in Figure 3 and using knowledge of the input x. Then the decryption key key is simulated by plugging in the holes in locations I with the correctly sampled garbled gates $\widetilde{g}_i$. There is some subtlety about how the input labels $K[i]$ and the output label maps $\widetilde{d}_j$ are created when computing $\widetilde{x}$:

- If all of the gates having in_i as an input wire are in $\mathsf{SimGate}$ mode, then $K[i] := k^0_{\mathsf{in}_i}$ else $K[i] := k^{x_i}_{\mathsf{in}_i}$.
- If the unique gate having out_j as an output wire is in $\mathsf{SimGate}$ mode, then we give the simulated output map $\widetilde{d}_j := [(k^{y_j}_{\mathsf{out}_j} \to 0), (k^{1-y_j}_{\mathsf{out}_j} \to 1)]$ else the real one $\widetilde{d}_j := [(k^0_{\mathsf{out}_j} \to 0), (k^1_{\mathsf{out}_j} \to 1)]$.

```
**Game** $\mathsf{Hyb}(I, (\mathsf{mode}_i)_{i\in[q]})$
Garble circuit $C$:
– **Garble Gates**
(Wires) $k^{\sigma}_{w_i} \leftarrow G(1^{\lambda})$ for $i \in [p]$, $\sigma \in \{0,1\}$.
(Gates) For each $\mathsf{gate}_i = (g, w_a, w_b, w_c)$ in $C$.
    – If $\mathsf{mode}_i = \mathsf{RealGate}$:
      run $\widetilde{g}_i \leftarrow \mathsf{GarbleGate}(g, \{k^{\sigma}_{w_a}, k^{\sigma}_{w_b}, k^{\sigma}_{w_c}\}_{\sigma\in\{0,1\}})$.
    – if $\mathsf{mode}_i = \mathsf{SimGate}$:
      run $\widetilde{g}_i \leftarrow \mathsf{GarbleSimGate}(\{k^{\sigma}_{w_a}, k^{\sigma}_{w_b}\}_{\sigma\in\{0,1\}}, k^{0}_{w_c})$.
– **Outer Encryption.**
    1. $(\mathsf{state}, \widetilde{C}) \leftarrow \mathsf{SimEnc}((\widetilde{g}_i)_{i\notin I}, I)$.
    2. Output $\widetilde{C}$.

Garble Input $x$:
(Compute adaptive gates)
For each $i \in I$ s.t. $\mathsf{mode}_i = \mathsf{InputDepSimGate}$:
      Let $\mathsf{gate}_i = (g_i, w_a, w_b, w_c)$, and let $v(c)$
      be the bit on the wire $w_c$ during the computation $C(x)$.
      Set $\widetilde{g}_i \leftarrow \mathsf{GarbleSimGate}((k^{\sigma}_{w_a}, k^{\sigma}_{w_b})_{\sigma\in\{0,1\}}, k^{v(c)}_{w_c})$.
(Decryption key) $\mathsf{key}' \leftarrow \mathsf{SimKey}(\mathsf{state}, (\widetilde{g}_i)_{i\in I})$
(Output tables) Let $y = C(x)$. For $j = 1, \ldots, m$:
Let $i$ be the index of the gate with output wire $\mathtt{out}_j$.
    – If $\mathsf{mode}_i \neq \mathsf{SimGate}$, set $\widetilde{d}_j := [(k^{0}_{\mathsf{out}_j} \rightarrow 0), (k^{1}_{\mathsf{out}_j} \rightarrow 1)]$,
    – else, set $\widetilde{d}_j := [(k^{y_j}_{\mathsf{out}_j} \rightarrow 0), (k^{1-y_j}_{\mathsf{out}_j} \rightarrow 1)]$.
(Select input keys) For $j = 1, \ldots, n$:
    – If all gates $i$ having $\mathsf{in}_j$ as an input wire satisfy $\mathsf{mode}_i = \mathsf{SimGate}$, then set $K[i] := k^{0}_{\mathsf{in}_i}$,
    – else set $K[i] := k^{x_i}_{\mathsf{in}_i}$.
**Output** $\widetilde{x}17517 := (K, \mathsf{key}', \{\widetilde{d}_j\}_{j\in[m]})$.
```

Figure 4: The Hybrid Game.

Real game and Simulated Game. If we look at the definition of adaptively secure garbled circuits (Definition 1) then:

- The real game $\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(1^\lambda, 0)$ is equivalent to $\mathsf{Hyb}(I = \emptyset, (\mathsf{mode}_i = \mathsf{RealGate})_{i\in[q]})$.
- The simulated game $\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(1^\lambda, 1)$ is equivalent to $\mathsf{Hyb}(I = \emptyset, (\mathsf{mode}_i = \mathsf{SimGate})_{i\in[q]})$.

Therefore, the main aim is to show that these hybrids are indistinguishable.[10]

5.2 Rules for Indistinguishable Hybrids

Next, we provide rules that allow us to move from one configuration to another and prove that the corresponding hybrid games are indistinguishable. We define three rules that allow us to do this. For convenience, let us define $\mathsf{mode} \stackrel{\text{def}}{=} (\mathsf{mode}_i)_{i\in[q]}$.

5.2.1 Indistinguishability Rule 1: Changing the Outer Encryption Mode $\mathsf{BindEnc} \leftrightarrow \mathsf{EquivEnc}$.

This rule allows to change the outer encryption of a single gate. It says that one can move from a valid circuit configuration (I, mode) to a circuit configuration (I', mode) where $I' = I \cup j$. Thus one more gate is now computed equivocally (and vice versa).

Lemma 1. *Let* (I, mode) *be any valid circuit configuration, let* $j \in [q] \setminus I$ *and let* $I' = I \cup j$. *Then* $\mathsf{Hyb}(I, \mathsf{mode}) \stackrel{\text{comp}}{\approx} \mathsf{Hyb}(I', \mathsf{mode})$ *are computationally indistinguishable as long as* $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{SimEnc}, \mathsf{SimKey})$ *is a somewhere equivocal encryption scheme with equivocation parameter* t *such that* $|I'| \leq t$.

Proof. Towards a contradiction, assume there exists a PPT distinguisher $\mathcal{A}$ that distinguishes the distributions $H_0 = \mathsf{Hyb}(I, \mathsf{mode})$ and $H_1 = \mathsf{Hyb}(I', \mathsf{mode})$ as defined in the Lemma.

We construct a distinguisher B for the security of somewhere equivocal encryption scheme as follows. Adversary B is formally described in Fig. 5. Informally, adversary B is playing in experiment $\mathsf{Exp}^{\mathsf{simenc}}_{B,\Pi}(1^\lambda, b)$ and uses her oracle access to SimEnc to reproduce the distribution of H_b. B, on input I, j and $\mathsf{mode} = \mathsf{mode}_1, \ldots, \mathsf{mode}_q$ computes each garbled gate $\widetilde{g}_i$ on its own exactly as in H_0/ H_1 accordingly to mode_i. B computes the outer encryptions of the gates by sending the gates, along with sets I, j to $\mathsf{Exp}^{\mathsf{simenc}}$.

In the on-line phase, after obtaining x from $\mathcal{A}$, B computes the values for the missing gates $(\widetilde{g}_i)_{i\in I}$ and send them to $\mathsf{Exp}^{\mathsf{simenc}}$, and obtain a key key'. B uses such key to compute the garbled inputs $\widetilde{x}$.

Now, if B is playing the game $\mathsf{Exp}^{\mathsf{simenc}}_{B,\Pi}(1^\lambda, b)$ with a bit b, then the view generated by B is distributed identically to H_b. Thus, B distinguishes whether it is playing the game with $b = 0$ or $b = 1$ with the same probability that $\mathcal{A}$ distinguishes H_0 from H_1.

□

5.2.2 Indistinguishability Rule 2. Changing the Garbling Mode $\mathsf{RealGate} \leftrightarrow \mathsf{InputDepSimGate}$

This rules allows us to change the mode of a gate j from $\mathsf{RealGate}$ to $\mathsf{InputDepSimGate}$ under the conditions that $j \in I$ and that $\mathsf{gate}_j = (g, w_a, w_b, w_c)$ has incoming wires w_a, w_b that are either input wires or are the outgoing wires of some predecessor gates both of which are in $\mathsf{InputDepSimGate}$ mode.

Definition 3 (Predecessor/Successor/Sibling Gates)**.** *Given a circuit* C *and a gate* $j \in [q]$ *of the form* $\mathsf{gate}_j = (g, w_a, w_b, w_c)$ *with incoming wires* w_a, w_b *and outgoing wire* w_c*:*

- *We define the* predecessors *of* j*, denoted by* $\mathsf{Pred}(j)$*, to be the set of gates whose outgoing wires are either* w_a *or* w_b*. If* w_a, w_b *are input wires then* $\mathsf{Pred}(j) = \emptyset$*, else* $|\mathsf{Pred}(j)| = 2$.
- *We define the* successors *of* j*, denoted by* $\mathsf{Succ}(j)$ *to be the set of gates that contain* w_c *as an incoming wire. If* w_c *is an output wires then* $\mathsf{Succ}(j) = \emptyset$.

[10]Note that, the games $\mathsf{Hyb}(\cdots)$ use the simulated encryption and key generation procedures of the somewhere equivocal encryption, while the games $\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(1^\lambda, b)$ only use the real key generation and encryption procedures. However, by definition, these are equivalent when $I = \emptyset$ (no "holes").

```
**Adversary $B$ (Reduction)**
**Input:** $(I, j, (\mathsf{mode}_i)_{i \in [q]})$, s.t. $|I| < t$

Garble Circuit $C$:
– **Garble Gates**
(Wires) $k^{\sigma}_{w_i} \leftarrow G(1^{\lambda})$ for $i \in [p]$, $\sigma \in \{0,1\}$.
(Gates) For each $\mathsf{gate}_i = (g, w_a, w_b, w_c)$ in $C$.
    – If $\mathsf{mode}_i = \mathsf{RealGate}$:
      run $\widetilde{g}_i \leftarrow \mathsf{GarbleGate}(g, \{k^{\sigma}_{w_a}, k^{\sigma}_{w_b}, k^{\sigma}_{w_c}\}_{\sigma \in \{0,1\}})$.
    – if $\mathsf{mode}_i = \mathsf{SimGate}$:
      run $\widetilde{g}_i \leftarrow \mathsf{GarbleSimGate}(\{k^{\sigma}_{w_a}, k^{\sigma}_{w_b}\}_{\sigma \in \{0,1\}}, k^{0}_{w_c})$.
– **Outer Encryption**
    1. Set $(m_i)_{i \notin I} := (\widetilde{g}_i)_{i \notin I}$. Send $I, j, (m_i)_{i \notin I}$ to the challenger in $\mathsf{Exp}^{\mathsf{simenc}}$.
    2. Obtain $\overline{c} = c_i, \ldots, c_q$ from the challenger.
    3. Set $\widetilde{C} \leftarrow \overline{c}$.

Send $\widetilde{C}$ to $\mathcal{A}$. Obtain $x$ from $\mathcal{A}$.

Garble Input $x$:
(Compute adaptive gates)
For each $i \in I$ s.t. $\mathsf{mode}_i = \mathsf{InputDepSimGate}$:
      Let $\mathsf{gate}_i = (g_i, w_a, w_b, w_c)$, and let $v(c)$
      be the bit on the wire $w_c$ during the computation $C(x)$.
      Set $\widetilde{g}_i \leftarrow \mathsf{GarbleSimGate}((k^{\sigma}_{w_a}, k^{\sigma}_{w_b})_{\sigma \in \{0,1\}}, k^{v(c)}_{w_c})$.
(Decryption key)
    – Send $(\widetilde{g}_i)_{i \in I}$ to $\mathsf{Exp}^{\mathsf{simenc}}$.
    – Obtain $\mathsf{key}'$.
(Output tables) Let $y = C(x)$. For $j = 1, \ldots, m$:
Let $i$ be the index of the gate with output wire $\mathtt{out}_j$.
    – If $\mathsf{mode}_i \neq \mathsf{SimGate}$, set $\widetilde{d}_j := [(k^{0}_{\mathsf{out}_j} \to 0), (k^{1}_{\mathsf{out}_j} \to 1)]$,
    – else, set $\widetilde{d}_j := [(k^{y_j}_{\mathsf{out}_j} \to 0), (k^{1-y_j}_{\mathsf{out}_j} \to 1)]$.
(Select input keys) For $j = 1, \ldots, n$:
    – If all gates $i$ having $\mathsf{in}_j$ as an input wire satisfy $\mathsf{mode}_i = \mathsf{SimGate}$, then set $K[i] := k^{0}_{\mathsf{in}_i}$,
    – else set $K[i] := k^{x_i}_{\mathsf{in}_i}$.
Set: $\widetilde{x} := (K, \mathsf{key}', \{\widetilde{d}_j\}_{j \in [m]})$.

Send $\widetilde{x}$ to $\mathcal{A}$.

**Output** whatever $\mathcal{A}$ outputs. If $\mathcal{A}$ aborts, output $b \xleftarrow{\$} \{0,1\}$.
```

Figure 5: Proof of security for rule 1: the reduction B uses an adversary $\mathcal{A}$ that distinguishes the hybrids to play the security game $\mathsf{Exp}^{\mathsf{simenc}}$ of the somewhere equivocal encryption scheme.

- *We define the* siblings *of j, denoted by* $\mathsf{Siblings}(j)$ *to be the set of gates that contain either w_a or w_b as an incoming wire.*

Lemma 2. *Let $(I, \mathsf{mode} = (\mathsf{mode}_i)_{i\in[q]})$ be a valid circuit configuration and let $j \in I$ be an index such that $\mathsf{mode}_j = \mathsf{RealGate}$ and for all $i \in \mathsf{Pred}(j)$: $\mathsf{mode}_i = \mathsf{InputDepSimGate}$. Let $\mathsf{mode}' = (\mathsf{mode}'_i)_{i\in[q]}$ be defined by $\mathsf{mode}'_i = \mathsf{mode}_i$ for all $i \neq j$ and $\mathsf{mode}'_j = \mathsf{InputDepSimGate}$. Then the games $\mathsf{Hyb}(I, \mathsf{mode}) \overset{\text{comp}}{\approx} \mathsf{Hyb}(I, \mathsf{mode}')$ are computationally indistinguishable as long as $\Gamma = (G, E, D)$ is an encryption scheme secure under* chosen double encryption *as per Definition 5.*

Proof. Let I, mode, j and mode' be as in the statement of the Lemma. Towards a contradiction, assume that there exists a PPT adversary $\mathcal{A}$ distinguishing distributions generated in $H^0 := \mathsf{Hyb}(I, \mathsf{mode})$ and $H^1 := \mathsf{Hyb}(I, \mathsf{mode}')$.

We construct an adversary B that breaks the CPA-security of the inner encryption scheme $\Gamma = (G, E, D)$ which is used to garble gates. More specifically, we show that B wins the chosen double encryption security game (Def. 5) which is implied by CPA security. The formal description of adversary B is provided in Fig. 6.

Informally, B, on input mode, I and target gate j aims to use her CPA-oracle access in $\mathsf{Exp}^{\mathsf{double}}(1^\lambda, b)$ to generate a distribution H^b. Recall that the only difference between H^0 and H^1 is in the way that the garble gate $\widetilde{g}_j$ is computed. On a high level, the reduction B will compute all garbled gates $\widetilde{g}_i$ for $i \neq j$, according to experiment $\mathsf{Hyb}(I, \mathsf{mode})$, and will compute the garbled gate $\widetilde{g}_j$ using the ciphertexts obtained as a challenge in the experiment $\mathsf{Exp}^{\mathsf{double}}(1^\lambda, b)$.

In mored detail, let $\mathsf{gate}_j = (g, w_a, w_b, w_c)$ be the target gate. Recall $j \in I$ and therefore the value $\widetilde{g}_j$ is only needed in the on-line phase. If the values going over the wires w_a, w_b during the computation $C(x)$ are α, β respectively, the reduction B will know all wire keys *except* for $k_{w_a}^{1-\alpha}, k_{w_b}^{1-\beta}$. To create the garbled gate $\widetilde{g}_j$ it will create the cihertext $c_{\alpha,\beta}$ as an encryption of $k_{w_c}^{g(\alpha,\beta)}$ on its own, but the remaining three cihertexts $c_{\alpha',\beta'}$ will come from the experiment $\mathsf{Exp}^{\mathsf{double}}(1^\lambda, b)$ as either encryptions of different values $k_{w_c}^{g(\alpha',\beta')}$ (real) or of the same value $k_{w_c}^{g(\alpha,\beta)}$.

The one subtlety is that reduction needs to create encryptions under the keys $k_{w_a}^{1-\alpha}, k_{w_b}^{1-\beta}$ to create garbled gates $\widetilde{g}_i$ for gates i that are siblings of gate j. It can do that by using the encryption oracles which are given to it as part of the experiment $\mathsf{Exp}^{\mathsf{double}}(1^\lambda, b)$. However, since some of the sibling gates i might be in $\mathsf{RealGate}$ or $\mathsf{SimGate}$ modes, the reduction needs to create these encryptions already in the offline phase and therefore needs to know the values of α, β in the offline phase before the input x is chosen. To deal with this, we simply have the reduction *guess* the bits α, β randomly in the offline phase. If in the online phase it finds out that the guess is incorrect it outputs a random bit and aborts, else continues. The formal description of the reduction B is provided in Fig. 6.

Let *Correct* be the event that the reduction B guesses α and β correctly. Then

$$\begin{aligned}
&|\Pr[\mathsf{Exp}_B^{\mathsf{double}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}_B^{\mathsf{double}}(1^\lambda, 1) = 1]| \\
&= \frac{1}{4}|\Pr[\mathsf{Exp}_B^{\mathsf{double}}(1^\lambda, 0) = 1|\mathit{Correct}] - \Pr[\mathsf{Exp}_B^{\mathsf{double}}(1^\lambda, 1) = 1|\mathit{Correct}]| \\
&= \frac{1}{4}|\Pr[H_{\mathcal{A}}^0(1^\lambda) = 1] - \Pr[H_{\mathcal{A}}^1(1^\lambda)]|
\end{aligned}$$

Meaning that

$$|\Pr[H_{\mathcal{A}}^0(1^\lambda) = 1] - \Pr[H_{\mathcal{A}}^1(1^\lambda)]| \leq 4|\Pr[\mathsf{Exp}_B^{\mathsf{double}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}_B^{\mathsf{double}}(1^\lambda, 1) = 1]| \leq \mathsf{negl}(\lambda)$$

which proves the Lemma.

□

5.2.3 Indistinguishability Rule 3. Changing the Garbling Mode: InputDepSimGate ↔ SimGate.

This rule allows us to change the mode of a gate j from $\mathsf{InputDepSimGate}$ to $\mathsf{SimGate}$ under the condition that all successor gates $i \in \mathsf{Succ}(j)$ satisfy that $\mathsf{mode}_i \in \{\mathsf{InputDepSimGate}, \mathsf{SimGate}\}$.

Adversary B (Reduction)

Input $(I, \mathsf{mode}), j$.

Let $\mathsf{gate}_j = (g, w_a, w_b, w_c)$ the target gate.
Guess two bits $\alpha, \beta \leftarrow \{0,1\}$ uniformly at random.

<u>Garble circuit C:</u>

(Wires) Sample $k^{\sigma}_{w_i} \leftarrow G(1^{\lambda})$ for all $i \in [p]$, $\sigma \in \{0,1\}$ *except* for the two keys $k^{1-\alpha}_{w_a}, k^{1-\beta}_{w_b}$.

Let: $x_0 = k^{g(\alpha,1-\beta)}_{w_c}, y_0 = k^{g(1-\alpha,\beta)}_{w_c}, z_0 = k^{g(1-\alpha,1-\beta)}_{w_c}$ and $x_1 = y_1 = z_1 = k^{g(\alpha,\beta)}_{w_c}$.
Give $k^{\alpha}_{w_a}, k^{\beta}_{w_b}$ and $(x_0, y_0, z_0), (x_1, y_1, z_1)$ to the challenger of $\mathsf{Exp}^{\mathsf{double}}(1^{\lambda}, b)$.
The challenger of $\mathsf{Exp}^{\mathsf{double}}(1^{\lambda}, b)$ chooses two keys which we implicitly define as $k^{1-\alpha}_{w_a}, k^{1-\beta}_{w_b}$.
It gives B the ciphertexts c_x, c_y, c_z and oracle access to $E_{k^{1-\alpha}_{w_a}}(\cdot)$ and $E_{k^{1-\beta}_{w_b}}(\cdot)$.

(Gates) For each $\mathsf{gate}_i = (g', w'_a, w'_b, w'_c)$ in C such that $i \neq j$:

- If $\mathsf{mode}_i = \mathsf{RealGate}$:
  run $\widetilde{g}_i \leftarrow \mathsf{GarbleGate}(g', \{k^{\sigma}_{w'_a}, k^{\sigma}_{w'_b}, k^{\sigma}_{w'_c}\}_{\sigma\in\{0,1\}})$. [a]
- if $\mathsf{mode}_i = \mathsf{SimGate}$:
  run $\widetilde{g}_i \leftarrow \mathsf{GarbleSimGate}(\{k^{\sigma}_{w'_a}, k^{\sigma}_{w'_b}\}_{\sigma\in\{0,1\}}, k^{0}_{w'_c})$. [a]

(Outer Encryption:) Sample $(\mathsf{state}, \widetilde{C}) \leftarrow \mathsf{SimEnc}((\widetilde{g}_i)_{i\notin I}, I)$.

Send $\widetilde{C}$ to $\mathcal{A}$. Obtain x from $\mathcal{A}$.

<u>Garble Input x:</u>

If the values going over the wires w_a, w_b during the computation $C(x)$ are not α, β respectively, then abort.
(Compute adaptive gates)
For each $\mathsf{gate}_i = (g, w'_a, w'_{b_i}, w'_{c_i})$ in C s.t. $\mathsf{mode}_i = \mathsf{InputDepSimGate}$:

  Let v be the bit going over wire w'_c during the computation $C(x)$.

  Set $\widetilde{g}_i \leftarrow \mathsf{GarbleSimGate}((k^{\sigma}_{w'_{a_i}}, k^{\sigma}_{w'_{b_i}})_{\sigma\in\{0,1\}}, k^{v}_{w'_c})$. [a, b]

For the gate j:

- Compute $c_{\alpha,\beta} \leftarrow E_{k^{\alpha}_{w_a}}(E_{k^{\beta}_{w_b}}(k^{g(\alpha,\beta)}_{w_c}))$. Set $c_{\alpha,1-\beta} := c_x$, $c_{\alpha,1-\beta} := c_y$, $c_{1-\alpha,1-\beta} := c_z$.
- Let $\widetilde{g}_j$ be a random ordering of $[c_{0,0}, c_{0,1}, c_{1,0}, c_{1,1}]$

(Decryption key) $\mathsf{key}' \leftarrow \mathsf{SimKey}(\mathsf{state}, (\widetilde{g}_i)_{i\in I})$
(Output tables) Let $y = C(x)$. For $j = 1, \ldots, m$:
Let i be the index of the gate with output wire out_j.

- If $\mathsf{mode}_i \neq \mathsf{SimGate}$, set $\widetilde{d}_j := [(k^{0}_{\mathsf{out}_j} \to 0), (k^{1}_{\mathsf{out}_j} \to 1)]$,
- else, set $\widetilde{d}_j := [(k^{y_j}_{\mathsf{out}_j} \to 0), (k^{1-y_j}_{\mathsf{out}_j} \to 1)]$.

(Select input keys) For $j = 1, \ldots, n$:

- If all gates i having in_j as an input wire satisfy $\mathsf{mode}_i = \mathsf{SimGate}$, then set $K[i] := k^{0}_{\mathsf{in}_i}$,
- else set $K[i] := k^{x_i}_{\mathsf{in}_i}$.

Set $\widetilde{x} := (K, \mathsf{key}', \{\widetilde{d}_j\}_{j\in[m]})$. Send $\widetilde{x}$ to $\mathcal{A}$ and output whatever $\mathcal{A}$ outputs.

---

[a] To compute $\widetilde{g}_i$ for gates $i \in \mathsf{Siblings}(j)$, the reduction B needs to create encryptions under unknown keys $k^{1-\alpha}_{w_a}, k^{1-\beta}_{w_b}$. It can do so by calling the encryption oracles $E_{k^{1-\alpha}_{w_a}}(\cdot)$ and $E_{k^{1-\beta}_{w_b}}(\cdot)$.

[b] To compute $\widetilde{g}_i$ for gates $i \in \mathsf{Pred}(j)$, which are all in $\mathsf{InputDepSimGate}$ mode and for which $w'_c \in \{w_a, w_b\}$, the reduction B does *not* need to know $k^{1-\alpha}_{w_a}, k^{1-\beta}_{w_b}$ since the values going over the wires w_a, w_b are α, β respectively.

Figure 6: Proof of security for rule 2: the reduction B uses an adversary $\mathcal{A}$ that distinguishes the hybrids to play the chosen double encryption security game (Def. 5) denoted by $\mathsf{Exp}^{\mathsf{double}}$.

Lemma 3. *Let* $(I, \mathsf{mode} = (\mathsf{mode}_i)_{i \in [q]})$ *be a valid circuit configuration and let* $j \in I$ *be an index such that* $\mathsf{mode}_j = \mathsf{InputDepSimGate}$ *and for all* $i \in \mathsf{Succ}(j)$ *we have* $\mathsf{mode}_i \in \{\mathsf{SimGate}, \mathsf{InputDepSimGate}\}$. *Let* $\mathsf{mode}' = (\mathsf{mode}'_i)_{i \in [q]}$ *be defined by* $\mathsf{mode}'_i = \mathsf{mode}_i$ *for all* $i \neq j$ *and* $\mathsf{mode}'_j = \mathsf{SimGate}$. *Then the games* $\mathsf{Hyb}(I, \mathsf{mode}) \equiv \mathsf{Hyb}(I, \mathsf{mode}')$ *are identically distributed.*

Proof. Define $H_0 := \mathsf{Hyb}(I, \mathsf{mode})$ and $H_1 := \mathsf{Hyb}(I, \mathsf{mode}')$. Let $\mathsf{gate}_j = (g, w_a, w_b, w_c)$, and let $v(c)$ be the bit on the wire w_c during the computation $C(x)$, which is defined in the on-line phase.

The main difference between the hybrids is how the garbled gate $\widetilde{g}_j$ is created:

- In H_0, we set $\widetilde{g}_j \leftarrow \mathsf{GarbleSimGate}((k^{\sigma}_{w_a}, k^{\sigma}_{w_b})_{\sigma \in \{0,1\}}, k^{v(c)}_{w_c})$.
- In H_1, we set $\widetilde{g}_j \leftarrow \mathsf{GarbleSimGate}((k^{\sigma}_{w_a}, k^{\sigma}_{w_b})_{\sigma \in \{0,1\}}, k^{0}_{w_c})$.

If j is not an output gate, and all successor gates $i \in \mathsf{Succ}(j)$ are in $\{\mathsf{InputDepSimGate}, \mathsf{SimGate}\}$ modes then the keys $k^0_{w_c}$ and $k^1_{w_c}$ are treated symmetrically everywhere in the game other than in $\widetilde{g}_j$. Therefore, by symmetry, there is no difference between using $k^0_{w_c}$ and $k^{v(c)}_{w_c}$ in $\widetilde{g}_j$

If j is an output gate then the keys $k^0_{w_c}$ and $k^1_{w_c}$ are only used in $\widetilde{g}_j$ and in the output map $\widetilde{d}_j$. Therefore, by symmetry, there is no difference between using $k^{y_j}_{w_c}$ in $\widetilde{g}_j$ and setting $\widetilde{d}_j := [(k^0_{\mathsf{out}_j} \to 0), (k^1_{\mathsf{out}_j} \to 1)]$ (in H_0) versus using $k^0_{w_c}$ in $\widetilde{g}_j$ and setting $\widetilde{d}_j := [(k^{y_j}_{\mathsf{out}_j} \to 0), (k^{1-y_j}_{\mathsf{out}_j} \to 1)]$ (in H_1).

One last difference between the hybrids occurs if some wire in_i becomes only connected to gates that are in $\mathsf{SimGate}$ in H_1. In this case, when we create the garbled input $\widetilde{x}$, then in H_0 we give $K[i] := k^{x_i}_{\mathsf{in}_i}$ but in H_1 we give $K[i] := k^0_{\mathsf{in}_i}$. Since the keys $k^0_{\mathsf{in}_i}, k^1_{\mathsf{in}_i}$ are treated symmetrically everywhere in the game (both in H_0 and H_1) other than in $K[i]$, there is no difference between setting $K[i] := k^0_{\mathsf{in}_i}$ versus $K[i] := k^{x_i}_{\mathsf{in}_i}$.

□

6 Pebbling and Sequences of Hybrid Games

In the last section we defined hybrid games parameterized by a configuration (I, mode). We also gave 3 rules, which describe ways that allow us to indisitnguishably move from one configuration to another. Now our goal is to use the given rules so as to define a *sequence of indistinguishable hybrid games* that takes us from the *real game* $\mathsf{Hyb}(I = \emptyset, (\mathsf{mode}_i = \mathsf{RealGate})_{i \in [q]})$ to the simulation $\mathsf{Hyb}(I = \emptyset, (\mathsf{mode}_i = \mathsf{SimGate})_{i \in [q]})$.

Pebbling Game. We show that the problem of finding such sequences of hybrid games can be captured by a certain type of *pebbling game* on the circuit C. Each gate can either have *no pebble*, a *black pebble*, or *a gray pebble* on it (this will correspond to $\mathsf{RealGate}, \mathsf{InputDepSimGate}$ and $\mathsf{SimGate}$ modes respectively). Initially, the circuit starts out with no pebbles on any gate. The game consist of the following possible moves:

Pebbling Rule A. We can place or remove a black pebble on a gate as long as both predecessors of that gate have black pebbles on them (or the gate is an input gate).

Pebbling Rule B. We can replace a black pebble with a gray pebble on a gate as long as all successors of that gate have black or gray pebbles on them (or the gate is an output gate).

A *pebbling* of a circuit C is a sequence of γ moves that follow rules A and B and that end up with a gray pebble on every gate. We say that a pebbling uses t black pebbles if this is the maximal number of black pebbles on the circuit at any point in time during the game.

From Pebbling to Sequence of Hybrids. In our next theorem we prove that any pebbling of a circuit C results in a sequence of hybrids that shows indistinguishability of the real and simulated games. The number of hybrids is proportional to the number of moves in the pebbling and the equivocation parameter is proportional to the number of black pebbles it uses.

Theorem 2. *Assume that there is a pebbling of the circuit* C *in* γ *moves. Then there is a sequence of* $2 \cdot \gamma + 1$ *hybrid games, starting with the real game* $\mathsf{Hyb}(I = \emptyset, (\mathsf{mode}_i = \mathsf{RealGate})_{i \in [q]})$ *and ending with the simulated game* $\mathsf{Hyb}(I = \emptyset, (\mathsf{mode}_i = \mathsf{SimGate})_{i \in [q]})$ *such that any two adjacent hybrid games in the sequence* *are indistinguishable by rules 1,2 or 3 from the previous section. Furthermore if pebbling uses t^* black pebbles then every hybrid $\mathsf{Hyb}(I, \mathsf{mode})$ in the sequence satisfies $|I| \le t^*$. In particular, indistinguishability holds as long as the equivocation parameter is at least t^*.*

Proof. A *pebble configuration* specifies whether each gate contains no pebble, a black pebble, or a gray pebble. A pebbling in γ moves gives rise to a sequence of $\gamma + 1$ pebble configurations starting with no pebbles and ending with a gray pebble on each gate. Each pebble configuration follows from the preceding one by a move that satisfies pebbling rules A or B.

We let each pebble configuration define a hybrid $\mathsf{Hyb}(I, \mathsf{mode})$ where:

- For every gate $i \in [q]$, we set $\mathsf{mode}_i = \mathsf{RealGate}$ if gate i has no pebble, $\mathsf{mode}_i = \mathsf{InputDepSimGate}$ if gate i has a black pebble, and $\mathsf{mode}_i = \mathsf{SimGate}$ if gate i has a gray pebble.
- We set I to be the set of gates with black pebbles on them.

Therefore a pebbling defines a sequence of hybrids $\mathsf{Hyb}_\alpha = \mathsf{Hyb}(I^\alpha, \mathsf{mode}^\alpha)$ for $\alpha = 0, \ldots, \gamma$ where $\mathsf{Hyb}_0 = \mathsf{Hyb}(\emptyset, (\mathsf{mode}_i^0 = \mathsf{RealGate})_{i \in [q]})$ is the real game and $\mathsf{Hyb}_\gamma = \mathsf{Hyb}(\emptyset, (\mathsf{mode}_i^\gamma = \mathsf{SimGate})_{i \in [q]})$ is the simulated game, and each Hyb_α is induced by the pebbling configuration after α moves. We will need to add additional intermediate hybrids (which we call "half steps") to ensure that each pair of consecutive hybrids is indistinguishable by rules 1,2 or 3. We do this as follows:

- Assume that move $\alpha + 1$ of the pebbling applies rule A to place a black pebble on gate j.

  Let $\mathsf{Hyb}_\alpha = \mathsf{Hyb}(I^\alpha, \mathsf{mode}^\alpha)$ and $\mathsf{Hyb}_{\alpha+1} = \mathsf{Hyb}(I^{\alpha+1}, \mathsf{mode}^{\alpha+1})$. Then $I^{\alpha+1} = I^\alpha \cup \{j\}$, $\mathsf{mode}_i^{\alpha+1} = \mathsf{mode}_i^\alpha$ for all $i \ne j$, and $\mathsf{mode}_j^\alpha = \mathsf{RealGate}, \mathsf{mode}_j^{\alpha+1} = \mathsf{InputDepSimGate}$.

  Define the intermediate "half-step" hybrid $\mathsf{Hyb}_{\alpha+\frac{1}{2}} := \mathsf{Hyb}(I^{\alpha+1}, \mathsf{mode}^\alpha)$.

  It holds that $\mathsf{Hyb}_\alpha \overset{\text{comp}}{\approx} \mathsf{Hyb}_{\alpha+\frac{1}{2}}$ by rule 1, and $\mathsf{Hyb}_{\alpha+\frac{1}{2}} \overset{\text{comp}}{\approx} \mathsf{Hyb}_{\alpha+1}$ by rule 2. The conditions needed to apply rule 2 are implied by pebbling rule A.

- Assume that move $\alpha + 1$ of the pebbling applies rule A to remove a black pebble from gate j.

  Let $\mathsf{Hyb}_\alpha = \mathsf{Hyb}(I^\alpha, \mathsf{mode}^\alpha)$ and $\mathsf{Hyb}_{\alpha+1} = \mathsf{Hyb}(I^{\alpha+1}, \mathsf{mode}^{\alpha+1})$. Then $I^{\alpha+1} = I^\alpha \setminus \{j\}$, $\mathsf{mode}_i^{\alpha+1} = \mathsf{mode}_i^\alpha$ for all $i \ne j$, and $\mathsf{mode}_j^\alpha = \mathsf{InputDepSimGate}, \mathsf{mode}_j^{\alpha+1} = \mathsf{RealGate}$.

  Define the intermediate "half-step" hybrid $\mathsf{Hyb}_{\alpha+\frac{1}{2}} := \mathsf{Hyb}(I^\alpha, \mathsf{mode}^{\alpha+1})$.

  It holds that $\mathsf{Hyb}_\alpha \overset{\text{comp}}{\approx} \mathsf{Hyb}_{\alpha+\frac{1}{2}}$ by rule 2, and $\mathsf{Hyb}_{\alpha+\frac{1}{2}} \overset{\text{comp}}{\approx} \mathsf{Hyb}_{\alpha+1}$ by rule 1. The conditions needed to apply rule 2 are implied by pebbling rule A.

- Assume that move $\alpha + 1$ of the pebbling applies rule B to replace a black pebble with a gray pebble on gate j.

  Let $\mathsf{Hyb}_\alpha = \mathsf{Hyb}(I^\alpha, \mathsf{mode}^\alpha)$ and $\mathsf{Hyb}_{\alpha+1} = \mathsf{Hyb}(I^{\alpha+1}, \mathsf{mode}^{\alpha+1})$. Then $I^{\alpha+1} = I^\alpha \setminus \{j\}$, $\mathsf{mode}_i^{\alpha+1} = \mathsf{mode}_i^\alpha$ for all $i \ne j$, and $\mathsf{mode}_j^\alpha = \mathsf{InputDepSimGate}, \mathsf{mode}_j^{\alpha+1} = \mathsf{SimGate}$.

  Define the intermediate "half-step" hybrid $\mathsf{Hyb}_{\alpha+\frac{1}{2}} := \mathsf{Hyb}(I^\alpha, \mathsf{mode}^{\alpha+1})$.

  It holds that $\mathsf{Hyb}_\alpha \overset{\text{comp}}{\approx} \mathsf{Hyb}_{\alpha+\frac{1}{2}}$ by rule 3, and $\mathsf{Hyb}_{\alpha+\frac{1}{2}} \overset{\text{comp}}{\approx} \mathsf{Hyb}_{\alpha+1}$ by rule 1. The conditions needed to apply rule 3 are implied by pebbling rule B.

Therefore the sequence $\mathsf{Hyb}_0, \mathsf{Hyb}_{\frac{1}{2}}, \mathsf{Hyb}_1, \mathsf{Hyb}_{1+\frac{1}{2}}, \mathsf{Hyb}_2, \ldots, \mathsf{Hyb}_\gamma$ consisting of $2\gamma + 1$ hybrids satisfies the conditions of the theorem.

□

Combining Theorem 2 and Theorem 1 we obtain the following corollary.

Corollary 1. *There exists an adaptively secure garbling scheme such that the following holds.*

Assuming the existence of one-way functions, there is an instantiation of the garbling scheme that has on-line complexity $(n + m + t^*)\mathsf{poly}(\lambda)$ for any circuit C that admits a pebbling with $\gamma = \mathsf{poly}(\lambda)$ moves and t^* black pebbles.

Furthermore, assuming the existence of sub-exponentially secure one-way functions, there is an instantiation of the garbling scheme that has on-line complexity $(n+m+t^*)\mathsf{poly}(\lambda, \log\gamma)$ *for any circuit* C *admits a pebbling strategy with* $\gamma = 2^{\mathsf{poly}(\lambda)}$ *moves and* t^* *black pebbles.*

Proof. We instantiate our construction from Section 5 with a CPA-secure "inner encryption" Γ having special correctness, and a somewhere-equivocal "outer encryption" Π from Section 4 using an equivocation parameter $t = t^*$. Both components can be instantiated from one-way functions.

Assuming that $\gamma = \mathsf{poly}(\lambda)$, Theorem 2 tells us that the resulting garbling scheme is adaptively as long as Γ, Π are. The on-line complexity consists of $n+m$ keys for Γ along with the key of Π for a total of $(n+m)\mathsf{poly}(\lambda) + t^*\mathsf{poly}(\lambda)$ as claimed.

When $\gamma = 2^{\mathsf{poly}(\lambda)}$, then Theorem 2 tells us that the resulting garbling scheme is adaptively as long as the schemes Γ, Π provide a higher level of security so as to survive $2\gamma+1$ hybrids, meaning that the distinguishing advantage for each of the schemes needs to be $2^{-(2\gamma+1)}\mathsf{negl}(\lambda)$. This can be accomplished assuming sub-exponentially secure one-way functions by setting the security parameter of Γ, Π to some $\lambda' = \mathsf{poly}(\lambda, \log\gamma)$ and results in on-line complexity $(n+m)\mathsf{poly}(\lambda, \log\gamma) + t^*\mathsf{poly}(\lambda, \gamma)$ as claimed.

□

6.1 Pebbling Strategies

In this section we give two pebbling strategies for arbitrary circuit with width w, depth d, and q gates. The first strategy uses $O(q)$ moves and $O(w)$ black pebbles. The second strategy uses $O(q2^d)$ moves and $O(d)$ black pebbles.

6.1.1 Strategy 1

To pebble the circuit proceed as follows:

```
Pebble($C$):
  1. Put a black pebble on each gate at the input level (level 1).
  2. For $i = 1$ to $d-1$, repeat:
     (a) Put a black pebble on each gate at level $i+1$.
     (b) For each gate at level $i$, replace the black pebble with a gray pebble.
     (c) $i \leftarrow i+1$
  3. For each gate at level $d$, replace the black pebble with a gray pebble.
```

This strategy uses $\gamma = 2q$ moves and $t^* = 2w$ black pebbles. By instantiating Corollary 1 with this strategy, we obtain the following corollary.

Corollary 2. *Assuming the existence of one-way functions there exists an adaptively secure garbling scheme with on-line complexity* $w \cdot \mathsf{poly}(\lambda)$*, where* w *is the width of the circuit.*

6.1.2 Strategy 2

This is a recursive strategy defined as follows.

```
• Pebble($C$):
    – For each gate $i$ in $C$ starting with the gates at the top level moving to the bottom level:
        1. RecPutBlack($C, i$)
        2. Replace the black pebble on gate $i$ with a gray pebble.

• RecPutBlack($C, i$): // Let LeftPred($C, i$) and RightPred($C, i$) are the two predecessors of gate $i$ in $C$.
    1. If gate $i$ is an input gate, put a black pebble on $i$ and **return**.
    2. Run RecPutBlack($C$, LeftPred($C, i$)), RecPutBlack($C$, RightPred($C, i$))
```

3. Put a black pebble on gate i.
4. Run $\mathsf{RecRemoveBlack}(C, \mathsf{LeftPred}(C,i))$ and $\mathsf{RecRemoveBlack}(C, \mathsf{RightPred}(C,i))$,

- $\mathsf{RecRemoveBlack}(C,i)$: This is the same as $\mathsf{RecPutBlack}$, except that instead of putting a black pebble on gate i, in steps 1 and 3, we remove it.

To analyze the correctness of this strategy, we note the following invariants: if the circuit C is in a configuration where it does not contain any pebbles at any level below that of gate i, then (1) the procedure $\mathsf{RecPutBlack}(C,i)$ results in a configuration where a single black pebble is added to gate i, but nothing else changes, (2) the procedure $\mathsf{RecRemoveBlack}(C,i)$ results in a configuration where a single black pebble is removed from gate i, but nothing else changes. Using these two invariants the correctness of of the entire strategy follows.

To calculate the number of black pebbles used and the number of moves that the above strategy takes to pebble C, we use the following simple recursive equations. Let $\#\mathsf{PebPut}(d)$ and $\#\mathsf{PebRem}(d)$ be the number of black pebbles on gate i and below it used to execute $\mathsf{RecPutBlack}$ and $\mathsf{RecRemoveBlack}$ on a gate at level d, respectively. We have,

$$\#\mathsf{PebPut}(1) = 1, \quad \#\mathsf{PebPut}(d) \leq \max(\#\mathsf{PebPut}(d-1), \#\mathsf{PebRem}(d-1)) + 2$$
$$\#\mathsf{PebRem}(1) = 1, \quad \#\mathsf{PebRem}(d) \leq \max(\#\mathsf{PebPut}(d-1), \#\mathsf{PebRem}(d-1)) + 2$$

Therefore the strategy requires at most $2d$ black pebbles to pebble the circuit.

To calculate the number of moves it takes run $\mathsf{Pebble}(C)$, we use the following recursive equations. Let $\#\mathsf{Moves}(d)$ be the number of moves it takes to put a black pebble on, or remove a black pebble from, a gate at level d. Then

$$\#\mathsf{Moves}(1) = 1, \quad \#\mathsf{Moves}(d) = 4(\#\mathsf{Moves}(d-1)) + 1$$

Hence, each call of $\mathsf{RecPutBlack}$ takes at most 4^d moves, and the total number of moves to pebble the circuit is at most $q4^d$.

In summary, the above gives us a strategy to pebble any circuit with at most $\gamma = q4^d$ moves and $t^* = 2d$ black pebbles. By instantiating Corollary 1 with the above strategy, we obtain the following corollary.

Corollary 3. *Assuming the existence of (standard) one-way functions, there exists an adaptively secure garbling schemes that has on-line complexity* $(n+m)\mathsf{poly}(\lambda)$ *for all circuits having depth* $d = O(\log \lambda)$*.*

Assuming the existence of sub-exponentially secure one-way functions, there exists an adaptively secure garbling scheme that has on-line complexity $(n+m)\mathsf{poly}(\lambda, d)$*, for arbitrary circuits of depth* $d = \mathsf{poly}(\lambda)$*.*

7 Conclusions

We have shown how to achieve adaptively secure garbling schemes under one-way functions by augmenting Yao's construction with an additional layer of somewhere-equivocal encryption. The on-line complexity in our constructions can be significantly smaller than the circuit size. In our main instantiation, the on-line complexity only scales with the width w of the circuit, which corresponds to the space complexity of the computation.

It remains as an open problem to get the optimal on-line complexity $(n+m)\mathsf{poly}(\lambda)$ which does not depend on the circuit depth or width. Currently, this is only known assuming the existence of indistinguishability obfuscation and therefore it remains open to achieve the above under one-way functions or even stronger assumptions such as DDH or LWE. It also remains open if Yao's scheme (or more precisely, a variant of it where the output map is sent in the on-line phase) can already achieve adaptive security without relying on somewhere-equivocal encryption. We have no proof nor a counter-example. It would be interesting to see if there is some simple-to-state standard-model security assumption that one could make on the encryption scheme used to create the garbled gates in Yao's construction (e.g., circular security, key-dependent message security, etc.), under which one could prove that the resulting garbling scheme is adaptively secure.

References

[ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 657–677. Springer, Heidelberg, August 2015.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *45th FOCS*, pages 166–175. IEEE Computer Society Press, October 2004.

[AIK05] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. In *20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA*, pages 260–274. IEEE Computer Society, 2005.

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *ICALP 2010, Part I*, volume 6198 of *LNCS*, pages 152–163. Springer, Heidelberg, July 2010.

[AIKW13] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 166–184. Springer, Heidelberg, August 2013.

[App11] Benny Applebaum. Key-dependent message security: Generic amplification and completeness. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 527–546. Springer, Heidelberg, May 2011.

[App14] Benny Applebaum. Bootstrapping obfuscators via fast pseudorandom functions. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 162–172. Springer, Heidelberg, December 2014.

[AS15] Prabhanjan Ananth and Amit Sahai. Functional encryption for turing machines. Cryptology ePrint Archive, Report 2015/776, 2015. http://eprint.iacr.org/.

[BGG$^+$14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Heidelberg, May 2014.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 337–367. Springer, Heidelberg, April 2015.

[BHHI10] Boaz Barak, Iftach Haitner, Dennis Hofheinz, and Yuval Ishai. Bounded key-dependent message security. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 423–444. Springer, Heidelberg, May 2010.

[BHR12a] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 134–153. Springer, Heidelberg, December 2012.

[BHR12b] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12*, pages 784–796. ACM Press, October 2012.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 276–288. Springer, Heidelberg, August 1984.

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

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 465–482. Springer, Heidelberg, August 2010.

[GI14] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658. Springer, Heidelberg, May 2014.

[GIS$^+$10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 308–326. Springer, Heidelberg, February 2010.

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 39–56. Springer, Heidelberg, August 2008.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 162–179. Springer, Heidelberg, August 2012.

[GWZ09] Juan A. Garay, Daniel Wichs, and Hong-Sheng Zhou. Somewhat non-committing encryption and efficient adaptively secure oblivious transfer. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 505–523. Springer, Heidelberg, August 2009.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015*, pages 163–172. ACM, January 2015.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[LR14] Yehuda Lindell and Ben Riva. Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 476–494. Springer, Heidelberg, August 2014.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 111–126. Springer, Heidelberg, August 2002.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 10*, pages 463–472. ACM Press, October 2010.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

A Symmetric-Key Encryption with Special Correctness [LP09]

In our construction of the garbling scheme, we use a symmetric-key encryption scheme $\Gamma = (G, E, D)$ which satisfies the standard definition of CPA security and an additional *special correctness* property below (this is a simplified and sufficient variant of the property described in from [LP09]). We need this property to ensure the correctness of our garbled circuit construction.

Definition 4 (Special Correctness)**.** *A CPA-secure symmetric-key encryption $\Gamma = (G, E, D)$ satisfies special correctness if there is some negligible function ν such that for any message m we have:*

$$\Pr[D_{k_2}(E_{k_1}(m)) \neq \bot \ : \ k_1, k_2 \leftarrow G(1^\lambda)]] \leq \nu(\lambda).$$

Construction. Let $F = \{f_k\}$ be a family of pseudorandom functions where $f_k : \{0,1\}^\lambda \to \{0,1\}^{\lambda+s}$, for $k \in \{0,1\}^\lambda$ and s is a parameter denoting the message length. Define $E_k(m) = (r, f_k(r) \oplus m0^\lambda)$ where $m \in \{0,1\}^s$, $r \xleftarrow{\$} \{0,1\}^\lambda$ and $m0^\lambda$ denotes the concatenation of m with a string of 0s of length λ. Define $D_k(c)$ which parses $c = (r, z)$, computes $w = z \oplus f_k(r)$ and if the last λ bits of w are 0's it outputs the first s bits of w, else it outputs $\bot$.

It's easy to see that this scheme is CPA secure and that it satisfies the special correctness property.

Double Encryption Encryption Security. For convenience, we define a notion of double encryption security, following [LP09]. This notion is implied by standard CPA security but is more convenient to use in our security proof of garbled circuit security.

Definition 5 (Double-encryption security.)**.** *An encryption scheme $\Gamma = (G, E, D)$ is secure under* chosen double encryption *if for every PPT machine $\mathcal{A}$, there exists a negligible function $\nu = \nu(\lambda)$ such that*

$$\Pr[\mathsf{Exp}^{\mathsf{double}}_{\mathcal{A}}(1^\lambda, 0) - \mathsf{Exp}^{\mathsf{double}}_{\mathcal{A}}(1^\lambda, 1)] \leq \nu(\lambda)$$

where the experiment $\mathsf{Exp}^{\mathsf{double}}_{\mathcal{A}}$ is defined as follows.

Experiment $\mathsf{Exp}^{\mathsf{double}}_{\mathcal{A}}(1^\lambda, b)$

1. *The adversary $\mathcal{A}$ on input 1^λ outputs two keys k_a and k_b of length λ and two triples of messages (x_0, y_0, z_0) and (x_1, y_1, z_1) where all messages are of the same length.*
2. *Two keys $k'_a, k'_b \xleftarrow{\$} G(1^\lambda)$ are chosen.*
3. *$\mathcal{A}^{E_{k'_a}(\cdot), E_{k'_b}(\cdot)}$ is given the challenge ciphertexts $c_x \leftarrow E_{k_a}(E_{k'_b}(x_b))$, $c_y \leftarrow E_{k'_a}(E_{k_b}(y_b))$, $c_z \leftarrow E_{k'_a}(E_{k'_b}(z_b))$ as well as* **oracle access** *to $E_{k'_a}(\cdot)$ and $E_{k'_b}(\cdot)$.*
4. *$\mathcal{A}$ outputs b' which is the output of the experiment.*

The following lemma is essentially immediate - see [LP09] for a formal proof.

Lemma 4. *If (G, E, D) is CPA-secure then it is secure under chosen double encryption.*

B Constructing Somewhere Equivocal Encryption

In this section, we prove Theorem 1 and show how to construct a somewhere equivocal symmetric-key encryption scheme from OWFs. We do this by introducing two intermediate and successively simpler primitives. First, in Section B.1 we introduce the notion of *t-Point Somewhere Equivocal PRF (t-SEPRF, for short)*, and we show that it implies somewhere equivocal symmetric-key encryption scheme with equivocation parameter t. We also show that 1-SEPRF implies t-SEPRF. Then, in Section B.2 we define an even simpler primitive called a *Two-Key Equivocal PRF (TEPRF, for short)* and show how to construct 1-SEPRFs from TEPRF. We then show how to construct TEPRF from one-way functions, following the approach of [BGI15] previously used to construct distributed point functions. Finally, in Section B.3 we finish the proof of Theorem 1 and summarize the parameters of the construction.

B.1 t-Point Somewhere Equivocal PRF (SEPRF)

B.1.1 Definition

We introduce the notion of a t-point Somewhere Equivocal PRF (t-SEPRF). Intuitively, in addition to the usual way of sampling PRF keys, there is a simulated method that creates a PRF key which is equivocal on a set of points X of size $|X| \leq t$: for any assignment of values $x_i \to r_i$ for $x_i \in X$ we can later create a key key' such that $\mathsf{PRF}(\mathsf{key}, x) = \mathsf{PRF}(\mathsf{key}', x)$ for all $x \notin X$ and $\mathsf{PRF}(\mathsf{key}', x_i) = r_i$ for $x_i \in X$. The security definition is analogous to somewhere equivocal encryption. We want to make sure that even if an adversary knows that the values X with $|X| < t$ are being equivocated to some arbitrary outputs, he cannot tell whether an additional point x is also being equivocated to a random output r or not.

Experiment $\mathsf{Exp}^{\mathsf{equivprf2}}_{\mathcal{A},\mathsf{SEPRF}}(1^\lambda)$

1. The adversary $\mathcal{A}$ on input 1^λ outputs a set $X \subseteq \{0,1\}^d$ s.t. $|X| < t$, and a challenge $j \in \{0,1\}^d \setminus X$.
2. A uniform bit $b \leftarrow \{0,1\}$ is chosen.
   - If $b = 0$, pick $(\mathsf{key}, \mathsf{state}) \xleftarrow{\$} \mathsf{Sim}_1(X)$; Set $r^*_j = \mathsf{PRF}(\mathsf{key}, j)$.
   - If $b = 1$, pick $r^*_j \xleftarrow{\$} \{0,1\}^s$; $(\mathsf{key}, \mathsf{state}) \xleftarrow{\$} \mathsf{Sim}_1(\{X \cup j\})$.
3. Send r^*_j
4. The adversary $\mathcal{A}^{\mathsf{PRF_p}(\mathsf{key},\cdot)}$ outputs values $R = \{x \to r^*_x\}_{x \in X}$. Where here, the adversary is given oracle access to the punctured oracle $\mathsf{PRF_p}$. The oracle is defined as
$$\mathsf{PRF_p}(\mathsf{key}, x) = \begin{cases} \mathsf{PRF}(\mathsf{key}, x) & \text{if } x \notin X \cup j \\ \bot & \text{if } x \in X \cup j \end{cases}$$
5. Send key' to the adversary $\mathcal{A}$ where:
   - If $b = 0$, compute key' as follows: $\mathsf{key}' \xleftarrow{\$} \mathsf{Sim}_2(\mathsf{state}, R)$.
   - If $b = 1$, compute key' as follows: $\mathsf{key}' \xleftarrow{\$} \mathsf{Sim}_2(\mathsf{state}, R \cup \{j \to r^*_j\})$.
6. $\mathcal{A}$ outputs b'.
7. Output $b = b'$ and halt.

Figure 7: Experiment $\mathsf{Exp}^{\mathsf{equivprf2}}_{\mathcal{A},\mathsf{SEPRF}}(1^\lambda)$

Definition 6 (t-Point Somewhere Equivocal PRF)**.** *A tuple of polynomial-time algorithms* $(\mathsf{ObvGen}, \mathsf{PRF}, \mathsf{Sim}_1, \mathsf{Sim}_2)$ *is t-point* Somewhere Equivocal PRF (t-SEPRF) *with* input size d *and* output size s *if it satisfies the following:*

Oblivious Key Generation: $\mathsf{key} \leftarrow \mathsf{ObvGen}(1^\lambda)$ *outputs a key such that* $\mathsf{PRF}(\mathsf{key}, \cdot) : \{0,1\}^d \to \{0,1\}^s$.

Equivocal Key Generation: *Two alternate key generation algorithms* $\mathsf{Sim}_1, \mathsf{Sim}_2$ *work as follows*

- *On input $X \subset \{0,1\}^d$ with $|X| \leq t$,* Sim_1 *outputs a key and a state* $(\mathsf{key}, \mathsf{state}) \xleftarrow{\$} \mathsf{Sim}_1(X)$.
- *On input a mapping $R = \{x_i \to r_i\}_{x_i \in X}$ with $r_i \in \{0,1\}^s$:* $\mathsf{key}' \xleftarrow{\$} \mathsf{Sim}_2(\mathsf{state}, \{x_i \to r_i\}_{x_i \in X})$ *outputs a key* key'.

Simulation with no Holes: *We require that the distribution of* key' *sampled via* $(\mathsf{key}, \mathsf{state}) \leftarrow \mathsf{Sim}_1(\emptyset)$, $\mathsf{key}' \leftarrow \mathsf{Sim}_2(\mathsf{state}, \emptyset)$ *is identical to* $\mathsf{key}' \leftarrow \mathsf{ObvGen}(1^\lambda)$.

Correctness: *If* $(\mathsf{key}, \mathsf{state}) \overset{\$}{\leftarrow} \mathsf{Sim}_1(X)$, *and* $\mathsf{key}' \overset{\$}{\leftarrow} \mathsf{Sim}_2(\mathsf{state}, R)$,

$$\mathsf{PRF}(\mathsf{key}, x) = \mathsf{PRF}(\mathsf{key}', x) \quad \textit{for } x \notin X$$
$$\mathsf{PRF}(\mathsf{key}', x_i) = r_i \quad \textit{for } x_i \in X \textit{ and } (x_i \to r_i) \in R$$

Equivocation Security. *For any PPT adversary $\mathcal{A}$, there exists a negligible function $\nu = \nu(\lambda)$ such that:*

$$\Pr[\mathsf{Exp}^{\mathsf{equivprf2}}_{\mathcal{A},\Pi}(1^\lambda) = 1] \leq \frac{1}{2} + \nu(\lambda)$$

where the experiment $\mathsf{Exp}^{\mathsf{equivprf2}}_{\mathcal{A},\Pi}$ is defined in Fig. 7.

B.1.2 From SEPRF to Somewhere Equivocal encryption

We now show how to construct somewhere equivocal encryption with equivocation parameter t from a t-SEPRF. The construction essentially uses the PRF outputs as a one-time pad and is shown in Fig. 8.

Somewhere Equivocal Encryption from Somewhere Equivocal PRF.
Let $(\mathsf{Gen}, \mathsf{PRF}, \mathsf{Sim}_1, \mathsf{Sim}_2)$ be a t-point somewhere equivocal PRF as per Definition 6.

- $\mathsf{KeyGen}(1^\lambda)$ runs $\mathsf{key} \overset{\$}{\leftarrow} \mathsf{Gen}(1^\lambda)$ and outputs key.
- $\mathsf{Enc}(\mathsf{key}, \overline{m})$ for i in $[n]$: $c_i = (\mathsf{PRF}(\mathsf{key}, i) \oplus m_i)$. Return ciphertext $\overline{c} = (c_1, \ldots, c_n)$.
- $\mathsf{Dec}(\mathsf{key}, \overline{c})$ for i in $[n]$, $m_i \leftarrow c_i \oplus \mathsf{PRF}(\mathsf{key}, i)$. Return plaintext $\overline{m} = (m_1, \ldots, m_n)$.
- $\mathsf{SimEnc}(I, (m_i)_{i \notin I})$ Run $(\mathsf{key}, \mathsf{state}) \overset{\$}{\leftarrow} \mathsf{Sim}_1(1^\lambda, I)$:

$$c_i \leftarrow (i, \mathsf{PRF}(\mathsf{key}, i)) \qquad \text{for } i \in [n] \setminus I:$$
$$c_i \leftarrow r_i; \text{ with } r_i \overset{\$}{\leftarrow} \{0,1\}^s \qquad \text{for } i \in I$$
$$\overline{c} \leftarrow (c_1, \ldots, c_n)$$

  Return $((\mathsf{state}, I, (r_i)_{i \in I}), \overline{c})$.

- $\mathsf{SimKey}((\mathsf{state}, I, (r_i)_{i\in I}), (m_i)_{i \in I})$:
  $y_i \leftarrow m_i \oplus r_i$ for $i \in I$; $\mathsf{key}' \overset{\$}{\leftarrow} \mathsf{Sim}_2(\mathsf{state}, \{i \to y_i\}_{i \in I})$
  Return key'.

Figure 8: Somewhere equivocal symmetric-key encryption scheme Π.

Claim 1. *Assume that $(\mathsf{Gen}, \mathsf{PRF}, \mathsf{Sim}_1, \mathsf{Sim}_2)$ is a t-SEPRF with input-length d, output length s and key-size k as in Definition 6. Then for any message-length $n \leq 2^d$ the encryption scheme described in Fig. 8 is a somewhere equivocal encryption scheme with equivocation parameter t, block-length s, and key-size k.*

Proof. The correctness and simulation with no holes properties of the encryption scheme follow from those of the SEPRF. Assume that there is an adversary $\mathcal{A}$ for the t-somewhere equivocal security of Construction 8, then we construct and adversary $\mathcal{A}_{\mathsf{seprf}}$ as follows.

- $\mathcal{A}_{\mathsf{seprf}}$ obtains sets I,$(m_i)_{i \notin I}$ and the challenge index j from $\mathcal{A}$. Let $I' \leftarrow I \cup j$. It sets $X = I$ and sends X, j to her challenger.
- $\mathcal{A}_{\mathsf{seprf}}$ receives value r^*_j from the challenger and oracle access to $\mathsf{PRF}_{\mathsf{p}}$. It queries the oracle on the values $i \in [n] \setminus I'$ and let us label the outputs $(v_i)_{i \notin I'}$. It computes the ciphertexts as follow.

  - for all $i \notin I'$, $c_i = v_i \oplus m_i$.
  - set $c_j = r_j^* \oplus m_j$.
  - for all $i \in I$ set $c_i \overset{\$}{\leftarrow} \{0,1\}^s$.

  Send $\bar{c}$ to $\mathcal{A}$.

- $\mathcal{A}$ sends the set of remaining messages $(m_i)_{i \in I}$ to $\mathcal{A}_{\mathsf{seprf}}$.
- $\mathcal{A}_{\mathsf{seprf}}$ sends the set $R = \{i \to r_i^*\}_{i \in I}$, where $r_i^* = c_i \oplus m_i$, to her challenger and obtains key key.
- $\mathcal{A}_{\mathsf{seprf}}$ forwards such key to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs. If $\mathcal{A}$ aborts, output a random bit b.

It's easy to see that the advantage of $\mathcal{A}_{\mathsf{seprf}}$ on the t-SEPRF is the same as the advantage of the adversary $\mathcal{A}$ on the somewhere equivocal encryption. □

B.1.3 From 1-SEPRF to t-SEPRF

In this section, we show how to amplify the equivocation parameter t. Indeed, starting with a 1-SEPRF, we can construct a t-SEPRF for an arbitrary t. We simply XOR together the outputs of t independent 1-SEPRFs. Firstly, we note that the security definition for 1-SEPRF becomes simpler and in particular, the experiment $\mathsf{Exp}^{\mathsf{equivprf2}}_{\mathcal{A},\mathsf{SEPRF}}(1^\lambda)$ becomes equivalent to the following game:

Definition 7 (Simplified 1-SEPRF Security). *We can re-write the security requirement of a 1-SEPRF from Definition 6, which syntactically simplifies to the following. For all PPT adversaries A there is some negligible function ν such that*

$$\left| \Pr\left[\begin{array}{c} x^* \overset{\$}{\leftarrow} A(1^\lambda),\ \mathsf{key} \overset{\$}{\leftarrow} \mathsf{ObvGen}(1^\lambda) \\ A(\mathsf{key}) = 1 \end{array} \right] - \Pr\left[\begin{array}{c} x^* \overset{\$}{\leftarrow} A(1^\lambda)\ ,\ r^* \overset{\$}{\leftarrow} \{0,1\}^s \\ (\mathsf{key}, \mathsf{state}) \overset{\$}{\leftarrow} \mathsf{Sim}_1(x^*) \\ \mathsf{key}' \overset{\$}{\leftarrow} \mathsf{Sim}_2(\mathsf{state}, x^* \to r^*) \\ A(\mathsf{key}') = 1 \end{array} \right] \right| < \nu(\lambda)$$

1-SEPRF is a Selectively-Secure PRF. As an aside (which we do not explicitly rely on in this paper), we note that any scheme which is a 1-SEPRF is also necessarily a *selectively secure* PRF. In a selective secure PRF, the distinguisher chooses a challenge point $x^* \in \{0,1\}^d$ in the domain of the PRF ahead of time and obtains an answer r^* which is either $\mathsf{PRF}(\mathsf{key}, x^*)$ or uniformly random. The distinguisher can then query the function $\mathsf{PRF}(\mathsf{key}, \cdot)$ on arbitrary inputs $x \neq x^*$. The reason that this is implication holds is that the distinguisher for 1-SEPRF can answer the queries of the PRF distinguisher by using its knowledge of key'.

Going from 1 to t. We now show how to go from a 1-SEPRF to a t-SEPRF for an arbitrary t.

Claim 2. *If there exists a 1-SEPRFs with input-size d, outputs-size s and key size k, then for any polynomial t there exist t-SEPRFs with the same input-size d and output-size s having key size $t \cdot k$.*

Proof. Let $(\mathsf{ObvGen}, \mathsf{PRF}, \mathsf{Sim}_1, \mathsf{Sim}_2)$ be a 1-SEPRF. We define $(\mathsf{ObvGen}', \mathsf{PRF}', \mathsf{Sim}_1', \mathsf{Sim}_2')$ by:

- $K \leftarrow \mathsf{ObvGen}'(1^\lambda)$: Define $K = \mathsf{key}_1, \ldots, \mathsf{key}_t$ where $\mathsf{key}_i \leftarrow \mathsf{ObvGen}(1^\lambda)$.
- $\mathsf{PRF}'(K, x) = \bigoplus_{i \in [t]} \mathsf{PRF}(\mathsf{key}_i, x)$.
- $(K, \mathsf{state}) \leftarrow \mathsf{Sim}_1'(X)$: Let $X = \{x_1, \ldots, x_q\}$. For each $i = 1, \ldots, q$, run $(\mathsf{key}_i, \mathsf{state}_i) \overset{\$}{\leftarrow} \mathsf{Sim}_1(x_i)$ and for $i = q+1, \ldots, t$ run $\mathsf{key}_i \leftarrow \mathsf{ObvGen}(1^\lambda)$. Define $K = \mathsf{key}_1, \ldots, \mathsf{key}_t$ and $\mathsf{state} = \mathsf{state}_1, \ldots, \mathsf{state}_q$. (Without loss of generality, we assume state_i contains key_i).
- $K' \leftarrow \mathsf{Sim}_2'(\mathsf{state}, \{x_i \to r_i\}_{x_i \in X})$: For $i = 1, \ldots, q$ compute $\mathsf{key}_i' \leftarrow \mathsf{Sim}_2(\mathsf{state}_i, \{x_i \to v_i\})$ where $v_i = r_i \bigoplus_{j \in [t] \setminus i} \mathsf{PRF}(\mathsf{key}_j, x_i)$. For $i = q+1, \ldots, t$ set $\mathsf{key}_i' = \mathsf{key}_i$. Output $K' = \mathsf{key}_1', \ldots, \mathsf{key}_t'$.

The correctness properties follow immediately. Assume there is and adversary $\mathcal{A}$ for the t-point SEPRF scheme constructed above. Then we construct an adversary B that breaks the security of the underlying 1-SEPRF. The high-level idea of the reduction is the following: an adversary for single-point SEPRF can compute the PRF evaluations for and adversary of t-point SEPRF by running the simulator for all points in I and by using its SEPRF challenger to compute the PRF evaluation for the challenge index j. Formally, adversary B works as follows.

- $\mathcal{A}$ outputs set $C = \{x_1, \ldots, x_q\}$ s.t. $|X| = q < t$ and a challenge index j.
- B gives j to its own challenger and gets back key_{q+1}. Let $r_j^* = \mathsf{PRF}(\mathsf{key}_1, j)$. It chooses $\mathsf{key}_1, \ldots, \mathsf{key}_q$ using $(\mathsf{key}_i, \mathsf{state}_i) \leftarrow \mathsf{Sim}_1(x_i)$ and $\mathsf{key}_{q+1}, \ldots, \mathsf{key}_t$ using $\mathsf{key}_i \leftarrow \mathsf{KeyGen}(1^\lambda)$. It sets $K = (\mathsf{key}_1, \ldots, \mathsf{key}_t)$ and uses K to simulate the PRF oracle for $\mathcal{A}$.
- Eventually $\mathcal{A}$ outputs a mapping $R = \{x_i \to r_i\}_{x_i \in X}$. B does the following. For $i = 1, \ldots, q$ it computes $\mathsf{key}'_i \leftarrow \mathsf{Sim}_2(\mathsf{state}_i, \{x_i \to v_i\})$ where $v_i = r_i \bigoplus_{j \in [t] \setminus i} \mathsf{PRF}(\mathsf{key}_j, x_i)$, and for $i = q+1, \ldots, t$ it sets $\mathsf{key}'_i = \mathsf{key}_i$. It gives $K = \mathsf{key}'_1, \ldots, \mathsf{key}'_t$ to $\mathcal{A}$.
- B outputs what $\mathcal{A}$ outputs.

It's easy to see that B breaks 1-SEPRF security with the same advantage that $\mathcal{A}$ break t-SEPRF security. □

B.2 Two-Key Equivocal PRFs

We now introduce a notion which is weaker than 1-SEPRF and which we call a *Two-Key* Equivocal PRF (TEPRF). Unlike an SEPRF, in a TEPRF, the simulator does not choose the value at the output. Instead, it can just create two keys key and key' which differ on a selected point x. We will consider TEPRFs that only have a single-bit output. Therefore, this essentially allows the simulator to open the value on this point to both bits.

B.2.1 Definition

Definition 8 (Two-Key Equivocal PRFs)**.** *A triple of polynomial-time algorithms* $(\mathsf{ObvGen}, \mathsf{Gen}, \mathsf{PRF})$ *is a Two-Key Equivocal PRFs (TEPRF) with input-size* d *and output size* 1 *if*

Oblivious Key Generation: ObvGen *outputs a key* $\mathsf{key} \xleftarrow{\$} \mathsf{ObvGen}(1^\lambda)$ *such that* $\mathsf{PRF}(\mathsf{key}, \cdot) : \{0,1\}^d \to \{0,1\}$.

Equivocal Key Generation: *On inputs* $x^* \in \{0,1\}^d$, Gen *outputs two keys* $(\mathsf{key}, \mathsf{key}') \xleftarrow{\$} \mathsf{Gen}(x^*)$

Equality *: If* $(\mathsf{key}, \mathsf{key}') \xleftarrow{\$} \mathsf{Gen}(x^*)$*, then*

$$\mathsf{PRF}(\mathsf{key}, x) = \mathsf{PRF}(\mathsf{key}', x) \text{ for all } x \neq x^*$$

Different Values on Target Path: *If* $(\mathsf{key}, \mathsf{key}') \xleftarrow{\$} \mathsf{Gen}(x^*)$*, then*

$$\mathsf{PRF}(\mathsf{key}, x^*) \neq \mathsf{PRF}(\mathsf{key}', x^*)$$

Indistinguishability: *For all PPT adversaries A there is some negligible function ν such that*

$$\left| \Pr\left[\begin{array}{c} (x^*) \xleftarrow{\$} A(1^\lambda) \\ \mathsf{key} \xleftarrow{\$} \mathsf{ObvGen}(1^\lambda) \\ A(\mathsf{key}) = 1 \end{array} \right] - \Pr\left[\begin{array}{c} (x^*) \xleftarrow{\$} A(1^\lambda) \\ (\mathsf{key}, \mathsf{key}') \xleftarrow{\$} \mathsf{Gen}(x^*) \\ A(\mathsf{key}) = 1 \end{array} \right] \right| < \nu(\lambda)$$

and

$$\left| \Pr\left[\begin{array}{c} (x^*) \xleftarrow{\$} A(1^\lambda) \\ \mathsf{key} \xleftarrow{\$} \mathsf{ObvGen}(1^\lambda) \\ A(\mathsf{key}) = 1 \end{array} \right] - \Pr\left[\begin{array}{c} (x^*) \xleftarrow{\$} A(1^\lambda) \\ (\mathsf{key}, \mathsf{key}') \xleftarrow{\$} \mathsf{Gen}(x^*) \\ A(\mathsf{key}') = 1 \end{array} \right] \right| < \nu(\lambda).$$

B.2.2 From TEPRFs to 1-SEPRFs

In this section, we show how to construct a 1-SEPRF from a TEPRF. Recall that a TEPRF has 1-bit output while a 1-SEPRF can have an arbitrary output length s. To build an SEPRF we simply concatenate s independent copies of a TERPF.

Claim 3 (TEPRFs to 1-SEPRFs)**.** *Given any TEPRF with input-length d (output-length 1) and key-length k, for any polynomial s there is a 1-SEPRF with input-length d, output length s and key length $s \cdot k$.*

Proof. Let $(\mathsf{ObvGen}, \mathsf{Gen}, \mathsf{PRF})$ be a TEPRF as in the claim. We construct a 1-SEPRF $(\mathsf{ObvGen}, \mathsf{PRF}, \mathsf{Sim}_1, \mathsf{Sim}_2)$ defined formally in Algorithms 1 - 3 below.

Algorithm 1 SEPRF Key Generation

```
function $\mathsf{ObvGen}'(1^\lambda)$
    for $i = 1, \ldots, s$ do
        $\mathsf{key}_i \xleftarrow{\$} \mathsf{ObvGen}(1^\lambda)$
    end for
    return $K = (\mathsf{key}_1, \ldots, \mathsf{key}_s)$
end function
```

Algorithm 2 SEPRF Evaluation

```
function $\mathsf{PRF}'(K, x^*)$
    for $i = 1, \ldots, s$ do
        $y_i \leftarrow \mathsf{PRF}(\mathsf{key}_i, x^*)$
    end for
    return $y = (y_1, \ldots, y_s)$
end function
```

Algorithm 3 SEPRF Simulator

```
function $\mathsf{Sim}_1(x^*)$
    for $i = 1, \ldots, s$ do
        $(\mathsf{key}_i^0, \mathsf{key}_i^1) \xleftarrow{\$} \mathsf{Gen}(x^*)$
    end for
    $\mathsf{state} = (\mathsf{key}_1^0, \mathsf{key}_1^1, \ldots, \mathsf{key}_s^0, \mathsf{key}_s^1)$, $K = (\mathsf{key}_1^0, \ldots, \mathsf{key}_s^0)$.
    return $\mathsf{state}, K$
end function
function $\mathsf{Sim}_2(\mathsf{state}, \{x^* \to r^*\})$
    Parse $(\mathsf{key}_1^0, \mathsf{key}_1^1, \ldots, \mathsf{key}_s^0, \mathsf{key}_s^1) \leftarrow \mathsf{state}$
    for $i = 1, \ldots, s$ do
        If $\mathsf{PRF}(\mathsf{key}_i^0, x^*) = r_i^*$ then $\mathsf{key}_i' = \mathsf{key}_i^0$ else $\mathsf{key}_i' = \mathsf{key}_i^1$.
    end for
    return $\mathsf{key}' = (\mathsf{key}_1', \ldots, \mathsf{key}_s')$
end function
```

(Technically, we also define $\mathsf{Sim}_1(\emptyset)$ to simply run $(\mathsf{state} = K, K) \leftarrow \mathsf{ObvGen}'(1^\lambda)$ and $K \leftarrow \mathsf{Sim}_2(\mathsf{state}, \emptyset)$ to outputs $K = \mathsf{state}$.)

The fact that the resulting construction satisfies the properties of an SEPRF follows immediately from the properties of the underlying TEPRF. Security follows from a simple hybrid argument. □

B.2.3 Construction of TEPRF

Finally, we give a general construction of a TEPRF based on a pseudo-random generator G. Since PRGs can be constructed from any one-way function [HILL99], this gives a black-box construction of TEPRFs from any one-way function. Our construction essentially follows the construction of distributed point functions from [BGI15].

Claim 4 (TEPRF)**.** *Assuming the existence of a length-doubling PRG with seed-length λ, the algorithms* $(\mathsf{ObvGen}, \mathsf{Gen}, \mathsf{PRF})$ *defined in Algorithms 6,7 and 8 below form a TEPRF, with input-length d and key size $O(d\lambda)$.*

The above claim is proved in below. The target value property is Claim 8. The indistinguishability is Claim 9. The equality follows from Claims 6 and 7.

The Construction. The construction is a tree-based construction similar to the GGM construction of a PRF from a PRG [GGM86]. To make the similarity clearer, we recall the GGM construction (Algorithm 4).

Algorithm 4 The GGM construction of a PRF from a PRG

```
function $F(s, \vec{x})$                                   ▷ $G : \{0,1\}^\lambda \to \{0,1\}^\lambda \times \{0,1\}^\lambda$
    if $\vec{x} = \emptyset$ then
        return $s$
    else
        Parse $x_1 \cdots x_\ell \xleftarrow{\$} \vec{x}$
        Parse $s_{\text{parent}} \leftarrow F(s, (x_1, \ldots, x_{\ell_1}))$
        Parse $s_0, s_1 \leftarrow G(s_{\text{parent}})$
        return $s_{x_\ell}$
    end if
end function
```

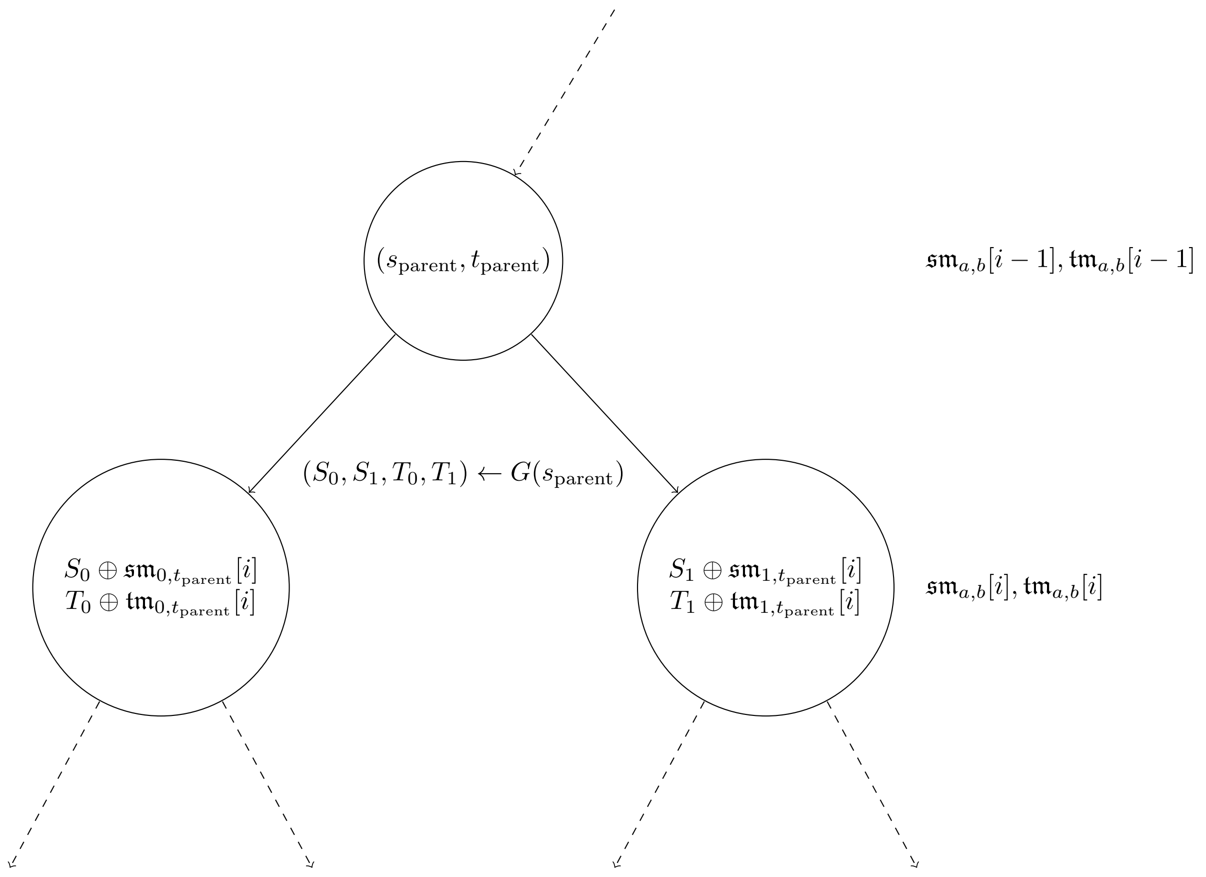
Our construction of a TEPRF will rely on a more complex PRG than the basic GGM construction. We will require: $G : \{0,1\}^\lambda \to \{0,1\}^\lambda \times \{0,1\}^\lambda \times \{0,1\} \times \{0,1\}$. In the GGM construction, each internal leaf can be viewed as holding a seed in $\{0,1\}^\lambda$ that is then expanded using the PRG to obtain the values for the leaf's children. Our tree will have two values on each internal leaf, a seed in $\{0,1\}^\lambda$ and a tag in $\{0,1\}$. The key for our TEPRF will include an initial seed for the G, as well as layer masks for each level of the tree. At level i in the tree there will be two types of masks "seed masks" $\mathfrak{sm}_{a,b}[i]$ and "tag masks" $\mathfrak{tm}_{a,b}[i]$, and there will be four masks of each type. Which mask is applied will depend on which child is being evaluated (left or right) and the value of the tag at the current leaf. To see the similarity with the GGM construction contrast Algorithms 4 and 5.

To equivocate, we imagine two trees, corresponding to keys key and key'. We would like the property that along the target path the two TEPRFs evaluate to different values, while along any other path, they are the same. To ensure this property, the masks will be carefully chosen to maintain two properties: (1) Along the target path, the tags corresponding to key and key' are always different (2) At the first deviation from the target path, the values on the internal nodes of the two trees become the same. Because the masks are the same for the two keys, once the internal values become the same, they will never deviate again. Property (1) is proven in Claim 5, and property (2) is proven in Claim 6.

The full specification of the Key Generation algorithm is given in Algorithm 6, while the evaluation algorithm is given in Algorithm 8 (which relies on Algorithm 5).

Claim 5. *Along the path given by $\vec{x}^*$ the second output of F on* key *and* key' *is different. More formally, for any prefix $\vec{y}$ of $\vec{x}^*$, if $(s_{parent}, t_{parent}) \leftarrow F(\mathsf{key}, \vec{y})$, and $(s'_{parent}, t'_{parent}) \leftarrow F(\mathsf{key}', \vec{y})$, then $t_{parent} \neq t'_{parent}$.*

Proof. We aim to show that $t_{\text{parent}} \neq t'_{\text{parent}}$. If $\ell = 1$, then this follows immediately because $t_{\text{parent}} = t$, and $t'_{\text{parent}} = t'$, and $t \neq t'$ (line 4 in Algorithm 6). Now, we proceed via induction. Assume that if $(s_{\text{parent}}, t_{\text{parent}}) \leftarrow F(\mathsf{key}, (x_1^*, \ldots, x_\ell^*))$, and $(s'_{\text{parent}}, t'_{\text{parent}}) \leftarrow F(\mathsf{key}', (x_1^*, \ldots, x_\ell^*))$, then $t_{\text{parent}} \neq t^*_{\text{parent}}$. Now, we need to If $\ell > 1$, then we proceed via induction. Let $(A, B) \leftarrow F(\mathsf{key}, (x_1^*, \ldots, x_{\ell+1}^*))$, and $(A', B') \leftarrow F(\mathsf{key}', (x_1^*, \ldots, x_{\ell+1}^*))$, We need to show that $B = B'$. From Algorithm 5, we have $(S_0, S_1, T_0, T_1) \leftarrow G(s_{\text{parent}})$, and $(S'_0, S'_1, T'_0, T'_1) \leftarrow G(s'_{\text{parent}})$, Then $B = T_{x_{\ell+1}} \oplus \mathfrak{tm}_{x^*_{\ell+1}, t_{\text{parent}}}$, and $B' = T'_{x_{\ell+1}} \oplus \mathfrak{tm}_{x^*_{\ell+1}, t'_{\text{parent}}}$, so by the definition of $\mathfrak{tm}_{x^*_{\ell+1},0}$ and $\mathfrak{tm}_{x^*_{\ell+1},1}$ (Line 12 in Algorithm 6) we conclude $B = B'$. □



$(s_{\text{parent}}, t_{\text{parent}})$
$\mathfrak{sm}_{a,b}[i-1], \mathfrak{tm}_{a,b}[i-1]$
$(S_0, S_1, T_0, T_1) \leftarrow G(s_{\text{parent}})$
$S_0 \oplus \mathfrak{sm}_{0,t_{\text{parent}}}[i]$
$T_0 \oplus \mathfrak{tm}_{0,t_{\text{parent}}}[i]$
$S_1 \oplus \mathfrak{sm}_{1,t_{\text{parent}}}[i]$
$T_1 \oplus \mathfrak{tm}_{1,t_{\text{parent}}}[i]$
$\mathfrak{sm}_{a,b}[i], \mathfrak{tm}_{a,b}[i]$

Figure 9: One step in the SEPRF tree

Algorithm 5 A Recursive Helper Function for the TEPRF

```
1: function $F(s, t, \vec{\mathfrak{sm}}_{a,b}, \vec{\mathfrak{tm}}_{a,b}, \vec{x})$
2:     ▷ $s \in \{0,1\}^\lambda$, $t \in \{0,1\}$
3:     ▷ $\mathfrak{sm}_{a,b}[i] \in \{0,1\}^\lambda$ for $i = 1, \ldots, \ell$ and $a, b \in \{0,1\}$
4:     ▷ $\mathfrak{tm}_{a,b}[i] \in \{0,1\}$ for $i = 1, \ldots, \ell$ and $a, b \in \{0,1\}$
5:     ▷ Length$(\vec{\mathfrak{sm}}_{a,b})$ = Length$(\vec{\mathfrak{tm}}_{a,b})$ = Length$(\vec{x})$
6:     if $\vec{x} = \emptyset$ then
7:         return $s, t$
8:     else
9:         Parse $x_1 \cdots x_\ell \leftarrow \vec{x}$
10:        Parse $\mathfrak{sm}_{a,b}[1] \cdots \mathfrak{sm}_{a,b}[\ell] \leftarrow \vec{\mathfrak{sm}}_{a,b}$ for $a, b \in \{0,1\}$
11:        Parse $\mathfrak{tm}_{a,b}[1] \cdots \mathfrak{tm}_{a,b}[\ell] \leftarrow \vec{\mathfrak{tm}}_{a,b}$ for $a, b \in \{0,1\}$
12:        $(s_{\text{parent}}, t_{\text{parent}}) \leftarrow F(s, t, (\mathfrak{sm}_{a,b}[1], \ldots, \mathfrak{sm}_{a,b}[\ell-1]), (\mathfrak{tm}_{a,b}[1], \ldots, \mathfrak{tm}_{a,b}[\ell-1]), (x_1, \ldots, x_{\ell-1}))$
13:        $(s_0, s_1, t_0, t_1) \leftarrow G(s_{\text{parent}})$
14:        $s \leftarrow s_{x_\ell} \oplus \mathfrak{sm}_{x_\ell, t_{\text{parent}}}[\ell]$
15:        $t \leftarrow t_{x_\ell} \oplus \mathfrak{tm}_{x_\ell, t_{\text{parent}}}[\ell]$
16:        return $s, t$
17:    end if
18: end function
```

Algorithm 6 Generating a pair of equivocable keys

```
1:  function Gen($x^*$)
2:      ▷ Target path $x^* \in \{0,1\}^d$
3:      Choose two random seeds, $s, s' \xleftarrow{\$} \{0,1\}^\lambda$
4:      Choose a random tag, $t \xleftarrow{\$} \{0,1\}$, set $t' = \neg t$
5:      for $i = 1, \ldots, d$ do
6:          $(s_{\text{parent}}, t_{\text{parent}}) \leftarrow F(s, t, (\mathfrak{sm}_{a,b}[1], \ldots, \mathfrak{sm}_{a,b}[i-1]), (\mathfrak{tm}_{a,b}[1], \ldots, \mathfrak{tm}_{a,b}[i-1]), (x^*_1, \ldots, x^*_{i-1}))$
7:          $(s'_{\text{parent}}, t'_{\text{parent}}) \leftarrow F(s', t', (\mathfrak{sm}_{a,b}[1], \ldots, \mathfrak{sm}_{a,b}[i-1]), (\mathfrak{tm}_{a,b}[1], \ldots, \mathfrak{tm}_{a,b}[i-1]), (x^*_1, \ldots, x^*_{i-1}))$
8:          $(S_0, S_1, T_0, T_1) = G(s_{\text{parent}})$
9:          $(S'_0, S'_1, T'_0, T'_1) = G(s'_{\text{parent}})$
10:         Choose two uniformly random seed masks $\mathfrak{sm}_{x^*_i,0}[i], \mathfrak{sm}_{x^*_i,1}[i] \xleftarrow{\$} \{0,1\}^\lambda$
11:         Choose two random seed masks subject to $\mathfrak{sm}_{\neg x^*_i,0}[i] \oplus S_{\neg x^*_i} = \mathfrak{sm}_{\neg x^*_i,1}[i] \oplus S'_{\neg x^*_i}$
12:         Choose two random tag masks subject to $\mathfrak{tm}_{x^*_i,0}[i] \oplus T_{\neg x^*_i} \neq \mathfrak{tm}_{x^*_i,1}[i] \oplus T'_{\neg x^*_i}$
13:         Choose two random tag masks subject to $\mathfrak{tm}_{\neg x^*_i,0}[i] \oplus T_{\neg x^*_i} = \mathfrak{tm}_{\neg x^*_i,1}[i] \oplus T'_{\neg x^*_i}$
14:     end for
15:     $(s'_{\text{final}}, t'_{\text{final}}) \leftarrow F(s', t', \vec{\mathfrak{sm}}_{a,b}, \vec{\mathfrak{tm}}_{a,b}, \vec{x}^*)$
16:     Let $y_{\text{final}}$ be the first bit of $s_{\text{final}}$
17:     Let $y'_{\text{final}}$ be the first bit of $s'_{\text{final}}$
18:     if $y'_{\text{final}} = y_{\text{final}}$ then go to 3
19:     end if
20:     $\mathsf{key} = (s, t, \vec{\mathfrak{sm}}_{a,b}, \vec{\mathfrak{tm}}_{a,b})$
21:     $\mathsf{key}' = (s', t', \vec{\mathfrak{sm}}_{a,b}, \vec{\mathfrak{tm}}_{a,b})$
22:     return $\mathsf{key}, \mathsf{key}'$
23: end function
```

Algorithm 7 Generating a single "honest" key

```
function ObvGen($1^\lambda$)
    Choose random seed, $s \xleftarrow{\$} \{0,1\}^\lambda$
    Choose a random tag, $t \xleftarrow{\$} \{0,1\}$
    for $i = 1, \ldots, d$ do
        for $a \in \{0,1\}$ do
            for $b \in \{0,1\}$ do
                $\mathfrak{sm}_{a,b}[i] \xleftarrow{\$} \{0,1\}^\lambda$
                $\mathfrak{tm}_{a,b}[i] \xleftarrow{\$} \{0,1\}$
            end for
        end for
    end for
    $\mathsf{key} = (s, t, \vec{\mathfrak{sm}}_{a,b}, \vec{\mathfrak{tm}}_{a,b})$
    return $\mathsf{key}$
end function
```

Algorithm 8 A TEPRF

```
1: function PRF($\mathsf{key}, \vec{x}$)
2:     Parse $\mathsf{key}$ as $(s, t, \vec{\mathfrak{sm}}_{a,b}, \vec{\mathfrak{tm}}_{a,b}) \leftarrow \mathsf{key}$
3:     $(s, t) \leftarrow F(s, t, \vec{\mathfrak{sm}}_{a,b}, \vec{\mathfrak{tm}}_{a,b}, \vec{x})$
4:     Parse $s$ as $(s_1, \ldots, s_\lambda) \leftarrow s$
5:     Set $y = s_1$
6: end function
```

Claim 6. *Suppose* $(\mathsf{key}, \mathsf{key}') \xleftarrow{\$} \mathsf{Gen}(\vec{x}^*)$*, and suppose* $\vec{x} \in \{0,1\}^\ell$ *satisfies* $(x_1, \ldots, x_{\ell-1}) = (x_1^*, \ldots, x_{\ell-1}^*)$ *and* $x_\ell = \neg x_{\ell^*}$*. Then* $F(\mathsf{key}, \vec{x}) = F(\mathsf{key}', \vec{x})$*.*

Proof. Let $\vec{y} = (x_1, \ldots, x_{\ell-1})$. Thus, by hypothesis, $\vec{y} = (x_1^*, \ldots, x_{\ell-1}^*)$ as well. Let $(S,T) = F(\mathsf{key}, \vec{x})$, and $(S',T') = F(\mathsf{key}', \vec{x})$. Thus our goal is to show that $(S,T) = (S',T')$. When $i = \ell$ then $(s_{\text{parent}}, t_{\text{parent}}) = F(s, t, \vec{\mathfrak{sm}}_{a,b}, \vec{\mathfrak{tm}}_{a,b}, \vec{y})$ in Line 6 of Algorithm 6. Similarly, $(s'_{\text{parent}}, t'_{\text{parent}}) = F(s', t', \vec{\mathfrak{sm}}_{a,b}, \vec{\mathfrak{tm}}_{a,b}, \vec{y})$ in Line 7 of Algorithm 6.

Now, from lines 13 - 15 in Algorithm 5, we have that $(s_0, s_1, t_0, t_1) \leftarrow G(s_{\text{parent}})$, and

$$\begin{aligned} S &= s_{x_\ell} \oplus \mathfrak{sm}_{x_\ell, t_{\text{parent}}}[\ell] \\ T &= t_{x_\ell} \oplus \mathfrak{tm}_{x_\ell, t_{\text{parent}}}[\ell] \end{aligned}$$

Similarly $(s'_0, s'_1, t'_0, t'_1) \leftarrow G(s'_{\text{parent}})$, and

$$\begin{aligned} S' &= s'_{x_\ell} \oplus \mathfrak{sm}_{x_\ell, t'_{\text{parent}}}[\ell] \\ T' &= t'_{x_\ell} \oplus \mathfrak{tm}_{x_\ell, t'_{\text{parent}}}[\ell] \end{aligned}$$

Now, since $x_\ell = \neg x_\ell^*$, and $t_{\text{parent}} \neq t'_{\text{parent}}$ (Claim 5), Line 11 in Algorithm 6 ensures that

$$S' = s'_{x_\ell} \oplus \mathfrak{sm}_{x_\ell, t'_{\text{parent}}}[\ell] = s_{x_\ell} \oplus \mathfrak{sm}_{x_\ell, t_{\text{parent}}}[\ell] = S$$

Similarly, line 13 ensures that

$$T = t_{x_\ell} \oplus \mathfrak{tm}_{x_\ell, t_{\text{parent}}}[\ell] = t'_{x_\ell} \oplus \mathfrak{tm}_{x_\ell, t'_{\text{parent}}}[\ell] = T'$$

□

Claim 7. *If* $\vec{y}$ *is a prefix of* $\vec{x}$*, and* $F(\mathsf{key}, \vec{y}) = F(\mathsf{key}', \vec{y})$*, then* $F(\mathsf{key}, \vec{x}) = F(\mathsf{key}', \vec{x})$*.*

Proof. This follows immediately from the fact that key and key' only differ in their initial values s, t and s', t', and the function F is recursive (line 12 of Algorithm 5). □

Claim 8. *Suppose* $(\mathsf{key}, \mathsf{key}') \xleftarrow{\$} \mathsf{Gen}(x^*)$*, Then* $\mathsf{PRF}(\mathsf{key}, x^*) \neq \mathsf{PRF}(\mathsf{key}', x^*)$*.*

Proof. This follows immediately from Line 18 in Algorithm 6. □

Claim 9. *The distribution of the first outputs of* $\mathsf{ObvGen}(1^\lambda)$ *and* $\mathsf{Gen}(x^*)$ *are computationally indistinguishable.*

Proof. If $\mathsf{key} \xleftarrow{\$} \mathsf{ObvGen}(1^\lambda)$, and $\mathsf{key} = (s, t, \vec{\mathfrak{sm}}_{a,b}, \vec{\mathfrak{tm}}_{a,b})$, then every component of key is uniformly random. Thus it suffices to show that if If $(\mathsf{key}, \mathsf{key}') \xleftarrow{\$} \mathsf{ObvGen}(x^*)$, and $\mathsf{key} = (s, t, \vec{\mathfrak{sm}}_{a,b}, \vec{\mathfrak{tm}}_{a,b})$, then the components are pseudorandom conditioned on x^*. But this follows immediately from Algorithm 6. From lines 3 and 4, we see that s and t are uniformly random. From lines 10 and 11, we see that $\mathfrak{sm}_{a,b}[i]$ are pseudorandom. From lines 12 and 13, we see that $\mathfrak{tm}_{a,b}[i]$ are pseudorandom. □

B.3 Theorem 1: Parameters of Somewhere Equivocal Encryption

Combining the above constructions of TEPRF from PRG (Claim 4), 1-SEPRF from TEPRF (Claim 3), t-SEPRF from 1-SEPRF (Claim 2) and finally somewhere equivocal encryption from t-SEPRF (Claim 1) we get the proof of Theorem 1. Table 1 summarizes the key sizes of the resulting primitives, assuming the existence of a length-doubling PRG with λ-bit seed, which follows from One-Way Functions.

Scheme	Key Size	Reference
TEPRF	$O(d \cdot \lambda)$	Claim 4
Single-point SEPRF	$O(d \cdot s \cdot \lambda)$	Claim 3
t-point SEPRF	$O(d \cdot t \cdot s \cdot \lambda)$	Claim 2
Somewhere Equivocal Encryption	$O(t \cdot s \cdot \lambda \cdot \log n)$	Claim 1

Table 1: Key sizes in our generic constructions. Here d denotes the input-length of the PRF constructions, λ denotes the security parameter, s denotes the output size of the PRF, and n denotes the message-length (in blocks) of the encryption scheme.

C Non-adaptive Somewhere-Equivocal Encryption

For sake of completeness, in this section we formally describe a simple non-adaptive security definition of somewhere equivocal encryption, as mentioned informally in Section 4. At first sight, it may not be obvious that this definition is implied by the more complex definition of adaptive security (Definition 2) and therefore we give a formal proof of this fact. We do not know of any simpler constructions that would achieve non-adaptive security but would not achieve adaptive security.

Definition 9 (Non-adaptive somewhere-equivocal encryption)**.** *A non-adaptive somewhere equivocal encryption scheme, is defined as in Definition 2 except that the security property is defined as follows.*

Non-adaptive Security. *For any PPT adversary $\mathcal{A}$, there exists a negligible function $\nu = \nu(\lambda)$ such that:*

$$\Pr[\mathsf{Exp}^{\mathsf{simenc\text{-}NA}}_{\mathcal{A},\Pi}(1^\lambda, 0)] - \Pr[\mathsf{Exp}^{\mathsf{simenc\text{-}NA}}_{\mathcal{A},\Pi}(1^\lambda, 1)] \leq \nu(\lambda)$$

where the experiment $\mathsf{Exp}^{\mathsf{simenc\text{-}NA}}_{\mathcal{A},\Pi}$ is defined as follows:

Experiment $\mathsf{Exp}^{\mathsf{simenc\text{-}NA}}_{\mathcal{A},\Pi}(1^\lambda, b)$

1. *The adversary $\mathcal{A}$ on input 1^λ outputs a vector $\overline{m} = m_1, \ldots, m_n$ with $|m_i| = s$ and a set I of size t.*
2. – *If $b = 0$, compute $\mathsf{key}, \overline{c}$ as follows: $\mathsf{key} \leftarrow \mathsf{KeyGen}(1^\lambda)$, $\overline{c} \leftarrow \mathsf{Enc}(\mathsf{key}, \overline{m})$.*
   – *If $b = 1$, compute $\mathsf{key}, \overline{c}$ as follows: $(\mathsf{state}, \overline{c}) \leftarrow \mathsf{SimEnc}((m_i)_{i \notin I}, I)$, $\mathsf{key} \leftarrow \mathsf{SimKey}(\mathsf{state}, (m_i)_{i \in I})$.*
3. *Send $\mathsf{key}, \overline{c}$ to the adversary $\mathcal{A}$.*
4. *$\mathcal{A}$ outputs b'.*
5. *Output $b = b'$ and halt.*

Proposition 1. *If Π is an adaptive-secure somewhere-equivocal encryption scheme according to Definition 2, then Π is also secure according to the non-adaptive security property of Definition 9.*

Proof. Towards a contradiction, assume that the exists a scheme Π that is secure according to Definition 2 but it does not satisfy Definition 9. This means that there exists an PPT adversary $\mathcal{A}$ that wins experiment $\mathsf{Exp}^{\mathsf{simenc\text{-}NA}}_{\mathcal{A},\Pi}(1^\lambda, b)$ with non-negligible probability. $\mathcal{A}$ is able to distinguish the case where $(\overline{c}, \mathsf{key})$ are computed with no holes ($I = \emptyset$) from the case where $(\overline{c}, \mathsf{key})$ are computed with t holes. We claim that there must be an index $j \in I$ for which $\mathcal{A}$ distinguishes if position j has a hole, and we can use such adversary to win the security experiment $\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{A},\Pi}(1^\lambda, b)$ with challenge index j. The proof goes by hybrids arguments. We define hybrid experiment H_k as follows.

Hybrid Experiment $H_k(I, (m)_{i \in I})$**.**
Let I_k be the set containing the first k indexes in I.

- $(\bar{c}, \mathsf{state}) \leftarrow \mathsf{SimEnc}(m_{i \notin I_k}, I_k)$.
- Receive $(m_i)_{i \in I_k}$.
- $\mathsf{key} \leftarrow \mathsf{SimKey}((m_i)_{i \in I_k}, \mathsf{state})$.
- Output $\bar{c}, \mathsf{key}$.

Note that $H_0 = \mathsf{Exp}^{\mathsf{simenc\text{-}NA}}_{\mathcal{A},\Pi}(1^\lambda, 0)$, this is because in H_0, $k = 0$ and $I = \emptyset$ (here we are implicitly using the "simulation with no holes property"); similarly, $H_t = \mathsf{Exp}^{\mathsf{simenc\text{-}NA}}_{\mathcal{A},\Pi}(1^\lambda, 1)$, because $k = t$ and hence $I_t = I$.

We want to argue that, if there exists $\mathcal{A}$ distinguishing H_0 from H_t then there must exist and index k such that $\mathcal{A}$ distinguishes H_k from H_{k+1}. Such $\mathcal{A}$ can be used by an adversary B playing in $\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{A},\Pi}(1^\lambda, b)$ as follows.

Reduction B**.**

- Obtain $I, m_1, \ldots, m_n$ from $\mathcal{A}$.
- Let $I_k \subset I$ be the set containing the first k positions of I. Let $j \in I$ be the $(k+1)^{th}$ position in I.
- B sends I_k, $(m_i)_{i \notin I_k}$ and challenge index j to her challenger, and obtains $\bar{c}$.
- B sends $(m_i)_{i \in I_k}$ and obtains key from her challenger.
- B sends $\bar{c}, \mathsf{key}$ to $\mathcal{A}$.
- B outputs whatever $\mathcal{A}$ outputs, and halts.

Note that if B is playing in $\mathsf{Exp}^{\mathsf{simenc}}_{B,\Pi}(1^\lambda, 0)$ then the distribution generated by B is identical to experiment H_k, otherwise, if B is playing in $\mathsf{Exp}^{\mathsf{simenc}}_{B,\Pi}(1^\lambda, 1)$ then the distribution generated by B is identical to experiment H_{k+1}. Thus the existence of $\mathcal{A}$ would contradict the adaptive security of Π. $\square$