Peeking Into the Future: MPC Resilient to Super-Rushing Adversaries

Gilad Asharov[1(✉)], Anirudh Chandramouli[1], Ran Cohen[2], and Yuval Ishai[3,4]

[1] Bar-Ilan University, Ramat Gan, Israel
{gilad.asharov,anirudh.chandramouli}@biu.ac.il
[2] Reichman University, Herzliya, Israel
cohenran@runi.ac.il
[3] Technion, Haifa, Israel
yuvali@cs.technion.ac.il
[4] AWS, New York, USA

Abstract. An important requirement in synchronous protocols is that, even when a party receives all its messages for a given round ahead of time, it must wait until the round officially concludes before sending its messages for the next round. In practice, however, implementations often overlook this waiting requirement. This leads to a mismatch between the security analysis and real-world deployments, giving adversaries a new, unaccounted-for capability: the ability to "peek into the future." Specifically, an adversary can force certain honest parties to advance to round $r+1$, observe their round $r+1$ messages, and then use this information to determine its remaining round r messages.

We refer to adversaries with this capability as "super-rushing" adversaries. We initiate a study of secure computation in the presence of super-rushing adversaries. We focus on understanding the conditions under which existing synchronous protocols remain secure in the presence of super-rushing adversaries. We show that not all protocols remain secure in this model, highlighting a critical gap between theoretical security guarantees and practical implementations. Even worse, we show that security against super-rushing adversaries is not necessarily maintained under sequential composition.

Despite those limitations, we present a general positive result: secret-sharing based protocols in the perfect setting, such as BGW, or those that are based on multiplication triplets, remain secure against super-rushing adversaries. This general theorem effectively enhances the security of

G. Asharov and A. Chandramouli—Research supported by the Israel Science Foundation (grant No. 2439/20), and by the European Union (ERC, FTRC, 101043243). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.
R. Cohen—Research supported in part by NSF grant no. 2055568 and by ISF grant 1834/23.
Y. Ishai—Research supported by ISF grants 2774/20 and 3527/24, BSF grant 2022370, and ISF-NSFC grant 3127/23.

S. Fehr and P.-A. Fouque (Eds.): EUROCRYPT 2025, LNCS 15605, pp. 390–420, 2025.
https://doi.org/10.1007/978-3-031-91092-0_14

such protocols "for free." It shows that these protocols do not require parties to wait for the end of a round, enabling potential optimizations and faster executions without compromising security. Moreover, it shows that there is no need to spend efforts to achieve perfect synchronization when establishing the communication networks for such protocols.

1 Introduction

Secure multiparty computation (MPC) protocols [15,22,40,53] are typically studied in two main models: the *synchronous* model, where messages exchanged between honest parties are guaranteed to arrive within a known, bounded delay, and the *asynchronous* model, where messages can be delayed arbitrarily, and parties proceed independently at their own pace [14,16]. Though more general, the asynchronous model is often seen as overly pessimistic since such arbitrary delays are rare in practice, and networks, in most cases, are reliable. Consequently, much of the MPC literature focuses on the synchronous model, which allows for better resilience, simpler protocols, stronger security guarantees, and improved concrete efficiency.

In the synchronous model, parties begin the protocol at the same time and a fixed upper bound on message delivery time between parties is assumed to be known in advance, allowing the protocol to progress in predefined rounds. All messages in a given round must be delivered within this specified time limit, which must be respected throughout the protocol's execution. All parties are perfectly coordinated and proceed together from one round to the next.

In practice, however, achieving and maintaining truly synchronous communication is challenging. The synchronous model is idealized, and can be realized using global synchronization mechanisms, such as a common clock shared by all participants and bounded-delay channels [45], but in the real world, parties may not be perfectly synchronized. Moreover, an implicit requirement in synchronous protocols is that even if a party receives all its messages for a given round early, it must wait until the round officially ends before sending the next-round's messages. This ensures that *all* honest parties receive their expected messages before proceeding. However, determining the end of a round, that is, establishing an appropriate message delivery bound, is problematic. Setting it too high results in inefficiency, while setting it too low risks security vulnerabilities or non-termination.

Unfortunately, in real-world implementations of MPC protocols, the requirement to wait (potentially idly) until the end of a round is often overlooked, and parties are instructed to proceed optimistically and send their round-$(r+1)$ messages as soon as they receive all their round-r messages. This creates a mismatch between security analyses and practical deployments. This mismatch could be an attempt to optimize efficiency, a result of ignorance, or a result of insufficient clarification in the way synchronous protocols are described in the literature, which separates the protocol layer and the communication layer.

This oversight introduces a potential security vulnerability. Specifically, it gives the adversary a new capability that is not accounted for in the security

analysis. The adversary can "peek into the future" by sending round-r messages to an honest party P_j, causing P_j to proceed and send its round-$(r+1)$ messages (once P_j receives all other round-r messages). The adversary can then base its remaining round-r message choices for the corrupted parties on this "future" information. This is a strictly stronger capability than traditional "rushing," where the adversary only views round-r messages from honest parties before deciding the round-r messages for the corrupted parties.

We illustrate this adversarial capability with a toy example involving three parties: P_1, P_2, and P_3, running the following synchronous two-round protocol:

- **Round I:** P_1 picks and sends a random $x_1 \leftarrow \{0,1\}^\kappa$ to P_3, and likewise, P_2 picks and sends a random $x_2 \leftarrow \{0,1\}^\kappa$ to P_3 (where κ is the security parameter). All other messages are dummy (P_3 to both P_1, P_2, and messages between P_1 and P_2).
- **Round II:** P_1 sends x_1 to P_2 (and all other parties exchange dummy messages).

When the protocol is executed optimistically, P_2 can first send its round-I dummy message to P_1 and wait. Once P_1 receives the round-I message also from P_3, it proceeds to round II and sends x_1 to P_2 (and other dummy messages to P_1). Now, P_2 can send $x_2 := x_1$ as its round-I message to P_3 and ensure that P_3 always receives the same message from both parties in round I. In contrast, such behavior is impossible in a standard synchronous protocol, where a malicious P_2 can ensure that $x_2 = x_1$ only with probability $2^{-\kappa}$, as x_1 and x_2 are chosen independently.

Super-Rushing Adversaries. We initiate a study of secure computation in the presence of *super-rushing adversaries* that have a limited ability to "peek into the future." Our goal is to bridge the gap between theoretical analyses of protocols and common practices in real-world implementations.

Our starting point is the synchronous model. The literature in this setting typically considers two types of adversaries with respect to their capability in ordering the message delivery: A *non-rushing* adversary must send the corrupted parties' messages in a given round *before* receiving their messages from the honest parties. A *rushing* adversary can arbitrarily order the message delivery within a round, enabling corrupted parties to choose their messages for the round *after* receiving the messages sent by honest parties.

In this work, we define stronger adversaries that we call *super-rushing*. Similarly to rushing adversaries, a super-rushing adversary can reorder messages within a given round, but, as opposed to rushing adversaries, it can also reorder messages *between different rounds*, i.e., deliver certain messages for round $r+1$ before all messages for round r have been delivered. The only constraint is that an honest party must receive all of its round r messages before sending its round $r+1$ messages.

We aim to understand the conditions under which existing synchronous protocols remain secure in the presence of super-rushing adversaries. That is, we ask

whether the mismatch between the theoretical analysis and the implementation is of real concern. Our central motivating question is:

Do synchronous MPC protocols remain secure even in the presence of super-rushing adversaries?

Our primary goal is *not* to design new protocols tailored to this model, but to investigate whether existing synchronous protocols can run "as-is," without any modification. This contrasts with the work of Kushilevitz, Lindell, and Rabin [46], who showed how to compile any perfect synchronous protocol into a secure protocol in the UC framework, which is inherently asynchronous (and thus allows such super-rushing attacks); this is achieved by changing the original protocol and adding a synchronization round after every communication round (thus doubling the round complexity). For further discussion, see Sect. 1.2.

Ideally, the hope is that since secure protocols are already robust against rushing adversaries and guarantee (at least) privacy and correctness, they will also remain secure against super-rushing adversaries. The best-case scenario is that the answer to the above question is affirmative. Such a result would have three important implications: First, for implementations that inadvertently bypass the requirement to wait until the end of the round, an affirmative answer would confirm that no real security risk is introduced. Second, for protocols that do enforce this waiting period, an affirmative answer would indicate that waiting is unnecessary, allowing for potential optimizations and faster execution without compromising security. Third, such a result would imply that there is no need to spend efforts to achieve perfect synchronization when establishing communication networks for secure protocols.

1.1 Our Results

We initiate a systematic study of protocols in the presence of super-rushing adversaries, with a focus on perfect security. At a very high level, our results show that not all protocols remain secure against super-rushing adversaries, highlighting a critical gap between theoretical security guarantees and practical implementations. Nevertheless, our exploration finds sufficient conditions for which protocols remain secure in the presence of super-rushing adversaries. These conditions imply that central secure computation protocols from the literature maintain their security even in the presence of super-rushing adversaries. Thus, our results effectively enhance their security "for free." We believe that the same applies to almost all general-purpose synchronous MPC protocols from the literature, and our work can be seen as the first evidence in this direction.

Defining Super-Rushing Adversaries. Before proceeding, we elaborate on the model. We consider synchronous protocols that are designed in a round-by-round manner and assume that communication between honest parties is private and reliable. As soon as P_j receives all the information it needs to compute the next-message function, P_j computes its outgoing messages and sends them. The

adversary can control the message delivery according to its power: non-rushing, rushing, or super-rushing. Finally, recall that in a synchronous model, if P_i does not send a message to P_j, then P_j can recognize this;[1] hence, a standard modeling simplification is that if the adversary does not intend to send a message to P_j, to consider that the adversary instead sends some default message to P_j to indicate that. We carry this simplification over to our setting, as this naturally holds in our motivating scenario: when executing a synchronous protocol optimistically, the adversary cannot stall its messages without being detected.

We formalize super-rushing by letting the adversary "pull" messages by request (from honest parties to corrupted parties) and provide corrupted parties' messages in its own schedule within a round. The adversary is allowed to request round $r + 1$ messages from an honest P_j under the restriction that it already sent all messages from corrupted parties to P_j in round r (see Sect. 3).

We also focus in this paper on protocols with all-to-all communication pattern in every round. This suffices to capture most of the multiparty protocols in the literature in the perfect setting, and already highlights the gap between super-rushing and rushing. On the other hand, in such protocols, the adversary can potentially see just one round into the future (due to all-to-all communication, the gap between every pair of honest parties can be at most one round). Our results naturally extend to message-oblivious protocols [24,25], where the communication pattern is predetermined and does not depend on inputs or randomness, even though in such protocols the adversary may have the ability to see beyond a single round into the future.

Single-Input Functionalities. As a warmup, we first investigate protocols where a single party holds an input. Despite being a simplified variant, such functionalities are of high importance. Single-input functionalities include broadcast protocols [47,52], where a sender wishes to distribute a message to all participants (and the parties must agree on the message received). Another important functionality is verifiable secret sharing (VSS) [23], where a dealer distributes a secret to n participants while the parties can verify that a well-defined secret was shared. Moreover, it allows, for instance, the distribution of "multiplication triplets with a dealer," a highly important building block in perfect MPC (e.g., [15]). It allows a dealer to distribute polynomials $A(x), B(x), C(x)$, such that each party P_i receives $A(i), B(i), C(i)$, while all parties guarantee that $A(x), B(x), C(x)$ are of degree-t, and that $C(0) = A(0) \cdot B(0)$. Using this primitive, parties generate multiplication triplets [12]. Our first result shows that:

Theorem 1.1. (Informal) *Let f be a single-input functionality, and consider a protocol π that perfectly securely realizes f in the presence of a rushing adversary. Moreover, assume that:*

[1] This can be achieved using timeouts when instantiating synchronous communication via a common clock and bounded-delay channels.

- *If the input provider is corrupted, then the input x can be extracted from the joint view of the honest parties;*
- *If the input provider is honest, then for every x and x' such that $f_I(x) = f_I(x')$ (where f_I denotes the outputs of the corrupted parties), the view of the corrupted parties in an execution of π with input x is identical to their view in an execution with x'.*

Then, π perfectly realizes f in the presence of a super-rushing adversary.

Note that verifiable secret sharing, generation of multiplication triplets, or functionalities with no privacy (such as broadcast), satisfy the conditions of Theorem 1.1, as well as other results in the literature, e.g., [9,38].

In fact, we prove an even stronger result than stated in Theorem 1.1: we can relax the requirements from the protocol and assume security against just a *non-rushing* adversary (which, in round r first provides its messages and only then receives the honest parties' round r messages). This suffices to automatically upgrade the security twice—ensuring security against both rushing and super-rushing adversaries, at no additional cost! In other words, when proving the security of such protocols in the past, we were overextending our efforts yet underestimating the results.

Two-Input Functionalities. After establishing that super-rushing is of no concern for single-input functionalities, we turn our attention to functionalities with two input providers. Specifically, we consider the simultaneous broadcast functionality [23], denoted as f_{sb}, where two designated parties, say P_1 and P_2 holding bits b_1 and b_2, respectively, wish to broadcast those bits in parallel, i.e., the output of the all participants should be (b_1, b_2). The critical security requirements here is *independence of inputs*, ensuring that no sender can choose its bit as a function of the other input. Here, things dramatically change:

Theorem 1.2. (informal) *There exists a protocol that implements the two-input functionality f_{sb} which is secure against a rushing adversary but is insecure against a super-rushing adversary.*

We consider the five-party protocol from [38], where in round 1 parties P_1 and P_2 send their bit to parties P_3, P_4, and P_5. In round 2, each of those parties echo the pair of bits it received to everyone, who then take the majority as their output. Intuitively, the super-rushing adversary, corrupting, say, P_1, can "peek into the future" and learn the bit b_2 of the other sender: this can be done by sending 0 to P_3 who will echo the pair $(0, b_2)$ to everyone including P_1 (here we rely on security against a single corruption, meaning that if P_3 is cheating with this behavior, it cannot affect correctness). Next, P_1 can pick its own input b_1 as a function of b_2 (e.g., set $b_1 = b_2$). That is, P_1 breaks the *independence of inputs* requirement of secure computation.

A closer look at this protocol reveals that the protocol does not have a "committal round" [34,50], which is a predefined round during which the parties' inputs become fixed (preventing the adversary from altering its input beyond

this point) while still hiding all information about the output. This leads to the conjecture of whether the existence of a committal round is a sufficient condition for guaranteeing resiliency to super-rushing. However, it does not:

Theorem 1.3. (informal) *There exists a protocol for f_{sb}, admitting a committal round* CR*, that is secure against a rushing adversary but is insecure against a super-rushing adversary.*

In this negative result, described in Sect. 2, the protocol is very natural and consists of a verifiable secret sharing phase (in which P_1 and P_2 "commit" to their respective input bits b_1 and b_2), followed by an "output revealing round," in which the parties learn the output. This "commit-then-reveal" structure is the standard approach for implementing coin-tossing mechanisms. As such, this result demonstrates that central building-blocks for MPC might go wrong when considering super-rushing adversaries.

Sequential Composition. We proceed to study sequential composition of protocols that are secure against super-rushing. Somewhat surprisingly, we show that even a sequential composition of two single-input protocols does not necessarily remain secure against a super-rushing adversary.

Theorem 1.4. *There exists a 4-ary single-input functionality f and a 4-party protocol π_f that securely realizes f against a super-rushing adversary corrupting a single party. Further, there exists a 4-ary functionality g and a 4-party protocol π in the f-hybrid model that consists of 2 calls to f and securely realizes g against a super-rushing adversary corrupting a single party. Nevertheless, the protocol $\pi^{f \to \pi_f}$ that replaces the calls to f with the execution of π_f does not securely realize g in the presence of a super-rushing adversary corrupting single party.*

General Possibility Result. In our negative results, we see that the adversary has the capability to peek into the future, learn something, and use this information to change its own input accordingly. Which protocols prevent such a behavior?

Our primary contribution is a general positive result, demonstrating that any perfectly secure protocol against a rushing adversary that satisfies certain natural conditions remains secure even against a super-rushing adversary. In other words, although the adversary can "peek into the future," this additional power does not compromise security.

These "natural conditions" required by our transformation align closely with the structure found in most perfect, secret-sharing-based secure protocols, such as the celebrated protocol of Ben Or, Goldwasser, and Wigderson [15]. The BGW protocol operates in three phases: (1) Input sharing phase (VSS)—by the end of this phase, the inputs of all parties are fixed and no information about the output is revealed; (2) Circuit emulation phase—the parties emulate the computation of the circuit gate-by-gate, computing shares on the output wire of each gate based on the shares of the input wires; notably, during this phase,

the simulator can carry out the simulation without needing to know the final outputs of the computation; and (3) Output reconstruction phase - the outputs are reconstructed (and are needed for the simulation of this phase).

Informally, our approach requires the protocol to include a "committal round" (denoted CR), a round in which the inputs are fixed and committed but the output remains hidden, and an "output revealing round" (denoted ORR), a round during which the outputs are disclosed. In BGW, the committal round is the end of the "input sharing phase," and the output revealing round is the last round, where the parties learn the outputs.

Recall that the negative example in Theorem 1.3 has a CR (round 1) and an ORR (round 2), which may sound contradicting to the discussion above; however, in this example it holds that ORR = CR+1. Our first positive result complements this example by showing that for all protocols satisfying ORR > CR + 1, that is, protocols for which "the circuit emulation" phase takes at least one round, super-rushing security follows from rushing security.

Theorem 1.5. (informal) *Every perfectly secret-sharing-based secure protocol that (1) has all-to-all communication, (2) admits a committal round* CR*, (3) admits an output revealing round* ORR*, and (4) satisfies* ORR > CR + 1*, is secure against a super-rushing adversary.*

We emphasize that this statement is informal, and there are several other requirements from the protocol that are not specified in the above statement. See Sect. 5 for the formal treatment.

Intuitively, the adversary can "peek into the future" by observing the next communication round. However, due to the all-to-all communication pattern, this peeking is restricted to just one round ahead. The adversary, thus, might look into round CR+1 while some parties have not yet completed their committal round. The condition ORR > CR + 1 ensures that the simulator can continue constructing the adversary's view at this point without relying on the actual inputs or outputs. By the time the adversary peeks into round ORR, where the outputs are actually required, all parties are guaranteed to have completed their committal round. At this point, the adversary can no longer cause any harm.

Theorem 1.5 is very powerful. It already captures natural protocols in the literature, such as BGW for non-linear circuits [15], protocols that follow a similar paradigm [3,9,31], and protocols that are based on the online-offline model (e.g., [1,26]). Our proof upgrades the security of all those protocols, for free, and without any modifications.

Extensions. We extend Theorem 1.5 in two central ways:

- We show that it is easy to transform any protocol that has a CR and an ORR but for which ORR = CR + 1 to be secure against super-rushing adversaries, by adding one more "dummy" round between CR and ORR.
- We consider the special case where the CR is over the broadcast channel (in which case if the adversary progresses party P_j to round $r+1$, it "commits" to the message is would send to all other honest parties in round r). In that case,

$\mathsf{ORR} = \mathsf{CR}+1$ suffices to guarantee security against super-rushing adversaries. This captures, for instance, the BGW protocol for linear functions, in which the final round of its VSS is a vote over the broadcast channel, and there is no need to add a synchronization round to BGW in that case. We conclude that BGW remains secure against super-rushing, regardless of whether the function is linear or not. This extension also captures the perfectly secure protocols of [5,6] for NC^1 circuits and upgrades their security against super-rushing adversaries.

Statistical Setting. After establishing an understanding of super-rushing in the perfect setting, we turn our attention to the statistical setting. We show that unfortunately, a protocol that satisfies all the requirements of Theorem 1.5, i.e., is secret-sharing based, admits a committal round, admits an output revealing round, and satisfies $\mathsf{ORR} > \mathsf{CR} + 1$, but is only *statistically secure*, does not provide security in the presence of a super-rushing adversary, although it is secure against a rushing adversary.

In fact, our negative results complement each one of the conditions of Theorem 1.5: We showed a negative result for a protocol that does not have a CR (Theorem 1.2); When there is a CR and an ORR but for which $\mathsf{ORR} = \mathsf{CR}+1$ (Theorem 1.3); or when the protocol admits a CR, an ORR, and satisfies $\mathsf{CR} > \mathsf{ORR}+1$ but is only statistically secure. Nevertheless, we emphasize again that the statement of Theorem 1.5 is simplified, and there are additional necessary requirements from the simulator of the rushing adversary that are not reflected in the informal statement.

Open Questions. We believe that a more systematic study of super-rushing is an interesting and well-motivated topic for future work. Our work leaves open several natural questions. What about protocols that do not have an all-to-all communication pattern, such as protocols with a "king-phase" communication pattern (such as [32,41])? Under what conditions composition of protocols secure against super-rushing remain secure? Which protocols remain secure in the statistical or computational setting?

Despite the seemingly pessimistic tone of our negative result, we conjecture that almost all, if not all, general-purpose protocols outside the scope of our current framework still remain secure against super-rushing adversaries. In particular, the negative result we present for the statistical setting is somewhat contrived, as it introduces a negligible error that is almost never exploitable by traditional rushing adversaries but becomes amplified and noticeable under super-rushing adversaries. The nature of this error in statistically secure protocols, and even in computationally secure protocols, is fundamentally different and does not seem to be affected by super-rushing adversaries, but we leave it for further exploration.

1.2 Related Work

Fully Asynchronous Model. The Universal Composability (UC) framework, introduced by Canetti [20], operates under a fully asynchronous and adversarial communication model, without any guarantees on message delivery. In this model, the adversary has complete control over the network, meaning that when one honest party sends a message to another, it may never reach its destination. This stands in contrast to our model, where reliable message delivery is assumed. Consequently, the UC framework inherently does not provide guaranteed termination, as the adversary has the ability to entirely block communication. Indeed, the adversary has inherent capabilities to "peek into the future." For example, it might observe future messages from one honest party P_j (if P_j has advanced in the protocol) while another party P_k lags behind.

Kushilevitz, Lindell, and Rabin [46] introduced a transformation that converts any information-theoretically secure protocol in the stand-alone, synchronous model (satisfying a few basic properties) into one that is secure in the UC setting. Their transformation alters the original protocol by introducing a synchronization round after every protocol round, where each party announces to all when it is ready to move on to the next round, and proceeds once reeving announcements from everyone. This indeed protects against super-rushing attacks, but effectively doubles the number of rounds required, and blows up the message complexity of each round to quadratic (even if the underlying protocol does not require so). Further, this approach inherently cannot guarantee termination, as a single corrupted party can block the entire computation by remaining silent. Our focus, in contrast, is on evaluating whether existing protocols remain secure as is, *without* any modification.

Several works establish synchronous communication over asynchronous networks [11,20,43–45,48,51]. Protocols in these models are designed in a lock-step manner, which prevents super-rushing attacks. However, such protocols may be vulnerable to super-rushing adversaries when executed optimistically.

Asynchronous Setting with Eventual Delivery. A long and fruitful line of work in the realm of distributed computing studies asynchronous protocol with eventual message delivery; that is, where the adversary can arbitrarily delay messages, though all messages must eventually be delivered. As such, adversaries in this model also have limited ability to "peek into the future." Ben-Or, Canetti, and Goldreich [14] initiated the study of MPC over asynchronous networks with eventual delivery (see, e.g., [2,14,16,27,30,42] and references therein). The inherent challenge in this model is that one cannot distinguish between an honest party whose message is delayed from a corrupted party who does not send the message at all. This uncertainty leads to certain impossibility results and reduced resilience compared to the synchronous model: e.g., impossibility of deterministic protocols for Byzantine agreement [36], impossibility of asynchronous Byzantine agreement and MPC with $t \geq n/3$ (even assuming cryptography) [14,35], and impossibility of perfect asynchronous MPC with $t \geq n/4$ [14]. In contrast, our model assumes that the corrupted parties always send messages when suppose

to, which is a realistic assumption when optimistically executing a synchronous protocol. Therefore, these impossibility results do not apply in our setting.

Hybrid Settings. Several works have looked at models that combine synchrony and asynchrony in various settings. A line of work initiated by Blum, Katz, and Loss [17] studies protocols that can obtain a certain security assuming synchrony, yet satisfy weaker security guarantees if this assumption does not hold and the communication is asynchronous; see, e.g., [4,10,18]. This model enables running synchronous protocols while ensuring security against super-rushing adversaries. However, as opposed to our results, such protocols achieve *weaker* security if the adversary is super-rushing (and, in particular, are limited by the lower bounds of asynchronous MPC); further, the goal in these works is different from our motivation of studying MPC protocol designed for the synchronous setting *without modifications*.

Another line of work, initiated by Beerliová-Trubíniová, Hirt, and Nielsen [13], studies protocols in which the protocol begins with synchronous rounds and proceeds with asynchronous communication. Such protocols may be vulnerable to super-rushing adversaries if the synchronous part uses point-to-point channels (as opposed to a broadcast channel) and is executed optimistically.

2 Technical Overview

In this section, we highlight some of our results while trying to give some intuition as to what might go wrong and when one can expect that super-rushing does not introduce security vulnerability.

What Might go Wrong? We start with one of our negative results, demonstrating the advantage a super-rushing adversary gains. Specifically, we consider the protocol and the adversary underlying Theorem 1.3, that admits a committal round CR, but for which $\mathsf{ORR} = \mathsf{CR} + 1$.

Consider the following simultaneous broadcast functionality between 5 parties where 2 parties provide inputs (and λ denotes the empty string):

$$f_{\mathsf{sb}}(b_1, b_2, \lambda, \lambda, \lambda) = (y, y, y, y, y) \qquad \text{where} \qquad y = (b_1, b_2).$$

To implement this functionality, we consider a protocol, denoted $\pi_{\mathsf{sb\text{-}vss}}$, in which P_1 and P_2 first use verifiable secret sharing (VSS) to distribute shares of b_1 and b_2, respectively, and then the parties jointly reconstruct those secrets to learn b_1 and b_2. Intuitively, this guarantees independence of inputs assuming synchronous communication and rushing adversaries: each one of the input providers must "commit" to its input before learning the other input. We remark that this is also the standard way to toss a coin: P_1 and P_2 can choose their secrets uniformly at random, commit to them, reveal them, and take a combination of the results.

Gennaro, Ishai, Kushilevitz, and Rabin [37] showed a 1-round VSS for 5 parties secure against one corruption. Specifically, the dealer distributes a secret s using a polynomial with degree 1. In the reconstruction phase, the dealer is not allowed to participate, and each party that receives a share simply forwards it to all others. The parties run Reed-Solomon decoding on the shares received. If there is no unique decoding (meaning that the dealer is cheating), the parties output a default value. Observe that:

- If the dealer is honest, then it must have sent valid shares to all parties, and privacy holds since $t = 1$ (and we assume just one corrupted party). During the reconstruction, the adversary can introduce only one error, which is eliminated using Reed Solomon decoding.
- If the dealer is corrupted and sent only one incorrect share (i.e., provided 3 shares that lie on the same line, and 1 that does not), then reconstruction will be successful. Specifically, since the participants are honest and forward the messages they received from the dealer, then all parties will have just a single error and 3 correct points, and therefore the single error can be eliminated using the Reed Solomon decoding. If the dealer distributed more than a single error, then no party will be able to decode, and all parties will output a default value.

This protocol $\pi_{\mathsf{sb\text{-}vss}}$ indeed perfectly implements the f_{sb} in the presence of a rushing adversary. However, it does not remain secure in the presence of a super-rushing adversary, who corrupts, say, P_1:

- In the first round, the honest P_2 shares its input b_2 to four shares s_1, s_3, s_4, s_5 by giving each party P_i its share s_i, for $i \in \{1, 3, 4, 5\}$.
- After the corrupted P_1 receives a share s_1 from P_2, party P_1 sends to some honest party, say P_3, a random point in the field as its "share."
- At this point, P_3 receives all its round-1 messages: a share from P_1 and a share from P_2, and proceeds to the next round—the reconstruction phase. In this phase, P_3 forwards the two shares it received to everyone (and, in particular, sends s_3 to everyone).
- The adversary P_1 receives s_3, and together with s_1 that it received directly from the dealer, P_2, it can reconstruct b_2.
- Finally, P_1 chooses b_1 as a function of b_2, and provides valid shares for b_1 to all parties. Note that it can even choose the polynomial such that it would even agree with the share it already provided to P_3, seemingly providing no error whatsoever!

The above attack shows that things might go wrong, and the super-rushing adversary can break the "independence of inputs" requirement in secure protocols. It is important to note also that the above protocol does have a "committal round" when considering rushing adversaries—there is a specific round in the protocol (round 1) in which the inputs of all parties (that provide inputs) is well-defined. As we show, the problem is that the "output revealing round" is too close to the committal round, and the ability of super-rushing adversaries that can progress some parties to the next round, enables learning the output *before* the conclusion of the committal round.

Single-Input Functionalities. A specific class of functionalities where "independence of inputs" is not a concern is the case where only one party provides an input. As a warmup towards our general positive result, we consider such functionalities first. We show that for single-input functionalities, any protocol that is secure against non-rushing adversaries, is already secure against super-rushing adversaries.

For simplicity, let us assume in this technical overview that the input provider is honest and holds input x. In the ideal model, the input provider sends its input to the trusted party, the trusted party computes $f(x) = (y_1, \ldots, y_n)$, and gives each honest party P_j its output y_j. The ideal adversary (the simulator) cannot influence this execution in any way. Focus for now just on the correctness of the protocol. The real-world adversary, has this extra capability of "peeking into the future." The question is whether this ability can be used to break the correctness of the protocol, that is, to cause some P_j output a value that is not y_j.

The key reason why super-rushing is not a concern for this class of functionalities, is that we can simulate round $r + 1$ message of every honest party. Specifically, we construct from the super-rushing adversary exponentially many adversaries—one for every possible execution that might occur with this particular super-rushing $\mathcal{A}'$. For every possible combination of random tapes of the honest parties $(\rho_j)_{j \notin I}$ (where I denotes the set of corrupted parties), we construct a rushing adversary $\mathcal{A}[(\rho_j)_{j \notin I}]$. This rushing adversary internally simulates the protocol, with each honest party P_j using ρ_j as its random tape (and no input), while the input provider input x, and interacting with the super-rushing adversary $\mathcal{A}'$. The rushing adversary then constructs all expected incoming and outgoing messages of the protocol in each round for all parties. After generating these messages, $\mathcal{A}[(\rho_j)_{j \notin I}]$ executes the protocol with the real honest parties. If, by chance, the actual honest parties use the same random tapes $(\rho_j)_{j \notin I}$, the adversary knows exactly how to behave and what messages to send in each round. Otherwise, the adversary aborts.

The key insight is that if the super-rushing adversary $\mathcal{A}'$ were to gain any advantage (i.e., break correctness), there would exist some execution where one of these rushing adversaries $\mathcal{A}$ also gains an advantage. However, the latter is impossible because of the *perfect* security of the protocol. The above intuition addresses just correctness, but a similar argument holds for other security properties. The formal proof also addresses a full simulation of the super-rushing adversary and the scenario where the input provider is corrupted. For the full details, we refer the reader to the full version of the paper.

Crucially, the above argument breaks down when there are two input providers. In that case, the ideal world requires this extra property of "independence of inputs." The real-world adversary can peek into the future, learn some information about the input of the honest party, and use this information to correlate its input. Such behavior is possible in the real world but is not allowed in the ideal world.

General Functionalities: Our Main Possibility Result. We now turn our attention to the crux of our contribution, which is a possibility result for general functionalities. We focus in this overview on the simpler case—protocols that admit a committal round CR and an output revealing round ORR, but for which $\mathsf{ORR} > \mathsf{CR} + 1$.

When considering rushing adversaries, the existence of CR and ORR guarantee the following in the real execution:

- **Committal round:** CR is the point in the execution where the inputs of all parties are committed, and the adversary can no longer influence the outputs of the honest parties after that point.
- **Output revealing round:** There is no leakage on the honest parties inputs' until ORR. That is, ORR is the first (and only point) in the execution where the parties learn the outputs, and the adversary learns at that point the information about the honest parties inputs' that is implied from those outputs.

Intuitively, in the real world, the adversary does have some extra power to look into future rounds. However, such protocols provide the adversary $\mathcal{A}'$ with no information on the outputs until ORR. Moreover, the adversary cannot look "too much" into the future, without progressing the protocol and providing messages. When $\mathsf{CR} > \mathsf{ORR} + 1$, the guarantee is that when $\mathcal{A}'$ peeks into round ORR, then $\mathcal{A}'$ must have already provided its inputs in CR, and therefore, $\mathcal{A}'$ cannot influence anything at this point.

To actually simulate the super-rushing adversary $\mathcal{A}'$, we use the simulator of the rushing adversary. The technical difficulty in the proof is that we need to invoke this underlying simulator multiple times on different behaviors of (rushing) adversaries and to show that all the invocations are consistent.

In more detail, the super-rushing adversary $\mathcal{A}'$ might ask for round $r + 1$ messages of some honest party P_j before it actually provided all the corrupted parties' messages for round r. The simulator $\mathcal{S}'$ of $\mathcal{A}'$ then feeds the rushing simulator $\mathcal{S}$ of all round-r messages that were sent from corrupted parties to P_j, and with dummy messages from the corrupted parties to the other honest parties. The simulator $\mathcal{S}'$ obtains back all the round $r + 1$ messages that the honest parties send to the corrupted parties, and extracts from this the messages that P_j sends in round $r + 1$. Based on this information, the adversary might provide messages from corrupted parties to honest parties in round r. We then will have to restart the simulator $\mathcal{S}$ and provide it with updated information of corrupted parties to honest parties in round r. However, since all the messages that P_j has received in round r are identical between the two executions of $\mathcal{S}$, it must provide the exact same outgoing messages in round $r + 1$ for P_j.

Our proof assumes some structure of the underlying simulator $\mathcal{S}$ that is used to simulate rushing adversaries. Nevertheless, we show that such requirements are quite natural and are satisfied for several protocols in the literature. We refer the reader to Sect. 5 for the formal treatment.

Organization. The rest of the paper is organized as follows. After the preliminaries in Sect. 3, we provide some separations between rushing and super-rushing in Sect. 4. The main theorem is given in Sect. 5, and we discuss the statistical setting in Sect. 6. Due to space limitation, we defer some of the proofs to the full version of the paper.

3 Preliminaries

Protocols. An n-party protocol $\pi = (P_1, \ldots, P_n)$ is an n-tuple of PPT interactive Turing machines (ITM). 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 $\rho_i \in \{0,1\}^*$. An *adversary* $\mathcal{A}$ is another ITM describing the behavior of the corrupted parties. We consider malicious, computationally unbounded adversaries, that can deviate from the protocol in an arbitrary way. For simplicity we focus on static corruptions, meaning that the adversary decides on the set of parties to corrupt before the beginning of the execution.[2] That is, the adversary starts the execution with input that contains the identities of the corrupted parties, their private inputs, and an additional auxiliary input z.

3.1 Communication Model

As standard for information-theoretic MPC, we assume that each party can communicate with every other party over an ideally secure communication channel, meaning that the adversary cannot eavesdrop nor alter honest-to-honest communication.

Synchronous Communication. We say that a protocol π is executed over a synchronous communication network if the protocol proceeds in a lock-step manner, round by round. That is, there is a global round counter that is known to all parties and advances in a sequential manner. 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). All parties begin the protocol at round 1, and for every round $r \in \mathbb{N}$, all parties advance from round r to round $r+1$ in a synchronized way in the sense that no party advances to round $r+1$ before all honest parties have completed their round r operations. Specifically, it is guaranteed that every message sent in round r is delivered to its destination, and this is done before any message is sent for round $r+1$. When a party P_i sends a message m to party P_j in round r, we denote the message together with the metadata by (i, j, r, m).

We will also consider protocols that are defined in the broadcast model. Here, in every round each party may send private messages using the point-to-point

[2] This makes our separations stronger, and in the perfect-security setting any statically secure protocol satisfying a few basic requirements is also adaptively secure [7,21,33].

channels, and also broadcast a message that will arrive to all parties in the same round.

In every round, every party can send a message to any other party as a function of its input, its randomness, its history of incoming messages, and the identity of the recipient. For simplicity, and unless stated otherwise, we will consider protocols where in each round there is all-to-all communication, i.e., every party sends a (possibly empty) message to every other party (either over point-to-point channels or via the broadcast channel), and where the number of rounds is fixed and known ahead of time. Our results can be extended to *message-oblivious* protocols [24,25] which means that the decision of whether party P_i should send a message to party P_j in round r is determined by the protocol, regardless of the input or the random tape. We further assume that every corrupted party always sends a message when supposed to (this is a standard assumption for synchronous protocols, as the recipient can identify the event where a party is cheating by not sending a message when it is supposed to, and can pretend receiving a default message instead).

We note that in a synchronous, message-oblivious protocol, each honest party knows how many messages it should receive in a given round and from whom. As such, every honest party can generate its messages for round $r+1$ immediately after receiving all of its messages for round r. The synchrony assumption, however, ensures that those messages are delivered to their destinations only after all round-r messages have been delivered to all honest parties.

Rushing and Non-rushing Adversaries. In the synchronous setting, two common types of adversaries are considered in the literature: *rushing* and *non-rushing*. A rushing adversary has the ability to schedule the order in which messages are delivered within a given round; as such, the corrupted parties can choose their messages to be sent in round r after receiving the messages sent by honest parties to corrupted parties in round r. That is, in round r, the adversary initially obtains all messages of the form $(j, i, r, \cdot)$ for an honest P_j and a corrupted P_i, and based on this information produces the messages $(i, j, r, \cdot)$ for each corrupted P_i to each an honest P_j.

In contrast, a non-rushing adversary must choose the corrupted parties' messages for round r independently of the messages sent by honest parties in round r. That is, in round r, the adversary initially generates all the messages $(i, j, r, \cdot)$ from a corrupted P_i and an honest P_j and only later obtains the messages of the form $(j, i, r, \cdot)$ for every honest P_j to every corrupted P_i.

We emphasize the both for rushing and non-rushing adversaries, it is guaranteed that every message of the form $(i, j, r, \cdot)$ is delivered at its destination before the delivery of any message of the form $(i', j', r+1, \cdot)$.

Super-Rushing Adversaries. We will also consider stronger adversaries that we call *super-rushing*. Similarly to rushing adversaries, a super-rushing adversary can reorder messages within a given round, but, as opposed to rushing adver-

saries, it can also reorder messages *between different rounds*, i.e., deliver certain messages for round $r+1$ before all messages for round r have been delivered.

Effectively, in an execution of a message-oblivious protocol, an honest party can generate its messages for round $r+1$ immediately after receiving all of its messages for round r; a super-rushing adversary can then deliver some of those messages before all honest parties have received all their messages for round r.

The Interface of the Adversary. For concreteness, and to simplify notations in our proofs, we will use the following interface between the adversary and the honest parties:

- The adversary can send a message from a corrupted party to an honest party.
- The adversary can request a message from an honest party to a corrupted party.

We assume that the adversary is well behaved in the following sense. First, the adversary always sends a message from a corrupted party to an honest party when supposed to, and second, if the adversary requests a message corresponding to $(j^*, i^*, r+1)$ for an honest P_{j^*} and a corrupted P_{i^*}, then the following holds:

- If the adversary is non-rushing, then the adversary must have sent all of the messages corresponding to (i, j, r') for every corrupted P_i, honest P_j, and $r' \leq r+1$.
- If the adversary is rushing, then the adversary must have sent all of the messages corresponding to (i, j, r') for every corrupted P_i, honest P_j, and $r' \leq r$.
- If the adversary is super-rushing, then the adversary must have sent all of the messages corresponding to (i, j, r') for every corrupted P_i, honest P_j, and $r' \leq r-1$, and also all of the messages corresponding to (i, j^*, r') for every corrupted P_i and $r' = r$.

We remark that the above requirement for super-rushing is for the case of all-to-all communication. In general, the requirement is that the adversary can request a message $(j^*, i^*, r+1)$ if P_{j^*} has received all the messages expected to arrive by the protocol in round r (recall that in the general case, we assume the protocol is message-oblivious, and so the communication pattern is fixed).

4 Separating Rushing from Super-Rushing

In this section, we show simple protocols that are secure against rushing adversaries but are insecure against super-rushing adversaries. We start by defining the $(2, 5)$-simultaneous-broadcast functionality, followed by two protocols that securely realize it against a rushing adversary corrupting one party but are insecure against a super-rushing adversary. Next, we show that as opposed to security against rushing adversaries that remains secure under sequential composition, security against super-rushing adversaries does not.

4.1 Rushing Security Does not Imply Super-Rushing Security

$(2,5)$ **simultaneous broadcast.** The functionality used in our separations is a variant of simultaneous broadcast [23], which involves five parties. Parties P_1 and P_2 hold input bits b_1 and b_2, respectively, whereas parties P_3, P_4, and P_5 have no input (formally, their input is the empty string λ). All parties learn the pair of bits (b_1, b_2). We emphasize that as opposed to naïve parallel composition of broadcast protocols, if P_i is corrupted for $i \in \{1,2\}$, then its input must remain *independent* of the input of the other honest party P_{3-i}. That is, the functionality to compute is

$$f_{\mathsf{sb}}(b_1, b_2, \lambda, \lambda, \lambda) = (y, y, y, y, y) \qquad \text{where} \qquad y = (b_1, b_2).$$

4.1.1 A Protocol with No Committal Round

We proceed to present the protocol $\pi_{\mathsf{sb\text{-}basic}}$ (Protocol 4.1) that securely realizes f_{sb} in the presence of a rushing adversary but not in the presence of a super-rushing adversary. This is a basic protocol, described in [38], in which each of the parties P_1 and P_2 send their input bits to the non-input parties P_3, P_4, and P_5, who then echo these messages to everyone.

Protocol 4.1. ($\pi_{\mathsf{sb\text{-}basic}}$ **(a basic** $(2,5)$**-simultaneous-broadcast protocol))**

Round 1:

- For $i \in \{1,2\}$, party P_i sends its input bit b_i to P_3, P_4, and P_5.
- For $j \in \{3,4,5\}$ and $i \in \{1,2\}$, denote by b_i^j the bit P_j received from P_i; in case P_j did not receive a message from P_i, or if the message is not a bit, then P_j sets $b_i^j := 0$.

Round 2:

- For $j \in \{3,4,5\}$, party P_j sends the pair of bits (b_1^j, b_2^j) to all parties.
- For $k \in [5]$ and $j \in \{3,4,5\}$, denote by $(b_1^{j\to k}, b_2^{j\to k})$ the pair of bits P_k received from P_j; in case P_k did not receive a message from P_j, or if the message is not a pair of bits, then P_k sets $b_i^{j\to k} := 0$ for $i \in \{1,2\}$.

Output:

- For $k \in [5]$ and $i \in \{1,2\}$, party P_k sets $c_i^k := \mathsf{maj}\{b_i^{3\to k}, b_i^{4\to k}, b_i^{5\to k}\}$.
- For $k \in [5]$, party P_k outputs (c_1^k, c_2^k).

We proceed to state in Lemma 4.2 that the protocol is secure in the presence of a rushing adversary, and refer the reader to the full version for its proof.

Lemma 4.2. *Protocol* $\pi_{\mathsf{sb\text{-}basic}}$ 1*-securely realizes* f_{sb} *with perfect security in the presence of a malicious rushing adversary.*

We proceed to show that $\pi_{\mathsf{sb\text{-}basic}}$ is vulnerable to attacks by a super-rushing adversary that cannot be simulated. Intuitively, a corrupted P_1 can send a round-1 message to P_3 and learn the input of P_2 before it sends the first-round messages to P_4 and P_5. This breaks the independence of inputs in f_{sb}.

Lemma 4.3. *Protocol $\pi_{\text{sb-basic}}$ does not 1-securely realize f_{sb} with perfect security in the presence of a super-rushing adversary.*

Proof. We start by constructing the following super-rushing adversary.

The Super-Rushing Adversary $\mathcal{A}_{\text{sr}}$. The adversary $\mathcal{A}_{\text{sr}}$ corrupts P_1 and proceeds as follows:

1. Send 0 to the party P_3 as the message in round 1.
2. Upon receiving the round-2 message from P_3 of the form $(b_1^{3\to 1}, b_2^{3\to 1})$, send the round-1 message $b_2^{3\to 1}$ to P_4 and P_5.
3. Receive round-2 messages: $(b_1^{4\to 1}, b_2^{4\to 1})$ from P_4 and $(b_1^{5\to 1}, b_2^{5\to 1})$ from P_5, and halt.

The proof of the lemma follows from Claims 4.4 and 4.5 showing that for any simulator $\mathcal{S}$, the distributions of $\text{REAL}_{\pi_{\text{sb-basic}},\{1\},\mathcal{A}_{\text{sr}}}$ and $\text{IDEAL}_{f_{\text{sb}},\{1\},\mathcal{S}}$ are not identical.

Claim 4.4. *Consider an execution of $\pi_{\text{sb-basic}}$ on uniformly sampled inputs $b_1, b_2 \leftarrow \{0,1\}$ with the adversary $\mathcal{A}_{\text{sr}}$ corrupting P_1. Denote by (y_1, y_2) the common output value of the honest parties. Define the event $\mathcal{E}_{\text{real}}$ where $y_1 = y_2$. Then, $\Pr[\mathcal{E}_{\text{real}}] = 1$, where the probability is over the choice of b_1 and b_2 (note that the protocol and the adversary are deterministic).*

Proof. The execution of $\pi_{\text{sb-basic}}$ on (b_1, b_2) with $\mathcal{A}_{\text{sr}}$ corrupting P_1 proceeds as follows:

- P_2 sends b_2 to P_3, P_4, and P_5.
- P_1 sends 0 to P_3.
- P_3 sends $(0, b_2)$ to P_1, P_2, P_4, and P_5.
- P_1 sends b_2 to P_4 and P_5.
- P_4 sends (b_2, b_2) to P_1, P_2, P_3, and P_5.
  Similarly, P_5 sends (b_2, b_2) to P_1, P_2, P_3, and P_4.
- For $k \in \{2,3,4,5\}$, party P_k outputs $c_1 = \mathsf{maj}\{0, b_2, b_2\}$ and $c_2 = \mathsf{maj}\{b_2, b_2, b_2\}$, i.e., outputs (b_2, b_2).

It is always the case that $y_1 = y_2$, and thus $\Pr[\mathcal{E}_{\text{real}}] = 1$ □

Claim 4.5. *Let $\mathcal{S}$ be a arbitrary simulator. Consider an ideal computation of f_{sb} on uniformly sampled inputs $b_1, b_2 \leftarrow \{0,1\}$ with the simulator $\mathcal{S}$ corrupting P_1. Denote by (y_1, y_2) the common output value of the honest parties. Define the event $\mathcal{E}_{\text{ideal}}$ where $y_1 = y_2$. Then, $\Pr[\mathcal{E}_{\text{ideal}}] = \frac{1}{2}$, where the probability is over the choice of b_1 and b_2 and the randomness of $\mathcal{S}$.*

Proof. The ideal computation of f_{sb} on (b_1, b_2) with $\mathcal{S}$ corrupting P_1 proceeds as follows:

- P_2 sends b_2 to the trusted party.
- P_1 sends a bit b to the trusted party.

- The trusted party sends (b, b_2) to all parties.

It follows that for every $b \in \{0,1\}$ it holds that

$$\begin{aligned}\Pr[\mathcal{E}_{\mathsf{ideal}}] &= \Pr[b = b_2] \\ &= \Pr[b = b_2 \mid b_2 = 0] \cdot \Pr[b_2 = 0] + \Pr[b = b_2 \mid b_2 = 1] \cdot \Pr[b_2 = 1] \\ &= 0 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} = \frac{1}{2}.\end{aligned}$$

□

This concludes the proof of Lemma 4.3. □

4.1.2 A Protocol with Committal Round but ORR = CR + 1 A closer look at the protocol $\pi_{\mathsf{sb\text{-}basic}}$ indicates that the protocol does not admit a committal round, i.e., a specific round CR such that the simulator can first simulate the protocol until round CR (including), next extract the corrupted party's input and send it to the trusted party to obtain the output, and then resume the simulation from round $\mathsf{CR} + 1$ till the end. Indeed, in case P_i with $i \in \{1,2\}$ is corrupted, the simulator must extract the corrupted party's input in the first round, whereas if P_j with $j \in \{3,4,5\}$ is corrupted, the simulator must obtain the output to simulate the first round. One may wonder whether the source of the separation between rushing and super-rushing adversaries lies in the lack of a committal round.

Our second separation shows that this is not the case. That is, we show a protocol for $(2,5)$ simultaneous broadcast, $\pi_{\mathsf{sb\text{-}vss}}$, that admits a committal round and is secure against a rushing adversary, yet is not secure against a super-rushing adversary. The protocol is based on the one-round verifiable secret sharing (VSS) from [37] for 5 parties with $t = 1$. The one-round sharing phase is vanilla Shamir sharing—the dealer picks a polynomial of degree 1 (a line), and gives each party one point. The dealer does not participate in the reconstruction phase, and during reconstruction the parties (excluding the dealer) simply exchange their shares. That is, each party receives four shares, and can reconstruct if there is at most one error; in case there is no unique decoding (i.e., if three shares are co-linear, and one share does not lie on the same line), the parties output a default value.

In our protocol, P_1 and P_2 share their input in the first round, and the parties reconstruct the pair of bits in the second round. The protocol is secure against a rushing adversary, and the first round is a committal round. However, as before, a super-rushing adversary can advance a single sender to proceed to round 2 to obtain an additional share on the honest-party's polynomial. That is, it can recover the entire sharing polynomial of the honest party from $t + 1 = 2$ shares, and thereby, the input of the other honest party. Thus, a super-rushing adversary in an execution of the protocol $\pi_{\mathsf{sb\text{-}vss}}$ can violate input independence.

We provide the full details of the protocol in the full version.

4.2 Super-Rushing Security is not Maintained Under Sequential Composition

Sequential composition is a basic requirement of MPC protocols; it enables a simple form of a modular design in the following sense. Given an n-ary functionality f with a protocol π_f that securely realizes f, one can construct another protocol π in the f-hybrid model that securely realizes another function g by invoking the ideal computation of f multiple times, such that the invocations of f are done sequentially and each new invocation of f begins after the previous invocation has completed. The sequential composition operation ensures security of the protocol $\pi^{f \to \pi_f}$ that is obtained by replacing each invocation of f with an execution of π_f. The standard definition of MPC ensures sequential composition in the presence of a rushing adversary [19,39].

In this section, we present a simple example highlighting that security is not maintained under sequential composition in the presence of a super-rushing adversary. The high-level idea is that a super-rushing adversary can learn information related to the second execution of π_f to influence the first execution. We note that the technique of [46] of adding a dummy synchronization round between the executions enables sequential composition; however, as before, our focus is on unmodified protocols.[3] We refer the reader to the full version for the counterexample and the formal theorem statement.

5 Sufficient Conditions for Perfect Security with Super-Rushing Adversary

In this section, we show sufficient conditions under which security in the presence of a rushing adversary implies security in the presence of a super-rushing adversary. Our theorem considers protocols that satisfy some requirements, such as the existence of a committal round. Moreover, it assumes that the protocol can be proven secure against a rushing adversary using a simulator the follows some specific structure. We start with the definitions of the requirements from the simulation.

5.1 Compatible Simulation

Our requirements from the simulation of the rushing adversary are as follows:

[3] We note that a similar issue arises when sequentially composing protocols with probabilistic termination, i.e., protocols in which fast parties may complete the execution before slow parties do, and further a party does not know whether it is fast or slow [28]. Such protocols naturally arise when using randomized Byzantine agreement as a subroutine in an MPC protocol. Sequential composition theorems in this setting indeed add a "local" dummy round in which a party does not communicate and ignores all incoming messages in this round to avoid super-rushing attacks [28,29].

Definition 5.1. *A simulator $\mathcal{S}(z, I, (x_i)_{i\in I}; \rho_{\mathcal{S}})$ for a protocol π simulating a rushing adversary $\mathcal{A}$, where z is the auxiliary input, I is the set of corrupted parties, x_i is P_i's original input for each $i \in I$, and $\rho_{\mathcal{S}}$ is the randomness of $\mathcal{S}$, parameterized by a set $\boldsymbol{\beta} = (\beta_j)_{j\notin I}$ from some domain B, is called* compatible *if it satisfies the following properties:*

- **Simulation via virtual honest parties:** *$\mathcal{S}$ computes the view of the adversary by simulating virtual honest parties with default inputs, and follows the structure as described in Structure 5.2.*
- **Simulation for every default inputs:** *The simulation works for every set of default input $(\beta_j)_{j\notin I}$ in some domain B over the possible inputs of the function. That is, we denote by $\mathcal{S}[\boldsymbol{\beta}]$ the simulator that uses $\boldsymbol{\beta} = (\beta_j)_{j\notin I}$ as default inputs. That is, for every real-world rushing adversary $\mathcal{A}$, and for every $\boldsymbol{\beta}$ it holds that:*

$$\left\{\text{REAL}_{\pi,\mathcal{A}(z),I}(\boldsymbol{x})\right\}_{\boldsymbol{x}\in(\{0,1\}^*)^n} \equiv \left\{\text{IDEAL}_{f,\mathcal{S}[\boldsymbol{\beta}](z),I}(\boldsymbol{x})\right\}_{\boldsymbol{x}\in(\{0,1\}^*)^n}$$

Structure 5.2. (Structure of simulation via virtual honest parties)

1. **Initialization:** $\mathcal{S}$ interacts with the real-world adversary $\mathcal{A}(z, I, (x_i)_{i\in I})$ as an oracle, in a round-by-round fashion. $\mathcal{S}$ initializes simulated honest parties denoted as $(\tilde{P}_j)_{j\notin I}$, and initializes each $\tilde{P}_j$ with default input β_j.
2. **Simulation for rounds** $r = 1, \ldots, \mathsf{CR}$: In each round $r \leq \mathsf{CR}$ the simulator sends to $\mathcal{A}$ all the messages from the honest parties to the corrupted parties, each is computed according to the next-message function of the protocol π on the default input and the simulated incoming messages. The adversary $\mathcal{A}$ then replies with all messages $\{m^r_{i\to j}\}_{j\notin I, i\in I}$ where $m^r_{i\to j}$ is the message from corrupted P_i to an honest $\tilde{P}_j$ in round r.
3. **At the end of round** CR: The simulator sends to the trusted party the input values of the corrupted parties, $\{x'_i\}_{i\in I}$.
4. **Simulation for rounds** $r = \mathsf{CR}+1, \ldots, \mathsf{ORR}-1$: Proceeds as in the simulation of rounds $r = 1, \ldots, \mathsf{CR}$.
5. **At the end of round** $\mathsf{ORR}-1$:
   The simulator receives from the trusted party the output values of the corrupted parties, $\{y_i\}_{i\in I}$.
6. **Simulator for round** $r = \mathsf{ORR}$: Let $J = [n] \setminus I$. For every party $j^* \notin I$, there exists a function SimOut_{j^*} such that:

$$m^{\mathsf{ORR}}_{j^*\to I} = \mathsf{SimOut}_{j^*}\left((y_i)_{i\in I},\quad m^{[\mathsf{ORR}-1]}_{J\to J},\quad m^{[\mathsf{ORR}-1]}_{I\to j^*}\right),$$

   where for $A, B \subseteq [n]$, and $R \subseteq [1, \ldots, \mathsf{ORR}]$, $m^R_{A,B} = (m^r_{a,b})_{a\in A, b\in B, r\in R}$. That is, the messages from P_{j^*} to the corrupted parties in the final round is a function of the outputs of the corrupted parties, the honest-to-honest communication in all rounds up to $\mathsf{ORR}-1$, and the messages P_{j^*} received from the corrupted parties up to round $\mathsf{ORR}-1$.

7. The simulator outputs whatever $\mathcal{A}$ outputs, without loss of generality, its view.

Intuitively, we assume that the simulator works by simulating honest parties with some default inputs (and that the simulation can work with any set of default inputs, as standard in secret-sharing-based simulations), and then at the ORR the simulator can "fix" the results. We highlight one important aspect of SimOut_{j^*}—the function to generate the messages that honest P_{j^*} sends in the last round—this function is based on all honest parties' communication up to round $\mathsf{ORR}-1$, on the outputs of the corrupted parties, and on all the messages P_{j^*} received in round $\mathsf{ORR}-1$; however, the function is *independent* of the messages other honest parties received in round $\mathsf{ORR}-1$. This clearly holds in the real protocol (as incoming messages that honest parties receive in round r will be propagated in outgoing messages in round $r+1$), and we require that it would also hold in the simulation.

Broadcast. For simplicity and to ease the presentation, we present the simulation where the parties use only point-to-point communication and no broadcast. For protocols with broadcast, we remark that SimOut should also obtain all the messages that P_{j^*} received over its point-to-point channels and over the broadcast channel.

5.2 The Main Theorem

We show the following theorem:

Theorem 5.3. *Let f be an n-ary functionality and let π be an n-party protocol with all-to-all communication. Assume that:*

- *π t-securely realizes f with perfect security against rushing adversaries;*
- *π admits a committal round CR, an output-revealing round ORR, and it holds that $\mathsf{ORR} > \mathsf{CR}+1$;*
- *The simulator $\mathcal{S}$ is compatible as per Definition 5.1.*

Then, π t-securely realizes f with perfect security against super rushing adversaries.

5.3 Examples

Before proving Theorem 5.3, we first demonstrate that several well-known secret-sharing-based protocols from the literature meet the conditions stated in Theorem 5.3. We assume the reader has some familiarity with those protocols.

The BGW Protocol. We consider the BGW protocol [15] and its proof of security [8]. The simulator described in [8, Theorem 7.2] simulates the protocol by considering virtual honest parties, assigning them default inputs of 0 (of course, this is arbitrary), and then executing the protocol honestly using those inputs. The committal round is the end of the VSS round, in which the simulator extracts the inputs of the corrupted parties.

VSS. Before proceeding, we provide further details on the VSS. We claim that the shared polynomial can be reconstructed by analyzing the all-to-all communication and the information transmitted over the broadcast channel. To see this, recall that to share a secret s, the dealer selects a bivariate polynomial $S(x, y)$ of degree t in both variables such that $S(0, 0) = s$. The dealer then sends each party P_j a pair of univariate polynomials, $(f_j(x), g_j(y)) = (S(x, j), S(j, y))$. The parties perform verification steps, during which each party P_k receives sub-shares $f_j(k)$ and $g_j(k)$. Finally, After some additional verification steps, the parties vote over the broadcast channel on whether to accept or reject the shares.

If the dealer is honest, the parties will always accept, and from the shares sent to all honest parties, the polynomial $S(x, y)$ of the dealer and the secret s can be determined. Similarly, if the dealer is corrupted and the shares are accepted, there must be at least $t+1$ honest parties that accepted, and their sub-shares can be used to reconstruct the univariate shares, and then the bivariate shares. The shared polynomial can be reconstructed by inspecting the all-to-all communication and the information transmitted over the broadcast channel.

The Circuit Emulation. The protocol maintains an invariant where the value of each wire in the emulated circuit is hidden by a random degree-t polynomial $F(x)$, with each party P_j holding the share $F(j)$. Similar to VSS, other sub-protocols of BGW, such as multiplication, adhere to this structure, where honest-to-honest communication (along with information on the broadcast channel) is sufficient to reconstruct the polynomials that hide the wire values, as well as the shares held by each party (and in fact, the entire adversary's view). It is important to note that in the simulation, the constant term of all internal wires is incorrect.

Simulating the Output Wires. Assume for simplicity that $|I| = t$. When reaching an output wire that is associated with some corrupted P_{i^*}, the simulator can use honest-to-honest communication to reconstruct the polynomial $F_{i^*}(x)$ that hides that output wire. The adversary already knows $F_{i^*}(i)$ for every $i \in I$, i.e., it knows t points on a degree-t polynomial, which still does not fully determine the polynomial. Moreover, the constant term of $F_{i^*}(x)$ in the simulation is incorrect.

At the point, the simulator "corrects" the polynomial by interpolating another degree-t polynomial $\widetilde{F}_{i^*}(x)$ such that $\widetilde{F}_{i^*}(i) = F_{i^*}(i)$ for every $i \in I$, and $\widetilde{F}_{i^*}(0) = y_i$, where y_i is the true output on that wire. Those $t + 1$ points

fully determine this polynomial. Each honest party P_j then sends to P_{i^*} the shares $\widetilde{F}_{i^*}(j)$. The functions SimOut_{j^*} are defined along those lines, while the important property is that the "invariant" of the simulation can be reconstructed from the honest-to-honest communication up until round $\mathsf{ORR} - 1$.

Other Protocols. We remark that the above simulation strategy also holds for other protocols that have a similar structure as BGW, such as [3,9,31], and protocols that are based on multiplication triplets with offline-online phases, e.g., [1,26].

5.4 Proof of Theorem 5.3

Proof. Fix a super-rushing adversary $\mathcal{A}'$. We construct a simulator $\mathcal{S}'$. The simulator is described as two processes that are run in parallel:

1. The interaction with the super-rushing adversary $\mathcal{A}'$; The simulator sends and receives messages to that adversary.
2. The interaction with the rushing simulator $\mathcal{S}$. The simulator $\mathcal{S}'$ (together with $\mathcal{A}'$) behaves as a rushing adversary when interacting with $\mathcal{S}$. Since we know exactly how $\mathcal{S}$ simulates the view of the adversary, we do not use it to generate messages; rather, we use it to extract the effective inputs from $\mathcal{A}'$.

The Super-Rushing Simulator $\mathcal{S}'(z, I, (x_i)_{i\in I})$. We now construct the simulator $\mathcal{S}'$ for the super-rushing adversary $\mathcal{A}'$. The simulator receives as input the auxiliary input z, the set of corrupted parties I, and the initial inputs of the corrupted parties. The simulator proceeds as follows:

1. Initialize $M_1, \ldots, M_{\mathsf{ORR}} \leftarrow \emptyset$, and output values $(y_i)_{i\in I} = \perp$.
2. Sample randomness $\rho_{\mathcal{S}}$ for the invocations of the underlying simulator $\mathcal{S}$, and invoke the simulator $\mathcal{S}(z, I, (x_i)_{i\in I}; \rho_{\mathcal{S}})$.
3. Invoke the super-rushing adversary $\mathcal{A}'(z, I, (x_i)_{i\in I})$.
4. $\mathcal{S}'$ invokes the simulated honest parties $\widetilde{P}_j$ for every $j \notin I$ with some default inputs. Extract the randomness for each simulator honest $\widetilde{P}_j$ from $\rho_{\mathcal{S}}$ in a similar way as $\mathcal{S}$ works.
5. **Interaction with $\mathcal{A}'$ for every** $r = 1, \ldots, \mathsf{ORR}$: Send and receive messages from $\mathcal{A}'$ as follows:
   Send messages to $\mathcal{A}'$: Whenever the adversary $\mathcal{A}'$ asks to receive the message (j^*, i^*, r) for $j^* \notin I$ and $i^* \in I$:
   (a) If there exists a message $(j^*, i^*, r, m^r_{j^*\to i^*}) \in M_r$, then give $m^r_{j^*,i^*}$ to $\mathcal{A}'$ as the result and break.
   (b) If there is no message $(j^*, i^*, r, m^r_{j^*\to i^*})$ in M_r, then verify that all messages $(i, j^*, r-1)$ are in M_{r-1} for all $i \in I$ (i.e., the adversary provided all the messages from the corrupted parties to P_{j^*} in round $r-1$; when $r = 1$ this holds vacuously) and all messages $(j, j^*, r-1)$ are also in M_{r-1} (again, for $r = 1$ this holds vacuously). If this does not hold, call this event $\mathsf{notSuperRushing}$ and abort.

(c) If $r = 1, \ldots, \mathsf{ORR} - 1$ and there is no message $(j^*, i^*, r, m^r_{j^* \to i^*})$ in M_r: Invoke the next-message function of the virtual honest party P_{j^*} on all its incoming messages at round $r - 1$. Add all the messages to M_r.

(d) If $r = \mathsf{ORR}$ and there is no message $(j^*, i^*, r, m^r_{j^* \to i^*})$ in M_r:

i. Verify that $(y_i)_{i \in I} \neq \perp$ are stored (see Step 6b). If not, call this event noOutputs and abort.

ii. Verify that all the messages from honest parties to honest parties are stored in all $M_1, \ldots, M_{\mathsf{ORR}-1}$. If not, call this event notReadyForORR and abort.

iii. Compute:

$$m^{\mathsf{ORR}}_{j^* \to I} = \mathsf{SimOut}_{j^*}\left((y_i)_{i \in I}, \quad m^{[\mathsf{ORR}-1]}_{J,J}, \quad m^{[\mathsf{ORR}-1]}_{I,j^*}\right).$$

Add those messages to M_{ORR}.

(e) Give to $\mathcal{A}'$ the message $(j^*, i^*, r, m^r_{j^* \to i^*})$ that was just generated.

Receive messages from the adversary: Whenever $\mathcal{A}'$ sends a message $m^r_{i,j}$ (a message from corrupted P_i to honest P_j in round r add the tuple $(i, j, r, m^r_{i,j})$ to M_r.

6. **Interaction with $\mathcal{S}$ for every** $r = 1, \ldots, \mathsf{ORR}$:

(a) Once all the messages $(k, \ell, r, \cdot) \in M_r$ for all $k \in [n]$ and $\ell \in [n]$:

i. Receive from $\mathcal{S}$ all the messages from the honest parties to the corrupted parties in round r. Verify that all the received messages are exactly the same in M_r. Otherwise, call this event notConsistent, and abort.

ii. Output all the messages from the corrupted parties to the honest parties that are in M_r.

(b) In round $\mathsf{CR} + 1$: In addition to the messages from the honest parties to the corrupted parties for round $\mathsf{CR} + 1$, the simulator $\mathcal{S}$ also outputs the extracted inputs $(x'_i)_{i \in I}$. Send $(x'_i)_{i \in I}$ to the trusted party as the effective inputs of the corrupted parties, and receive back $(y_i)_{i \in I}$. Store those values.

7. Output whatever $\mathcal{A}'$ outputs, and halt.

We now show that the simulator is valid. We have the following claims, which are proven in the full version.

Claim 5.4. *The following holds during an execution of a simulator $\mathcal{S}'$:*

1. *$\mathcal{S}'$ never aborts due to* noSuperRushing *event in Step 5b.*
2. *$\mathcal{S}'$ never aborts due to* noOutputs *event in Step 5(d)i.*
3. *$\mathcal{S}'$ never aborts due to* notReadyForORR *event in Step 5(d)ii.*
4. *$\mathcal{S}'$ never aborts due to* notConsistent *event in Step 6(a)i.*

The simulator $\mathcal{S}'$ perfectly simulates $\mathcal{A}'$. We now show that the outputs of the real and ideal are identically distributed. We prove this via the following hybrid experiments:

1. HYB_0: This is the ideal execution; The output of this execution is the view of the adversary $\mathcal{A}'$ and the output of the honest parties in the ideal world.
2. HYB_1: In this hybrid experiment, the simulator $\mathcal{S}'$ receives from the trusted party the inputs of the honest parties $(x_j)_{j \notin I}$ as input. Then, we modify both $\mathcal{S}'$ and $\mathcal{S}$, and use the real inputs $(x_j)_{j \notin I}$ instead of default inputs for the simulator honest parties $\widetilde{P}_j$ for every $j \notin I$.
3. HYB_2: We change the generation of the messages $m_{j \to i}^{\mathsf{ORR}}$ for every $j \notin I$ and $i \in I$ inside $\mathcal{S}'$ and $\mathcal{S}$. Instead of generating them according to the simulator using SimOut, we generate each $m_{j \to i}$ using the next-message function of P_j on all its incoming messages in round $\mathsf{ORR} - 1$ (and its private input and private state).
4. HYB_3: We change the outputs of the honest parties. Instead of using the outputs as generated by the trusted party, we use the outputs as generated by the simulated honest parties executed by $\mathcal{S}$.
5. HYB_4: This is the real execution: The output is the view of the adversary $\mathcal{A}'$ and the outputs of the honest parties.

We will conclude the proof by showing that any two consecutive hybrids are identically distributed. See full details in the full version. □

5.5 Extensions

In this section, we show extensions of Theorem 5.3. Before proceeding, we note that our definition of a committal round CR and an output revealing round ORR, implicitly requires that $\mathsf{ORR} > \mathsf{CR}$.

Input Committal Synchronization. Suppose that we have a protocol π that securely realizes a functionality f with compatible simulation but for which $\mathsf{ORR} = \mathsf{CR} + 1$. Consider the following protocol π' that is identical to π with the following instruction added for each party after the committal round CR:

1. If all round CR messages of the protocol π are received, then send ready to all the parties.
2. Proceed to round $\mathsf{CR} + 1$ of π only after receiving ready from all parties.

This results in a protocol where $\mathsf{ORR} > \mathsf{CR} + 1$. We term this transformation, $\mathsf{input\ committal\ synchronization}$ and state the following corollary.

Corollary 5.5. *Let f be an n-ary functionality, and suppose that the protocol π satisfies all the requirements of Theorem 5.3, except for $\mathsf{ORR} > \mathsf{CR}+1$. Then, the protocol π' with $\mathsf{input\ committal\ synchronization}$ securely realizes f with perfect security against super-rushing adversaries.*

Committal Round Over Broadcast (a special case of [15]). Consider the BGW protocol for the computation of a linear function. This protocol satisfies all of our sufficient conditions except the condition that $\mathsf{ORR} > \mathsf{CR} + 1$. To see this, observe that the BGW protocol for the computation of linear functions is as follows:

1. Each party with an input, executes a VSS of its input.
2. The parties perform a local computation on the shares they have received from all the VSS instances to obtain shares on the output wires.
3. The parties execute a reconstruction of the outputs by reconstructing the sharings on the output wires.

The protocol admits a committal round (the last round of the VSS) and an output revealing round that immediately follows the committal round. Hence, for this protocol, $\mathsf{ORR} = \mathsf{CR} + 1$.

We can derive the security of BGW for linear functions against super-rushing adversaries using Corollary 5.5 at the cost of incurring an additional round. However, we observe that the committal round CR of this protocol is over the broadcast channel. Therefore, if an honest party completes CR, i.e., if the adversary provided all messages to some honest P_{j^*} and requires to see its round $\mathsf{CR}+1$ messages, then the view of all other honest parties P_j in round $\mathsf{CR}+1$ is also determined and must be the same set of messages. We show the proof of the following corollary in the full version:

Corollary 5.6. *Let f be an n-ary functionality, and suppose that π satisfies all the requirements of Theorem 5.3, except for $\mathsf{ORR} > \mathsf{CR}+1$, however, CR occurs over the broadcast channel. Then, the protocol π securely realizes f with perfect security against super-rushing adversaries.*

6 A Separation for the Statistical Setting

Let f be an n-party functionality and let π be a protocol that satisfies all our sufficient conditions in Theorem 5.3, except that π achieves only statistical security against rushing adversaries. We show that security against super-rushing adversaries is not always implied by security against rushing adversaries.

To that end, consider the BGW protocol for $n = 5$ parties with $t = 1$ corruptions (which satisfies all our sufficient conditions, including perfect security) with the following modifications:

1. The parties first execute an instance of the $(2, 5)$ simultaneous broadcast protocol $\pi_{\mathsf{sb\text{-}basic}}$ (from Sect. 4.1.1) on uniformly random inputs $r_1, r_2 \leftarrow \{0,1\}^\kappa$, where κ denotes the statistical security parameter. Let (y_1, y_2) denote the output of the parties in this phase.
2. If $y_1 = y_2$, the parties reveal their private inputs.
3. The parties then execute the BGW protocol on their actual inputs.

The above protocol securely computes f with perfect security against rushing adversaries conditioned on $y_1 \neq y_2$. From Lemma 4.2, we have that $\pi_{\mathsf{sb\text{-}basic}}$ is perfectly secure in the presence of a rushing adversary. Therefore, the input r_1 of P_1 is independent of the input r_2 of P_2, and vice versa, even if one of them is cheating. As a result, $\mathsf{Pr}[y_1 = y_2] \leq 2^{-\kappa}$ in the presence of a rushing adversary and the modified BGW protocol securely realizes f with statistical security in the presence of a rushing adversary. In fact, the simulator can completely ignore the

bad event in the simulation, and the statistical distance would be $2^{-\kappa}$. As such, the modified protocol still admits a committal round and an output revealing round.

From Claim 4.4, we have that, in the presence of a super-rushing adversary in the protocol, $\mathsf{Pr}[y_1 = y_2] = 1$. As a result, in the real execution of the above protocol, there exists a super-rushing adversary that learns the inputs of the honest parties before the execution of the modified BGW protocol. Such behavior cannot be simulated by any simulator in the ideal execution and therefore, the above protocol is not secure against a super-rushing adversary.

References

1. Abraham, I., Asharov, G., Patil, S., Patra, A.: Detect, pack and batch: perfectly-secure MPC with linear communication and constant expected time. In: EUROCRYPT '23, Part II. vol. 14005, pp. 251–281 (2023)
2. Abraham, I., Asharov, G., Patil, S., Patra, A.: Perfect asynchronous MPC with linear communication overhead. In: EUROCRYPT '24, Part V. vol. 14655, pp. 280–309 (2024)
3. Abraham, I., Asharov, G., Yanai, A.: Efficient perfectly secure computation with optimal resilience. In: TCC '21, pp. 66–96 (2021)
4. Appan, A., Chandramouli, A., Choudhury, A.: Network agnostic perfectly secure multiparty computation against general adversaries. IEEE Trans. Inf. Theory **71**(1), 644–682 (2025)
5. Applebaum, B., Brakerski, Z., Tsabary, R.: Degree 2 is complete for the round-complexity of malicious MPC. In: EUROCRYPT '19, Part II. vol. 11477, pp. 504–531 (2019)
6. Applebaum, B., Kachlon, E., Patra, A.: The round complexity of perfect MPC with active security and optimal resiliency. In: FOCS '20, pp. 1277–1284 (2020)
7. Asharov, G., Cohen, R., Shochat, O.: Static vs. adaptive security in perfect MPC: a separation and the adaptive security of BGW. In: ITC '22. vol. 230, pp. 15:1–15:16 (2022)
8. Asharov, G., Lindell, Y.: A full proof of the BGW protocol for perfectly secure multiparty computation. J. Cryptol. **30**(1), 58–151 (2017)
9. Asharov, G., Lindell, Y., Rabin, T.: Perfectly-secure multiplication for any $t<n/3$. In: CRYPTO '11, pp. 240–258 (2011)
10. Bacho, R., Collins, D., Liu-Zhang, C., Loss, J.: Network-agnostic security comes (almost) for free in DKG and MPC. In: CRYPTO '23, Part I. vol. 14081, pp. 71–106 (2023)
11. Baum, C., David, B., Dowsley, R., Nielsen, J.B., Oechsner, S.: TARDIS: A foundation of time-lock puzzles in UC. In: EUROCRYPT '21, Part III. vol. 12698, pp. 429–459 (2021)
12. Beaver, D.: Foundations of secure interactive computing. In: CRYPTO '91. vol. 576, pp. 377–391 (1991)
13. Beerliová-Trubíniová, Z., Hirt, M., Nielsen, J.B.: On the theoretical gap between synchronous and asynchronous MPC protocols. In: PODC '10, pp. 211–218 (2010)
14. Ben-Or, M., Canetti, R., Goldreich, O.: Asynchronous secure computation. In: STOC '93, pp. 52–61 (1993)

15. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: STOC '88, pp. 1–10 (1988)
16. Ben-Or, M., Kelmer, B., Rabin, T.: Asynchronous secure computations with optimal resilience (extended abstract). In: PODC '94, pp. 183–192 (1994)
17. Blum, E., Katz, J., Loss, J.: Synchronous consensus with optimal asynchronous fallback guarantees. In: TCC '19, Part I. vol. 11891, pp. 131–150 (2019)
18. Blum, E., Zhang, C.L., Loss, J.: Always have a backup plan: fully secure synchronous MPC with asynchronous fallback. In: CRYPTO '20, Part II. vol. 12171, pp. 707–731 (2020)
19. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptol. **13**(1), 143–202 (2000)
20. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: FOCS '01, pp. 136–145 (2001)
21. Canetti, R., Damgård, I., Dziembowski, S., Ishai, Y., Malkin, T.: On adaptive vs. non-adaptive security of multiparty protocols. In: EUROCRYPT '01, pp. 262–279 (2001)
22. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: STOC '88, pp. 11–19 (1988)
23. Chor, B., Goldwasser, S., Micali, S., Awerbuch, B.: Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In: FOCS '85, pp. 383–395 (1985)
24. Chor, B., Kushilevitz, E.: A communication-privacy tradeoff for modular addition. Inf. Process. Lett. **45**(4), 205–210 (1993)
25. Chor, B., Merritt, M., Shmoys, D.B.: Simple constant-time consensus protocols in realistic failure models (extended abstract). In: PODC '85, pp. 152–162 (1985)
26. Choudhury, A., Patra, A.: An efficient framework for unconditionally secure multiparty computation. IEEE Trans. Inf. Theory **63**(1), 428–468 (2017)
27. Cohen, R.: Asynchronous secure multiparty computation in constant time. In: PKC '16, Part II. vol. 9615, pp. 183–207 (2016)
28. Cohen, R., Coretti, S., Garay, J.A., Zikas, V.: Probabilistic termination and composability of cryptographic protocols. J. Cryptol. **32**(3), 690–741 (2019)
29. Cohen, R., Coretti, S., Garay, J.A., Zikas, V.: Round-preserving parallel composition of probabilistic-termination cryptographic protocols. J. Cryptol. **34**(2), 12 (2021)
30. Coretti, S., Garay, J.A., Hirt, M., Zikas, V.: Constant-round asynchronous multiparty computation based on one-way functions. In: ASIACRYPT '16, Part II. vol. 10032, pp. 998–1021 (2016)
31. Cramer, R., Damgård, I., Maurer, U.M.: General secure multi-party computation from any linear secret-sharing scheme. In: EUROCRYPT '00. vol. 1807, pp. 316–334 (2000)
32. Damgård, I., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: CRYPTO '07. vol. 4622, pp. 572–590 (2007)
33. Damgård, I., Nielsen, J.B.: Adaptive versus static security in the UC model. In: Chow, S.S.M., Liu, J.K., Hui, L.C.K., Yiu, S.M. (eds.) ProvSec 2014. LNCS, vol. 8782, pp. 10–28. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12475-9_2
34. Dodis, Y., Micali, S.: Parallel reducibility for information-theoretically secure computation. In: CRYPTO '00. vol. 1880, pp. 74–92 (2000)
35. Dwork, C., Lynch, N.A., Stockmeyer, L.J.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)

36. Fischer, M.J., Lynch, N.A., Paterson, M.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985)
37. Gennaro, R., Ishai, Y., Kushilevitz, E., Rabin, T.: The round complexity of verifiable secret sharing and secure multicast. In: STOC '01, pp. 580–589 (2001)
38. Gennaro, R., Ishai, Y., Kushilevitz, E., Rabin, T.: On 2-round secure multiparty computation. In: CRYPTO '02. vol. 2442, pp. 178–193 (2002)
39. Goldreich, O.: The Foundations of Cryptography - Volume 2: Basic Applications. Cambridge University Press (2004)
40. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: STOC '87, pp. 218–229 (1987)
41. Goyal, V., Liu, Y., Song, Y.: Communication-efficient unconditional MPC with guaranteed output delivery. In: CRYPTO '19, Part II, pp. 85–114 (2019)
42. Hirt, M., Nielsen, J.B., Przydatek, B.: Cryptographic asynchronous multi-party computation with optimal resilience (extended abstract). In: EUROCRYPT '05. vol. 3494, pp. 322–340 (2005)
43. Hofheinz, D., Müller-Quade, J.: A synchronous model for multi-party computation and the incompleteness of oblivious transfer. IACR Cryptol. ePrint Arch, p. 16 (2004). http://eprint.iacr.org/2004/016
44. Kalai, Y.T., Lindell, Y., Prabhakaran, M.: Concurrent composition of secure protocols in the timing model. J. Cryptol. **20**(4), 431–492 (2007)
45. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: TCC '13. vol. 7785, pp. 477–498 (2013)
46. Kushilevitz, E., Lindell, Y., Rabin, T.: Information-theoretically secure protocols and security under composition. In: STOC '06, pp. 109–118 (2006)
47. Lamport, L., Shostak, R.E., Pease, M.C.: The Byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
48. Liu-Zhang, C., Maurer, U.: Synchronous constructive cryptography. In: TCC '20, Part II. vol. 12551, pp. 439–472 (2020)
49. Micali, S., Rogaway, P.: Secure computation (abstract). In: CRYPTO '91. vol. 576, pp. 392–404 (1991)
50. Micali, S., Rogaway, P.: Secure computation (unpublished manuscript) (1992), preliminary version in [49]. (All references within refer to the unpublished manuscript.)
51. Nielsen, J.B.: On Protocol Security in the Cryptographic Model. Ph.D. thesis, University of Aarhus (2003)
52. Pease, M.C., Shostak, R.E., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)
53. Yao, A.C.: How to generate and exchange secrets (extended abstract). In: FOCS '86, pp. 160–164 (1986)