Static vs. Adaptive Security in Perfect MPC: A Separation and the Adaptive Security of BGW*

Gilad Asharov† Ran Cohen‡ Oren Shochat§

October 29, 2023

Abstract

Adaptive security is a highly desirable property in the design of secure protocols. It tolerates adversaries that corrupt parties as the protocol proceeds, as opposed to static security where the adversary corrupts the parties at the onset of the execution. The well-accepted folklore is that static and adaptive securities are *equivalent* for perfectly secure protocols. Indeed, this folklore is backed up by the transformation of Canetti et al. (EUROCRYPT'01), showing that any perfectly secure protocol that is statically secure and satisfies some basic requirements is also adaptively secure. Yet, this transformation results in an adaptively secure protocol with *inefficient simulation* (i.e., where the simulator might run in super-polynomial time even if the adversary runs just in polynomial time). Inefficient simulation is problematic when using the protocol as a sub-routine in the computational setting.

Our main question is whether an alternative *efficient* transformation from static to adaptive security exists. We show an inherent difficulty in achieving this goal generically. In contrast to the folklore, we present a protocol that is perfectly secure with efficient static simulation (therefore also adaptively secure with inefficient simulation), but for which efficient adaptive simulation does not exist (assuming the existence of one-way permutations).

In addition, we prove that the seminal protocol of Ben-Or, Goldwasser, and Wigderson (STOC'88) is secure against adaptive, semi-honest corruptions with *efficient* simulation. Previously, adaptive security of the protocol, as is, was only known either for a restricted class of circuits or for all circuits but with inefficient simulation.

Keywords: Secure multiparty computation; perfect security; adaptive security; BGW protocol.

*A preliminary version of this work appeared in ITC'22.

†Department of Computer Science, Bar-Ilan University. E-mail: gilad.asharov@biu.ac.il. Sponsored by the Israel Science Foundation (grant No. 2439/20), by JPM Faculty Research Award, by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office, and by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 891234.

‡Efi Arazi School of Computer Science, Reichman University. E-mail: cohenran@runi.ac.il. Research partially supported by NSF grant no. 2055568.

§Department of Computer Science, Bar-Ilan University. E-mail: oren.shochat6@gmail.com. Sponsored by the Israel Science Foundation (grant No. 2439/20).

Contents

1 Introduction

Secure multiparty computation (MPC) [Yao82, GMW87] enables a set of mutually distrustful parties to compute a joint function while keeping the privacy of their inputs and the correctness of their outputs. The seminal results from the '80s [Yao82, GMW87, BGW88, CCD88, RB89] as well as the vast majority of MPC protocols in the literature were proven secure with respect to a *static* adversary; that is, security is guaranteed as long as the adversary decides which parties to corrupt at the onset of the execution. A more realistic setting, first considered by Beaver and Haber [BH92] and by Canetti et al. [CFGN96], considers an *adaptive* adversary, who can dynamically decide which parties to corrupt during the course of the protocol. Adaptive security is known to be strictly stronger than static security, with many impossibility results separating the two notions, e.g., [CFGN96, CDD+04].

In this work, we focus on the well-studied setting of *perfect security*, where all existing separations from the literature no longer hold. Perfectly secure protocols guarantee security facing computationally unbounded adversaries without any error probability. This is a highly desirable property that, in some sense, provides the strongest security notion for MPC. For example, Kushilevitz et al. [KLR10] showed that any perfectly secure protocol that is proven secure in the standalone model using a straight-line and black-box simulation[1] automatically guarantees security under universal composition (UC) [Can01].[2] Most relevant to our work, Canetti et al. [CDD+04] showed that *any* perfect statically secure protocol, satisfying basic requirements, remains secure also in the presence of adaptive adversaries. This result, however, comes with a caveat, since the transformation from static to adaptive security does not preserve the efficiency of the simulation.

Roughly speaking, a protocol is deemed secure if any attack on an execution of the protocol in the real world can be simulated also in an ideal world where a trusted party receives the inputs from all the parties and computes the function on their behalf. It is desirable that the simulator, i.e., the adversary who simulates the attack in the ideal world, will use roughly the same resources as the adversary who carries out the attack on the real-world protocol. In particular, we would like the simulator to run in polynomial time with respect to the running time of the real-world adversary; if so, we say that the simulation is *efficient*.

Unfortunately, the adaptive simulator in [CDD+04] does *not* run in polynomial time with respect to the real-world adversary. This means that given a perfectly secure protocol against static adversaries with an efficient simulator, the transformation in [CDD+04] guarantees adaptive security, albeit with an inefficient simulator. This leads us to the following fundamental question:

Does perfect, static security with efficient simulation imply
perfect, adaptive security with efficient simulation?

Stated differently, is there another generic transformation from static to adaptive security, other than [CDD+04], that preserves the efficiency of the simulation? Are there other assumptions on the structure of the protocol and/or on the static simulation that might lead to such an efficient transformation? Or, perhaps, do there exist statically secure protocols for which *efficient* adaptive simulation simply does not exist?

Efficient vs. inefficient simulation. One may ask whether the efficiency loss in the simulation makes a difference when considering perfect security: If security is anyway guaranteed against

[1] A simulator is called *straight-line* if it does not rewind the adversary and is called *black-box* if it does not rely on the code of the adversary.

[2] We note that Backes et al. [BMU07] showed that this transformation no longer holds if the simulator is not straight-line.

computationally unbounded adversaries, does it matter if the simulator is inefficient? Indeed, inefficient simulation is a weaker-yet-acceptable security notion when considering information-theoretic security.

It turns out that inefficient simulation has an undesirable impact when considering *composition* of secure protocols. For example, consider a perfectly secure protocol π for computing a function f that is defined over secure communication channels, and consider a computationally secure realization of secure channels over authenticated channels (e.g., using non-committing encryption [CFGN96]). If π is secure with efficient simulation, then a composition theorem (e.g., from [Can00, Can01]) can be used to derive a computationally secure protocol for f over authenticated channels. However, if the simulation is *inefficient* then the composition will not go through since it will result in a super-polynomial time adversary that can break the cryptographic assumptions used to realize secure channels.

A case study: on the adaptive security of the BGW protocol. The seminal results of Ben-Or, Goldwasser, and Wigderson [BGW88] and of Chaum, Crépeau, and Damgård [CCD88] show that any function f can be computed by an n-party protocol with perfect security as long as $t < n/2$ of the parties are corrupted in the semi-honest setting, and as long as $t < n/3$ are corrupted in the malicious setting. A full proof of security for the BGW protocol was given in 2011 by Asharov and Lindell [AL11, AL17]. The proof was specified in the static, standalone setting, and universally composable security and security against adaptive adversaries were derived using the transformations of Kushilevitz et al. [KLR10] and Canetti et al. [CDD+04], respectively. However, as discussed above, adaptive security is obtained with an inefficient simulation.

This issue was revisited by Damgård and Nielsen [DN14], who showed that the semi-honest version of the BGW protocol achieves adaptive security with efficient simulation. However, the result of [DN14] holds only for circuits where each output wire is a direct output of a multiplication gate. Obviously, one can manually add such "multiplications with 1" to each output wire. While this seems sufficient, there are two reasons why we revisit this problem: First, for linear functions, the semi-honest version of the BGW protocol can tolerate up to n corruptions, whereas the requirement that each output wire is a direct output of a multiplication gate reduces the corruption threshold to $t < n/2$. Second, this subtlety has been neglected by prior works that relied on [DN14], e.g., Lin et al. [LLW20, Lem. 6.2] claimed that any degree-2 function could be computed by the BGW protocol in two rounds with adaptive security and efficient simulation. Yet, when adding multiplication gates, the round complexity increases, which implies a gap in the literature as there is no proof for the claim mentioned in [LLW20].[3]

Alternatively, one can interpret this additional restriction on the circuit as adding some re-randomization step at the very end of the protocol before the parties reconstruct their output. Is this step essential to achieve adaptive security for all circuits? Can one prove the adaptive security of the original protocol directly, without the additional communication round?

1.1 Our Results

Our work revisits the question of static versus adaptive in perfectly secure multiparty computation. We show that in contrast to the "weaker" definition of adaptive security (i.e., inefficient simulation), perfect static security no longer implies perfect adaptive security when demanding the simulation

[3] As discussed below, one can add a re-randomization for the output wires, and this re-randomization can already be sent in the first round, leading to a 2-round protocol; however, this modification of BGW is also not proven anywhere. In any case, we emphasize that the results in [LLW20] are correct, and merely were lacking a proof of the lemma.

to be efficient, even when merely considering semi-honest adversaries. Complementarily, we show that the BGW protocol is adaptively secure with efficient simulation even without changing the underlying circuit; this answers an open question posed by Damgård and Nielsen [DN14]. We focus on semi-honest security, which is enough to highlight the subtleties that arise; indeed, the gap in the literature discussed above already appears in this setting. We conjecture that the analysis extends to the malicious case using standard secret-sharing techniques.

We proceed to describe our results in more detail.

Separating perfect adaptive security from perfect static security. Our first result shows that under some cryptographic assumptions, there is no hope of finding an alternative (efficient) transformation to that of Canetti et al. [CDD+04], and that the inefficiency of the adaptive simulation is inherent. More precisely, there exist protocols that admit perfect, static security with efficient simulation but for which an efficient adaptive simulation does not exist.

Theorem 1.1. *Assume the existence of a one-way permutation. Then, there exists an n-party functionality f and a protocol that securely computes f with efficient, perfect static simulation, but for which efficient, perfect adaptive simulation does not exist.*

The theorem is proven by showing a protocol for which all the additional requirements of [CDD+04] hold and therefore (inefficient) adaptive simulation *does* exist for this protocol, but an efficient adaptive simulator can be used to invert the one-way permutation with inverse-polynomial probability. Interestingly, this implies that the protocol is regarded as adaptively secure in the perfect setting, but the exact same protocol is not adaptively secure in the *computational* setting, where both the adversary and the simulator should run in polynomial time in the security parameter. We, therefore, derive the following somewhat counter-intuitive corollary.

Corollary 1.2. *Assume the existence of a one-way permutation. Then, there exists an n-party functionality f and a protocol that securely computes f with perfect adaptive security, but that does not securely compute f with computational adaptive security.*

Revisiting adaptive security in the standalone setting. The above does not rule out the possibility of finding additional requirements from the protocol that would imply efficient adaptive simulation. This is exactly the approach taken by Damgård and Nielsen [DN14], who showed that under additional requirements of the protocol, an efficient adaptive simulation exists.

The transformation of [DN14] is directly proved in the UC framework with its full generality, capturing reactive functionalities and concurrency issues at once. However, the strong guarantees do not come without a price, since the requirements from the static simulator must capture multiple input phases (as required for reactive computations) and deal with the technical overhead needed for concurrent composition, e.g., incorporating an "online" environment to the definition. As a small side contribution, we simplify this transformation.

Specifically, in addition to proving perfect static security, the transformation of [DN14] requires an additional (efficient!) algorithm, called "Patch," for sampling randomness that explains the simulated protocol whenever a corruption occurs. The adaptive simulator invokes the static simulator on a dynamically growing set of corrupted parties (initially empty). At any point, the simulation of the protocol towards the adaptive adversary is done by forwarding messages from the adaptive adversary to the static simulator, and vice-versa. Upon a corruption of a new party, say of P_i, the Patch algorithm receives the state of the static simulator until this point together with the input and output of P_i, and outputs a new state for the static simulator that allows the continuation of the simulation as if P_i was statically corrupted from the beginning.

We then propose an alternative recipe for proving universal composability and (efficient) adaptive security in the perfect setting:

1. Prove that the protocol is (efficiently) statically secure in the perfect *standalone* setting and that the protocol satisfies some natural requirements (similar to those in [CDD+04]).
2. Show the existence of an efficient "Patch" algorithm corresponding to the static simulator.
3. We show a transformation (which is essentially a distilled version of [DN14]) that the protocol is perfectly adaptively secure with an efficient simulator in the standalone setting.
4. Using [KLR10], the protocol is also secure in the perfect adaptive setting with efficient simulation and with universal composability.

We remark that this result is not technically novel and is inspired by [DN14]. We hope that providing an alternative definition in the standalone setting would simplify proofs of efficient adaptive security in the future, as the protocol designer can focus on the standalone setting.

Adaptive security of the BGW protocol. Finally, we follow our recipe proposed above and show that the (semi-honest) BGW protocol is adaptively secure with efficient simulation. No additional step to the protocol is needed, and the proof works for any circuit (as opposed to the proof of [DN14]). This result solves an open problem raised by [DN14], whether the assumption on the circuit (that each output wire is a direct output of a multiplication gate) is necessary. This reaffirms the security of the BGW protocol [BGW88], and closes a gap in the proofs of [AL11, DN14], providing sound foundations for cryptography.

Theorem 1.3. *Let f be a deterministic n-party functionality. The BGW protocol securely computes f with perfect adaptive security and efficient simulation facing a semi-honest adversary corrupting $t < n/2$ parties.*

Further, if f is a linear function, the BGW protocol securely computes f with perfect adaptive security and efficient simulation facing a semi-honest adversary corrupting $t < n$ parties.

1.2 Technical Overview

We provide a technical overview of our results. In Section 1.2.1 we review our separation result, showing the existence of a protocol that has inefficient adaptive simulation but no efficient adaptive simulation (under some cryptographic assumptions). In Section 1.2.2, we review the adaptive security of the BGW protocol.

1.2.1 Static Security Does Not Imply Adaptive Security

Definition of adaptive security. We first recall the definition of adaptive security. We remark that since the transformation of [CDD+04] assumes some additional properties from the statically secure protocol and its simulation (specifically, it assumes that the protocol has a straight-line, black-box simulation), our description here incorporates those properties in the informal definition.

Given a protocol π, an adaptive adversary might corrupt parties on the fly as the protocol proceeds. Upon corruption, the adversary sees the corrupted party's random tape, the input it used, and all the messages it has received so far. From that point on, the adversary completely controls the behavior of P_i. As the protocol proceeds, the adversary might decide to corrupt additional parties, as a function of whatever it saw so far.

We follow the ideal/real simulation paradigm and say that a protocol is adaptively secure if for every such an adversary in the real world, there exists a simulator in the ideal world that simulates

its behavior. In the ideal world, the simulator invokes the real-world adversary and simulates the honest parties sending their message to the corrupted parties (without knowing the inputs of the honest parties). At every round, the adversary might ask to corrupt some party P_i in the simulated protocol, and the simulator corrupts the corresponding party in the ideal world. Upon corruption, the simulator learns the input of P_i (and its output, if it was already computed by the trusted party), and it has to provide the adversary the input *together with randomness* that explains the messages sent by this party in the simulation until this point (also known as "equivocation" in the literature).

This is the challenging part of the proof as the simulator has to first simulate P_i without knowing its input, "commit" to some messages on its behalf, and later upon corruption find a randomness r_i that explains all the messages that were sent so far according to the input x_i. Note that the simulator is not allowed to rewind the adversary at any point of the execution, i.e., the simulation is "straight-line."

First attempt. The transformation of [CDD+04] shows that for any perfectly secure protocol, randomness r_i describing the behavior of P_i so far, exists. This implies that the simulator can *always* find it; however, finding it might not be an efficient procedure. Our first attempt is adding some cryptographic hardness while targeting exactly this procedure of finding the matching randomness. That is, our goal is to make the finding of the randomness a computationally hard problem.

It is intuitive to simply take the BGW protocol, even just for semi-honest security and for computing a linear function, with one modification:[4] Before a party starts the protocol, it takes its random tape $\boldsymbol{\rho}$, and uses $\mathsf{OWP}(\boldsymbol{\rho})$ as its randomness in the protocol, where OWP is a one-way permutation. The intuition is that whenever the adversary asks to corrupt some party, the simulator effectively has to invert the one-way permutation, which is computationally infeasible. The construction is still statically secure since the static simulator only goes "forward"; that is, it chooses some randomness $\boldsymbol{\rho}$ and then uses $\mathsf{OWP}(\boldsymbol{\rho})$ as the randomness of the simulated honest party. However, it seems that an adaptive adversary will have to move "backward" and invert the one-way permutation.

To elaborate further, let n be the number of parties, and consider the function $f(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i$; that is, all parties receive the same output, which is the sum of their inputs. We consider some finite field $\mathbb{F}$ with $|\mathbb{F}| > n$. The protocol is defined as follows:

- **Input:** P_i has $x_i \in \mathbb{F}$ as input, and randomness $\boldsymbol{\rho}_i \in \mathbb{F}^{n-1}$.
- **The protocol:**
  1. P_i computes $(r_{i\to 1}, \ldots, r_{i\to n-1}) = \mathsf{OWP}(\boldsymbol{\rho}_i)$, where each $r_{i\to j} \in \mathbb{F}$.
  2. For every $j \in [n-1]$ P_i sends $r_{i\to j}$ to party P_j. Next, P_i defines
  $$r_{i\to n} := x_i - (r_{i\to 1} + \ldots + r_{i\to n-1})$$
  and sends $r_{i\to n}$ to party P_n.
  3. Upon receiving $r_{1\to i}, \ldots, r_{n\to i}$ from all the parties, party P_i computes $\beta_i := \sum_{j=1}^{n} r_{j\to i}$ and sends β_i to all other parties.
  4. Upon receiving $\beta_1, \ldots, \beta_n$, each party computes $y := \beta_1 + \ldots + \beta_n$, and outputs y.

The above protocol can be statically simulated for every $t < n$ corruptions. Moreover, an inefficient adaptive simulator exists by [CDD+04]. What about efficient adaptive simulation?

[4] Since the function is linear, security holds for any $t < n$ number of corruptions. For brevity, we change the underlying secret sharing from threshold-Shamir sharing to additive secret sharing.

The problem – the reduction to OWP. Unfortunately, assuming the existence of an efficient adaptive simulator, it is hard to construct an inverter to the one-way permutation. In order to see that, let's simply try to construct such a reduction. Recall that the inverter receives some challenge $y^* = (r^*_1, \ldots, r^*_{n-1}) \in \mathbb{F}^{n-1}$ and it has to find $\boldsymbol{\rho} \in \mathbb{F}^{n-1}$ such that $y^* = \mathsf{OWP}(\boldsymbol{\rho})$. Consider an efficient adaptive simulator, and assume that the adversary have already corrupted $n-2$ parties at the onset of the execution, say $P_1, \ldots, P_{n-2}$. The simulator then simulates just two honest parties: P_{n-1} and P_n. As it does not know their inputs, it must give the adversary $2 \cdot (n-2)$ independent random values as the messages of the two honest parties it simulates (those are the shares that P_{n-1} and P_n send to the corrupted parties, $P_1, \ldots, P_{n-2}$). For concreteness, assume that the messages that the simulated P_n sent to the adversary are $(\gamma_{n\to 1}, \ldots, \gamma_{n\to n-2})$.

If it happened to be that $(\gamma_{n\to 1}, \ldots, \gamma_{n\to n-2}) = (r^*_1, \ldots, r^*_{n-2})$ then potentially we can use the simulator to invert: The adversary asks to corrupt P_n and it might receive back the pre-image of $(r^*_1, \ldots, r^*_{n-1})$, as required. Note that the simulator did not commit to $\gamma_{n\to n-1}$, and therefore there are $|\mathbb{F}|$ many different other pre-images that the efficient adaptive simulator might give us (there are $|\mathbb{F}|$ possible ways to set $\gamma_{n\to n-1}$; only if $\gamma_{n\to n-1} = r^*_{n-1}$ then the simulator must invert the challenge). In any case, there is no guarantee whatsoever that $(\gamma_{n\to 1}, \ldots, \gamma_{n\to n-2}) = (r^*_1, \ldots, r^*_{n-2})$. Likewise, we can try to adaptively corrupt P_{n-1} instead of P_n if it happened to be that the shares P_{n-1} sent match the challenge, but again there is no such guarantee. In particular, inverting is possible only if the simulator explains the view in a way that connects the challenge that the inverter receives and the partial view that the simulator generated earlier.

Second attempt. To solve the above technicality, we change the functionality and the protocol. Instead of having the input of each party P_i be just $x_i \in \mathbb{F}$, it is augmented to be an ordered set of values $(x_i, (a_{i\to 1}, \ldots, a_{i\to n-1}))$ over $\mathbb{F}$. The functionality is then defined as:

$$f\Big((x_1, (a_{1,1}, \ldots, a_{1,n-1})), \ldots, (x_n, (a_{n,1}, \ldots, a_{n,n-1}))\Big) = \sum_{i=1}^{n} x_i.$$

Note that the parties input n elements, while all but the first ones do not affect the output of the computation. In the protocol, each party P_i then invokes $(r_{i,1}, \ldots, r_{i,n-1}) = \mathsf{OWP}(\boldsymbol{\rho}_i)$ where $\boldsymbol{\rho}_i$ is its random tape, and then defines the share $\gamma_{i\to j} = a_{i,j} + r_{i,j}$ for party P_j for every $j \in [n-1]$, and the share $\gamma_{i\to n} = x_i - ((a_{i,1} + r_{i,1}) + \ldots + (a_{i,n-1} + r_{i,n-1}))$ for party P_n. It then sends to each other party P_j the corresponding share, just as in Step 2 in the previous protocol.

The difference is that now, no matter what points $(\gamma_{n\to 1}, \ldots, \gamma_{n\to n-2})$ the adversary commits to as the messages P_n had sent, for any challenge $y^* = (r^*_1, \ldots, r^*_{n-1})$, the inverter of the OWP can choose an input $(x_n, (a_{n,1}, \ldots, a_{n,n-1}))$ such that $a_{n,j} = \gamma_{n\to j} - r^*_j$ for every $j \in [n-2]$.

To see that, given a challenge $y^* = (r^*_1, \ldots, r^*_{n-1})$ to the inverter, and given $(\gamma_{n\to 1}, \ldots, \gamma_{n\to n-2})$ that were chosen by the simulator as the messages P_n had sent in the first round, the inverter chooses two additional points x_{n-1} and $a_{n,n-1}$ at random. It gives $(x_{n-1}, (a_{n,1}, \ldots, a_{n,n-1}))$ to the adaptive simulator as the input of P_n.

The simulator now has to reply with some randomness $\boldsymbol{\rho}'$ for which $(r'_{n,1}, \ldots, r'_{n,n-1}) = \mathsf{OWP}(\boldsymbol{\rho}')$ such that $\gamma_{n\to j} = a_{n,j} + r'_{n,j}$ for every $j \in [n-2]$. Since $\gamma_{n\to n-1}$ is not determined, the simulator essentially has $|\mathbb{F}|$ different randomness to choose from. However, *one of them* corresponds to the challenge y^*. Moreover, y^* is distributed uniformly in the support of all valid solutions for the simulator. Since the adaptive simulator simulates with perfect security, the inverter succeeds in inverting y^* with probability $1/|\mathbb{F}|$. By tuning the parameters (such that $|\mathbb{F}|$ is polynomial in the security parameter), this is an inverse-polynomial advantage. Assuming that OWP is a one-way permutation, this implies that the adaptive simulator cannot run in polynomial time.

1.2.2 Adaptive Security of the BGW Protocol

The BGW Protocol. We briefly recall the semi-honest version of the BGW protocol [BGW88] that is secure for $t < n/2$ corruptions; see [AL11] for more details.

Input sharing: Initially, each party P_i secret shares its input x_i using a $(t+1)$-out-of-n Shamir secret sharing (t is the privacy threshold and $t+1$ are needed for reconstruction), and sends the j^{th} share to P_j. Specifically, P_i chooses a random polynomial q_i of degree (at most) t with constant term x_i, and sends to P_j the share $q_i(\alpha_j)$.

Circuit emulation: The parties evaluate the circuit gate by gate on their shares. Addition and scalar-multiplication gates are evaluated locally; that is, each party performs the operation on its shares. Multiplication requires a dedicated sub-protocol in which each party inputs shares for two values a and b, and obtains a share for the product ab as output. For simplicity of the technical overview, we consider that this operation is done using the help of a trusted party (i.e., we work in the f_{mult}-hybrid model).

Output reconstruction: Upon conclusion of the circuit emulation, each party holds one share for each output wire. For simplicity, assume that output wire o_i is the private output for P_i. Then, each party P_j sends the relevant share $\beta_{j\to i}$ for P_i, who can reconstruct the polynomial g_i interpolating the points $(\alpha_1, \beta_{1\to i}), \ldots, (\alpha_n, \beta_{n\to i})$, and obtain $g_i(0)$ as its output.

Static simulation. To simulate the adversary's view for the set of corrupted parties I of cardinality at most t, the static simulator simulates the input-sharing phase by choosing randomness for the corrupted parties (which, in turn, determines the polynomials they use in the input-sharing phase). Moreover, for the honest parties, the simulator just chooses $|I|$ random elements for each honest party and simulates the honest parties sending their shares on their inputs to the adversary. The simulator can then locally compute the shares of the corrupted parties on each internal wire of the circuit, while for simulating an invocation of f_{mult}, the simulator again just provides the adversary with $|I|$ random shares. In the output-reconstruction phase, the simulator has to provide all shares on the output wires of corrupted parties. It knows the constant term of each such wire, and it knows $|I|$ shares on each wire, and therefore it is possible to interpolate a random polynomial that passes through the $|I| \leq t$ shares and the known constant term, and simulate the honest parties sending shares on this polynomial to the adversary. When $|I| = t$, this is a deterministic process as we already have $t+1$ shares that are determined on the output wires; however, when $|I| < t$, the simulator needs to generate some new shares on that output wire.

The assumption on the circuit. Damgård and Neilsen assumed that the output wires are direct outputs of multiplication gates. As a result, the shares on each output wire are independent of each other – and the simulator can just choose additional $|I|-t$ random shares on the output wire to interpolate the polynomial on that wire. When considering arbitrary circuits, output wires that are not a result of a multiplication gate may have some linear relation between them. To illustrate the challenge, consider output wires o_1, o_2, and o_3, corresponding to three corrupted parties P_1, P_2, and P_3, respectively, such that $o_3 = o_1 + o_2$. Denote the output values by y_1, y_2, and y_3, respectively. The shares that the parties hold on the output wires define polynomials $g_1(x), g_2(x)$ and $g_3(x)$, respectively. In the real execution, it holds that $g_3(x) = g_1(x) + g_2(x)$, and therefore the simulator must choose the shares on the output wires wisely to guarantee the same dependency, which makes the simulation more challenging.

Our static simulator. To guarantee such consistencies, our static simulator generates t random shares on each input wire and each output of a multiplication wire. That is, while the adversary corrupts some set I, the simulator, "in its head," chooses an arbitrary set $\hat{I} \supseteq I$ of cardinality exactly t and simulates the shares for all parties in $\hat{I}$, while giving the adversary only shares for the parties in I. As a result, we have no random choice when simulating the output wires. The simulator holds t shares on each output wire, and also knows the constant term on the output wires of the corrupted parties. It can deterministically interpolate the polynomials on the output wire, and simulate the honest parties sending their shares on those wires. This guarantees that if there is any dependency between output wires, then the simulator provides consistent shares.

Our adaptive simulator. Assume that the adaptive adversary corrupts some party P_{i^*}. If $i^* \in \hat{I} \setminus I$, then providing the view of that party is easy. The simulator has already sampled all the shares that P_{i^*} is supposed to receive. On the other hand, if $i^* \notin \hat{I} \setminus I$, then we face a new obstacle. This is because the simulator has already defined t shares on each wire, and defining an additional share on each wire would completely determine each polynomial. Let $j^* \in \hat{I} \setminus I$. The key idea of our adaptive simulator is to "replace" the sampling of shares for party P_{j^*} with sampling shares for party P_{i^*}. Specifically, we show that each random choice made for P_{j^*} that leads to the current view of the adversary, can also be interpreted by a random choice made for P_{i^*} that leads to the exact same view. This procedure then changes the set of the simulator and "forgets" all the random choices made for P_{j^*}, but instead samples matching choices for P_{i^*}. This is essentially sampling random shares for P_{i^*} on the input wires of all honest parties and the outputs of f_{mult}, under the constraints posed by the shares of P_{j^*} on the output wires. Such sampling can be performed efficiently by solving a linear set of equations. Then, we are back to the previous case, where the corruption is made on some $i^* \in \hat{I} \setminus I$.

1.3 Related Work

Adaptive security is known to be strictly stronger than static security in many settings, with many impossibility results separating the two notions. Below, we compare our separation to existing separations between static and adaptive security from the literature.

The first separation was presented by [CFGN96] and relied on a positive error probability of the statically secure protocol. They considered a dealer that secret shares a value to a random set of parties and later announces their identities; an adaptive adversary can corrupt the members of the set and learn the value while a static adversary can only guess this set ahead of time and succeed with negligible (yet positive) probability. In this setting, Cramer et al. [CDD+99] showed that certain protocols from [RB89] guarantee statistical information-theoretic security against static corruptions but are not secure in the adaptive setting. Separations based on statistical security are different than ours as we consider *perfect* protocols that have zero error probability.

In the computational-security setting, many statically secure primitives do not remain secure under adaptive corruptions. For example, Beaver [Bea96] showed that if the zero-knowledge proofs of GMW [GMW86] are adaptively secure then the polynomial hierarchy collapses. Nielsen [Nie02] ruled out public-key encryption for unbounded messages in the non-programmable random oracle model. Canetti et al. [CDD+04] separated adaptively secure commitments from statically secure ones. Lindell and Zarosim [LZ11] showed that achieving adaptively secure oblivious transfer (OT) requires stronger assumptions than statically secure OT. Katz et al. [KTZ13] ruled out adaptively secure fully homomorphic encryption; the latter result was generalized in [CsW19]. Separations based on computational assumptions are different than ours as we consider *information-theoretic* protocols that remain secure against computationally unbounded adversaries.

Canetti et al. [CDD+04] showed a separation based on the ability of the adversary to corrupt a party and change its input to the protocol based on messages that have already been transmitted. Similar separations were illustrated also for broadcast protocols [HZ10, GKKZ11, CGZ23]. Canetti et al. [CDD+04] showed that such separations no longer hold when considering protocols that have a *committal round* [MR91], i.e., a fixed round in which all inputs to the protocol are committed. These separations hold only for malicious adversaries that can deviate from the protocol; our separation applies for semi-honest adversaries that cannot deviate from the protocol in any way.

Other separations are known when considering restricted interaction patterns, as was shown by Garay et al. [GIOZ17] for protocols with sublinear communication, and by Boyle et al. [BCDH18] for protocols whose communication graph admits a sublinear cut. Our separation relies on the BGW protocol that induces a complete communication graph.

Finally, Garg and Sahai [GS12] showed that constant-round MPC with black-box simulation in the plain model cannot tolerate corruption of all of the parties. Our result holds irrespective of the number of rounds and does not require corrupting all the parties.

2 Preliminaries

Our results hold in any natural model that captures perfect adaptive security, for example, [CFGN96, Can00, DM00, Can01, KTR20, HLM21]. For concreteness, we will state our results using the modular (non-concurrent) composability framework of Canetti [Can00]. Indeed, the separation in this limited setting extends to any of the models listed above, whereas our positive results translate to the universal-composability setting via the transformation in [KLR10]. Before describing the security model, we give basic notation and define the cryptographic primitive used in our separation.

Notation. We denote by λ the security parameter. For $n \in \mathbb{N}$, let $[n] = \{1, \ldots, n\}$. Let poly denote the set of all positive polynomials and let PPT denote a probabilistic algorithm that runs in *strictly* polynomial time. A function $\nu \colon \mathbb{N} \to [0,1]$ is *negligible* if $\nu(\lambda) < 1/p(\lambda)$ for every $p \in \mathsf{poly}$ and large enough λ. Given a random variable X, we write $x \leftarrow X$ to indicate that x is selected according to X, and given a set $\mathcal{X}$ we write $x \leftarrow \mathcal{X}$ to indicate that x is selected uniformly at random from $\mathcal{X}$.

2.1 One-Way Permutations

Our separation in Section 3 will rely on the existence of a one-way permutation (OWP); that is a one-way function which is length preserving and one-to-one; we refer the reader to [Gol04] for more details.

Definition 2.1 (OWP)**.** *A polynomial-time function $f : \{0,1\}^* \to \{0,1\}^*$ is a one-way permutation if the following conditions are satisfied.*

1. *For every $\lambda \in \mathbb{N}$, f induces a permutation over $\{0,1\}^\lambda$, i.e., $f : \{0,1\}^\lambda \to \{0,1\}^\lambda$ is one-to-one and onto.*
2. *There exists a deterministic polynomial-time algorithm A such that on input $x \in \{0,1\}^*$ the algorithm A outputs $f(x)$.*
3. *For every PPT algorithm $\mathcal{A}$, every positive polynomial $p(\cdot)$, and all sufficiently large λ's, it holds that*

$$\Pr_{x \leftarrow \{0,1\}^\lambda}\Big[\mathcal{A}(f(x)) = x\Big] < \frac{1}{p(\lambda)}.$$

2.2 Secret Sharing and Hiding

The BGW protocol is based on Shamir's secret sharing [Sha79]. In our analysis of the protocol in Section 5 we rely on the following claim, taken from [AL11, Claim 3.2].

Claim 2.2. *For any set of distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$, any pair of values $s, s' \in \mathbb{F}$, any subset $I \subset [n]$ where $|I| = \ell \leq t$ and every $\vec{y} \in \mathbb{F}^\ell$ it holds that:*

$$\Pr\left[\vec{y} = (f(\alpha_i))_{i \in I}\right] = \Pr\left[\vec{y} = (g(\alpha_i))_{i \in I}\right] = \frac{1}{|\mathbb{F}|^\ell}$$

where the probability on the left (resp. right) is taken over the choice of $f(x)$ (reps. $g(x)$) uniformly at random from the set of all polynomials of degree-t with s (resp. s') as its constant term.

2.3 Security Model: Static Corruptions

The definition in [Can00] is based on the simulation paradigm, where the security of a protocol running in the real world is compared to an ideal computation of the function. We start by defining secure protocols in the non-adaptive case. We first formalize the real-life model; next we describe the ideal process; finally the notion of emulation of the ideal process by a computation in the real-life model is presented.

Real-world execution. An n-party protocol $\pi = (P_1, \ldots, P_n)$ is an n-tuple of PPT interactive Turing machines (ITMs). The term *party* P_i refers to the i^{th} ITM. Each party P_i starts with input $x_i \in \{0,1\}^*$ and random coins $r_i \in \{0,1\}^*$. An *adversary* $\mathcal{A}$ is another ITM describing the behavior of the corrupted parties. It starts the execution with input that contains the identities of the corrupted parties and their private inputs, an additional auxiliary input, and random coins $r_{\mathcal{A}} \in \{0,1\}^*$. The parties execute the protocol in a synchronous network. That is, the execution proceeds in rounds: Each round consists of a *send phase* (where parties send their messages for this round) followed by a *receive phase* (where they receive messages from other parties). The adversary is assumed to be *rushing*, which means that it can see the messages the honest parties send in a round before determining the messages that the corrupted parties send in that round. The parties communicate over a fully connected point-to-point network of ideally private communication lines, meaning that the adversary cannot modify messages sent between two honest parties nor read them.

Throughout the execution of the protocol, all the honest parties follow the instructions of the prescribed protocol, whereas the corrupted parties receive their instructions from the adversary. A *semi-honest* adversary always instructs the parties to continue the execution honestly, whereas a *malicious* adversary can instruct the corrupted parties to deviate from the protocol in any arbitrary way. At the conclusion of the execution, the honest parties output their prescribed output from the protocol, the corrupted parties do not output anything and the adversary outputs an (arbitrary) function of its view of the computation (containing the views of the corrupted parties). The view of a party in a given execution of the protocol consists of its input, its random coins, and the messages it sees throughout this execution.

Definition 2.3 (real-world execution: static case)**.** *Let $\pi = (P_1, \ldots, P_n)$ be an n-party protocol and let $I \subseteq [n]$ denote the set of indices of the parties corrupted by $\mathcal{A}$. Denote by $\text{REAL}_{\pi,I,\mathcal{A}}(\mathbf{x}, z, \mathbf{r})$ the output vector of $P_1, \ldots, P_n$ and $\mathcal{A}$ resulting from the protocol interaction on input vector $\mathbf{x} = (x_1, \ldots, x_n)$, auxiliary input z, and randomness $\mathbf{r} = (r_{\mathcal{A}}, r_1, \ldots, r_n)$. Let $\text{REAL}_{\pi,I,\mathcal{A}}(\mathbf{x}, z)$ denote the probability distribution of $\text{REAL}_{\pi,I,\mathcal{A}}(\mathbf{x}, z, \mathbf{r})$ where $\mathbf{r}$ is uniformly chosen.*

Ideal-world execution. The ideal-world computation is parameterized by a (potentially randomized) n-party functionality $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ to compute. An ideal computation of f on input $\mathbf{x} = (x_1, \ldots, x_n)$ for parties $(P_1, \ldots, P_n)$ in the presence of an adversary (a simulator) $\mathcal{S}$ controlling the parties indexed by $I \subseteq [n]$, proceeds via the following steps.

***Sending inputs to trusted party*:** An honest party P_i sends its input x_i to the trusted party. If the adversary is semi-honest it sends x_i to the trusted party for every corrupted party P_i; if the adversary is malicious it may send to the trusted party arbitrary inputs for the corrupted parties. Let x'_i be the value actually sent as the input of party P_i.

***Computation stage*:** If x'_i is outside of the domain for P_i or if P_i sends no input, the trusted party sets x'_i to be some predetermined default value. Next, the trusted party samples randomness $r_f \leftarrow \{0,1\}^*$, computes $(y_1, \ldots, y_n) = f(x'_1, \ldots, x'_n; r_f)$, and sends y_i to party P_i.

***Outputs*:** Honest parties always output the message received from the trusted party while the corrupted parties output nothing. The adversary $\mathcal{S}$ outputs an arbitrary function of the initial inputs $\{x_i\}_{i \in I}$, the messages received by the corrupted parties from the trusted party, and its auxiliary input.

Definition 2.4 (ideal computation: static case)**.** *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality and let $I \subseteq [n]$ be the set of indices of the corrupted parties. Denote by $\text{IDEAL}_{f,I,\mathcal{S}}(\mathbf{x}, z, \mathbf{r})$ the output vector of $P_1, \ldots, P_n$ and $\mathcal{S}$ resulting from the above described ideal process on input vector $\mathbf{x} = (x_1, \ldots, x_n)$, auxiliary input z, and randomness $\mathbf{r} = (r_f, r_{\mathcal{S}})$. Let $\text{IDEAL}_{f,I,\mathcal{S}}(\mathbf{x}, z)$ denote the probability distribution of $\text{IDEAL}_{f,I,\mathcal{S}}(\mathbf{x}, z, \mathbf{r})$ where $\mathbf{r}$ is uniformly chosen.*

Security definition. Having defined the real and ideal computations, we can now define static security of protocols.

Definition 2.5 (static security)**.** *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality. A protocol π statically t-securely computes f with perfect security if for every real-world adversary, there exists an ideal-world adversary $\mathcal{S}$, such that for every $I \subseteq [n]$ of size at most t, it holds that*

$$\{\text{REAL}_{\pi,I,\mathcal{A}}(\mathbf{x}, z)\}_{(\mathbf{x},z) \in (\{0,1\}^*)^{n+1}} \equiv \{\text{IDEAL}_{f,I,\mathcal{S}}(\mathbf{x}, z)\}_{(\mathbf{x},z) \in (\{0,1\}^*)^{n+1}} .$$

If $\mathcal{S}$ runs in polynomial time with respect to the running time of $\mathcal{A}$ than we say that π statically t-securely computes f with perfect security and efficient simulation.

If $\mathcal{A}$ and $\mathcal{S}$ are semi-honest, we say that π statically t-privately computes f with perfect security and efficient simulation.

2.4 Security Model: Adaptive Corruptions

In the adaptive setting, the adversary has the ability to corrupt parties during the protocol execution and learn their internal state. This means that the adversary learns their input, output, incoming messages, and random coins used in the current execution of the protocol, but also their internal states from previous runs. To capture information about previous executions, the model includes an additional ITM called the *environment*, denoted $\mathcal{Z}$, whose role is to provide the adversary with auxiliary information about newly corrupted parties. The environment may also issue corruption requests *after* the conclusion of the protocol, in the *post-execution corruption* stage.

Real-world execution. An execution of a protocol is similar to the non-adaptive case with small adjustments. As before, an n-party protocol $\pi = (P_1, \ldots, P_n)$ consists of n PPT ITMs, where every party P_i starts with input $x_i \in \{0,1\}^*$ and random coins $r_i \in \{0,1\}^*$. An *adversary* $\mathcal{A}$ is an ITM that starts with random coins $r_{\mathcal{A}} \in \{0,1\}^*$, and the environment $\mathcal{Z}$ is an ITM that starts with auxiliary information z and random coins $r_{\mathcal{Z}}$. We say that the adversary is t-limited if it can corrupt at most t parties during the protocol (including the post-execution corruption stage).

At the beginning of the execution, the adversary receives a message from $\mathcal{Z}$ (corresponding to the auxiliary information seen by $\mathcal{A}$ in the static case), and then the execution proceeds in synchronous rounds, where each round proceeds in mini-rounds as follows: Each mini-round starts by allowing $\mathcal{A}$ to corrupt parties one by one in an adaptive way. Next, $\mathcal{A}$ chooses an uncorrupted party P_i that was not yet activated in this round and activates it. Upon activation, P_i receives the messages sent to it in the previous round, generates its messages for this round, and the next mini-round begins. The adversary learns the messages sent by P_i to all (currently) corrupted parties. Once all the uncorrupted parties were activated, $\mathcal{A}$ generates the messages to be sent by the corrupted parties that were not yet activated in this round, and the next round begins.

Once a party is corrupted, the party's input, random input, and the entire history of the messages sent and received by the party become known to $\mathcal{A}$. In addition, $\mathcal{Z}$ learns the identity of the corrupted party, and hands additional auxiliary information to $\mathcal{A}$. From this point, $\mathcal{A}$ learns all the messages received by the party. If $\mathcal{A}$ is semi-honest, then the corrupted parties continue running protocol π, whereas if $\mathcal{A}$ is malicious, it can instruct the corrupted party to deviate from the protocol in any arbitrary way.

At the conclusion of the execution, the honest parties output their prescribed output from the protocol, the corrupted parties do not output anything and the adversary outputs an (arbitrary) function of its view of the computation (containing the views of the corrupted parties).

Next, a "post-execution corruption process" begins. Initially, $\mathcal{Z}$ learns the outputs of all the parties and of the adversary, and later $\mathcal{Z}$ and $\mathcal{A}$ interact in rounds, where in each round $\mathcal{Z}$ sends a "corrupt P_i" request to $\mathcal{A}$, who may corrupt more parties as before (in which case $\mathcal{Z}$ learns their identity) and hands $\mathcal{Z}$ some arbitrary response. The interaction continues until $\mathcal{Z}$ halts with some output.

Definition 2.6 (real-world execution: adaptive case)**.** *Let $\pi = (P_1, \ldots, P_n)$ be an n-party protocol. Denote by $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(\mathbf{x}, z, \mathbf{r})$ the output of $\mathcal{Z}$ resulting from the protocol interaction, as described above, on input vector $\mathbf{x} = (x_1, \ldots, x_n)$, auxiliary input z, and randomness $\mathbf{r} = (r_{\mathcal{Z}}, r_{\mathcal{A}}, r_1, \ldots, r_n)$. Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(\mathbf{x}, z)$ denote the random variable describing $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(\mathbf{x}, z, \mathbf{r})$ where $\mathbf{r}$ is uniformly chosen.*

Ideal-world execution. As in the static case, the ideal-world computation is parameterized by a (potentially randomized) n-party functionality $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ to compute. An ideal computation of f on input $\mathbf{x} = (x_1, \ldots, x_n)$ for parties $(P_1, \ldots, P_n)$ in the presence of an adversary (a simulator) $\mathcal{S}$ and an environment $\mathcal{Z}$, proceeds via the following steps.

***First corruption stage*:** Initially, $\mathcal{S}$ receives auxiliary information from $\mathcal{Z}$. Next, $\mathcal{S}$ proceeds in iterations, where in each iteration $\mathcal{S}$ may decide to corrupt some party, based on its random input and the information gathered so far. Once a party is corrupted its input becomes known to $\mathcal{S}$. In addition, $\mathcal{Z}$ learns the identity of the corrupted party and hands some extra auxiliary information to $\mathcal{S}$. Let I denote the set of corrupted parties at the end of this stage.

***Sending inputs to trusted party*:** An honest party P_i sends its input x_i to the trusted party. If the adversary is semi-honest it sends x_i to the trusted party for every corrupted party P_i; if

the adversary is malicious it may send to the trusted party arbitrary inputs for the corrupted parties. Let x'_i be the value actually sent as the input of party P_i.

***Computation stage*:** If x'_i is outside of the domain for P_i or if P_i sends no input, the trusted party sets x'_i to be some predetermined default value. Next, the trusted party samples $r_f \leftarrow \{0,1\}^*$, computes $(y_1, \ldots, y_n) = f(x'_1, \ldots, x'_n; r_f)$, and sends y_i to party P_i.

***Second corruption stage*:** After learning the output, $\mathcal{S}$ proceeds in another sequence of iterations, where in each iteration $\mathcal{S}$ may decide to corrupt some additional party, based on the information gathered so far. Upon corruption, $\mathcal{Z}$ learns the identity of the corrupted party, $\mathcal{S}$ sees the corrupted party's input and output, plus some additional information from $\mathcal{Z}$ as before.

***Outputs*:** Honest parties always output the message received from the trusted party while the corrupted parties output nothing. The adversary $\mathcal{S}$ outputs an arbitrary function of its view in the computation.

***Post-execution corruption (PEC)*:** Once the outputs are generated, $\mathcal{S}$ and $\mathcal{Z}$ proceed in rounds where in each round $\mathcal{Z}$ generates some "corrupt P_i" request, and $\mathcal{S}$ generates some arbitrary answer based on its view of the computation so far. For this purpose, $\mathcal{S}$ may corrupt more parties as described in the second corruption stage. The interaction continues until $\mathcal{Z}$ halts with an arbitrary output.

Definition 2.7 (ideal computation: adaptive case)**.** *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality. Denote by $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}}(\mathbf{x}, z, \mathbf{r})$ the output of $\mathcal{Z}$ resulting from the above described ideal process on input vector $\mathbf{x} = (x_1, \ldots, x_n)$, auxiliary input z, and randomness $\mathbf{r} = (r_f, r_{\mathcal{Z}}, r_{\mathcal{S}})$. Let $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}}(\mathbf{x}, z)$ denote the probability distribution of $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}}(\mathbf{x}, z, \mathbf{r})$ where $\mathbf{r}$ is uniformly chosen.*

Security definition. We can now ready to define adaptive security of protocols.

Definition 2.8 (adaptive security)**.** *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality. A protocol π adaptively t-securely computes f with perfect security if for every real-world t-limited adversary $\mathcal{A}$ and every environment $\mathcal{Z}$, there exists an ideal-world adversary $\mathcal{S}$, such that*

$$\{\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(\mathbf{x}, z)\}_{(\mathbf{x},z)\in(\{0,1\}^*)^{n+1}} \equiv \{\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}}(\mathbf{x}, z)\}_{(\mathbf{x},z)\in(\{0,1\}^*)^{n+1}} .$$

If $\mathcal{S}$ runs in polynomial time with respect to the running time of $\mathcal{A}$ than we say that π adaptively t-securely computes f with perfect security and efficient simulation.

If $\mathcal{A}$ and $\mathcal{S}$ are semi-honest, we say that π adaptively t-privately computes f with perfect security and efficient simulation.

2.5 Static vs. Adaptive for Perfect Security

Canetti [Can00] proved the intuitive result that adaptive security implies static security; i.e., if a protocol adaptively t-securely computes f according to Definition 2.8 than it also statically t-securely computes f according to Definition 2.5.

Canetti et al. [CDD+04] showed that under a few assumptions on the static simulation, the other direction also holds, i.e., that static security implies adaptive security. Namely, they considered the following:

- The simulation is *black-box* with respect to the adversary, in the sense that the simulator has an oracle access to the adversary and does not rely on its code.
- The simulation is *straight-line* in the sense that the simulator does not rewind the adversary.
- There exists an explicit *committal round*, denoted CR, in which all of the parties' inputs are committed and cannot be changed (as in [MR91]). Specifically, the simulation can be "split" into two phases. The first is up to and including the committal round: at the conclusion of this phase, $\mathcal{S}$ outputs a partial view of $\mathcal{A}$ up to and including CR together with the inputs to be sent to the trusted party on behalf of the corrupted parties $\{x_i \mid i \in I\}$. After sending these inputs to the trusted party and receiving the output, the second phase is carried out, where the simulator generates a partial view of $\mathcal{A}$ from the round after CR till the end of the protocol.

Theorem 2.9 ([CDD+04]). *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, let t be an integer, and let π be a protocol that statically t-securely computes f with perfect security with a black-box, straight-line simulation with a committal round. Then, π also adaptively t-securely computes f with perfect security.*

We remark, however, that the adaptive simulator in the theorem above might run in super-polynomial time in the running time of the adaptive adversary.

2.6 The Hybrid Model and Sequential Composition

Canetti [Can00] showed that the definitions presented in Sections 2.3 and 2.4 satisfy a natural sequential composition operation. We formalize it below in the adaptive setting, and refer the reader to [Can00] for the static analogue.

The hybrid model. The *hybrid model* is a model that extends the real model with a trusted party that provides ideal computation for specific functionalities. The parties communicate with this trusted party in exactly the same way as in the ideal model described above.

An execution of a protocol π computing a functionality g in the f-hybrid model involves the parties sending normal messages to each other (as in the real model) and in addition, having access to a trusted party computing f. It is essential that the invocations of f are done sequentially, meaning that before an invocation of f begins, the preceding invocation of f must finish. In particular, there is at most one call to f per round, and no other messages are sent during any round in which f is called. Specifically, during a round in which the trusted party is called, the adversary in the hybrid model serves as an ideal-world adversary toward the trusted party. A round in which the trusted party is invoked proceeds as follows.

- The adversary may first adaptively corrupt parties (as in the ideal model) and learn their inputs for the current computation of f. Further, the adversary learns the corrupted party's internal state in the computation of π until this point; this corresponds to the auxiliary information received from the environment.

- The parties hand their inputs to the trusted party. The values for the honest parties are determined by the protocol and the values for the corrupted party are determined by the adversary (in case the adversary is semi-honest the corrupted parties' values are also determined by the protocol). Once received all inputs, the trusted party computes the function f and sends to each party the corresponding output.

- The adversary can corrupt parties as before, and the round completes.

- A corruption of a party *after* the round completes corresponds to the post-execution corruption. In this case, the adversary learns the entire state of the party including its input and output to the trusted party.

We denote by $\text{HYBRID}^{f}_{\pi,\mathcal{A},\mathcal{Z}}(\mathbf{x}, z)$ the random variable consisting of the output of $\mathcal{Z}$ following an execution of π in the f-hybrid model described above (with ideal calls to a trusted party computing f according to the ideal model) on input vector $\mathbf{x}$ and auxiliary input z given to $\mathcal{Z}$.

Definition 2.10 (adaptive security in hybrid model)**.** *Let f and g be n-party functionalities. A protocol π adaptively t-securely computes g with perfect security in the f-hybrid model if for every t-limited adversary $\mathcal{A}$ for the f-hybrid model and every environment $\mathcal{Z}$, there exists an ideal-world adversary $\mathcal{S}$, such that*

$$\left\{\text{HYBRID}^{f}_{\pi,\mathcal{A},\mathcal{Z}}(\mathbf{x}, z)\right\}_{(\mathbf{x},z)\in(\{0,1\}^*)^{n+1}} \equiv \left\{\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}}(\mathbf{x}, z)\right\}_{(\mathbf{x},z)\in(\{0,1\}^*)^{n+1}}.$$

If $\mathcal{S}$ runs in polynomial time with respect to the running time of $\mathcal{A}$ than we say that π adaptively t-securely computes f with perfect security and efficient simulation in the f-hybrid model.

Sequential composition. The sequential composition theorem of Canetti [Can00, Cor. 12] states the following. Let ρ be a protocol that securely computes f in the ideal model. Then, if a protocol π computes g in the f-hybrid model, then the protocol π^{ρ}, that is obtained from π by replacing all ideal calls to the trusted party computing f with the protocol ρ, securely computes g in the real model. We adjust the statement of the theorem to our setting.

Theorem 2.11 ([Can00])**.** *Let f and g be a n-party functionalities, let ρ be a protocol that adaptively t-securely computes f with perfect security and efficient simulation, and let π be a protocol that adaptively t-securely computes g with perfect security and efficient simulation in the f-hybrid model.*

Then, protocol π^{ρ} adaptively t-securely computes g with perfect security and efficient simulation in the real world.

3 Static Security Does Not Imply Adaptive Security

In this section we prove Theorem 1.1. We show a functionality together with a protocol that privately computes it with perfect static security and efficient simulation. Further, we show that if the protocol privately computes the functionality with perfect *adaptive* security and efficient simulation, then one-way permutations do not exist.

We start by defining the functionality and the protocol. Next, in Lemma 3.2 we prove static security and in Lemma 3.5 we prove that adaptive security with efficient simulation cannot be achieved assuming OWP. Combined, this proves Theorem 1.1.

3.1 The Functionality

The n-party functionality f_{sum} is parametrized by a finite binary field $\mathbb{F}_{2^\ell}$ such that $2^\ell > n$. Looking ahead, having a binary field will enable using a one-way permutation $\mathsf{OWP} : \{0,1\}^* \to \{0,1\}^*$ that is applied on a vector of field elements in a clean way. During the proof below, we will denote the field by $\mathbb{F} := \mathbb{F}_{2^\ell}$. The private input of every party consists of n elements over the field $\mathbb{F}$, and the common output is the sum of the first element of each input.

- **Input:** The input of party P_i is a set of n elements $(x_i, (a_{i,1}, \ldots, a_{i,n-1}))$.
- **Output:** The output of every party is $y = \sum_{i=1}^{n} x_i$.

3.2 The Protocol

The n-party protocol π_{sum} is parametrized by the field $\mathbb{F}$ and assumes the existence of a one-way permutation $\mathsf{OWP} : \mathbb{F}^{n-1} \to \mathbb{F}^{n-1}$, i.e., $\mathsf{OWP} : \{0,1\}^{\ell(n-1)} \to \{0,1\}^{\ell(n-1)}$.

Protocol 3.1: Separating Protocol π_{sum}

- **Private input:** The input of party P_i is a set of n elements $(x_i, (a_{i,1}, \ldots, a_{i,n-1})) \in \mathbb{F}^n$.
- **Randomness:** The random tape of party P_i is $\boldsymbol{\rho}_i \leftarrow \mathbb{F}^{n-1}$.
- **The protocol: code for party P_i:**
  1. Compute $(r_{i,1}, \ldots, r_{i,n-1}) = \mathsf{OWP}(\boldsymbol{\rho}_i)$.
  2. For every $j \in [n-1]$ compute the share $\gamma_{i \to j} := r_{i,j} + a_{i,j}$.
  3. Compute the share $\gamma_{i \to n} := x_i - \sum_{j=1}^{n-1} \gamma_{i \to j}$.
  4. Send to every P_j its share $\gamma_{i \to j}$.
  5. Having received all shares $\gamma_{1 \to i}, \ldots, \gamma_{n \to i}$, party P_i locally computes $\beta_i = \sum_{j=1}^{n} \gamma_{j \to i}$ and sends β_i to all other parties..
  6. Having received $\beta_1, \ldots, \beta_n$, party P_i computes $y := \beta_1 + \ldots + \beta_n$.
- **Output:** y.

3.3 Static Security

We start by proving static security of π_{sum}. We note that static security holds even if OWP is simply a permutation that is not necessarily one way and can be inverted efficiently.

Lemma 3.2. *Protocol π_{sum} statically $(n-1)$-privately computes f_{sum} with perfect semi-honest security and efficient simulation.*

Proof. Let $\mathcal{A}$ be a static semi-honest adversary and let $I \subset [n]$ of size $|I| < n$ denote the set of corrupted parties' indices. We construct a simulator $\mathcal{S}$ as follows:

1. The simulator initially receives auxiliary information z and the inputs of the corrupted parties $(x_i, (a_{i,1}, \ldots, a_{i,n-1}))_{i \in I}$. First, $\mathcal{S}$ sends $(x_i, (a_{i,1}, \ldots, a_{i,n-1}))_{i \in I}$ to the trusted party and receives back the output value y. Next, $\mathcal{S}$ invokes $\mathcal{A}$ on z and $(x_i, (a_{i,1}, \ldots, a_{i,n-1}))_{i \in I}$.
2. To simulate the first round, for every $j \notin I$, the simulator chooses a random $\boldsymbol{\rho}_j \leftarrow \mathbb{F}^{n-1}$ and computes $(r_{j,1}, \ldots, r_{j,n-1}) = \mathsf{OWP}(\boldsymbol{\rho}_j)$. Next, $\mathcal{S}$ chooses random elements $(\tilde{x}_j, (\tilde{a}_{j,1}, \ldots, \tilde{a}_{j,n-1}))_{j \notin I}$, under the constraint that

$$\sum_{j \notin I} \tilde{x}_j = y - \sum_{i \in I} x_i,$$

   Next, for every $k \in [n-1]$ the simulator computes the share $\gamma_{j \to k} := r_{j,k} + \tilde{a}_{j,k}$. Then, the simulator computes $\gamma_{j \to n} := \tilde{x}_j - \sum_{k=1}^{n-1} \gamma_{j \to k}$. Finally, for every $j \notin I$ and $i \in I$, the simulator sends $\gamma_{j \to i}$ to $\mathcal{A}$ as the message from an honest P_j to a corrupted P_i, and receives the messages $\gamma_{i \to j}$ from $\mathcal{A}$ as the message from a corrupted P_i to an honest P_j.
3. To simulate the second round, for every $j \notin I$, the simulator computes $\beta_j := \sum_{k=1}^{n} \gamma_{k \to j}$ and sends β_j as the message from P_j. Next, $\mathcal{S}$ receives the message β_i from $\mathcal{A}$ on behalf of every corrupted P_i.
4. Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs, and halts.

Note that by construction, the simulator $\mathcal{S}$ runs in polynomial time in the running time of $\mathcal{A}$.

Claim 3.3. $\{\text{REAL}_{\pi_{\mathsf{sum}},I,\mathcal{A}}(\mathbf{x},z)\}_{(\mathbf{x},z)\in(\{0,1\}^*)^{n+1}} \equiv \{\text{IDEAL}_{f_{\mathsf{sum}},I,\mathcal{S}}(\mathbf{x},z)\}_{(\mathbf{x},z)\in(\{0,1\}^*)^{n+1}}$.

Proof. Note that the only difference between the real execution of the protocol and the simulated execution in the ideal world is the construction of the first elements of each honest party:

- In the real protocol, these elements are part of the original inputs of the honest parties $(x_j,(a_{j,1},\ldots,a_{j,n-1}))_{j\notin I}$. In this case it holds that the output is $y=\sum_{i=1}^{n}x_i$.
- In the simulated execution, these elements are part of the elements $(\tilde{x}_j,(\tilde{a}_{j,1},\ldots,\tilde{a}_{j,n-1}))_{j\notin I}$. These elements are generated under the constraint that $\sum_{j\notin I}\tilde{x}_j = y-\sum_{i\in I}x_i$, where y is computed by the trusted party to be $\sum_{i=1}^{n}x_i$. It follows that

$$\sum_{j\notin I}\tilde{x}_j=\sum_{j\notin I}x_j.$$

The claim now follows by the perfect privacy of the secret sharing. □

This concludes the proof of Lemma 3.2. □

3.4 Inefficient Adaptive Security

By the construction of the static simulator it is clear that the simulation is straight-line and black-box; further, since we consider a semi-honest adversary that in particular does not change the corrupted parties' inputs, "round zero" (the beginning of the protocol) can be set as the committal round. Therefore, we can use Theorem 2.9 to derive the following corollary.

Corollary 3.4. *Protocol π_{sum} adaptively $(n-1)$-privately computes f_{sum} with perfect semi-honest security.*

3.5 No Efficient Adaptive Simulator

We proceed to prove that an efficient adaptive simulator can be used to invert the one-way permutation.

Lemma 3.5. *Assume that* OWP *is a one-way permutation. Then, Protocol π_{sum} does* not *adaptively $(n-1)$-privately computes f_{sum} with perfect semi-honest security and efficient simulation.*

Proof. Let λ denote the security parameter, i.e., the one-way permutation is defined as $\mathsf{OWP}:\{0,1\}^{\lambda}\to\{0,1\}^{\lambda}$. For simplicity, assume that $\lambda=(n-1)\log(n+1)$ for some n such that $n+1$ is a power of 2 (this holds without loss of generality, since the security of the OWP holds for all sufficiently large λ's). We consider the n-party functionality f_{sum} that is defined with respect to the field $\mathbb{F}=\mathbb{F}_{2^\ell}$ where $\ell=\log(n+1)$; then, indeed, $|\mathbb{F}|>n$ as required and OWP induces a permutation over $\mathbb{F}^{n-1}$.

Defining the adversary and the environment. Consider the following adaptive, semi-honest, $(n-1)$-limited adversary $\mathcal{A}$ and the environment $\mathcal{Z}$ for π_{sum}.

1. The environment does not send any auxiliary information to the adversary at the beginning.
2. The adversary $\mathcal{A}$ starts by corrupting the parties $P_1,\ldots,P_{n-2}$ and learns their inputs $(x_1,(a_{1,1},\ldots,a_{1,n-1})),\ldots,(x_{n-2},(a_{n-2,1},\ldots,a_{n-2,n-1}))$ and random tapes $(\boldsymbol{\rho}_1,\ldots,\boldsymbol{\rho}_{n-2})$. The environment does not send any auxiliary information to the adversary for these corruptions.

3. Next, $\mathcal{A}$ receives first-round messages $\gamma_{n-1\to 1},\ldots,\gamma_{n-1\to n-2}$ from P_{n-1} and $\gamma_{n\to 1},\ldots,\gamma_{n\to n-2}$ from P_n.
4. The adversary completes the execution honestly and outputs its view.
5. The environment corrupts P_n in the PEC phase and learns (amongst other things) its input $(x_n,(a_{n,1},\ldots,a_{n,n-1}))$, randomness $\boldsymbol{\rho}_n$, and the values $(r_{n,1},\ldots,r_{n,n-1})$ computed in Step 1 of π_{sum}.
6. The environment checks whether $(r_{n,1},\ldots,r_{n,n-1}) = \mathsf{OWP}(\boldsymbol{\rho}_n)$; if so it outputs real and otherwise ideal.

The corresponding adaptive simulator. By the assumed security of π_{sum}, there exists an efficient adaptive simulator $\mathcal{S}$ for $\mathcal{A}$ and $\mathcal{Z}$. Note that by construction, the adversary $\mathcal{A}$ runs in polynomial time with respect to the parameter λ; hence, $\mathcal{S}$ is PPT with respect to λ. We will use $\mathcal{S}$ to construct a PPT inverter $\mathcal{D}$ for the OWP.

Constructing the inverter from the simulator. The inverter $\mathcal{D}$ receives as input the challenge $y^*\in\{0,1\}^\lambda$. We will consider y^* as an element of $\mathbb{F}^{n-1}$, i.e., $y^*=(r_1^*,\ldots,r_{n-1}^*)\in\mathbb{F}^{n-1}$. The inverter $\mathcal{D}$ proceeds as follows:

1. $\mathcal{D}$ invokes the simulator $\mathcal{S}$ and emulates the ideal computation of f_{sum} toward $\mathcal{S}$; initially, $\mathcal{D}$ sends nothing as the auxiliary information to $\mathcal{S}$.
2. When $\mathcal{S}$ corrupts the parties $P_1,\ldots,P_{n-2}$ in the emulated ideal computation, $\mathcal{D}$ chooses arbitrary elements $(x_1,(a_{1,1},\ldots,a_{1,n-1})),\ldots,(x_{n-2},(a_{n-2,1},\ldots,a_{n-2,n-1}))$ over $\mathbb{F}$ and hands $(x_i,(a_{i,1},\ldots,a_{i,n-1}))$ to $\mathcal{S}$ as the input value of P_i; $\mathcal{D}$ sends nothing as the auxiliary information to $\mathcal{S}$.
3. When $\mathcal{S}$ sends inputs to the trusted party on behalf of the corrupted parties, $\mathcal{D}$ responds with an arbitrary output value $y\in\mathbb{F}$.
4. Once $\mathcal{S}$ generates its output, containing the view of $\mathcal{A}$, the inverter $\mathcal{D}$ extracts the first-round messages that each corrupted party received from the honest party P_n, denoted $\gamma_{n\to 1},\ldots,\gamma_{n\to n-2}$.
5. $\mathcal{D}$ samples two field elements x_n and $a_{n,n-1}$ and for every $i\in[n-2]$ computes $a_{n,i}=\gamma_{n\to i}-r_i^*$.
6. $\mathcal{D}$ sends a "corrupt P_n" request to $\mathcal{S}$ during the PEC phase. When $\mathcal{S}$ corrupts P_n in the emulated ideal computation, $\mathcal{D}$ sets the input of P_n to be $(x_n,(a_{n,1},\ldots,a_{n,n-1}))$. Next, $\mathcal{S}$ responds with the view of P_n, containing the content of its random tape $\boldsymbol{\rho}_n$.
7. $\mathcal{D}$ outputs $\boldsymbol{\rho}_n$.

Efficiently inverting the OWP. First, notice that by construction, the running time of the inverter $\mathcal{D}$ is polynomial with respect to its input y^* and to the running time of $\mathcal{S}$; therefore, $\mathcal{D}$ is PPT with respect to the parameter λ. We proceed to show that the success probability of $\mathcal{D}$ in inverting a random challenge $y^*\leftarrow\{0,1\}^\lambda$ is $1/|\mathbb{F}|$.

Claim 3.6. $\Pr_{y^*\leftarrow\mathbb{F}^{n-1}}\left[\mathsf{OWP}(\mathcal{D}(y^*))=y^*\right]=1/|\mathbb{F}|$.

Proof. In our proof, we do not assume any specific *behavior* of the adaptive simulator $\mathcal{S}$ (i.e., we do not know *how* the simulator generates its output messages). However, based on the assumed perfect security of the protocol, the adaptive simulator $\mathcal{S}$ operates according to the following interface:

1. $\mathcal{S}$ begins by corrupting the parties $P_1,\ldots,P_{n-2}$ and learning their corresponding inputs $(x_1,(a_{1,1},\ldots,a_{1,n-1})),\ldots,(x_{n-2},(a_{n-2,1},\ldots,a_{n-2,n-1}))$.

2. $\mathcal{S}$ sends the inputs $(x_1, (a_{1,1}, \ldots, a_{1,n-1})), \ldots, (x_{n-2}, (a_{n-2,1}, \ldots, a_{n-2,n-1}))$ to the trusted party and obtains the output value y.
3. $\mathcal{S}$ generates the simulated view of the adversary, which in particular contains the messages $\gamma_{n\to 1}, \ldots, \gamma_{n\to n-2}$ from P_n to parties $P_1, \ldots, P_{n-2}$.
4. $\mathcal{S}$ corrupts party P_n and learns its input $(x_n, (a_{n,1}, \ldots, a_{n,n-1}))$.
5. $\mathcal{S}$ outputs the view of P_n, including its random tape $\boldsymbol{\rho}_n$ and values $(r_{n,1}, \ldots, r_{n,n-1}) = \mathsf{OWP}(\boldsymbol{\rho}_n)$, such that $\gamma_{n\to i} = a_{n,i} + r_{n,i}$ for $i \in [n-1]$.

Recall that in Step 5 of the construction of the inverter, $\mathcal{D}$ computes $a_{n,i} := \gamma_{n\to i} - r_i^*$ for every $i \in [n-2]$. Therefore, for every $i \in [n-2]$ it holds that $r_{n,i} = r_i^*$. It follows that the inverter succeeds in inverting y^* if and only if $r_{n,n-1} = r_{n-1}^*$. Since party P_{n-1} remains honest throughout the simulation, the message $\gamma_{n\to n-1}$ remains unspecified when $\mathcal{S}$ outputs the view of P_n, including $(r_{n,1}, \ldots, r_{n,n-1}) = \mathsf{OWP}(\boldsymbol{\rho}_n)$. Hence, the support of $\mathcal{S}$ for the random tape $\boldsymbol{\rho}_n$ includes all possibilities of $r_{n,n-1}$, which is of size $|\mathbb{F}|$. Since r_{n-1}^* is uniformly distributed in $\mathbb{F}$ and $r_{n,n-1}$ is computed by $\mathcal{S}$ independently of r_{n-1}^*, we conclude that $r_{n,n-1} = r_{n-1}^*$ with probability $1/|\mathbb{F}|$. □

Finally, note that $y^* \in \mathbb{F}^{n-1}$, i.e., $y^* \in \{0,1\}^{(n-1)\log(n+1)} = \{0,1\}^{\lambda}$. However, the inverting probability is $1/|\mathbb{F}|$, which is non-negligible in λ. We conclude that $\mathcal{D}$ is PPT in λ and succeeds in inverting OWP with non-negligible probability. This contradicts the assumption that OWP is a one-way permutation. □

4 Boosting Static Security to Adaptive Security

Damgård and Nielsen [DN14] showed that any prefect UC-secure protocol against static adversaries that satisfies several additional properties is also perfect UC-secure with efficient simulation against *adaptive* adversaries. Due to the inherent technical complexity of the UC framework, the requirements from the statically secure protocol in [DN14] simultaneously need to address multiple input/output phases and concurrent security. In this section we distil the transformation to only require simpler, standalone requirements from the underlying statically secure protocol. UC security can be automatically derived using the transformation of [KLR10].

In Section 4.1 we define *compatible* static simulation, in Section 4.2 show how this notion yields adaptive security with efficient simulation, and in Section 4.3 simplify the requirements for the semi-honest setting.

4.1 Compatible Static Simulation

We start by describing the requirements from the static simulator in the standalone model and the Patch algorithm to be compatible with the transformation of [DN14]. We emphasize that these requirements are in the standalone model as opposed to the UC framework used in [DN14]. Later, in Section 4.2, we will show how to efficiently achieve adaptive security.

The static simulator. The basic requirements from the static simulator $\mathcal{S}_{\mathsf{static}}$ are to be black-box and straight-line, and that the simulation admits a committal round CR (as defined in Section 2.5). Further, given a static adversary $\mathcal{A}_{\mathsf{static}}$ and a set of corrupted parties $I \subseteq [n]$, the simulator $\mathcal{S}_{\mathsf{static}}$ is assumed to invoke a single instance of $\mathcal{A}_{\mathsf{static}}$ and interact with $\mathcal{A}_{\mathsf{static}}$ in a round-by-round fashion. In this case we say that this is a $\mathsf{round\text{-}by\text{-}round\ simulation}$.

Below we explicitly describe the structure of a round-by-round simulation, focusing on the *state* of the simulator and the *interface* of the simulator toward every corrupted party:

- The state of the simulator, denoted state, consists of its input, all messages received by the simulator (from the adversary and from the trusted party) and its random coins. The state fully determines the behavior of the simulator.
- For every corrupted party P_i, the interface of the simulator toward P_i, denoted $\mathsf{interface}^i$, consists of all messages sent to/recieved from P_i during the simulation, as well as the input value sent by the simulator to the trusted party for P_i and the corresponding output.

Looking ahead, we will use these values when boosting static security to adaptive security, observing that the state of the simulator fully determines the messages generated by the simulator, and the interface can be viewed as values "committed" by the simulator.

- Initially, the internal state of $\mathcal{S}_{\mathsf{static}}$ consists of the set of corrupted parties $I \subseteq [n]$, their original input values $(x_i)_{i\in I}$, the auxiliary information z, and random coins $r_{\mathcal{S}}$. $\mathcal{S}_{\mathsf{static}}$ invokes $\mathcal{A}_{\mathsf{static}}$ on I, $(x_i)_{i\in I}$, z, and on random coins $r_{\mathcal{A}}$ (that are derived from $r_{\mathcal{S}}$).
  For every corrupted $\tilde{P}_i$, initialize the interface $\mathsf{interface}^i$ to be the empty string.
- To simulate rounds $1,\ldots,\mathsf{CR}$, the simulator sets up virtual honest parties, $\tilde{P}_1,\ldots,\tilde{P}_n$, and runs $\{\tilde{P}_h\}_{h\notin I}$ honestly on arbitrary input values (e.g., on zeros). That is, in round k, $\mathcal{A}_{\mathsf{static}}$ selects an honest party $\tilde{P}_h$ and $\mathcal{S}_{\mathsf{static}}$ sends to $\mathcal{A}_{\mathsf{static}}$ a message on behalf of $\tilde{P}_h$ to each corrupted party $\tilde{P}_i$, denoted by $m^k_{h\to i}$. After $\mathcal{A}_{\mathsf{static}}$ selected all honest parties, $\mathcal{S}_{\mathsf{static}}$ receives from $\mathcal{A}_{\mathsf{static}}$ a message from each corrupted party $\tilde{P}_i$ to each honest party $\tilde{P}_h$, denoted $m^k_{i\to h}$, and the next round begins (note that this models a rushing adversary). For every corrupted $\tilde{P}_i$, all messages $m^k_{h\to i}$ and $m^k_{i\to h}$, for $h\notin I$, are concatenated to $\mathsf{interface}^i$.
  In case the protocol is defined in an f-hybrid model, for some functionality f, the simulator computes the messages that the virtual honest parties send to f, receives messages from $\mathcal{A}_{\mathsf{static}}$ on behalf of every corrupted party, and based on those produces an output for each party and simulates sending the output to each party. For every corrupted $\tilde{P}_i$, the message sent as output from f is concatenated to $\mathsf{interface}^i$.
  Throughout this part, every message received from $\mathcal{A}_{\mathsf{static}}$ is concatenated to the state of $\mathcal{S}_{\mathsf{static}}$.
- Once round CR is completed in the simulation, $\mathcal{S}_{\mathsf{static}}$ extracts an input value x'_i for every corrupted party $\tilde{P}_i$ (recall that this can be done by the definition of a committal round) and sends it to the trusted party; next, $\mathcal{S}_{\mathsf{static}}$ receives the output value y_i for each corrupted party. Those values $(x'_i,y_i)_{i\in I}$ are concatenated to the state of $\mathcal{S}_{\mathsf{static}}$, and for every corrupted $\tilde{P}_i$, the values x'_i and y_i are concatenated to $\mathsf{interface}^i$.
- Based on the output values for all corrupted parties and the state of the simulation so far, $\mathcal{S}_{\mathsf{static}}$ continues simulating rounds $\mathsf{CR}+1$ till the completion of the protocol, as before.
- Finally, $\mathcal{S}_{\mathsf{static}}$ outputs whatever $\mathcal{A}_{\mathsf{static}}$ outputs.

We emphasize that the state of $\mathcal{S}_{\mathsf{static}}$ is well-defined and updates in a monotonic manner after every message it receives from $\mathcal{A}_{\mathsf{static}}$ or from the trusted party. Denote by

$$\mathsf{state}^{\rho,\mathcal{H}}(\mathcal{S}_{\mathsf{static}},I,\mathbf{x},z,r_{\mathcal{S}},r_f)$$

the state of $\mathcal{S}_{\mathsf{static}}$ in the ρ^{th} round after a subset of honest parties, denoted by $\mathcal{H}$, sent messages (i.e., $\mathcal{H}\subseteq[n]\setminus I$), when simulating $\mathcal{A}_{\mathsf{static}}$ on corrupted set I, inputs $\mathbf{x}=(x_1,\ldots,x_n)$, auxiliary input z, random coins $r_{\mathcal{S}}$ for $\mathcal{S}_{\mathsf{static}}$ and r_f for the trusted party.

The patch algorithm. Given an adaptive adversary $\mathcal{A}_{\mathsf{adaptive}}$, the idea of [DN14] is to construct an adaptive simulator $\mathcal{S}_{\mathsf{adaptive}}$ as follows. $\mathcal{S}_{\mathsf{adaptive}}$ uses the static simulator $\mathcal{S}_{\mathsf{static}}$ for simulating

the (so-far) honest parties towards $\mathcal{A}_{\mathsf{adaptive}}$ in a round-by-round fashion. Say that the corrupted parties are indexed by the set I (initially, this set is empty) and $\mathcal{A}_{\mathsf{adaptive}}$ corrupts party $\tilde{P}_j$ with $j \notin I$. Denote the round number by ρ and the set of honest parties who sent messages so far in round ρ by $\mathcal{H}$. The (pre-corruption) state of the simulator at this point is denoted by

$$\mathsf{state}_{\mathsf{pre}} := \mathsf{state}^{\rho,\mathcal{H}}(\mathcal{S}_{\mathsf{static}}, I, \mathbf{x}, z, r_{\mathcal{S}}, r_f).$$

At this point $\mathcal{S}_{\mathsf{adaptive}}$ corrupts the dummy party P_j in the ideal computation and learns its input x_j, its output y_j (the output tape of P_j may still be empty at this point), and the auxiliary information z_j received from the environment.

It is required that there exists a PPT algorithm Patch that on input $\mathsf{state}_{\mathsf{pre}}$, j, x_j, y_j, and z_j, outputs a new (post-corruption) state $\mathsf{state}_{\mathsf{post}}$ and random coins r_j such that the following is satisfied:

- The new state $\mathsf{state}_{\mathsf{post}}$ corresponds to a static simulation of $\mathcal{S}_{\mathsf{static}}$ on corrupted set $I \cup \{j\}$ on the same values, i.e., $\mathsf{state}_{\mathsf{post}} = \mathsf{state}^{\rho,\mathcal{H}}(\mathcal{S}_{\mathsf{static}}, I \cup \{j\}, \mathbf{x}, z, r'_{\mathcal{S}}, r_f)$, for some $r'_{\mathcal{S}}$.
- For every $i \in I$, denote by $\mathsf{interface}^i_{\mathsf{pre}}$ the interface of $\tilde{P}_i$ in the simulation of $\mathcal{S}_{\mathsf{static}}$ on state $\mathsf{state}_{\mathsf{pre}}$ (with corrupted parties I and honest parties $[n] \setminus I$), and by $\mathsf{interface}^i_{\mathsf{post}}$ the interface of $\tilde{P}_i$ in the simulation of $\mathcal{S}_{\mathsf{static}}$ on state $\mathsf{state}_{\mathsf{post}}$ (with corrupted parties $I \cup \{j\}$ and honest parties $[n] \setminus (I \cup \{j\})$). Denote by $\mathsf{interface}^{i\setminus j}_{\mathsf{pre}}$ the values in $\mathsf{interface}^i_{\mathsf{pre}}$ after removing all messages sent/received from $\tilde{P}_j$. Then, it is required that $\mathsf{interface}^{i\setminus j}_{\mathsf{pre}} = \mathsf{interface}^i_{\mathsf{post}}$.
  Stated differently, messages from honest parties to corrupted parties (other than $\tilde{P}_j$) remain the same.
- For every round $k \in [\rho]$ and every $i \in I$, let $m^k_{i\to j}$ and $m^k_{j\to i}$ be the messages sent between $\tilde{P}_i$ to $\tilde{P}_j$ in round k in the simulation of $\mathcal{S}_{\mathsf{static}}$ on state $\mathsf{state}_{\mathsf{pre}}$ (as set in $\mathsf{interface}^i_{\mathsf{pre}}$); for every $h \notin I \cup \{j\}$, let $m^k_{h\to j}$ and $m^k_{j\to h}$ be the messages sent between $\tilde{P}_h$ to $\tilde{P}_j$ in round k in the simulation of $\mathcal{S}_{\mathsf{static}}$ on state $\mathsf{state}_{\mathsf{post}}$ (as set in $\mathsf{interface}^j_{\mathsf{post}}$). Then, these messages are compatible with an honest execution of $\tilde{P}_j$ on input x_j on random coins r_j.
  Stated differently, the interface of an honest $\tilde{P}_j$ with corrupted parties in the simulation on $\mathsf{state}_{\mathsf{pre}}$ can be explained using random coins r_j and messages from honest parties to (the now corrupted) $\tilde{P}_j$ in the simulation on $\mathsf{state}_{\mathsf{pre}}$, as if $\tilde{P}_j$ played honestly until this point in the simulation.

Definition 4.1 (Compatible simulation)**.** *Let f be an n-party functionality and let π be a protocol that statically t-securely computes f with perfect security. We say that the simulation is efficient and compatible if for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that:*

- *The running time of $\mathcal{S}$ is polynomial in the running time of $\mathcal{A}$ (i.e., the simulation is efficient).*
- *The simulation is black-box, straight-line, and admits a committal round* CR*.*
- *The simulation is round-by-round.*
- *There exists a PPT* Patch *algorithm as described above.*

4.2 Static Security implies Adaptive Security

The following theorem is a simplified version of [DN14, Thm. 1]. The proof follows the same arguments from [DN14], with the difference of working in the simpler, standalone model as opposed to the UC framework.

Theorem 4.2. *Let f be an n-party functionality and let π be an n-party protocol that statically t-securely computes f with perfect security and with efficient and compatible simulation. Then, π adaptively t-securely computes f with perfect security and efficient simulation.*

Proof. Let $\mathcal{A}_{\mathsf{adaptive}}$ be an adversary and let $\mathcal{S}_{\mathsf{static}}$ be an efficient static simulator that is guaranteed to exist from the efficient static security of π. We will construct an efficient adaptive simulator $\mathcal{S}_{\mathsf{adaptive}}$ such that no environment can distinguish between a real execution of π with $\mathcal{A}_{\mathsf{adaptive}}$ and an ideal execution of f with $\mathcal{S}_{\mathsf{adaptive}}$. Let $\mathcal{Z}$ be an environment.

1. Initially, $\mathcal{S}_{\mathsf{adaptive}}$ receives as input auxiliary information z from $\mathcal{Z}$. The simulator initializes the dynamic set of corrupted parties $I = \emptyset$, samples random coins $r_{\mathcal{S}}$ for $\mathcal{S}_{\mathsf{static}}$, sets the dynamic state to $\mathsf{state} := (I, z, r_{\mathcal{S}})$, invokes $\mathcal{S}_{\mathsf{static}}$ on state, and invokes $\mathcal{A}_{\mathsf{adaptive}}$ on z.
2. To simulate rounds $1, \ldots, \mathsf{CR}$, in the protocol, let state denote the current state of the static simulator and let I denote the current set of corrupted parties. When $\mathcal{A}_{\mathsf{adaptive}}$ activates an honest party for this round, $\mathcal{S}_{\mathsf{adaptive}}$ forwards the message to $\mathcal{S}_{\mathsf{static}}$ and once receiving from $\mathcal{S}_{\mathsf{static}}$ the messages to be sent for this honest party, $\mathcal{S}_{\mathsf{adaptive}}$ forwards them to $\mathcal{A}_{\mathsf{adaptive}}$. Once $\mathcal{A}_{\mathsf{adaptive}}$ finished activating all honest parties, it sends messages on behalf of corrupted parties, and $\mathcal{S}_{\mathsf{adaptive}}$ forwards them to $\mathcal{S}_{\mathsf{static}}$.
   Every message received from $\mathcal{A}_{\mathsf{adaptive}}$ is appended to state.
3. In round CR, $\mathcal{S}_{\mathsf{adaptive}}$ receives from $\mathcal{S}_{\mathsf{static}}$ input values $(x'_i)_{i \in I}$ and forwards them to the trusted party. Once receiving output values $(y_i)_{i \in I}$ from the trusted party, $\mathcal{S}_{\mathsf{adaptive}}$ forwards them to $\mathcal{S}_{\mathsf{static}}$ and appends them to the state.
4. To simulate rounds $\mathsf{CR}+1$ till the end, $\mathcal{S}_{\mathsf{adaptive}}$ proceeds as in Step 2.
5. Upon a corruption request by $\mathcal{A}_{\mathsf{adaptive}}$ of party P_j, the simulator $\mathcal{S}_{\mathsf{adaptive}}$ corrupts the dummy party P_j and learns its original input x_j, and possibly its output y_j. In addition, $\mathcal{Z}$ sends auxiliary information z_j for P_j. Next, denote by $\mathsf{state}_{\mathsf{pre}}$ the current (pre-corruption) state of the static simulation. $\mathcal{S}_{\mathsf{adaptive}}$ calls $\mathsf{Patch}(\mathsf{state}_{\mathsf{pre}}, j, x_j, y_j, z_j)$ to receive the post-corruption state $\mathsf{state}_{\mathsf{post}}$ and random coins r_j for P_j.
   Next, $\mathcal{S}_{\mathsf{adaptive}}$ sets the state of the static simulation to be $\mathsf{state}_{\mathsf{post}}$ and responds to $\mathcal{A}_{\mathsf{adaptive}}$ with the internal state of P_j set as follows:
   - The content of the input tape is x_j.
   - The content of the output tape is y_j (possibly empty, if the corruption is before round CR).
   - The content of the random tape is r_j.
   - The content of the incoming communication tapes from every corrupted party P_i with $i \in I$ is set according to the interface $\mathsf{interface}^i_{\mathsf{pre}}$ that is derived from $\mathsf{state}_{\mathsf{pre}}$. That is, as the messages sent/received on behalf of P_j to P_i during the simulation thus far.
   - The content of the incoming communication tapes from every honest party P_h with $h \notin I \cup \{j\}$ is set according to the interface $\mathsf{interface}^j_{\mathsf{post}}$ that is derived from $\mathsf{state}_{\mathsf{post}}$.
   - The content of the incoming communication tapes from calls to ideal functionalities are set according to the interface $\mathsf{interface}^j_{\mathsf{post}}$ that is derived from $\mathsf{state}_{\mathsf{post}}$.

Decomposing the adaptive adversary. We proceed to prove that the output of the ideal computation is identically distributed as the output of the real computation. To see this, we "decompose" the t-limited adaptive adversary $\mathcal{A}_{\mathsf{adaptive}}$ into $(t+1)$ static adversaries. Fix a sequence $(k_1, \rho_1), \ldots, (k_t, \rho_t)$ that corresponds to an ordered corruption pattern $\mathcal{A}_{\mathsf{adaptive}}$, where party P_{k_i} is the i^{th} corrupted party in round ρ_i. Denote by $\mathcal{E}_i$ the event that the first i corruptions are according to the sequence $(k_1, \rho_1), \ldots, (k_t, \rho_t)$. If $\Pr[\mathcal{E}_i > 0]$, then conditioned on $\mathcal{E}_i$ we can define the following static adversaries from $\mathcal{A}_{\mathsf{adaptive}}$.

- The first adversary $\mathcal{A}^0_{\mathsf{static}}$ does not corrupt any party.

- For every $i \in [t]$, the i^{th} adversary $\mathcal{A}^i_{\mathsf{static}}$ statically corrupts parties $\{P_{k_1}, \ldots, P_{k_i}\}$. The adversary $\mathcal{A}^i_{\mathsf{static}}$ controls $\{P_{k_1}, \ldots, P_{k_{i-1}}\}$ just like $\mathcal{A}^{i-1}_{\mathsf{static}}$ till round ρ_i. Further, $\mathcal{A}^i_{\mathsf{static}}$ plays with P_{k_i} honestly until round ρ_i. $\mathcal{A}^i_{\mathsf{static}}$ continues to play according to $\mathcal{A}_{\mathsf{adaptive}}$ till round ρ_{i+1} (where the $(i+1)^{\text{th}}$ corruption occurs).

We define the corresponding static simulators as follows:

- The first simulator $\mathcal{S}^0_{\mathsf{static}}$ is exactly $\mathcal{S}_{\mathsf{static}}$ running on $I = \emptyset$.
- For every $i \in [t]$, the i^{th} simulator $\mathcal{S}^i_{\mathsf{static}}$ statically corrupts parties $\{P_{k_1}, \ldots, P_{k_i}\}$. The simulator $\mathcal{S}^i_{\mathsf{static}}$ proceeds just like $\mathcal{A}^{i-1}_{\mathsf{static}}$ till round ρ_i. If $\rho_i \geq \mathsf{CR}$, $\mathcal{S}^i_{\mathsf{static}}$ uses the original input value x_{k_i} for P_{k_i}. Next, in round ρ_i, $\mathcal{S}^i_{\mathsf{static}}$ calls Patch on its current state, denoted $\mathsf{state}_{\mathsf{pre}}$, on i, on x_i, on y_i (which could be empty), and on the auxiliary information z together with the view of P_{k_i} in the simulation thus far. Upon receiving $\mathsf{state}_{\mathsf{post}}$ as the output from Patch, the simulator $\mathcal{S}^i_{\mathsf{static}}$ continues its simulation on state $\mathsf{state}_{\mathsf{post}}$.

Given a sequence $(k_1, \rho_1), \ldots, (k_t, \rho_t)$ we define two games as follows.

- The first game, denoted $\text{REAL}^{i,\rho}_{\pi,\mathcal{A},I}(\mathbf{x}, z)$ outputs the view of $\mathcal{A}$ until round ρ when the corrupted parties' indices are $I = \{k_1, \ldots, k_i\}$, together with the output of the honest parties $[n] \setminus I$. In case the protocol did not complete yet, these values are bottom; otherwise, these values are computed as in $\text{REAL}_{\pi,\mathcal{A},I}(\mathbf{x}, z)$.
- The second game, denoted $\text{IDEAL}^{i,\rho}_{f,\mathcal{S},I}(\mathbf{x}, z)$ outputs a simulated view generated by a compatible simulator $\mathcal{S}$ until round ρ when the corrupted parties are $I = \{k_1, \ldots, k_i\}$, together with the output of the honest parties $[n] \setminus I$. In case the honest parties did not receive output yet, these values are bottom; otherwise, these values are computed as in $\text{IDEAL}_{f,\mathcal{S},I}(\mathbf{x}, z)$.

Claim 4.3. *For every sequence $(k_1, \rho_1), \ldots, (k_t, \rho_t)$ and every $i \in \{0, \ldots, t\}$, if $\Pr[\mathcal{E}_i] > 0$ then conditioned on $\mathcal{E}_i$ it holds that for $I = \{k_1, \ldots, k_i\}$:*

$$\left\{\text{REAL}^{i,\rho_i}_{\pi,\mathcal{A}^i_{\mathsf{static}},I}(\mathbf{x}, z)\right\}_{(\mathbf{x},z)\in(\{0,1\}^*)^{n+1}} \equiv \left\{\text{IDEAL}^{i,\rho_i}_{f,\mathcal{S}^i_{\mathsf{static}},I}(\mathbf{x}, z)\right\}_{(\mathbf{x},z)\in(\{0,1\}^*)^{n+1}}.$$

Proof. We will prove the claim by induction on i. The base case is immediate since no party is corrupted. Let $i \in [t]$ and assume the claim holds for $i-1$; we will prove the claim for i. Conditioned on the event $\mathcal{E}_i$, it holds by the induction hypothesis that the view simulated by $\mathcal{S}_{\mathsf{static}}$ on corrupted parties $\{P_{k_1}, \ldots, P_{k_{i-1}}\}$ until round ρ_{i-1} is identically distributed as the view of $\mathcal{A}^{i-1}_{\mathsf{static}}$ until round ρ_{i-1}.

Now, $\mathcal{A}^i_{\mathsf{static}}$ proceeds as $\mathcal{A}^{i-1}_{\mathsf{static}}$ where party P_{k_i} is also corrupted but plays honestly until round ρ_i. In round ρ_i, the simulator $\mathcal{S}^i_{\mathsf{static}}$ calls Patch on its current state, and on the simulated view of party P_i. By the requirement on the Patch algorithm, the new state $\mathsf{state}_{\mathsf{post}}$ corresponds to the internal state of $\mathcal{S}_{\mathsf{static}}$ till round ρ_i when simulating the adversary $\mathcal{A}^i_{\mathsf{static}}$. By the perfect security of $\mathcal{S}_{\mathsf{static}}$, the simulation can resume and perfectly simulate the view of $\mathcal{A}^i_{\mathsf{static}}$ till round ρ_{i+1}. □

Suppose that there are $\ell \leq t$ corruptions till the completion of the protocol, and potentially additional $t - \ell$ corruptions requests in the PEC stage. It follows from Claim 4.3 that the only difference between $\mathcal{S}^\ell_{\mathsf{static}}$ and $\mathcal{S}_{\mathsf{adaptive}}$ is in answering those corruption requests. Again, relying on the assumption of the Patch algorithm, $\mathcal{S}_{\mathsf{adaptive}}$ can respond with the internal view of a party in the PEC stage in a way that is compatible with the simulation thus far. We therefore conclude that

$$\left\{\text{REAL}_{\pi,\mathcal{A}_{\mathsf{adaptive}},\mathcal{Z}}(\mathbf{x}, z)\right\}_{(\mathbf{x},z)\in(\{0,1\}^*)^{n+1}} \equiv \left\{\text{IDEAL}^{i,\rho_i}_{f,\mathcal{S}_{\mathsf{adaptive}},\mathcal{Z}}(\mathbf{x}, z)\right\}_{(\mathbf{x},z)\in(\{0,1\}^*)^{n+1}}.$$

□

Using [KLR10] we derive the following corollary.

Corollary 4.4. *Let f be an n-party functionality and let π be an n-party protocol that statically t-securely computes f with perfect security and with efficient and compatible simulation. Then, π UC-realizes f with perfect adaptive security and efficient simulation against t-adversaries.*

4.3 A Simplification for Semi-Honest Adversaries

In case the adversary is semi-honest and all corrupted parties follow the protocol, the requirements from the Patch algorithm are much simpler. In this case, the simulator knows exactly how each corrupted party will behave, and once sampling their random coins the simulator can simply run the code of each corrupted party honestly.

In this case, the state of the simulation is defined by the random coins sampled for each party, the random coins sampled for each call to an ideal functionality, and the input/output values of the corrupted parties. The entire simulated execution is set according to those values.

The Patch algorithm receives the state of a simulated execution with a corrupted set I, and the input x_i and output y_i of the newly corrupted party, and should produce a new state that explains the simulation when the corrupted set is $I \cup \{i\}$. Specifically, the random coins of all corrupted parties and all communication involving corrupted parties must remain the same; however, the Patch algorithm may adjust the random coins of honest parties and of P_i, as well as honest-to-honest communication and honest-to-P_i communication, to retain compatibility with x_i and y_i.

In this case it is also sufficient to compare the output produced by $\mathcal{S}_{\mathsf{static}}$ on $\mathsf{state}_{\mathsf{pre}}$ and on $\mathsf{state}_{\mathsf{post}}$ and show that they are identically distributed, as opposed to compare the partial state of the simulation at the point the Patch was invoked. This is because in the semi-honest setting the state fully determines the rest of the simulation.

In Section 5 we will prove the semi-honest security of the BGW protocol using this simplified requirement.

5 Adaptive Security of the BGW Protocol

In this section we prove Theorem 1.3. In Section 5.1 we prove efficient adaptive security of the multiplication protocol and in Section 5.2 we prove efficient adaptive security of the BGW protocol in the f_{mult}-hybrid model. The proof of Theorem 1.3 follows from the composition theorem of Canetti (Theorem 2.11). Throughout this section, we denote by $\mathbb{F}$ a finite field satisfying $|\mathbb{F}| > n$.

5.1 Computing a Multiplication Gate

We start by defining the multiplication functionality f_{mult} that will be used to compute a multiplication gate in the circuit. The functionality receives from each party P_i a pair of points $q_a(\alpha_i)$ and $q_b(\alpha_i)$, where $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ are public. The functionality interpolates these points to recover two degree-t polynomials $q_a(x)$ and $q_b(x)$, randomly samples a degree-t polynomial $q_{ab}(x)$ satisfying $q_{ab}(0) = q_a(0) \cdot q_b(0)$, and returns to each P_i the value $q_{ab}(\alpha_i)$. That is,

$$f_{\mathsf{mult}}\Big((q_a(\alpha_1), q_b(\alpha_1)), \ldots, (q_a(\alpha_n), q_b(\alpha_n))\Big) = \Big(q_{ab}(\alpha_1), \ldots, q_{ab}(\alpha_n)\Big) \ .$$

The protocol. We present here the multiplication protocol of [GRR98]. Each party holds shares β_i and γ_i corresponding to some polynomials $q_a(\alpha_i)$ and $q_b(\alpha_i)$. The goal is to obtain shares on a random degree-t polynomial $q_c(x)$ satisfying $q_c(0) = q_a(0) \cdot q_b(0)$. Towards that end, each party

multiplies its two shares $\beta_i \cdot \gamma_i$, which is essentially a share on the degree-$2t$ polynomial $h(x) := q_a(x) \cdot q_b(x)$. Note that there exists fixed, public coefficients (known as Lagrange coefficients) $\lambda_1, \ldots, \lambda_n$, satisfying $h(0) = \lambda_1 \cdot h(\alpha_1) + \ldots + \lambda_n \cdot h(\alpha_n)$. The protocol proceeds by instructing each party P_i to secret share $h(\alpha_i)$ using some degree-t polynomial $g_i(x)$, and then given $g_1(\alpha_i), \ldots, g_n(\alpha_i)$, output $\delta_i = \sum_{k=1}^{n} \lambda_k \cdot g_k(\alpha_i)$. That is, the parties jointly compute shares on the polynomial $H(x) := \sum_{k=1}^{n} \lambda_k \cdot g_k(x)$, which is a random degree-t polynomial satisfying

$$H(0) = \sum_{k=1}^{n} \lambda_k \cdot g_k(0) = \lambda_1 \cdot h(\alpha_1) + \ldots + \lambda_n \cdot h(\alpha_n) = h(0) = q_a(0) \cdot q_b(0) \ .$$

Protocol 5.1: Realizing f_{mult}

- **Private input:** Each party P_i holds as input $\beta_i, \gamma_i \in \mathbb{F}$.
- **Auxiliary input:** Each party holds $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ and Lagrange coefficients $\lambda_1, \ldots, \lambda_n$.
- **The protocol:**
  1. P_i chooses a random polynomial $g_i(x)$ of degree at most t such that $g_i(0) = \beta_i \cdot \gamma_i$ and sends to each party P_j the point $g_i(\alpha_j)$.
  2. **Output:** Given $g_1(\alpha_i), \ldots, g_n(\alpha_i)$, output $\delta_i = \sum_{k=1}^{n} \lambda_k \cdot g_k(\alpha_i)$.

5.1.1 Static Security

Lemma 5.2. *Let $t < n/2$. Then, Protocol 5.1 statically t-privately computes f_{mult} in the presence of a semi-honest adversary with efficient simulation.*

Proof. We start by describing the static simulator.

The static simulator.

- **Input:** The simulator $\mathcal{S}$ receives the inputs β_i and γ_i for every $i \in I$. Initially, $\mathcal{S}$ sends these inputs to the trusted party computing f_{mult} and receives the output value δ_i for every $i \in I$.
- **The simulation:** The simulation consists of two parts:
  1. **Sample randomness for corrupted parties:**
     For every $i \in I$, sample the random tape r_i uniformly at random. Note that this defines the polynomials $g_i(x)$ that the corrupted P_i uses to hide $\beta_i \cdot \gamma_i$.
  2. **Sample random points for honest parties:**
     For every $j \notin I$ and $i \in I$, choose points $g_j(\alpha_i)$ uniformly and at random in $\mathbb{F}$, under the constraint that for every $i \in I$:

     $$\sum_{k=1}^{n} \lambda_k \cdot g_k(\alpha_i) = \delta_i \ .$$

     Note that this defines at most $|I| \leq t$ points on each polynomial and *does not* uniquely define the polynomial. A possible way to sample all the points is as follows: Set $j^* \notin I$ arbitrarily. For every $j \notin I \cup \{j^*\}$ and $i \in I$, the simulator chooses $g_j(\alpha_i)$ uniformly and independently at random, and then determines:

     $$g_{j^*}(\alpha_i) := \frac{1}{\lambda_{j^*}} \cdot \Big(\delta_i - \sum_{k \in [n] \setminus \{j^*\}} \lambda_k \cdot g_k(\alpha_i)\Big).$$

- **Output:** The simulator outputs the view of the adversary in the simulated execution. This includes r_i for every $i \in I$ as the random tapes of the corrupted parties, and all the messages $(g_j(\alpha_i))_{j \notin I, i \in I}$.

We now show that the joint distribution of the outputs of all parties and the view of the corrupted parties in the real world is identically distributed to the joint distribution of the output of all parties and the output of the simulator in the ideal execution. Towards that end, we show a sequence of hybrid experiments:

1. $\mathsf{Hyb_0}$: This is the real execution. Each P_k starts with input β_k and γ_k. The output of this hybrid is the output values of all parties and the view of the corrupted parties.
2. $\mathsf{Hyb_1}$: In this hybrid execution, we consider a simulator that receives the inputs and outputs of all parties, i.e., receives $\beta_j, \gamma_j, \delta_j$ for $j \notin I$ in addition to the inputs and outputs of the corrupted parties. Then, the simulator chooses polynomials $(g_j(x))_{j \notin I}$ of degree-t uniformly at random under the following constraints:

$$g_j(0) = \beta_j \cdot \gamma_j \qquad \text{and} \qquad \sum_{k=1}^{n} \lambda_k \cdot g_k(\alpha_j) = \delta_j \ .$$

   A possible way to sample such polynomials is as follows: Set $j^* \notin I$ arbitrarily. Then, for every $j \notin I \cup \{j^*\}$ and $i \in I$, the simulator chooses $g_j(x)$ uniformly and independently at random such that $g_j(0) = \beta_j \cdot \gamma_j$. This uniquely determines the polynomial

$$g_{j^*}(x) := \frac{1}{\lambda_{j^*}} \cdot \Big(q_{ab}(x) - \sum_{k \in [n] \setminus \{j^*\}} \lambda_k \cdot g_k(x) \Big).$$

   Given all those polynomials, the simulator outputs $(g_j(\alpha_i))_{j \notin I, i \in I}$ in addition to the random tapes of the corrupted parties. The output of this experiment is the output of all parties as determined by the trusted party, and the output of the modified simulator.
3. $\mathsf{Hyb_2}$: This hybrid is similar to the previous one, except that the modified simulator is not given $\beta_j, \gamma_j, \delta_j$ (for $j \notin I$). Instead, it is only given the $|I|$ shares of the corrupted parties, i.e., $(\beta_i, \gamma_i, \delta_i)_{i \in I}$. It then works as the simulator in $\mathsf{Hyb_1}$, while choosing the constant terms of each polynomial $g_j(x)$ for $j \notin I \cup \{j^*\}$ to be 0; further, $g_{j^*}(x)$ is determined in a similar way to $\mathsf{Hyb_1}$. The output of this experiment is the output of all parties as determined by the trusted party, and the output of the modified simulator.
4. $\mathsf{Hyb_3}$: This is the ideal execution. The output of this hybrid is the output of all parties as determined by the trusted party, and the output of the simulator as described in the proof.

The only difference between $\mathsf{Hyb_0}$ and $\mathsf{Hyb_1}$ is in the choice of the polynomial $g_{j^*}(x)$, and whether the output of all parties is determined by the output of the protocol or by the trusted party. In $\mathsf{Hyb_0}$, we first choose $g_{j^*}(x)$ as a random degree-t polynomial with the constant term $q_a(\alpha_i) \cdot q_b(\alpha_i)$ and then derive $q_{ab}(x) = \sum_{k=1}^{n} \lambda_k \cdot g_k(x)$. Note that

$$q_{ab}(0) = \sum_{k=1}^{n} \lambda_k \cdot g_k(0) = \sum_{k=1}^{n} \lambda_k \cdot q_a(\alpha_i) \cdot q_b(\alpha_i) = q_a(0) \cdot q_b(0),$$

and so $q_{ab}(x)$ is a random degree-t polynomial with constant term $q_a(0) \cdot q_b(0)$. In $\mathsf{Hyb_1}$, the trusted party first chooses $q_{ab}(x)$ as a random degree-t polynomial with $q_a(0) \cdot q_b(0)$ as its constant term, and then after choosing random polynomials $\{g_k(x)\}_{k \neq j^*}$ we determine $g_{j^*}(x)$ such that $q_{ab}(x) = \sum_{k=1}^{n} \lambda_k \cdot g_k(x)$. Those two processes are therefore, equivalent.

The difference between Hyb_1 and Hyb_2 is the following: In Hyb_1 we choose the polynomials $g_j(x)$ for $j \notin I \cup \{j^*\}$ uniformly and independently at random with constant term $\beta_j \cdot \gamma_j$, and in Hyb_2 we choose each such polynomial with constant term 0. In both experiments we determine $g_{j^*}(x)$ deterministically in the same way. In both executions, the view of the adversary consists of at most $|I| \leq t$ points on each such polynomial, which according to Claim 2.2 implies that these two processes have the exact same distribution.

The difference between Hyb_2 and Hyb_3 is that in Hyb_3 the simulator now does not even choose the polynomials $g_j(x)$ and just chooses the points on the polynomials uniformly at random. However, one way to perform such a random choice is as described in Hyb_2, and the two processes are identical. □

5.1.2 The Patch Algorithm

We now describe the Patch algorithm when corrupting some party P_{i^*} for $i^* \notin I$. The input to Patch is a state of the static simulator $\mathsf{state}_{\mathsf{pre}}$, which consists of the inputs and outputs of the corrupted parties, the randomness sampled for the corrupted parties and the points sampled on the polynomials of the honest parties. Moreover, it receives the index i^* of the corruption request, and the input and output of that corrupted party. The output of Patch is a new state for the simulator $\mathsf{state}_{\mathsf{post}}$ that is consistent with $\mathsf{state}_{\mathsf{pre}}$. Namely, the simulator would generate the exact same messages sent/received by corrupted parties in I as with $\mathsf{state}_{\mathsf{pre}}$, and will use the same randomness for those parties. Moreover, $\mathsf{state}_{\mathsf{post}}$ corresponds to statically corrupting $I \cup \{i^*\}$. For the sake of clarity, we explicitly add the set of corrupted parties to the input of Patch (although it can be derived from $\mathsf{state}_{\mathsf{pre}}$); further, as we focus on semi-honest adversaries that do not rely on auxiliary information, we omit it from the description of Patch.

The algorithm $\mathsf{Patch}\,(\mathsf{state}_{\mathsf{pre}}, I, i^*, (\beta_{i^*}, \gamma_{i^*}), \delta_{i^*})$:

1. First, decompose $\mathsf{state}_{\mathsf{pre}}$ to the inputs, outputs, and randomness of corrupted parties, and the points that were sampled on the polynomials of the honest parties (which determine the honest-to-corrupt messages). That is, $(\beta_i, \gamma_i, \delta_i, r_i)_{i \in I}$ and $(g_j(\alpha_i))_{j \notin I, i \in I}$.
2. **Sample randomness for P_{i^*}:** In $\mathsf{state}_{\mathsf{pre}}$, P_{i^*} was honest and the simulator had sampled points $g_{i^*}(\alpha_i)$ for every $i \in I$. Choose a random polynomial $g'_{i^*}(x)$ under the following constraints:

   $$g'_{i^*}(\alpha_i) = g_{i^*}(\alpha_i) \;\text{ for every } i \in I, \qquad \text{and} \qquad g'_{i^*}(0) = \beta_{i^*} \cdot \gamma_{i^*} \;.$$

   There are $|\mathbb{F}|^{t-|I|}$ such polynomials. Sample one of them uniformly at random (this can be done, if $|I| < t$, by choosing $t - |I|$ random points and then interpolate the polynomial). Let r_{i^*} be the corresponding random tape that leads to the polynomial $g'_{i^*}(x)$. By abuse of notation, we set $g_{i^*}(x) := g'_{i^*}(x)$.
3. **Sample points on polynomials of honest parties:** For every $j \notin I \cup \{i^*\}$, sample an additional point on each of the polynomials $g_j(x)$. That is, sample $\{g_j(\alpha_{i^*})\}_{j \notin I \cup \{i^*\}}$ such that

   $$\delta_{i^*} = \sum_{k=1}^{n} \lambda_k \cdot g_k(\alpha_{i^*}).$$

   Note that so far we sampled $|I| < t$ points on each polynomial, so this can be chosen, for instance, in a similar way as the simulator (choosing independent points for all honest parties except for one, and then uniquely determining its point).

4. **Output:** The output is the updated state $\mathsf{state}_{\mathsf{post}}$ consisting of $(\beta_i, \gamma_i, \delta_i, r_i)_{i \in I \cup \{i^*\}}$ and $(g_j(\alpha_i))_{j \notin I \cup \{i^*\}, i \in I \cup \{i^*\}}$.

By inspection, it is clear that the view of the adversary is compatible with the view it already received. Before proving correctness of the Patch algorithm, we introduce the following notation. Given the simulator $\mathcal{S}$ for the semi-honest adversary, we denote by $\mathcal{S}(\beta_I, \gamma_I, \delta_I, I)$ the state of $\mathcal{S}$ upon producing the output when simulating corrupted set I on input β_I, γ_I and output δ_I. Recall that by Section 4.3, it is sufficient to compare the output produced by $\mathcal{S}$ on $\mathsf{state}_{\mathsf{pre}}$ and on $\mathsf{state}_{\mathsf{post}}$ and show that they are identically distributed.

Claim 5.3. *Let $t < n$. For every $I \subseteq [n]$ of cardinality at most t and $i^* \notin I$, for every pair of degree-t polynomials $q_a(x), q_b(x)$, and for every degree-t polynomial $q_{ab}(x)$ satisfying:*

- $q_{ab}(0) = q_a(0) \cdot q_b(0)$, *and*
- $(\delta_1, \ldots, \delta_n) = f_{\mathsf{mult}}((\beta_1, \gamma_1), \ldots, (\beta_n, \gamma_n))$ *where* $\beta_j = q_a(\alpha_j), \gamma_j = q_b(\alpha_j), \delta_j = q_{ab}(\alpha_j)$ *for every* $j \in [n]$,

it holds that:

$$\left\{ \mathsf{Patch}\left(\mathcal{S}(\beta_I, \gamma_I, \delta_I, I), i^*, (\beta_{i^*}, \gamma_{i^*}), \delta_{i^*}\right) \right\} \equiv \left\{ \mathcal{S}(\beta_{I \cup \{i^*\}}, \gamma_{I \cup \{i^*\}}, \delta_{I \cup \{i^*\}}, I \cup \{i^*\}) \right\} .$$

Proof. Fix some $\mathbf{z}$ in the support of $\mathcal{S}(\beta_{I \cup \{i^*\}}, \gamma_{I \cup \{i^*\}}, \delta_{I \cup \{i^*\}}, I \cup \{i^*\})$. Let $k' = |I \cup \{i^*\}|$. Note that each element in the support is obtained with exactly the same probability, as $\mathcal{S}$ just makes independent random choices:

1. For each $i \in I \cup \{i^*\}$, the simulator chooses the randomness r_{i^*} for the polynomial $g_i(x)$ uniformly and independently at random. This is equivalent to choosing $(|I| + 1) \cdot t$ random field elements (recall that the constant term is fixed).
2. For each $i \notin I \cup \{i^*\}$, the simulator chooses the view of P_i, which consists of all messages honest parties send to P_i. This includes sampling $n - (|I| + 1) - 1 = n - |I| - 2$ random points (as the last one is determined). In total, here we have $(|I| + 1)(n - |I| - 2)$ random choices.

In total, sampling a view consists of $(|I| + 1) \cdot t + (|I| + 1)(n - |I| - 2)$. Thus, each element in the support is chosen with probability $(1/|\mathbb{F}|)^{(|I|+1)(n+t-|I|-2)}$.

We now consider the process $\mathsf{Patch}\left(\mathcal{S}(\beta_I, \gamma_I, \delta_I, I), i^*, (\beta_{i^*}, \gamma_{i^*}), \delta_{i^*}\right)$. We first choose an output for $\mathcal{S}(\beta_I, \gamma_I, \delta_I, I)$ to obtain some output $\mathbf{w}$. Each element in the support of $\mathcal{S}(\beta_I, \gamma_I, \delta_I, I)$ is chosen with probability $(1/|\mathbb{F}|)^{|I|(n+t-|I|-1)}$.

Then, we run $\mathsf{Patch}\left(\mathbf{w}, i^*, (\beta_{i^*}, \gamma_{i^*}), \delta_{i^*}\right)$. For choosing r_{i^*}, Patch samples additional $t - |I|$ random points. For sampling its view, Patch samples additional $n - |I \cup \{i^*\}| - 1 = n - |I| - 2$ points, i.e., Patch samples $n + t - 2|I| - 2$ elements in total. Thus, sampling $\mathbf{w}$ and then running Patch results in a total of $(|I| + 1)(n + t - |I| - 2)$ random choices. Moreover, it is easy to see that for every possible vector $\mathbf{z}$ in the support of $\mathcal{S}(\beta_{I \cup \{i^*\}}, \gamma_{I \cup \{i^*\}}, \delta_{I \cup \{i^*\}}, I \cup \{i^*\})$ there exists one unique possible $\mathbf{w}'$ such that $\mathsf{Patch}\left(\mathbf{w}', i^*, (\beta_{i^*}, \gamma_{i^*}), \delta_{i^*}\right)$ leads to $\mathbf{z}$: Given $\mathbf{z}$, removing $g_{i^*}(0)$, $g_{i^*}(\alpha_j)$ for every $j \notin I \cup \{i^*\}$, and $g_j(\alpha_{i^*})$ for every $j \notin I \cup \{i^*\}$, fully defines $\mathbf{w}'$. Thus, the two distributions are identical. □

5.1.3 Efficient Adaptive Security

Lemma 5.2 proves static security of Protocol 5.1. By construction, it holds that the static simulator is efficient, black-box, straight-line, and round-by-round. It also admits a committal round since the inputs never change in the semi-honest setting. Finally, by Claim 5.3 there exists an efficient Patch algorithm. Therefore, Theorem 4.2 yields the following corollary.

Corollary 5.4. *Let $t < n/2$. Then, Protocol 5.1 adaptively t-privately computes f_{mult} in the presence of a semi-honest adversary with efficient simulation.*

5.2 The BGW Protocol in the f_{mult}-Hybrid Model

We now present the BGW protocol in the f_{mult}-hybrid model. Given an n-party functionality $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, we consider an arithmetic circuit C over $\mathbb{F}$ that computes f. The following is taken verbatim from [AL11]:

Protocol 5.5: Computing f in the f_{mult}-hybrid model

- **Private input:** Each party P_i has a private input $x_i \in \mathbb{F}$.
- **Auxiliary input:** Each party P_i holds an arithmetic circuit C over the field $\mathbb{F}$, such that for every $\mathbf{x} \in \mathbb{F}^n$ it holds that $C(\mathbf{x}) = f(\mathbf{x})$, where $f : \mathbb{F}^n \to \mathbb{F}^n$. The parties also hold distinct non-zero values $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.
- **The protocol:**
  1. **The input-sharing stage:** Each party P_i chooses a polynomial $q_i(x)$ uniformly from the set of all polynomials of degree at most t with constant term x_i. For every $j \in [n]$, party P_i sends to party P_j the value $q_i(\alpha_j)$. Each party P_i records the values $q_1(\alpha_i), \ldots, q_n(\alpha_i)$ that it received.
  2. **The circuit-emulation stage:** Let $G_1, \ldots, G_\ell$ be a predetermined topological ordering of the gates of the circuit. For $k = 1, \ldots, \ell$ the parties proceed as follows:
     - *If G_k is an addition gate:* Let β_i^k and γ_i^k be the shares of input wires held by party P_i. Then, P_i defines its share of the output wire to be $\delta_i^k = \beta_i^k + \gamma_i^k$.
     - *If G_k is a multiplication-by-a-constant gate with constant $c \in \mathbb{F}$:* Let β_i^k be the share of the input wire held by party P_i. Then, P_i defines its share of the output wire to be $\delta_i^k = c \cdot \beta_i^k$.
     - *If G_k is a multiplication gate:* Let β_i^k and γ_i^k be the shares of input wires held by party P_i. Then, P_i sends (β_i^k, γ_i^k) to the ideal functionality f_{mult} and receives back a value δ_i^k. Party P_i defines its shares of the output wire to be δ_i^k.
  3. **The output-reconstruction stage:** Let $o_1, \ldots, o_n$ be the output wires, where party P_i's output is the value on wire o_i. For every $k = 1, \ldots, n$ denote by $\beta_1^k, \ldots, \beta_n^k$ the shares that the parties hold for wire o_k. Then, each P_i sends to P_k the share β_i^k. Upon receiving all shares $(\beta_1^k, \ldots, \beta_n^k)$, P_k finds the unique degree-t polynomial $g_k(x)$ satisfying $g_k(\alpha_j) = \beta_j^k$ for every $j \in [n]$.
- **Output:** Each party P_k outputs $g_k(0)$.

5.2.1 Static Security

After recalling the protocol, we prove static security for completeness.

Lemma 5.6. *Let f be a deterministic n-party functionality. Protocol 5.5 statically t-privately computes f in the f_{mult}-hybrid model with perfect security and efficient simulation facing a semi-honest adversary, for $t < n/2$.*

Further, if f is a linear function, Protocol 5.5 statically t-privately computes f in the plain model with perfect security and efficient simulation facing a semi-honest adversary, for $t < n$.

Proof. We start by constructing a static simulator. Let $I \subseteq [n]$ of size $|I| \leq t$.

The Simulator

- **Inputs:** The simulator $\mathcal{S}$ receives the inputs of the corrupted parties $(x_i)_{i\in I}$. $\mathcal{S}$ sends these inputs to the trusted party computing f and receives the output values $(y_i)_{i\in I}$.
- The simulator is parameterized by an arbitrary subset $\hat{I} \subseteq [n]$ of cardinality exactly t such that $I \subseteq \hat{I}$ (if $|I| = t$ then $\hat{I} = I$).
- **The simulator:**
  1. First, $\mathcal{S}$ initializes two sets $\mathsf{view} = \emptyset$ and $\mathsf{secretChoices} = \emptyset$.
  2. Simulating the input-sharing stage:
     (a) For every $i \in I$, the simulator $\mathcal{S}$ chooses a uniformly distributed random tape for P_i denoted as r_i; this random tape and the input x_i fully determine the degree-t polynomial $q_i(x)$ with $q_i(0) = x_i$ chosen by P_i in the input-sharing phase.
     (b) For every $j \notin I$, the simulator $\mathcal{S}$ chooses random points $q_j(\alpha_i)$ for every $i \in \hat{I}$ uniformly and independently at random.
     (c) Add $(r_i)_{i\in I}$ and $(q_j(\alpha_i))_{j\notin I, i\in I}$ to view, and $(q_j(\alpha_i))_{j\notin I, i\in \hat{I}\setminus I}$ to $\mathsf{secretChoices}$.
  3. Simulating the circuit-emulation stage: For every $G_k \in \{G_1, \ldots, G_\ell\}$ proceed as follows.
     (a) *G_k is an addition gate:* Let $(f_a(\alpha_i))_{i\in\hat{I}}$ and $(f_b(\alpha_i))_{i\in\hat{I}}$ be the shares of the input wires of the corrupted parties that were generated by $\mathcal{S}$. For every $i \in \hat{I}$, the simulator $\mathcal{S}$ computes $f_a(\alpha_i) + f_b(\alpha_i) = (f_a + f_b)(\alpha_i)$ which defines the shares of the output wire of G_k for the set $\hat{I}$.
     (b) *G_k is a multiplication-with-constant gate:* Let $(f_a(\alpha_i))_{i\in\hat{I}}$ be the shares of the input wire and let $c \in \mathbb{F}$ be the constant of the gate. $\mathcal{S}$ computes $c \cdot f_a(\alpha_i) = (c \cdot f_a)(\alpha_i)$ for every $i \in \hat{I}$ which defines the shares of the output wire of G_k for the set $\hat{I}$.
     (c) *G_k is multiplication gate:* $\mathcal{S}$ chooses $|\hat{I}| = t$ points $f_{ab}(\alpha_i)$ uniformly at random for every $i \in \hat{I}$, and defines the shares of the set $\hat{I}$ of the output wire of G_k to be $(f_{ab}(\alpha_i))_{i\in\hat{I}}$. Add $(f_{ab}(\alpha_i))_{i\in I}$ to view and $(f_{ab}(\alpha_i))_{i\in\hat{I}\setminus I}$ to $\mathsf{secretChoices}$.
     $\mathcal{S}$ adds these shares to the corrupted parties' views.
  4. Simulating the output-reconstruction stage: Let $o_1, \ldots, o_n$ be the output wires. We now focus on the output wires of the corrupted parties. For every $k \in I$, the simulator $\mathcal{S}$ has already defined exactly t shares on the polynomial that hides the value on the wire o_k, denoted as $g_k(\alpha_i)$ (for every $i \in \hat{I}$).
     Then, find the unique degree-t polynomial $g'_k(x)$ such that $g'_k(\alpha_i) = g_k(\alpha_i)$ for every $i \in \hat{I}$, and $g'_k(0) = y_k$. Note that $\hat{I}$ defines exactly t points, and $g'_k(0)$ is one additional point, and therefore $g'_k(x)$ is fully determined. Add the shares $(g'_k(\alpha_j))_{j\notin I}$ to view as simulating the shares from the honest parties to the adversary.
- **Output:** Output view as the view of the adversary, and store $\mathsf{secretChoices}$ for further adaptive corruption (as explained in Section 5.2.2).

We now show that the joint distribution of the outputs of all parties and the view of the corrupted parties in the real world, is identically distributed as the joint distribution of the output of all parties and the output of the simulator in the ideal execution. Towards that end, we show a sequence of hybrid experiments:

1. Hyb_0: This is the real execution. We run the protocol on inputs as determined by $(x_i)_{i\in[n]}$. The output of this hybrid is the output values of all parties and the view of the corrupted parties.
2. Hyb_1: In this hybrid execution, we consider a simulator that receives the input values of *all* parties (i.e., receives $(x_k)_{k\in[n]}$) and also receives the outputs of the corrupted parties $(y_i)_{i\in I}$ as

computed by the trusted party. It then works as the simulator where instead of just choosing the points that are sent to the honest parties, it chooses random degree-t polynomial with the correct constant term, and adds to the view of the adversary only the shares of corrupted parties, I. In the output stage, for each output wire of a corrupted party, it just sends the shares that the simulated honest parties hold on that wire. The output of this experiment is the output of all parties as determined by the trusted party, and view as generated by the modified simulator.

3. Hyb_2: In this hybrid execution, we again consider a simulator that receives all the inputs of all parties and the outputs of the corrupted parties. It works exactly as the simulator of Hyb_1 with one modification: In the output wires, instead of just sending the shares on the output wires, we first take the y_i as the output of the circuit, we take the t shares on the output wire of the parties in $\hat{I}$, and interpolate them to the polynomial $g'_k(x)$ as the simulator in the ideal process. It then simulate the honest parties sending shares on $g'_k(x)$. The output of this experiment is the output of all parties as determined by the trusted party, and the view as generated by the modified simulator.
4. Hyb_3: This is the ideal execution. The output of this hybrid is the output of all parties as determined by the trusted party, and view as generated by the simulator.

There is just one difference between Hyb_0 and Hyb_1: In Hyb_0, the output of all honest parties is determined by the protocol whereas in Hyb_1 the output is determined by the trusted party. In Hyb_1, the output is simply $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$. In Hyb_0, the parties emulate the computation of the circuit C on the inputs $x_1, \ldots, x_n$. We have the invariant that the value on each wire is hidden using secret sharing, i.e., the shares that the parties hold define a unique polynomial with constant term that correspond to the value on the wire when computing $(y_1, \ldots, y_n) = C(x_1, \ldots, x_n)$. This is true for the input-sharing phase, as each party P_i shares its input x_i. Then, this holds from the properties of the secret-sharing scheme (addition and multiplication-by-a-constant gate) and as guaranteed from f_{mult}. We then obtain that the output wires carry the correct values $y_1, \ldots, y_n$, and in the output-reconstruction phase, each party simply reconstruct its output. We conclude that the output of all parties is exactly the same in both executions and that the output of Hyb_0 and Hyb_1 is exactly the same.

The difference between Hyb_1 and Hyb_2 is in the output stage only. In Hyb_1, the simulator computes the polynomials of the output wires according to the circuit. In Hyb_2 it computes the polynomials on the output wires, but then takes just t shares on that polynomial (corresponding to the set of parties $\hat{I}$), takes the value on the output wire, and interpolate them to a polynomial $g'_k(x)$. It then adds to the view the shares of all honest parties on the polynomials $g'_k(x)$, for $k \in I$. We claim that the two processes are *exactly* the same. Namely, as follows from the computation of the circuit, $g_k(0)$ is y_k where $(y_1, \ldots, y_n) = C(x_1, \ldots, x_n)$, and the simulator also receives y_k as input as computed from the trusted party. The two polynomials are identical, and the view of the adversary is therefore identical.

The difference between Hyb_2 and Hyb_3 is as follows. In Hyb_2, the simulator receives the inputs of all parties, and therefore computes all polynomials with the correct constant term. In Hyb_3, the simulator just chooses random points uniformly at random on each input polynomial of the honest parties, and each polynomial of f_{mult}. In both executions, the output phase is a deterministic process of the shares defined before. We conclude that the two outputs are identically distributed, as follows from Claim 2.2. □

5.2.2 The Patch Algorithm

We now describe the Patch algorithm when corrupting some party P_{i^*} for $i^* \notin I$. The input to Patch is a pre-corruption state of the static simulator $\mathsf{state}_{\mathsf{pre}}$, which consists of the inputs and outputs of the corrupted parties, the view of the adversary view, secretChoices, and the set $\hat{I} \supseteq I$. We have two cases to consider:

1. When $i^* \in \hat{I}$, the Patch algorithm that we will present is deterministic and make no random choices.
2. When $i^* \notin \hat{I}$, we first present a procedure called Replace, which re-samples a state for the simulator corresponding to some set I' in which $i^* \in I'$ instead of $\hat{I}$. For that, it needs to remove from its secret state shares that were sampled for some $j^* \in \hat{I} \setminus I$. We then simply run Replace to replace j^* with i^*, followed by Patch for i^*.

We proceed with describing the above two procedures.

The first Patch algorithm. We first describe the Patch algorithm for the case where $i^* \in \hat{I} \setminus I$.

$\underline{\mathsf{Patch}(\mathsf{state}_{\mathsf{pre}}, I, i^*, x_{i^*}, y_{i^*}) \text{ for } i^* \in \hat{I} \setminus I.}$

1. **Input:** $\mathsf{state}_{\mathsf{pre}}$ includes the inputs and outputs of all corrupted parties $(x_i, y_i)_{i \in I}$, the random tapes $(r_i)_{i \in I}$ as chosen by the simulator, the view of the adversary view which includes all $q_j(\alpha_i)$ for $i \in I$ for each input of an honest party $j \notin I$, the shares on each output of f_{mult}, i.e., $f_{ab}(\alpha_i)$ for $i \in I$ and the polynomials $g_i(x)$ for each output wire $i \in I$. In addition, it has secretChoices which include of $q_j(\alpha_i)$ for $i \in \hat{I} \setminus I$ and $f_{ab}(\alpha_i)$ for $i \in \hat{I} \setminus I$.
2. Since $i^* \in \hat{I} \setminus I$, the values $q_j(\alpha_{i^*})$ have already been sampled for every $j \notin I$, as well as $f_{ab}(\alpha_{i^*})$ for each multiplication gate. All that is left is to sample the random tape of P_{i^*} and the shares it receives in the output stage.
   (a) **Random tape:** The simulator already sampled $q_{i^*}(\alpha_i)$ for every $i \in \hat{I}$. Reconstruct the unique polynomial $q_{i^*}(x)$ that agrees with the points that were sampled, and for which $q_{i^*}(0) = x_{i^*}$. Let r_{i^*} be the random tape that corresponds to $q_{i^*}(x)$. Add to view the random tape r_{i^*} and move $q_j(\alpha_{i^*})$ for every $j \notin I$ from secretChoices to view.
   (b) **Output wire:** Compute the t shares on the output wire o_{i^*}, corresponding to the set $\hat{I}$. Reconstruct the unique degree-t polynomial $g_{i^*}(x)$ that agrees with all the shares defined on o_{i^*} for the set $\hat{I}$, and with the output y_{i^*}. Add to view the shares $g_{i^*}(\alpha_j)$ for every $j \notin I \cup \{i^*\}$.

   Note that there are no random choices here.
3. **Output:** the post-corruption state $\mathsf{state}_{\mathsf{post}}$ which includes all the inputs and outputs of all corrupted parties $(x_i, y_i)_{i \in I \cup \{i^*\}}$, the random tapes $(r_i)_{i \in I \cup \{i^*\}}$ of the corrupted parties, the modified view and modified secretChoices.

By construction, the new view is compatible with the previous one (i.e., agrees with all the messages and the randomness that the corrupted parties received).

Claim 5.7. *Let $t < n$. For every $I \subset [n]$ of cardinality at most t, for every $\hat{I} \supset I$ of cardinality exactly t and $i^* \in \hat{I} \setminus I$, for every $x_1, \ldots, x_n \in \mathbb{F}^n$, $f : \mathbb{F}^n \to \mathbb{F}^n$ and $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, it holds that:*

$$\left\{\mathsf{Patch}\left(\mathcal{S}(x_I, y_I, I, \hat{I}), i^*, x_{i^*}, y_{i^*}\right)\right\} \equiv \left\{\mathcal{S}\left(x_{I \cup \{i^*\}}, y_{I \cup \{i^*\}}, I \cup \{i^*\}, \hat{I}\right)\right\} \ .$$

Proof. Fix some $\mathbf{z}$ in the support of $\mathcal{S}(x_{I\cup\{i^*\}}, y_{I\cup\{i^*\}}, I\cup\{i^*\}, \hat{I})$. Note that each element in the support is obtained with exactly the same probability, as $\mathcal{S}$ just makes independent random choices:

1. For each $i \in I\cup\{i^*\}$, the simulator chooses t coefficients for the polynomial $q_i(x)$ uniformly and independently at random. Those are $|I\cup\{i^*\}|\cdot t$ random choices.
2. For each $j \notin I\cup\{i^*\}$, the simulator chooses t random points $q_j(\alpha_i)$ for every $i \in \hat{I}$. Those are $(n - |I\cup\{i^*\}|)\cdot t$ random choices.
3. For each multiplication gate $G_1, \ldots, G_k$, the simulator chooses t random points $h_1(\alpha_i), \ldots, h_k(\alpha_i)$ for every $i \in \hat{I}$. Those are $k\cdot t$ random choices.

All other values (such as the polynomials on the output wires) are deterministically determined given those random choices. Each element in the support of $\mathcal{S}(x_{I\cup\{i^*\}}, y_{I\cup\{i^*\}}, I\cup\{i^*\}, \hat{I})$ is chosen with probability $\epsilon := (1/|\mathbb{F}|)^{t\cdot(n+k)}$.

We now consider the process $\mathsf{Patch}(\mathcal{S}(x_I, y_I, I, \hat{I}), i^*, x_{i^*}, y_{i^*})$. We first choose an output $\mathbf{w}$ for $\mathcal{S}(x_I, y_I, I, \hat{I})$. It is easy to see that each element in the support of $\mathcal{S}(x_I, y_I, I, \hat{I})$ is chosen with probability ϵ as well. Next, we run $\mathsf{Patch}\,(\mathbf{w}, i^*, x_{i^*}, y_{i^*})$. Since Patch has no random choices, observe that each $\mathbf{w}$ defines a unique value in the support of $\mathcal{S}(x_{I\cup\{i^*\}}, y_{I\cup\{i^*\}}, I\cup\{i^*\}, \hat{I})$. Moreover, for every possible element $\mathbf{z}$ in the support of $\mathcal{S}(x_{I\cup\{i^*\}}, y_{I\cup\{i^*\}}, I\cup\{i^*\}, \hat{I})$ there exists a unique $\mathbf{w}'$ such that $\mathsf{Patch}\,(\mathbf{w}', i^*, x_{i^*}, y_{i^*})$ might lead to $\mathbf{z}$. We conclude that the two distributions are identical. □

The second Patch algorithm: Replace. As mentioned above, when $i^* \notin \hat{I}\setminus I$, we first choose some $j^* \in \hat{I}\setminus I$ for which we already sampled a view, and replace it with i^*, i.e., sample a view for i^* that is compatible with the consequences of the random choices we made for j^*. This is the algorithm $\mathsf{Replace}$.

$\mathsf{Replace}(\mathsf{state}_{\mathsf{pre}}, I, j^*, i^*)$ for $j^* \in \hat{I}$ and $i^* \notin \hat{I}$. Recall that the simulator sampled also shares for parties not in I, those are the parties in $\hat{I}\setminus I$. When corrupting a party P_{i^*} for $i^* \notin \hat{I}$, we first want to replace the underlying corruption set to $I' = (\hat{I}\setminus\{j^*\})\cup\{i^*\}$ and then invoke Patch as for $i^* \in I'\setminus I$. The replace is performed as follows:

1. **Input:** $\mathsf{state}_{\mathsf{pre}}$ as above, and $j^* \in \hat{I}$ and $i^* \notin \hat{I}$.
2. The simulator already sampled $q_j(\alpha_{j^*})$ for every $j \notin I$. Moreover, let $G_1, \ldots, G_k$ be the multiplication gates, and let $h_1(x), \ldots, h_k(x)$ be the associated polynomials with the multiplication gates. The simulator already sampled $h_1(\alpha_{j^*}), \ldots, h_k(\alpha_{j^*})$. All of those are part of $\mathsf{secretState}$. Finally, let $(g_i(x))_{i\in I}$ be the polynomial associated with the output wires of the corrupted parties, and recall that those are part of the view that the simulator has committed to. Note also that each output wire is a *linear* combination of the input wires and the outputs of the multiplication wires.
   The simulator now samples $q_j(\alpha_{i^*})$ for every $j \notin I$ and $h_1(\alpha_{i^*}), \ldots, h_k(\alpha_{i^*})$ that agree with $g_i(\alpha_{i^*})$ for every $i \in I$. We have $|I|$ constraints, and a total of $n - |I| + k$ variables, where k is the number of multiplication gates. We claim below that this set of equations always has a solution, as we show below. Standard Gauss elimination enables an efficient sampling of a random solution.
3. **Output:** Remove from $\mathsf{secretChoices}$ all samples $q_j(\alpha_{j^*})$ for $j \notin I$ and $h_1(\alpha_{j^*}), \ldots, h_k(\alpha_{j^*})$. Include instead $q_j(\alpha_{i^*})$ for every $j \notin I$ and $h_1(\alpha_{i^*}), \ldots, h_k(\alpha_{i^*})$. The view of the adversary is the exact same view. The output is $\mathsf{state}_{\mathsf{post}} = (\mathsf{view}, \mathsf{secretChoices})$.

We claim that the output distribution of $\mathcal{S}(x_I, y_I, I, I')$ (the joint distribution of view and $\mathsf{secretChoices}$) is distributed identically to running $\mathsf{Replace}(\mathcal{S}(x_I, y_I, I, \hat{I}), j^*, i^*)$, where the view obtained in the two processes is exactly the same. We have:

Claim 5.8. *Let $t < n$. For every $I \subset [n]$ of cardinality at most t, for every $\hat{I} \supset I$ of cardinality exactly t, for every $j^* \in \hat{I} \setminus I$ and $i^* \notin \hat{I}$, for every $x_1, \ldots, x_n \in \mathbb{F}^n$, deterministic $f : \mathbb{F}^n \to \mathbb{F}^n$ and $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, it holds that:*

$$\left\{\mathsf{Replace}(\mathcal{S}(x_I, y_I, I, \hat{I}), j^*, i^*)\right\} \equiv \left\{\mathcal{S}(x_I, y_I, I, I')\right\} ,$$

where $I' = \hat{I} \setminus \{j^*\} \cup \{i^*\}$.

Proof. Recall that the circuit has n input wires marked as $i_1, \ldots, i_n$ which hold $x_1, \ldots, x_n$ as the input values, and k multiplication gates $G_1, \ldots, G_k$ which hold $w_1, \ldots, w_k$ as the values of the output wires of the gates. For some known matrix M the output wires of the protocol can be represented as:

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = M \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_n \\ w_1 \\ \vdots \\ w_k \end{pmatrix} = \begin{pmatrix} m_{1,1} & \ldots & m_{1,n+k} \\ \vdots & \ddots & \vdots \\ m_{n,1} & \ldots & m_{n,n+k} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_n \\ w_1 \\ \vdots \\ w_k \end{pmatrix} .$$

For the input x_I and output y_I, we say that a possible sequence of values $(x'_1, \ldots, x'_n, w'_1, \ldots, w'_k)$ is $\mathsf{compatible}$ with x_I and y_I if the following holds:

1. $x'_I = x_I$;
2. Let $(y'_1, \ldots, y'_n) = M \cdot (x'_1, \ldots, x'_n, w'_1, \ldots, w'_k)$. Then, $y'_I = y_I$.

Let $\mathsf{compatible}(x_I, y_I)$ be the set of all (x', w') that are compatible with x_I an y_I, and denote its size as $T := |\mathsf{compatible}(x_I, y_I)|$. Note that $T \geq 1$, as the true inputs that trusted party used (which fully determine $w_1, \ldots, w_k$) are always compatible with x_I and y_I.

Fix some $\mathbf{z}$ in the support of $\mathcal{S}(x_I, y_I, I, I')$. Note that each element in the support is obtained with exactly the same probability, as $\mathcal{S}$ just makes independent random choices:

1. For each $i \in I$, the simulator chooses t coefficients for the polynomial $q_i(x)$ uniformly and independently at random.
2. For each $j \notin I$, the simulator chooses t random points $q_j(\alpha_i)$ for every $i \in I'$.
3. For each multiplication gate $G_1, \ldots, G_k$, the simulator chooses t random points $h_1(\alpha_i), \ldots, h_k(\alpha_i)$ for every $i \in I'$.

All other values (such as the polynomials on the output wires) are deterministically determined given those random choices. Thus, each element in the support of $\mathcal{S}(x_I, y_I, I, I')$ is obtained with probability $\epsilon := (1/|\mathbb{F}|)^{t \cdot |I| + t \cdot (n - |I|) + kt}$.

We now consider the process $\mathsf{Replace}(\mathcal{S}(x_I, y_I, I, \hat{I}), j^*, i^*)$. We first choose an output for $\mathcal{S}(x_I, y_I, I, \hat{I})$ to obtain some output $\mathbf{w}$. Then, we run $\mathsf{Replace}(\mathbf{w}, j^*, i^*)$. We claim that for each $\mathbf{w}$ there are T possible values in the support of $\mathcal{S}(x_I, y_I, I, I')$ that it might be mapped to, and each is obtained with probability $1/T$. Specifically, each $\mathbf{w}$ and set of values $(\mathbf{x}', w') \in \mathsf{compatible}(x_I, y_I)$ defines a unique possible output $\mathbf{z}$: Since $\mathbf{w}$ contains all the shares on all wires of exactly t parties, it is possible to completely determine the polynomials on the wires given the constant terms. The set of values $\mathbf{x}'$ defines the constant terms on the input wires, and the set of values w' defines the constant terms on the output wires of the multiplication gates.

Thus, the shares of P_{i^*} are uniquely determined, and are consistent with the view of the adversary on all wires, as well as the output wires. We claim that the set of shares are different for

each pair of different values $(\mathbf{x}', w'), (\mathbf{x}'', w'') \in \mathsf{compatible}(x_I, y_I)$. The two vectors of values are different on at least one input wire of an honest party P_j (or one output wire of multiplication gate G_l). We claim that $q_j(\alpha_{i^*})$ (or $h_l(\alpha_{i^*})$) is different in the two executions; otherwise, the two polynomials equal on $t+1$ points (the t point of the parties in $\hat{I}$ and on i^*) and therefore are equal; thus, the input of P_j is the same in both executions, yielding a contradiction.

Next, we claim that for every possible vector $\mathbf{z}$ in the support of $\mathcal{S}(x_I, y_I, I, I')$ there exist exactly T possible $\mathbf{w}'$ such that $\mathsf{Replace}(\mathbf{w}', j^*, i^*)$ might lead to $\mathbf{z}$. Similarly to the above, given $\mathbf{z}$, each set of values $(\mathbf{x}', w') \in \mathsf{compatible}(x_I, y_I)$ defines a unique $\mathbf{w}'$ for which $\mathsf{Replace}(\mathbf{w}', j^*, i^*)$ might lead to $\mathbf{z}$. Let $\mathsf{Supp}(\mathbf{z})$ define the set of possible such outputs, and recall that $|\mathsf{Supp}(\mathbf{z})| = |\mathsf{compatible}(x_I, y_I)| = T$. Thus, we can conclude that for every $\mathbf{z}$ in the support of $\mathcal{S}(x_I, y_I, I, I')$:

$$\begin{aligned}
&\Pr\left[\mathsf{Replace}\left(\mathcal{S}\left(x_I, y_I, I, \hat{I}\right), j^*, i^*\right) = \mathbf{z}\right] \\
&= \sum_{w \in \mathsf{Supp}(\mathbf{z})} \Pr\left[\mathcal{S}(x_I, y_I, I, \hat{I}) = \mathbf{w}\right] \cdot \Pr\left[\mathsf{Replace}(\mathbf{w}, j^*, i^*) = \mathbf{z}\right] \\
&= T \cdot \epsilon \cdot \frac{1}{T} = \epsilon = \Pr\left[\mathcal{S}(x_I, y_I, I, I') = \mathbf{z}\right].
\end{aligned}$$

This concludes the proof of Claim 5.8. □

5.2.3 Efficient Adaptive Security

Lemma 5.6 proves static security of Protocol 5.5. By construction, it holds that the static simulator is efficient, black-box, straight-line, and round-by-round. It also admits a committal round since the inputs never change in the semi-honest setting. Finally, by Claims 5.7 and 5.8 there exists an efficient patch algorithm. Therefore, Theorem 4.2 yields the following corollary.

Corollary 5.9. *Let f be a deterministic n-party functionality. Protocol 5.5 adaptively t-privately computes f in the f_{mult}-hybrid model with perfect security and efficient simulation facing a semi-honest adversary, for $t < n/2$.*

Further, if f is a linear function, Protocol 5.5 adaptively t-privately computes f in the plain model with perfect security and efficient simulation facing a semi-honest adversary, for $t < n$.

References

[AL11] Gilad Asharov and Yehuda Lindell. A full proof of the BGW protocol for perfectly-secure multiparty computation. IACR Cryptol. ePrint Arch., 2011.

[AL17] Gilad Asharov and Yehuda Lindell. A full proof of the BGW protocol for perfectly secure multiparty computation. *Journal of Cryptology*, 30(1):58–151, 2017.

[BCDH18] Elette Boyle, Ran Cohen, Deepesh Data, and Pavel Hubáček. Must the communication graph of MPC protocols be an expander? In *Advances in Cryptology – CRYPTO 2018, part III*, pages 243–272, 2018.

[Bea96] Donald Beaver. Adaptive zero knowledge and computational equivocation (extended abstract). In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)*, pages 629–638, 1996.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 1988.

[BH92] Donald Beaver and Stuart Haber. Cryptographic protocols provably secure against dynamic adversaries. In *Advances in Cryptology – EUROCRYPT 1992*, pages 307–323, 1992.

[BMU07] Michael Backes, Jörn Müller-Quade, and Dominique Unruh. On the necessity of rewinding in secure multiparty computation. In *Proceedings of the Fourth Theory of Cryptography Conference, TCC 2007*, pages 157–173, 2007.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 11–19, 1988.

[CDD+99] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Advances in Cryptology – EUROCRYPT 1999*, pages 311–326, 1999.

[CDD+04] Ran Canetti, Ivan Damgård, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. Adaptive versus non-adaptive security of multi-party protocols. *Journal of Cryptology*, 17(3):153–207, 2004.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)*, pages 639–648, 1996.

[CGZ23] Ran Cohen, Juan A. Garay, and Vassilis Zikas. Completeness theorems for adaptively secure broadcast. In *Advances in Cryptology – CRYPTO 2023, part I*, pages 3–38, 2023.

[CsW19] Ran Cohen, abhi shelat, and Daniel Wichs. Adaptively secure MPC with sublinear communication complexity. In *Advances in Cryptology – CRYPTO 2019, part II*, pages 30–60, 2019.

[DM00] Yevgeniy Dodis and Silvio Micali. Parallel reducibility for information-theoretically secure computation. In *Advances in Cryptology – CRYPTO 2000*, pages 74–92, 2000.

[DN14] Ivan Damgård and Jesper Buus Nielsen. Adaptive versus static security in the UC model. In *Proceedings of the 8th International Conference on Provable Security (ProvSec)*, Lecture Notes in Computer Science, pages 10–28, 2014.

[GIOZ17] Juan A. Garay, Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. The price of low communication in secure multi-party computation. In *Advances in Cryptology – CRYPTO 2017, part I*, pages 420–446, 2017.

[GKKZ11] Juan A. Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively secure broadcast, revisited. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 179–186, 2011.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In *Advances in Cryptology – CRYPTO 1986*, pages 171–185, 1986.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[Gol04] Oded Goldreich. *Foundations of Cryptography – VOLUME 2: Basic Applications*. Cambridge University Press, 2004.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 101–111, 1998.

[GS12] Sanjam Garg and Amit Sahai. Adaptively secure multi-party computation with dishonest majority. In *Advances in Cryptology – CRYPTO 2012*, pages 105–123, 2012.

[HLM21] Martin Hirt, Chen-Da Liu-Zhang, and Ueli Maurer. Adaptive security of multi-party protocols, revisited. In *Proceedings of the 19th Theory of Cryptography Conference, TCC 2021, part I*, pages 686–716, 2021.

[HZ10] Martin Hirt and Vassilis Zikas. Adaptively secure broadcast. In *Advances in Cryptology – EUROCRYPT 2010*, pages 466–485, 2010.

[KLR10] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. *SIAM Journal on Computing*, 39(5):2090–2112, 2010.

[KTR20] Ralf Küsters, Max Tuengerthal, and Daniel Rausch. The IITM model: A simple and expressive model for universal composability. *Journal of Cryptology*, 33(4):1461–1584, 2020.

[KTZ13] Jonathan Katz, Aishwarya Thiruvengadam, and Hong-Sheng Zhou. Feasibility and infeasibility of adaptively secure fully homomorphic encryption. In *Proceedings of the 16th International Conference on the Theory and Practice of Public-Key Cryptography (PKC)*, pages 14–31, 2013.

[LLW20] Huijia Lin, Tianren Liu, and Hoeteck Wee. Information-theoretic 2-round MPC without round collapsing: Adaptive security, and more. In *Proceedings of the 18th Theory of Cryptography Conference, TCC 2020, part II*, pages 502–531, 2020.

[LZ11] Yehuda Lindell and Hila Zarosim. Adaptive zero-knowledge proofs and adaptively secure oblivious transfer. *Journal of Cryptology*, 24(4):761–799, 2011.

[MR91] Silvio Micali and Phillip Rogaway. Secure computation (abstract). In *Advances in Cryptology – CRYPTO 1991*, pages 392–404, 1991.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *Advances in Cryptology – CRYPTO 2002*, pages 111–126, 2002.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 73–85, 1989.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.