Round-Preserving Compilers for Super-Rushing Secure MPC

Michele Ciampi[1], Divya Ravi[2], and Mingrui Zou[2]

[1] University of Edinburgh;
[2] University of Amsterdam, Netherlands;

Abstract. Practical implementations of synchronous MPC protocols typically require each party to advance to the next round as soon as they have received all expected messages. This deviates from the theoretical synchronous round-based model, where instead each party advances in the next round after a timeout. To capture this gap between theory and practice, Asharov, Chandramouli, Cohen and Ishai in Eurocrypt 2025 proposed a new model where the adversary is *super-rushing*. In this, the adversary can see future messages of some honest parties before delivering current-round messages to slower ones. In this work, we study super-rushing security in both the computational and statistical settings, and design round-preserving compilers that transform standard synchronous MPC protocols into ones secure against super-rushing adversaries. Ours is the first work to investigate the security of computational MPC protocols against a super-rushing adversary.

Keywords: Multi-party Computation · Super-Rushing · Round Complexity

1 Introduction

Secure Multi-Party Computation (MPC) [Yao86,CDvdG88,GMW87,BGW88] allows a set of mutually distrusting parties to jointly compute a function on their inputs while achieving both *correctness* (i.e. the output computed is the correct function output) as well as *privacy* (i.e. nothing beyond the output of the function is revealed) simultaneously. Applications of MPC range from privacy-preserving healthcare data analysis [BCB+24] to enabling banks to collaboratively detect fraud across customer datasets without exposing individual financial details [SvHA+19].

Since its introduction in the 1980s, MPC has evolved from a theoretical notion to an active research line of interest to both theoreticians and practitioners [Lin20]. With tremendous efficiency improvements enabling real-world deployments of MPC [BCD+09,BLV17,BLW08], there is now a growing emphasis on modeling more realistic adversaries and ensuring that the underlying theory accurately reflects practical implementations.

One mismatch between theoretical security framework and practical deployments, highlighted by [ACCI25], concerns the underlying network model. Most MPC protocols are designed for a *synchronous* network, where all parties are assumed to be synchronized and start the protocol simultaneously. It is further assumed that there is a known upper bound on message delivery time, which lets the protocol proceed in clearly defined rounds. In the standard security framework for synchronous protocols, an (often implicit) requirement is that even if an honest party receives all messages for round r early, it must wait until the round officially ends before sending its messages for round $r + 1$.

In practice, choosing this message delivery time bound is difficult: if it is set too small, security may be violated, while setting it too large prevents implementations from exploiting fast networks. In fact, some real-world implementations overlook this requirement of waiting until the end of the round. Instead, parties are instructed to proceed optimistically and send their round-$(r + 1)$ messages as soon as they have received all round-r messages, without waiting for the end of the round. This leads to a significant gap between the assumptions used in security proofs and how protocols are actually run. The effect is that in practical deployments some honest parties may move ahead faster than others. An adversary can then observe the next-round messages sent by these faster honest parties and use this information to choose or adapt its own previous-round messages to the slower honest parties.

Asharov et al. [ACCI25] capture this behavior by introducing the notion of a *super-rushing adversary*. In this model, the adversary can first see (i.e., 'peek into the future') some honest parties' round-$(r+1)$ messages *before* choosing its own round-r messages to other honest parties. This differs from the well-studied *rushing*

adversary model in the MPC literature, where the adversary may choose its outgoing round-r messages after seeing the incoming messages in the same round r. The super-rushing adversary is strictly more powerful than a rushing adversary, as it lets the adversary choose its round-r messages based not only on the round-r messages of honest parties, but also on the round-$(r+1)$ messages of those honest parties that have already advanced to the next round.

A natural way to sidestep this timeout dilemma altogether is to instead design protocols for the *asynchronous* network model, where messages may be delayed arbitrarily and parties advance as soon as they have received what they need for the current step. This avoids the problem above, but at a cost: asynchronous protocols must settle for weaker guarantees than their synchronous counterparts, e.g., tolerating fewer corruptions and allowing up to t honest parties' inputs to be excluded from the computation [BOCG93,BOKR94].

The super-rushing model allows capturing the best of both worlds. Parties still keep a fixed per-round timeout as a fallback, but may ignore it and advance as soon as they have received all the messages they need. When the network is fast, the protocol runs as fast as messages actually arrive; when it is not, it degrades gracefully to a synchronous protocol that simply waits for the timeout.

1.1 Our Contribution

A natural question is whether existing MPC protocols that are designed for and proven secure against rushing adversaries are already secure against the stronger super-rushing adversaries. Asharov et al. [ACCI25] give a two-fold answer to this question. First, they show that rushing security does not in general imply super-rushing security, by providing counterexamples of protocols that are secure against rushing adversaries but not against super-rushing adversaries. Second, they prove that if a perfectly secure synchronous protocol has a certain structure, namely that the output reconstruction round is at least two rounds after the round in which inputs are committed, then that protocol is secure against super-rushing adversaries.

While this largely settles the question in the setting of perfect security, it leaves open the study of statistically and computationally secure super-rushing protocols. This naturally leads to the following question:

Can we design statistically and computationally secure MPC protocols that are secure against super-rushing adversaries?

Since the super-rushing model is motivated by its alignment with practical deployments, it is further important to understand whether this can be achieved while retaining efficiency (in terms of round, communication and computation complexity) comparable to protocols secure against standard rushing adversaries. In other words, *can we hope to get super-rushing security for free?*

We show that we can in fact get super-rushing security (almost) for free in certain settings. In more detail, we design efficient *round-preserving* compilers that transform protocols secure against rushing adversaries into protocols secure against super-rushing adversaries. Our compilers work for any corruption threshold, covering both honest and dishonest-majority settings. More precisely,

1) Broadcast-Only Protocols (Section 3). First, we observe that any broadcast-only protocol that is secure against rushing adversaries is already secure against super-rushing adversaries. Intuitively, the extra power of a super-rushing adversary only appears in the presence of point-to-point communication: once an adversarial message has been broadcast, it cannot be changed after the adversary observes next-round messages from honest parties. As in much of the MPC literature (in particular, constant-round MPC), these results assume an ideal broadcast channel.

By an *ideal broadcast channel*, we mean an *atomic, simultaneous* broadcast: a party's round-r broadcast message is fixed and delivered identically to every party in one indivisible step. It remains an interesting open question whether one can instead realize such a broadcast channel from point-to-point channels while retaining security against super-rushing adversaries. Notably, simultaneous broadcast protocols realized from point-to-point channels are not, in general, secure against super-rushing adversaries [ACCI25].

Next, we turn to n-party MPC protocols that use point-to-point communication and obtain the following results.

2) Round-Preserving Information-Theoretic Compiler with Selective Abort (Section 4). Our main result is an information-theoretic compiler that, given any R-round statistically (resp. computationally) secure MPC protocol against rushing adversaries, produces an R-round statistically (resp. computationally)

secure protocol against super-rushing adversaries. The compiled protocol achieves security with selective abort (privacy holds, but the adversary can prevent some of the honest parties from receiving the output). The computational overhead is minimal, as the compiler itself is purely information-theoretic, relying only on additive secret sharing and information-theoretic message authentication codes. Regarding communication, our compiled protocol has total communication complexity $\mathcal{O}(n^2 \cdot CC)$, where CC denotes the total communication complexity (expressed as number of field elements) of the underlying rushing protocol.

A key feature of our compiler is that it preserves the communication pattern of the underlying protocol: if a given round of the original protocol does not use broadcast, then the corresponding round of the compiled protocol does not use broadcast either. Furthermore, the compiler does not introduce any additional setup: if the underlying protocol is in the plain model, then so is the compiled protocol.

3) Round-Preserving Compiler for Unanimous and Identifiable Abort (Section 5). First, using digital signatures, we upgrade security to unanimous abort (UA), where either all honest parties obtain the output or none do. This transformation preserves the setup and round complexity of the underlying protocol. In terms of computational tools, we use digital signatures. The total communication complexity of our compiled protocol is $\mathcal{O}(n \cdot CC)$ (where CC denotes the total communication complexity of the underlying rushing protocol), with an additional overhead of $\mathcal{O}(n \cdot CC)$ signatures, each authenticating a single field element.

Next, we further upgrade to identifiable abort (IA), where, in the event of an abort, the honest parties can identify at least one corrupt party. In the dishonest-majority setting, this compiler requires a CRS setup; in the honest-majority setting, no additional setup is needed. The round complexity is preserved, and the construction uses non-interactive key exchange (NIKE) together with a NIZK proof (to certify the correctness of the NIKE computation) per party per round, whose size is independent of the circuit being computed. The communication *overhead* of our compiled IA protocol is $\mathcal{O}(n \cdot CC')$ as well as NIZK proofs on $\mathcal{O}(n \cdot CC')$ statements (each of constant size). Here, CC' refers to the communication complexity of Round 2 of the underlying rushing protocol (assumed to be using only broadcast from Round 2 onwards). We leave open the question of improving the communication and computation overhead of our IA compiler while preserving the round complexity.

4) Round-Optimality. It is known that there is a separation between non-rushing versus rushing adversaries in terms of round optimal MPC [Kat07,AKP22]. Here, non-rushing adversaries are not allowed to see the honest parties' messages from the current round before sending their own, whereas rushing adversaries are. It is thus natural to ask *'Is there a separation between rushing and super-rushing adversaries in terms of round optimal MPC?'*

We show that for *computationally* secure MPC, there is no separation between the two notions in terms of optimal round complexity. This follows from the fact that (i) lower bounds for the round complexity of MPC against rushing adversaries automatically extend to super-rushing adversaries, as super-rushing adversaries are strictly stronger and (ii) our compilers are *round preserving*. Taken together, this settles the question of the optimal round complexity of computational MPC with super-rushing security.

Table 1 summarizes our bounds in the computational setting. Establishing the optimal round complexity for information-theoretic MPC against super-rushing adversaries is an interesting direction for future work. We provide a partial answer to this only for selective abort (as our selective abort compiler preserves statistical security), while the corresponding question for other security notions remains open.

1.2 Technical Overview

We adopt the same model of super-rushing adversaries as [ACCI25], where the protocols are assumed to have an all-to-all communication pattern; under this assumption, the adversary can peek into at most one round in the future (see Section 2.1 for the formal definition). First, we note that if a PKI can be established, then any rushing protocol can be changed to a broadcast-only protocol (simply by encrypting the private messages with the public key of the recipient and broadcasting the encryption). This transformation would already yield a protocol secure against super-rushing adversaries. Indeed, as noted above, broadcast-only protocols are inherently secure against super-rushing adversaries: once a party has sent its message via a broadcast channel, it is effectively committed and cannot change that message after seeing other parties' next-round messages, so looking ahead provides no additional advantage.

Table 1: Round-Optimal computational MPC protocols against super-rushing adversaries. For each setting, we denote the round complexity, the round-optimal rushing protocol, and our relevant round-preserving compiler that upgrades it to super-rushing security. Arrows indicate implication between security guarantees.

Threshold	Setup	Selective Abort (SA)	Unanimous Abort (UA)	Identifiable Abort (IA)	Fairness (FAIR)	Guaranteed Output Delivery (GOD)
$t < n$	Plain	4 [CCG$^+$20,CDR$^+$23] Thm. 2	4 [CCG$^+$20,CDR$^+$23] Cor. 1	4 [CRSW22] Thm. 1	✗ [Cle86]	✗ [Cle86]
	CRS	2 [GS18,BL18,CGZ20] Thm. 2	2 [GS18,BL18,CGZ20] Cor. 1	2 [CGZ20] Thm. 3		
	PKI	↓	↓	↓		
$t < n/2$	Plain	2 [ACGJ18,ABT19][ACGJ19]* Thm. 2	2 [ACGJ18,ABT19] Cor. 1	3[ACGJ18] Cor. 2	←	3 [ACGJ18] Thm. 1
	CRS	2 [CGZ20] Thm. 2	2 [CGZ20] Cor. 1	2 [CGZ20] Thm. 3	←	3 [GLS15] Thm. 1
	PKI	↓	↓	2 [DMR$^+$21] Thm. 3	←	2 [GLS15] Thm. 1

* achieves statistical (information-theoretic) security restricted to NC^1 circuits.

This approach however has several downsides: (1) It is either not round preserving, because setting up the public keys requires an additional initial broadcast round, or it requires PKI as a form of setup (2) Routing all communication via broadcast incurs a significant efficiency overhead (3) Finally, relying on PKI inherently gives only computational security. Our selective-abort compiler avoids these drawbacks. It (a) preserves the round complexity of the underlying protocol, (b) maintains the same communication pattern (for example, if the original protocol does not use broadcast in a particular round, neither does the compiled one), and (c) achieves information-theoretic security.

The main difficulty in designing a compiler upgrading from rushing to super-rushing security is that we can no longer rely directly on the security of the underlying rushing-secure protocol.

Indeed, such a protocol provides no guarantees once the adversary can peek at next-round messages of honest parties before completing its current-round communication with all the other honest parties. Our approach is therefore to make such peeked messages 'useless' to the adversary. At a high level, we modify the starting protocol to prevent the adversary from seeing the i-th round message of an honest party before having sent all the messages in the $(i-1)$-th round.

In more detail, instead of sending messages in the clear, parties send them encrypted under one-time pads. Each one-time pad is additively secret-shared among all parties, with the shares exchanged in the first round over point-to-point channels. Now, even if the adversary gains early access to say Round $(r+1)$ messages of some honest parties before sending its Round r messages to all, these Round $(r+1)$ honest messages are masked and thus yield no useful information until the adversary can unmask them. Unmasking is possible only after obtaining all additive shares, but since an honest party gives its additive share if it has received the previous round messages from the adversary; it forces the adversary to have completed its previous round communication. This makes it equivalent to the rushing model, as the adversary must now complete Round r before receiving any useful information related to Round $(r+1)$.

The above approach introduces a new issue: honest-to-honest messages are now masked, and their eventual unmasking depends on the additive share provided by the adversary. To continue relying on the security of the underlying protocol, which assumes that honest-to-honest messages are not tampered with, we must

ensure that the adversary cannot effectively modify these messages. We resolve this by authenticating the additive shares using information-theoretic MACs.

These ideas suffice for achieving selective abort security. Next, we observe that replacing information-theoretic MACs with digital signatures upgrades this approach to unanimous abort security. To further obtain identifiable abort security, we must ensure that any misbehavior in the communication of authenticated additive shares is detected. To this end, instead of communicating these shares over point-to-point channels, we derive the additive shares using *Non-Interactive Key-Exchange* NIKE. This prevents misbehavior such as selectively dropping shares, and allows parties to accompany their messages with NIZK proofs showing that the NIKE-derived keys (and thus the resulting shares) are correct. This completes a high-level overview of our techniques and we refer to the corresponding technical section for details.

1.3 Related Work

As mentioned above, the most relevant prior work is by Asharov et al. [ACCI25], who introduced the model of super-rushing adversaries and provided a characterization of perfectly secure MPC protocols that remain secure in this model.

Apart from the synchronous network model, another well-studied network model in MPC is the *asynchronous* network model, where the adversary can arbitrarily delay all messages in the network (while messages are still guaranteed to be eventually delivered). This gives the adversary unlimited peeking power but leads to strictly lower resilience compared to the synchronous setting, as shown in the foundational works on asynchronous MPC [BOCG93,BOKR94]. A recent line of work studies *network-agnostic* protocols [BKL19,LZLM+20,BCLZL23,AES26,JLZS26], which are designed to achieve security simultaneously in both synchronous and asynchronous networks. Partially synchronous models, in which protocols are essentially asynchronous but include either a synchronization point or an initial synchronous phase, have been explored in [BTHN10,PR18b,DGKN09]. Another related goal is *optimistic responsiveness* [ADLS91,PS18,SARN20], which allows running protocols as fast as actual network delays allow rather than worst-case delays. Liu-Zhang et al. [LZLM+20] achieved this notion for MPC by compiling together synchronous and asynchronous protocols.

Lastly, we outline the works that establish the round complexity for standard MPC. In dishonest majority ($t < n$), fairness (where, either all parties obtain the output or none do) and guaranteed output delivery (honest parties obtain the output, irrespective of adversarial behavior) are impossible regardless of the number of rounds [Cle86]. For the remaining security notions, the optimal round complexity varies based on setup. In the plain model, 4 rounds are necessary [GMPP16] and sufficient for security with abort [HHPV18,ACJ17,BGJ+18,CCG+20] and security with identifiable abort [CRSW22]. With a CRS or PKI, the round complexity drops to 2 rounds [GLS15,MW16,GS18,BL18], which is optimal since one-round MPC is insufficient for general functionalities [HLP11].

In the honest-majority setting ($t < n/2$), fairness requires at least 3 rounds even with a CRS [GIKR02,PR18a], and since guaranteed output delivery implies fairness, the same bound applies to guaranteed output delivery. The respective upper bounds appear in [ACGJ18,GLS15]. Identifiable abort is impossible in fewer than 3 rounds in the plain model with honest majority [GJPR21]. In the information-theoretic setting, 4 rounds are known to be necessary for guaranteed output delivery both for perfect security [AKP20b] and statistical security [AKP20a,AKP23].

2 Definition and Notation

Below, we recall the communication and security model of [ACCI25], which introduced super-rushing adversaries.

Setting. An n-party protocol $\Pi = (\mathcal{P}_1, \ldots, \mathcal{P}_n)$ is an n-tuple of PPT interactive Turing machines (ITM). Each party $\mathcal{P}_i$ 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 *static* and *malicious* adversaries i.e. the centralized adversary decides which parties to corrupt before execution begins and corrupt parties can deviate from the protocol in an arbitrary way.

2.1 Communication Model

Synchronous model. There is a global clock counter that all parties can access, and advance in a sequential manner. The protocol proceeds in a round-by-round manner, according to the global clock. Each round consists of a *send phase* (where parties send their messages for that round) followed by a *receive phase* (where they receive messages from other parties). All parties start 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 round r. Specifically, it is guaranteed that every message sent in round r is delivered to its destination before any message is sent for round $r+1$. We denote the message sent from $\mathcal{P}_i$ to $\mathcal{P}_j$ in round r as $\mathsf{msg}^r_{i\to j}$.

We consider protocols that are defined in the broadcast model. In every round, each party may send private messages using point-to-point channels, and/or broadcast messages that will arrive to all parties in the same round.

We consider protocols with all-to-all communication: in each round, every party sends a (possibly empty) message to every other party (either over point-to-point channels or via the broadcast channel), and the number of rounds is fixed and known ahead of time. We further assume that every corrupted party always sends a message when it is supposed to (this is a standard assumption for synchronous protocols, as the recipient can identify the event where the party is cheating by not sending a message when supposed to, and can pretend receiving a default message instead).

Rushing and Non-rushing Adversary In the synchronous setting, the two common adversary types are *non-rushing* and *rushing* adversaries.

A *non-rushing* adversary must choose the corrupted parties' round-r messages to honest parties independently of the messages sent by honest parties in round r. Formally, for each round r, every message $\mathsf{msg}^r_{i\to j}$ from a corrupted $\mathcal{P}_i$ to an honest $\mathcal{P}_j$ must be fixed and delivered prior to the adversary receiving any message $\mathsf{msg}^r_{j\to i}$ from an honest $\mathcal{P}_j$ to a corrupted $\mathcal{P}_i$.

A *rushing* adversary is more powerful: it may schedule the order in which messages are delivered within a given round, and can choose the corrupted parties' round-r messages to honest parties *after* receiving the messages sent by honest parties to corrupted parties in round r. Formally, in round r the adversary first receives all messages $\mathsf{msg}^r_{j\to i}$ from every honest $\mathcal{P}_j$ to every corrupt party $\mathcal{P}_i$, and based on this information produces the messages $\mathsf{msg}^r_{i\to j}$ for each corrupted party $\mathcal{P}_i$ to each honest party $\mathcal{P}_j$.

Super-rushing Adversary We consider a strictly stronger adversary type, called *super-rushing*, which is the main focus of this work.

A *super-rushing* adversary can not only reorder the messages within a given round r like a rushing adversary, but also reorder messages across different rounds, i.e., it can choose to deliver some round-$(r+1)$ messages of the honest parties before all messages for round r are delivered.

Concretely, consider any honest party $\mathcal{P}_j$ in round r. An honest party can generate its messages for round $r+1$ immediately after receiving all of its round-r messages. Once $\mathcal{P}_j$ has received all messages it expects in round r, a super-rushing adversary can then deliver $\mathcal{P}_j$'s *round*-$(r+1)$ message $\mathsf{msg}^{r+1}_{j\to i}$ to all corrupted parties $\mathcal{P}_i$, before all honest parties have received their round-r messages.

Such a request to 'peek' into an honest $\mathcal{P}_j$'s round $(r+1)$ message is permitted only if the adversary has already delivered:

- all past messages $\mathsf{msg}^{r'}_{i\to j'}$ from every corrupted $\mathcal{P}_i$ to every honest $\mathcal{P}_{j'}$ for every $r' \le r-1$, and
- all round-r messages $\mathsf{msg}^r_{i\to j}$ from every corrupted $\mathcal{P}_i$ to the targeted honest party $\mathcal{P}_j$.

In other words, the super-rushing adversary can use its peeking power to access $\mathcal{P}_j$'s round $(r+1)$ message to influence its own round-r messages to the remaining honest parties, who have not yet entered round $r+1$. This strictly separates the super-rushing model from the standard rushing model: a rushing adversary sees all honest round-r messages but cannot peek into round $r+1$ until every party has completed round r.

The interface of the adversary. For concreteness, 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 the message $\mathsf{msg}_{j^* \to i^*}^{r+1}$ for an honest $\mathcal{P}_{j^*}$ and a corrupted $\mathcal{P}_{i^*}$, then the following holds:

– If the adversary is *non-rushing*, then the adversary must have sent all messages $\mathsf{msg}_{i \to j}^{r'}$ for every corrupted $\mathcal{P}_i$, honest $\mathcal{P}_j$, and $r' \leq r+1$.
– If the adversary is *rushing*, then the adversary must have sent all messages $\mathsf{msg}_{i \to j}^{r'}$ for every corrupted $\mathcal{P}_i$, honest $\mathcal{P}_j$, and $r' \leq r$.
– If the adversary is *super-rushing*, then the adversary must have sent all messages $\mathsf{msg}_{i \to j}^{r'}$ for every corrupted $\mathcal{P}_i$, honest $\mathcal{P}_j$, and $r' \leq r-1$, and also all messages $\mathsf{msg}_{i \to j^*}^{r}$ for every corrupted $\mathcal{P}_i$.

We note that the above requirement for super-rushing is only for the case of all-to-all communication. In general, the requirement is that the adversary can request a message $\mathsf{msg}_{j^* \to i^*}^{r+1}$ if $\mathcal{P}_{j^*}$ has received all the messages expected to arrive by the protocol in round r.

2.2 Secure Computation

We follow the real/ideal world simulation paradigm and we adopt the security model of [CGZ20].

Real execution. We describe the real world execution of a protocol Π. During the execution, the corrupted parties receive arbitrary instructions from the adversary $\mathcal{A}$, while the honest parties follow the protocol faithfully. The scheduling of message delivery follows the rules of the adversary type under consideration: non-rushing, rushing, or super-rushing (as defined in Section 2.1). At the end of the protocol execution, the honest parties produce output, the corrupted parties produce no output, and the adversary outputs an arbitrary function of its view. The view of a party during the execution includes its input, random coins, and all messages it sees during the execution.

Definition 1 (Real execution). *Let $\Pi = (\mathcal{P}_1, \ldots, \mathcal{P}_n)$ be an n-party protocol, where each party $\mathcal{P}_i$ starts with input $x_i \in \{0,1\}^*$ and private random coins $\rho_i \in \{0,1\}^*$; we write $\mathbf{x} = (x_1, \ldots, x_n)$ for the input vector. Let $\mathcal{I} \subseteq [n]$, of size at most t, be the set of indices of parties corrupted by the adversary $\mathcal{A}$. and auxiliary input $z \in \{0,1\}^*$. The joint execution of Π under $(\mathcal{A}, \mathcal{I})$ in the real world, on input vector $\mathbf{x}$, auxiliary input z and security parameter λ, denoted $\textsc{Real}_{\Pi,\mathcal{I},\mathcal{A}(z)}(\mathbf{x}, \lambda)$, is defined as the output vector of $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and $\mathcal{A}(z)$ resulting from the protocol execution.*

Ideal execution. We describe the ideal world execution with selective abort (SA) and unanimous abort (UA), identifiable abort (IA), fairness (Fair), and guaranteed output delivery (GOD).

Definition 2 (Ideal execution). *For an n-party ideal functionality $\mathcal{F}$ computing an n-party function $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, and let $\mathcal{I} \subseteq [n]$, of size at most t, be the set of indices of parties corrupted by the adversary. Consider* $\mathsf{type} \in \{\mathsf{SA}, \mathsf{UA}, \mathsf{IA}, \mathsf{Fair}, \mathsf{GOD}\}$. *The joint ideal execution of f under $(\mathcal{S}, \mathcal{I})$ on input vector $\mathbf{x}$, auxiliary input z to $\mathcal{S}$ and security parameter λ, denoted $\textsc{Ideal}_{\mathcal{F},\mathcal{I},\mathcal{S}(z)}^{\mathsf{type}}(\mathbf{x}, \lambda)$, is defined as the output vector of $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and $\mathcal{S}(z)$ resulting from the following ideal process.*

1. Parties send inputs to $\mathcal{F}$: *Every honest party $\mathcal{P}_i$ sends its input x_i to $\mathcal{F}$.*
   $\mathcal{S}$ may send to $\mathcal{F}$ arbitrary inputs for the corrupted parties. Let x_i' be the value actually sent as the input of party $\mathcal{P}_i$.
2. $\mathcal{F}$ computes and speaks to $\mathcal{S}$: *$\mathcal{F}$ computes $(y_1, \ldots, y_n) = f(x_1', \ldots, x_n')$. If there are no corrupted parties or* $\mathsf{type} = \mathsf{GOD}$, *proceed to step 4.*
   – *if* $\mathsf{type} = \{\mathsf{SA}, \mathsf{UA}, \mathsf{IA}\}$, *$\mathcal{F}$ sends $\{y_i\}_{i \in \mathcal{I}}$ to $\mathcal{S}$.*
   – *if* $\mathsf{type} = \mathsf{Fair}$, *$\mathcal{F}$ sends* ready *to $\mathcal{S}$.*
3. $\mathcal{S}$ responds to $\mathcal{F}$:
   – *If* $\mathsf{type} = \mathsf{SA}$*: $\mathcal{S}$ can select a set $\mathcal{J} \subseteq [n] \setminus \mathcal{I}$ of honest parties to be denied output (possibly empty, allowing all parties to obtain the output). It sends* $(\mathsf{abort}, \mathcal{J})$ *to $\mathcal{F}$.*
   – *If* $\mathsf{type} = \mathsf{UA}$*: $\mathcal{S}$ can send* abort *to $\mathcal{F}$. If it does, we take $\mathcal{J} = [n] \setminus \mathcal{I}$.*

– *If* $\mathsf{type} = \mathsf{IA}$*: If* $\mathcal{S}$ *chooses to abort, it can select a corrupted party* $\mathcal{P}_{i^*} \in \mathcal{I}$ *who will be blamed, and send* (abort, i^*) *to* $\mathcal{F}$*. If it does, we take* $\mathcal{J} = [n] \setminus \mathcal{I}$*.*
– *if* $\mathsf{type} = \mathsf{Fair}$*:* $\mathcal{S}$ *can send* abort *to* $\mathcal{F}$*. If it does, we take* $\mathcal{J} = [n]$*.*

4. $\mathcal{F}$ answers parties:
   (a) *If* $\mathcal{F}$ *received* abort *from the simulator* $\mathcal{S}$*:*
      i. It sets the abort message abortmsg*, as follows:*
         – *if* $\mathsf{type} \in \{\mathsf{SA}, \mathsf{UA}, \mathsf{Fair}\}$*, we let* $\mathsf{abortmsg} = \perp$*.*
         – *if* $\mathsf{type} = \mathsf{IA}$*, we let* $\mathsf{abortmsg} = (\perp, i^*)$*.*
      ii. $\mathcal{F}$ *then sends* abortmsg *to every party* $\mathcal{P}_j$*,* $j \in \mathcal{J}$*, and* y_j *to every party* $\mathcal{P}_j$*,* $j \in [n] \setminus \mathcal{J}$*.*
      Note that, if $\mathsf{type} = \mathsf{GOD}$, we will never be in this setting, since $\mathcal{S}$ was not allowed to ask for an abort.
   (b) *Otherwise, it sends* y_j *to every* $\mathcal{P}_j$*,* $j \in [n]$*.*
5. Outputs: *Honest parties always output the message received from* $\mathcal{F}$ *while the corrupt parties output nothing. The simulator* $\mathcal{S}$ *outputs an arbitrary function of the initial inputs* $\{x_i\}_{i \in \mathcal{I}}$*, the messages received by the corrupt parties from the trusted party, and its auxiliary input.*

Definition 3. *Consider* $\mathsf{type} \in \{\mathsf{SA}, \mathsf{UA}, \mathsf{IA}, \mathsf{Fair}, \mathsf{GOD}\}$*. A protocol* Π t-securely computes f with type security *if for every* PPT *(resp. unbounded) adversary* $\mathcal{A}$ *there exists an expected polynomial-time simulator* $\mathcal{S}$ *such that for all* $\mathcal{I} \subseteq [n]$ *with* $|\mathcal{I}| \leq t$*, the following two distributions:*

$$\left\{\textsc{Real}_{\Pi,\mathcal{I},\mathcal{A}(z)}(\mathbf{x}, \lambda)\right\}_{\mathbf{x} \in (\{0,1\}^*)^n,\, \lambda \in \mathbb{N}}, \left\{\textsc{Ideal}_{f,\mathcal{I},\mathcal{S}(z)}(\mathbf{x}, \lambda)\right\}_{\mathbf{x} \in (\{0,1\}^*)^n,\, \lambda \in \mathbb{N}}$$

are computationally (resp. perfectly / statistically) indistinguishable.

2.3 Building Blocks

We recall the key building blocks used in our construction below, and defer the remaining definitions to Appendix A.

Definition 4 (Message Authentication Code[Gol04]). *A* message authentication code *(MAC) for message space* $\mathcal{M}$ *is a triple of algorithms* $(\mathsf{Setup}, \mathsf{Auth}, \mathsf{Vrfy})$*:*

– $\mathsf{Setup}(1^\lambda)$*: on input a unary representation of the security parameter* 1^λ*, outputs a key* κ*.*
– $\mathsf{Auth}(\kappa, m)$*: on input a key* κ *and a message* $m \in \mathcal{M}$*, outputs a tag* τ*.*
– $\mathsf{Vrfy}(\kappa, m, \tau)$*: on input a key* κ*, a message* $m \in \mathcal{M}$*, and a tag* τ*, outputs a bit* $b \in \{0, 1\}$*.*

Correctness. *For all* $\lambda \in \mathbb{N}$*,* $\kappa \leftarrow \mathsf{Setup}(1^\lambda)$*, and* $m \in \mathcal{M}$*, we have*

$$\Pr[\mathsf{Vrfy}(\kappa,\, m,\, \mathsf{Auth}(\kappa, m)) = 1] = 1.$$

Definition 5 (EUF-CMA security). *Let* $(\mathsf{Setup}, \mathsf{Auth}, \mathsf{Vrfy})$ *be a MAC scheme and* $\mathcal{A}$ *a* PPT *adversary. Define the* MAC forgery experiment $\mathsf{EUF\text{-}CMA}^{\mathsf{MAC}}(\lambda, \mathcal{A})$*:*

1. *The challenger samples* $\kappa \leftarrow \mathsf{Setup}(1^\lambda)$ *and initialises* $Q \leftarrow \emptyset$*.*
2. $\mathcal{A}$ *adaptively submits authentication queries: on query* m*, the challenger returns* $\tau \leftarrow \mathsf{Auth}(\kappa, m)$ *and records* $Q \leftarrow Q \cup \{m\}$*.*
3. $\mathcal{A}$ *outputs* (m^*, τ^*)*.*
4. $\mathcal{A}$ wins *if* $\mathsf{Vrfy}(\kappa, m^*, \tau^*) = 1$ *and* $m^* \notin Q$*.*

The MAC *is* EUF-CMA secure *if for every* PPT $\mathcal{A}$*:* $\Pr[\mathsf{EUF\text{-}CMA}^{\mathsf{MAC}}(\lambda, \mathcal{A}) = 1] \leq \mathrm{negl}(\lambda)$*.*

Definition 6 (EUF-1-CMA security). *Let* $(\mathsf{Setup}, \mathsf{Auth}, \mathsf{Vrfy})$ *be a MAC scheme and* $\mathcal{A}$ *an unbounded adversary. Define the* one-time MAC forgery experiment $\mathsf{EUF\text{-}1\text{-}CMA}^{\mathsf{MAC}}(\lambda, \mathcal{A})$ *identically to* $\mathsf{EUF\text{-}CMA}^{\mathsf{MAC}}(\lambda, \mathcal{A})$*, except that* $\mathcal{A}$ *is permitted at most one authentication query. The* MAC *is* EUF-1-CMA secure *(or* one-time MAC secure*) if for every* $\mathcal{A}$*:*

$$\Pr\left[\mathsf{EUF\text{-}1\text{-}CMA}^{\mathsf{MAC}}(\lambda, \mathcal{A}) = 1\right] \leq \mathrm{negl}(\lambda).$$

Remark 1 (One-time MAC instantiation). A canonical instantiation of an EUF-1-CMA secure information-theoretic MAC is the *linear MAC* over a finite field $\mathbb{F}$ with $|\mathbb{F}| \geq |\mathcal{M}|$ and $|\mathbb{F}| > 2^\lambda$.

- $\mathsf{Setup}(1^\lambda)$: sample $(a, b) \leftarrow \mathbb{F}^2$ uniformly; output $\kappa = (a, b)$.
- $\mathsf{Auth}(\kappa, m)$: output $\tau = a \cdot m + b \in \mathbb{F}$.
- $\mathsf{Vrfy}(\kappa, m, \tau)$: output 1 if $\tau = a \cdot m + b$, else 0.

One-time security. Given a single valid pair (m, τ) with $\tau = a \cdot m + b$, there are $|\mathbb{F}|$ possible keys (a, b) consistent with it (one for each choice of a). Hence for any $m^* \neq m$, the correct tag $am^* + b$ is uniform over $\mathbb{F}$ from the adversary's view, so the forgery probability is exactly $1/|\mathbb{F}| \leq \mathrm{negl}(\lambda)$.

Non-interactive Key Exchange A non-interactive key exchange protocol allows two parties to jointly compute a cryptographic key.

Definition 7 (Non-Interactive Key Exchange (NIKE) [DMR+21,FHKP13]). *A* non-interactive key exchange *(NIKE) scheme, parameterized by a distribution $\mathbb{D}$, is a tuple of efficient algorithms* $(\mathsf{NIKE.KeyGen}, \mathsf{SharedKey})$ *defined as follows:*

- $\mathsf{NIKE.KeyGen}(1^\lambda; r) \to (\mathsf{pk}, \mathsf{sk})$*: takes as input the security parameter 1^λ and outputs a public/secret key pair* $(\mathsf{pk}, \mathsf{sk})$*.*
- $\mathsf{SharedKey}(\mathsf{pk}_i, \mathsf{sk}_j) \to k$*: takes as input a public key pk_i and a secret key sk_j from a different key pair, and outputs a shared key k for the holder of the public key and the holder of the private key. Importantly, the algorithm returns the same key when run on $\mathcal{P}_i$'s public key and $\mathcal{P}_j$'s secret key, and vice versa.*

Correctness. *A non-interactive key exchange NIKE scheme is* correct *if*

$$\Pr\left[\begin{array}{l}(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{NIKE.KeyGen}(1^\lambda; r_0);\\ (\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{NIKE.KeyGen}(1^\lambda; r_1);\\ \mathsf{SharedKey}(\mathsf{pk}_0, \mathsf{sk}_1) = \mathsf{SharedKey}(\mathsf{pk}_1, \mathsf{sk}_0);\end{array}\right] \geq 1 - \mathrm{negl}(\lambda).$$

Security. *A non-interactive key exchange scheme is* secure *if for all* PPT *adversaries $\mathcal{A}$, the joint pair of keys is indistinguishable from two independent random values:*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{l}(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{NIKE.KeyGen}(1^\lambda; r_0);\\ (\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{NIKE.KeyGen}(1^\lambda; r_1);\\ b \leftarrow \{0, 1\};\\ k_0 \leftarrow \mathsf{SharedKey}(\mathsf{pk}_0, \mathsf{sk}_1);\\ k_1 \leftarrow \mathbb{D};\\ b' \leftarrow \mathcal{A}(\mathsf{pk}_0, \mathsf{pk}_1, k_b)\end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda).$$

Definition 8 (Non-Interactive Zero-Knowledge Arguments of Knowledge (NIZKAoK) [GM17]). *A* non-interactive zero-knowledge argument of knowledge *(*NIZK*) scheme is a tuple of efficient algorithms* $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{Prove}, \mathsf{Verify})$ *defined as follows:*

- $\mathsf{NIZK.Setup}(1^\lambda, \mathcal{R}) \to \texttt{crs}$*: takes as input the security parameter $\lambda \in \mathbb{N}$, and outputs the global common reference string* $\texttt{crs}$*.*
- $\mathsf{Prove}(\texttt{crs}, \phi, w) \to \pi$*: takes as input the common reference string* $\texttt{crs}$ *for a relation $\mathcal{R}$, a statement ϕ and a witness w, and outputs a proof π that $(\phi, w) \in \mathcal{R}$.*
- $\mathsf{Verify}(\texttt{crs}, \phi, \pi) \to \mathsf{accept/reject}$*: takes as input the common reference string* $\texttt{crs}$ *for a relation $\mathcal{R}$, a statement ϕ and a proof π, and verifies whether π proves the existence of a witness w such that $(\phi, w) \in \mathcal{R}$.*

The rest of the algorithms are used in the security proof, but will not be used in the real world:

- $\mathsf{simSetup}(1^\lambda, \mathcal{R}) \to (\texttt{crs}, \tau)$*: takes as input the security parameter $\lambda \in \mathbb{N}$, and outputs a simulated common reference string* $\texttt{crs}$ *and a simulation trapdoor τ.*
- $\mathsf{simP}(\texttt{crs}, \tau, \phi) \to \pi$*: takes as input the common reference string* $\texttt{crs}$ *for a relation $\mathcal{R}$, the trapdoor τ and a statement ϕ, and outputs a simulated proof of the statement ϕ.*

We require the following properties of a NIZK *scheme:*

Correctness. *We say that* NIZK *satisfies* correctness *if for any* $(\phi, w) \in \mathcal{R}$*, we have that*

$$\Pr\left[\mathsf{Verify}(\mathit{crs}, \phi, \pi) = 1 \;\middle|\; \begin{array}{l}(\mathit{crs}, \tau) \leftarrow \mathsf{NIZK.Setup}(1^\lambda, \mathcal{R}) \\ \pi \leftarrow \mathsf{Prove}(\mathit{crs}, \phi, w)\end{array}\right] \geq 1 - \mathrm{negl}(\lambda)$$

(where the randomness is taken over the internal coin tosses of NIZK.Setup*,* Prove *and* Verify*).*

Zero Knowledge. *We say that* NIZK *satisfies* zero-knowledge *if for all* PPT *adversaries* $\mathcal{A}$*,*

$$\Pr[\mathcal{A} \textit{ wins}] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$$

in the following experiment:

Adversary $\mathcal{A}$		Challenger $\mathcal{C}$
		$b \leftarrow \{0,1\}$
	$\xleftarrow{crs}$	*if* b=0*:* $\mathit{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, \mathcal{R})$
		if b=1*:* $(\mathit{crs}, \tau) \leftarrow \mathsf{simSetup}(1^\lambda, \mathcal{R})$
	$\xrightarrow{\phi, w}$	
Repeat $\mathrm{poly}(\lambda)$ *times*		*if* b=0*:* $\pi \leftarrow \mathsf{Prove}(\mathit{crs}, \phi, w)$
		if b=1*:* $\pi \leftarrow \mathsf{simP}(\mathit{crs}, \tau, \phi)$
	$\xleftarrow{\pi}$	
	$\xrightarrow{b'}$	

$\mathcal{A}$ *wins if* $b = b'$

Simulation Soundness. *We say that* NIZK *satisfies* simulation soundness *if for all* PPT *adversaries* $\mathcal{A}$ *that*

$$\Pr[\mathcal{A} \textit{ wins}] \leq \mathrm{negl}(\lambda)$$

in the following experiment:

Adversary $\mathcal{A}$		Challenger $\mathcal{C}$
	$\xleftarrow{crs}$	$(\mathit{crs}, \tau) \leftarrow \mathsf{simSetup}(1^\lambda, \mathcal{R})$
		$Q_{sim} = \emptyset$
	$\xrightarrow{\phi}$	
Repeat $\mathrm{poly}(\lambda)$ *times*		$\pi \leftarrow \mathsf{simP}(\mathit{crs}, \tau, \phi)$
		add π *to* Q_{sim}
	$\xleftarrow{\pi}$	
	$\xrightarrow{\phi^*, \pi^*}$	

If all of the following checks pass, $\mathcal{A}$ *wins:*

$\phi^* \notin \mathcal{L}$ $\quad \mathsf{Verify}(\mathit{crs}, \phi^*, \pi^*) = \mathsf{accept}$ $\quad \pi^* \notin Q_{sim}$

3 Broadcast-only Protocols

In this section, we show that any protocol secure against rushing adversaries that uses *only* broadcast communication is also secure against super-rushing adversaries. This result formalizes that the power of super-rushing adversary heavily relies on point-to-point messages being involved.

Intuitively, if a protocol uses communication only over ideal broadcast (say, instantiated via a public bulletin board), then the power of super-rushing is rendered useless. This is because, once the adversary broadcasts its round r message, it is committed to it. Then, even if it can peek into the round $r+1$ messages of some honest parties, this gives it no additional power: it cannot change the round r broadcast message that other parties receive.

Security follows from the fact that, by the time the adversary requests to peek at an honest party's next-round message (say, in round $r+1$), it must have already sent all of its own broadcast messages for round r. Consequently, the simulator for the super-rushing protocol already knows all adversarial round-r messages and can invoke the simulator for the rushing protocol to obtain the honest parties' round-$(r+1)$ messages, which it then forwards to answer the adversary's peek request. Notably, this compiler preserves the type of security guarantee (i.e. security with abort / fairness / guaranteed output delivery) of the underlying protocol as well.

We state the formal theorem below and refer to Appendix B for the proof.

Theorem 1. *Let Π be a synchronous R-round computationally (resp. perfectly / statistically) secure n-party protocol computing a functionality f over broadcast channels only, tolerating at most t static corruptions. Then Π is computationally (resp. perfectly / statistically) secure against super-rushing adversaries tolerating at most t static corruptions. That is, for every PPT (resp. unbounded) super-rushing adversary $\mathcal{A}_{\mathsf{sr}}$ there exists an expected polynomial-time simulator $\mathcal{S}_{\mathsf{sr}}$ such that for all $\mathcal{I} \subseteq [n]$ with $|\mathcal{I}| \leq t$, the following two distributions are computationally (resp. perfectly / statistically) indistinguishable:*

$$\left\{\textsc{Real}_{\Pi,\mathcal{I},\mathcal{A}_{\mathsf{sr}}(z)}(\mathbf{x},\lambda)\right\}_{\mathbf{x}\in(\{0,1\}^*)^n,\,\lambda\in\mathbb{N}},\left\{\textsc{Ideal}_{f,\mathcal{I},\mathcal{S}_{\mathsf{sr}}(z)}(\mathbf{x},\lambda)\right\}_{\mathbf{x}\in(\{0,1\}^*)^n,\,\lambda\in\mathbb{N}}.$$

4 Round-Preserving Compiler with Selective Abort Security

Below we present our *round-preserving* compiler that transforms any protocol secure against rushing adversary (say Π) and transforms it into a protocol $\Pi^{\mathsf{sr}}_{\mathsf{SA}}$ that is secure against super-rushing adversaries with selective abort.

The main challenge in lifting security from rushing to super-rushing adversaries in the presence of point-to-point communication is that, unlike in the broadcast-only setting (Section 3), a super-rushing adversary can request for the round-$(r+1)$ message of some honest party $\mathcal{P}_{i^*}$ before having sent all of its own round-r messages. In this case, we can no longer directly rely on the simulator for the underlying rushing protocol Π, since that simulator typically requires the entire set of adversarial round-r messages in order to generate the honest parties' round-$(r+1)$ messages.

We now give an informal description of our solution to this issue. For simplicity, and without loss of generality, assume that the underlying rushing protocol Π uses point-to-point communication only in the first round and that all subsequent rounds use only broadcast communication[3]. We emphasize that this assumption is made solely for clarity of exposition; it can be lifted, and we present a communication-pattern preserving compiler in Section 4.3.

If a super-rushing adversary requests an honest party's Round-2 message before sending all of its Round-1 messages, we render this information useless by masking Round-2 messages with a mask, whose additive shares are distributed in Round 1. In Round 2, an honest party sends the additive shares it received in Round 1 only to parties from which it received a Round-1 message. Intuitively, even if the super-rushing adversary obtains the masked Round-2 messages early, it cannot learn anything useful from them until it has sent all of its Round-1 messages. This effectively forces the adversary to commit to all Round-1 communication before learning any honest Round-2 message.

[3]This is without loss of generality: any private communication in later rounds can be emulated by having parties exchange pairwise random masks in the first round, and then broadcasting masked messages instead of sending them privately.

Concretely, in Round 1 of $\Pi^{\mathsf{sr}}_{\mathsf{SA}}$, in addition to executing the first-round messages of Π, the parties also perform an additive sharing of random masks that will later be used to hide their Round-2 broadcast messages. In Round 2 of $\Pi^{\mathsf{sr}}_{\mathsf{SA}}$, each honest party broadcasts its Round-2 message of Π masked with the corresponding random mask, and also sends the additive shares it received in Round 1. At the end of Round 2 of $\Pi^{\mathsf{sr}}_{\mathsf{SA}}$, each party can reconstruct the mask from the additive shares and then unmask the masked Round-2 messages to recover the Round-2 messages of Π. From Round 3 onwards, since the protocol uses only broadcast, $\Pi^{\mathsf{sr}}_{\mathsf{SA}}$ proceeds identically to Π.

This structure enables simulation in the super-rushing setting: as long as the adversary has not sent its Round-1 messages to all honest parties, the simulator can answer peek requests for Round-2 messages with random values (indistinguishable from real masked messages since the adversary is missing at least one additive share). Once the adversary has completed Round 1 with every honest party, the simulator invokes the rushing simulator for Π to obtain the honest Round-2 messages, and then programs the final additive shares so that unmasking reveals exactly those messages. Lastly, we note that we only need to handle the peeking requests for Round 2 messages of honest parties (as Round 1 involves point-to-point communication). Since Π uses only broadcast from Round 2 onwards, any peek requests for later rounds can be easily handled (recall that the broadcast-only protocol steps are directly secure against super-rushing adversaries, as shown in Section 3).

4.1 The Selective Abort Compiler

We give a formal description of our compiler below. The compiler takes the following building blocks:

- An R-round rushing protocol Π in which all communication in rounds $2, \ldots, R$ is via broadcast only, expressed as a set of algorithms

$$\{(\mathsf{fst\text{-}msg}_i, \mathsf{nxt\text{-}msg}^r_i, \mathsf{output}_i)\}_{i\in[n], r\in\{2,\ldots,R\}}$$

  - $\left\{\left(\mathsf{BC}^1_i, \{\mathsf{msg}^1_{i\to j}\}_{j\in[n]}\right) \leftarrow \mathsf{fst\text{-}msg}_i(x_i, \rho_i)\right\}_{i\in[n]}$
  - $\{\mathsf{BC}^r_i \leftarrow \mathsf{nxt\text{-}msg}^r_i(x_i, \rho_i, T_i^{<r})\}_{i\in[n], r\in\{2,\ldots,R\}}$
  - $\left\{y_i \leftarrow \mathsf{output}_i(x_i, \rho_i, T_i^{<R+1})\right\}_{i\in[n]}$

  where r represents the current round number, BC^r_i represents the broadcast messages of party $\mathcal{P}_i$ in round r, $\mathsf{msg}^1_{i\to j}$ represents the point-to-point messages from $\mathcal{P}_i$ to $\mathcal{P}_j$ in the first round, y_i represents the output of $\mathcal{P}_i$ and $T_i^{<r}$ represents its transcript that includes messages until (but not including) round r.
- An n-out-of-n additive secret sharing scheme $(\mathsf{Share}, \mathsf{Rec})$ over $(\mathbb{F}, +)$, where $\mathbb{F}$ is a field into which all messages of Π are embedded (see Definition 11)
- A one-time $\mathsf{EUF\text{-}1\text{-}CMA}$-secure MAC scheme $(\mathsf{Setup}, \mathsf{Auth}, \mathsf{Vrfy})$ with message space $\mathbb{F}$. (see Definition 6)

Protocol $\Pi^{\mathsf{sr}}_{\mathsf{SA}}$

Input: Each party $\mathcal{P}_i$ has input x_i and randomness ρ_i.

Protocol Description:

- **Round 1:** Each party $\mathcal{P}_i$ does the following:
  - Run $\mathsf{Setup}(1^\lambda)$ to generate MAC keys $\left\{\kappa^j_{i,k}\right\}_{j,k\in[n]}$, and send $\left\{\kappa^j_{i,k}\right\}_{k\in[n]}$ to $\mathcal{P}_j$ over point-to-point channels for all $j \in [n]$.
  - Sample masks M_i uniformly from $\mathbb{F}$. Compute an additive sharing of M_i as $(m_{i,1}, \ldots, m_{i,n}) \leftarrow \mathsf{Share}(M_i)$.
  - For each share $m_{i,k}$, compute $\tau^j_{i,k} \leftarrow \mathsf{Auth}(\kappa^j_{i,k}, m_{i,k})$ for all $j \in [n]$. Send $\left(m_{i,k}, \left\{\tau^j_{i,k}\right\}_{j\in[n]}\right)$ over point-to-point channels to $\mathcal{P}_k$.

- Compute $\left(\mathsf{BC}_i^1, \{\mathsf{msg}_{i\to j}^1\}_{j\in[n]}\right) \leftarrow \mathsf{fst\text{-}msg}_i(x_i, \rho_i)$. Send BC_i^1 over broadcast channel and $\mathsf{msg}_{i\to j}^1$ over point-to-point channels to $\mathcal{P}_j$ for all $j \in [n]$.

– **Round 2:** Each party $\mathcal{P}_i$ does the following:
- If $\mathcal{P}_i$ is missing any share from round 1, it outputs $\perp$.
- Else, update the transcript $T_i^{<2} = \left\{\mathsf{BC}_j^1, \mathsf{msg}_{j\to i}^1\right\}_{j\in[n]}$.
- Broadcast the authenticated shares $\left(m_{j,i}, \left\{\tau_{j,i}^k\right\}_{k\in[n]}\right)$ for all $j \in [n]$.
- Compute $\mathsf{BC}_i^2 \leftarrow \mathsf{nxt\text{-}msg}_i^2(x_i, \rho_i, T_i^{<2})$.
- Send $\widetilde{\mathsf{BC}_i^2} = \mathsf{BC}_i^2 + M_i$ over broadcast.

– **Local Computation (at the end of Round 2):** After round 2, each party $\mathcal{P}_i$ does the following (this could take place at the start of round 3 (if $R > 2$), or at the start of the output stage if $R = 2$):
- If $\mathcal{P}_i$ is missing any share from round 2, it outputs $\perp$.
- For every $k, \ell \in [n]$, run the verification algorithm $\mathsf{Vrfy}(\kappa_{k,\ell}^i, m_{k,\ell}, \tau_{k,\ell}^i)$. If any of the verification fails, output $\perp$.
- Else, reconstruct the masks $\{M_k\}_{k\in[n]}$: $M_k \leftarrow \mathsf{Rec}(m_{k,1}, \ldots, m_{k,n})$.
- For every masked message $\widetilde{\mathsf{BC}_k^2}$ received, recover $\mathsf{BC}_k^2 \leftarrow \widetilde{\mathsf{BC}_k^2} - M_k$, obtaining $\left\{\mathsf{BC}_k^2\right\}_{k\in[n]}$.
- Update the transcript $T_i^{<3} = T_i^{<2} \cup \left\{\mathsf{BC}_j^2\right\}_{j\in[n]}$.

– **Round** r $(3 \le r \le R)$:
- Update the transcript $T_i^{<r} = T_i^{<r-1} \cup \left\{\mathsf{BC}_j^{r-1}\right\}_{j\in[n]}$.
- Compute and broadcast $\mathsf{BC}_i^r \leftarrow \mathsf{nxt\text{-}msg}_i^r(x_i, \rho_i, T_i^{<r})$.

– **Output:** Update the transcript $T_i^{<R+1} = T_i^{<R} \cup \left\{\mathsf{BC}_j^R\right\}_{j\in[n]}$ and compute the output as $y_i \leftarrow \mathsf{output}_i(x_i, \rho_i, T_i^{<R+1})$.

Fig. 1: Compiler for Selective Abort

4.2 Security

Theorem 2. *Let Π be any R-round statistically (resp. computationally) secure n-party protocol tolerating a rushing adversary corrupting at most t static corruptions. Without loss of generality, assume Π uses only broadcast communication from (and including) round 2. Then $\Pi_{\mathsf{SA}}^{\mathsf{sr}}$ is statistically (resp. computationally) secure with selective abort against super-rushing adversaries tolerating at most t static corruptions.*

We defer the formal proof to Appendix D and give only a proof sketch here.

Fix an arbitrary super-rushing adversary $\mathcal{A}_{\mathsf{sr}}$. We construct a simulator $\mathcal{S}_{\mathsf{sr}}$ that answers peek requests from $\mathcal{A}_{\mathsf{sr}}$ as follows. Consider a peek request for Round 2 of an honest party P_{j^*}. If the adversary has not yet sent its Round 1 messages to all honest parties, then $\mathcal{S}_{\mathsf{sr}}$ samples a random value as the Round 2 message of P_{j^*} and random additive shares. This is indistinguishable from the masked Round 2 message and the additive shares that the adversary receives in the real protocol.

Now consider the peek request for the last honest party (say P_{k^*}). At this point, $\mathcal{A}_{\mathsf{sr}}$ must have already sent all of its Round 1 messages. Thus, $\mathcal{S}_{\mathsf{sr}}$ is now ready to invoke the simulator for the rushing protocol Π and obtain simulated Round 2 broadcast messages for all honest parties. It then programs the final additive share sent on behalf of P_{k^*} so that these simulated messages are exactly what is revealed to $\mathcal{A}_{\mathsf{sr}}$.

Security follows from the security of the underlying rushing protocol and of the additive secret-sharing scheme. In addition, the security of the MACs guarantees that the adversary can tamper with the honest parties' shares only with negligible probability. We note that misbehavior with respect to additive shares involving a pair of corrupt parties is translated to malicious behavior in the underlying protocol. Finally, from Round 3 onwards, since only broadcast communication is used, security against super-rushing adversaries is for free (as shown in Section 3).

4.3 Communication-Pattern Preserving Compiler

In the above description, we have assumed that the rushing protocol Π uses point-to-point communication only in the first round. While this assumption is without loss of generality, in the interest of efficiency, it is also natural to ask whether there exists a round-preserving compiler that additionally preserves the type of communication in the underlying protocol (for instance, whether a rushing protocol that uses only point-to-point channels can be compiled to be secure against super-rushing adversaries without using an additional broadcast channel). We show in Appendix C that this is indeed possible: instead of using a single mask per party for the second-round broadcast message, we apply the additive-sharing-of-masks idea in every round, using separate masks for each private and broadcast message. This yields a compiler that preserves not only the number of rounds, but also the type of communication channels used in each round of the underlying protocol.

4.4 Extension to Unanimous Abort

In this section, we describe how a minor modification to $\Pi^{\mathsf{sr}}_{\mathsf{SA}}$ yields a protocol $\Pi^{\mathsf{sr}}_{\mathsf{UA}}$ that achieves unanimous-abort security (UA) against super-rushing adversaries.

We begin by analyzing why the construction in Figure 1 achieves only selective abort, meaning that some honest parties may obtain the output while others do not. The key reason is that in $\Pi^{\mathsf{sr}}_{\mathsf{SA}}$, each honest party verifies shares using a different MAC key, which it received *privately* in the first round. A corrupted party $\mathcal{P}_i$, which knows all the MAC keys it generated, can therefore compute and send a valid MAC tag for some honest parties and an invalid tag for others. Consequently, the same (broadcast) share may pass the MAC check for some honest parties but fail for others, causing only a subset of the honest parties to abort.

We observe that replacing the one-time MAC scheme $(\mathsf{Setup}, \mathsf{Auth}, \mathsf{Vrfy})$ with an $\mathsf{EUF\text{-}CMA}$-secure digital signature scheme $(\mathsf{Sig.KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ (see Definition 9 and 10) suffices to guarantee unanimity among the honest participants. In more detail,

- Each party $\mathcal{P}_i$ runs $(vk_i, sk_i) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$ in round 1, broadcasts vk_i alongside its protocol message BC^1_i, and sends $\{\mathsf{msg}^1_{i\to j}\}_{j\in[n]}$ via point-to-point channels as in $\Pi^{\mathsf{sr}}_{\mathsf{SA}}$.
- The masks are sampled and additively shared in Round 1 as in $\Pi^{\mathsf{sr}}_{\mathsf{SA}}$. The difference lies only in the authentication of the additive shares: each party $\mathcal{P}_i$ signs each share $m_{i,k}$ it sends to $\mathcal{P}_k$ as $\sigma_{i,k} \leftarrow \mathsf{Sign}(sk_i, m_{i,k})$, and sends the pair $(m_{i,k}, \sigma_{i,k})$ via point-to-point channels.
- In round 2, each party $\mathcal{P}_i$ outputs $\perp$ if it is missing any share from round 1; otherwise, it broadcasts all signed shares $(m_{j,i}, \sigma_{j,i})_{j\in[n]}$ received in round 1 as well as the protocol message BC^2_i.
- At the end of round 2, each party outputs $\perp$ if it has not received shares with valid signatures in round 2. Otherwise, it proceeds identically to $\Pi^{\mathsf{sr}}_{\mathsf{SA}}$.

To summarize, using digital signatures with a single public verification key allows us to ensure that all honest parties are in agreement.

Corollary 1. *Let Π be any R-round computationally secure n-party protocol tolerating at most t static corruptions. Without loss of generality, assume Π uses only broadcast communication from (and including) round 2. Then $\Pi^{\mathsf{sr}}_{\mathsf{UA}}$ is computationally secure with* unanimous abort *against super-rushing adversaries tolerating at most t static corruptions, assuming an $\mathsf{EUF\text{-}CMA}$-secure signature scheme.*

Lastly, we note that the above approach of using digital signatures instead of MACs can also be applied to our communication-pattern preserving compiler (see Section 4.3). Specifically, this approach lets us transform a UA protocol secure against rushing adversaries into a UA protocol $\Pi^{cp,\mathsf{sr}}_{\mathsf{UA}}$ secure against super-rushing adversaries, which preserves the communication pattern in the intermediate rounds 2 to $R-1$, and introduces broadcast only in the first and last round. We defer the details to Section C.1.

4.5 Communication Complexities

Let CC, CC_{BC} and CC_{P2P} denote the total communication complexity, broadcast communication complexity and point-to-point communication complexity of the rushing protocol Π respectively. We assume that the messages of Π are embedded in a field $\mathbb{F}$, and express the communication complexities in terms of number of field elements. The total communication complexity of our compiled super-rushing secure protocols are as follows.

- $\Pi^{\mathsf{sr}}_{\mathsf{SA}}$ (Figure 1) has a total communication complexity of $CC + \mathcal{O}(n^2 \cdot CC'_{\mathsf{BC}})$, where CC'_{BC} denotes the broadcast communication complexity in Round 2. Recall that each Round 2 broadcast message requires n additive shares, each of which are authenticated with n IT-MAC tags; contributing $\mathcal{O}(n^2)$ per message.
- $\Pi^{\mathsf{sr}}_{\mathsf{UA}}$ (Section 4.4) has a total communication complexity of $CC + \mathcal{O}(n \cdot CC'_{\mathsf{BC}}) + \sigma(n \cdot CC'_{\mathsf{BC}})$. Here, CC'_{BC} denotes the broadcast communication complexity in Round 2 and $\sigma(m)$ denotes the total communication complexity (expressed as number of field elements) of m signatures, each authenticating a single field element. Recall that each Round 2 broadcast message requires n additive shares, each of which are now authenticated using a single signature.
- $\Pi^{cp,\mathsf{sr}}_{\mathsf{SA}}$ (Figure 7) has a total communication complexity of $\mathcal{O}(n^2 \cdot CC_{\mathsf{BC}} + n \cdot CC_{\mathsf{P2P}})$. Recall that each broadcast message incurs $\mathcal{O}(n^2)$ overhead (one MAC key and tag per share per verifier); each P2P message incurs $\mathcal{O}(n)$ (one MAC tag per share, since only the recipient is required to verify).
- $\Pi^{cp,\mathsf{sr}}_{\mathsf{UA}}$ (Section C.1) has a total communication complexity of $\mathcal{O}(n \cdot CC) + \sigma(n \cdot CC)$, where $\sigma(m)$ denotes the total communication complexity of m signatures, each authenticating a single field element. Recall that each message (point-to-point as well as broadcast) requires n additive shares, each of which are authenticated using a single signature.

5 Round-Preserving Compiler for Identifiable Abort

We observe that the constructions in Section 4 do not achieve *identifiable* abort: our compiler involves sending authenticated additive shares privately in the first round. If a party, say $\mathcal{P}_i$ simply sends nothing (or an invalid message) to $\mathcal{P}_j$, then $\mathcal{P}_j$ cannot proceed. Even if $\mathcal{P}_j$ complains regarding the missing message, the other parties would not be able to tell if $\mathcal{P}_i$ is corrupt or $\mathcal{P}_j$ is lying. A possible approach to make the communication of shares verifiable is to move share distribution onto the broadcast channel (e.g. by having $\mathcal{P}_i$ broadcast shares encrypted to $\mathcal{P}_j$'s public key). However, this requires each party's public encryption key to be known *before* round 1, which does not hold in the absence of a PKI.

We resolve the above issue by exploiting the following observation: the private messages that need to be verifiable are random shares, so they need not be sent directly; instead, they can be deterministically derived from pairwise non-interactive key exchange (NIKE) keys.

Concretely, in round 1 each party $\mathcal{P}_i$ generates $n-1$ independent NIKE key pairs $(pk_{i,j}, sk_{i,j})$, one for each other party $\mathcal{P}_j$, and broadcasts all of its public keys $\{pk_{i,j}\}_{j \neq i}$. We slightly abuse notation by assuming that the key-agreement algorithm $\mathsf{SharedKey}$ outputs an ordered pair of shared keys:

$$(s_{j\to i}, s_{i\to j}) \leftarrow \mathsf{SharedKey}(pk_{i,j}, sk_{j,i}) \text{ and } (s_{i\to j}, s_{j\to i}) \leftarrow \mathsf{SharedKey}(pk_{j,i}, sk_{i,j})$$

In round 2, each party $\mathcal{P}_i$ defines the additive share of its mask intended for recipient $\mathcal{P}_j$ as $s_{i\to j}$ for all $j \neq i$. The term $s_{i\to i}$ is then sampled at random, and the mask is set to $M_i := \sum_{j\in[n]} s_{i\to j}$. Now, exactly as in Figure 1, $\mathcal{P}_i$ uses this mask to hide its Round 2 broadcast message, setting $\widetilde{\mathsf{BC}^2_i} = \mathsf{BC}^2_i + M_i$.

In addition, $\mathcal{P}_i$ broadcasts the shares it holds that contribute to other parties' masks, i.e., $\{s_{j\to i}\}_{j\in[n]}$, together with a NIZK proof that these shares were correctly derived from the corresponding NIKE key pairs. Since this information is broadcast, every party can verify the proofs and thus identify any party whose proof fails, thereby resolving the identifiability issue outlined above.

The NIZK proof for party $\mathcal{P}_i$ establishes the following relation $\mathcal{R}$ for each share $s_{j\to i}$:

$$\mathcal{R} \;=\; \left\{ \begin{array}{l} \phi = (pk_{j,i}, pk_{i,j}, s_{j\to i}) \\ w = (sk_{i,j}, r_{i,j}) \end{array} \;\middle|\; \begin{array}{l} (pk_{i,j}, sk_{i,j}) \leftarrow \mathsf{NIKE.KeyGen}(1^\lambda; r_{i,j}) \\ (*, s_{j\to i}) = \mathsf{SharedKey}(pk_{j,i}, sk_{i,j}) \end{array} \right\}.$$

We give a formal description of our compiler below. The compiler takes the following building blocks:

– An R-round rushing protocol Π in which all communication in rounds $2, \ldots, R$ is via broadcast only, expressed as a set of algorithms

$$\{(\mathsf{fst\text{-}msg}_i, \mathsf{nxt\text{-}msg}_i^r, \mathsf{output}_i)\}_{i\in[n], r\in\{2,\ldots,R\}}$$

- $\left\{\left(\mathsf{BC}_i^1, \{\mathsf{msg}_{i\to j}^1\}_{j\in[n]}\right) \leftarrow \mathsf{fst\text{-}msg}_i(x_i, \rho_i)\right\}_{i\in[n]}$
- $\{\mathsf{BC}_i^r \leftarrow \mathsf{nxt\text{-}msg}_i^r(x_i, \rho_i, T_i^{<r})\}_{i\in[n], r\in\{2,\ldots,R\}}$
- $\left\{y_i \leftarrow \mathsf{output}_i(x_i, \rho_i, T_i^{<R+1})\right\}_{i\in[n]}$

– A non-interactive key exchange scheme $(\mathsf{NIKE.KeyGen}, \mathsf{SharedKey})$ with key space $\mathbb{D} = \mathbb{F} \times \mathbb{F}$, where $\mathbb{F}$ is a field into which all messages of Π are embedded (see Definition 7).

– A non-interactive zero-knowledge proof system satisfying simulation soundness $(\mathsf{NIZK.Setup}, \mathsf{Prove}, \mathsf{Verify})$ (see Definition 8).

IA Protocol $\Pi_{\mathsf{IA}}^{\mathsf{sr}}$

Input: Each party $\mathcal{P}_i$ has input x_i and randomness ρ_i.

Setup: Set up the common reference string $\mathtt{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, \mathcal{R})$.

Construction:

– **Round 1:** Each party $\mathcal{P}_i$ does the following:

- For each $j \in [n] \setminus \{i\}$, run $(pk_{i,j}, sk_{i,j}) \leftarrow \mathsf{NIKE.KeyGen}(1^\lambda; r_{i,j})$. Broadcast $\{pk_{i,j}\}_{j\in[n]\setminus\{i\}}$.
- Compute $\left(\mathsf{BC}_i^1, \{\mathsf{msg}_{i\to j}^1\}_{j\in[n]}\right) \leftarrow \mathsf{fst\text{-}msg}_i(x_i, \rho_i)$. Broadcast BC_i^1 and send $\mathsf{msg}_{i\to j}^1$ via point-to-point to $\mathcal{P}_j$ for all $j \in [n]$.

– **Round 2:** Each party $\mathcal{P}_i$ does the following:

- Update the transcript $T_i^{<2} = \left\{\mathsf{BC}_j^1, \mathsf{msg}_{j\to i}^1\right\}_{j\in[n]}$.
- For all $j \in [n] \setminus \{i\}$, compute the NIKE shared key

$$(s_{i\to j}, s_{j\to i}) \leftarrow \mathsf{SharedKey}(pk_{j,i}, sk_{i,j}).$$

- Compute the mask $M_i \leftarrow \sum_{j\in[n]} s_{i\to j}$.
- Compute the broadcast message $\mathsf{BC}_i^2 \leftarrow \mathsf{nxt\text{-}msg}_i^2(x_i, \rho_i, T_i^{<2})$.
- Compute the masked message $\widetilde{\mathsf{BC}_i^2} = \mathsf{BC}_i^2 + M_i$.
- For all $j \in [n] \setminus \{i\}$, set $\phi_{j,i} = (pk_{j,i}, pk_{i,j}, s_{j\to i})$ and $w_{i,j} = (sk_{i,j}, r_{i,j})$. Compute $\pi_{j,i} \leftarrow \mathsf{Prove}(\mathtt{crs}, \phi_{j,i}, w_{i,j})$.
- Broadcast the masked messages, the shares along with the corresponding NIZK proofs:

$$\left\{\widetilde{\mathsf{BC}_i^2}, \{(s_{j\to i}, \pi_{j,i})\}_{j\in[n]}\right\}.$$

– **After Round** 2: After finishing round 2, each party $\mathcal{P}_i$ does the following (this could take place at the start of round 3 (if $R > 2$), or at the start of the output stage if $R = 2$):

- For each $j, k \in [n]$, if $\mathsf{Verify}(\mathtt{crs}, \phi_{j,k}, \pi_{j,k}) \neq 1$, identify $\mathcal{P}_k$ as the cheater.
- For each $k \in [n]$, reconstruct $M_k \leftarrow \sum_{j\in[n]} s_{k\to j}$ using the received shares $\{s_{k\to j}\}_{j\in[n]}$ (each $s_{k\to j}$ was broadcast by $\mathcal{P}_j$ in round 2).
- Recover $\mathsf{BC}_k^2 \leftarrow \widetilde{\mathsf{BC}_k^2} - M_k$ for every $k \in [n]$.

– **Round** r **(**$3 \le r \le R$**):** Each party $\mathcal{P}_i$ does the following:

- Update the transcript $T_i^{<r} = T_i^{<r-1} \cup \left\{\mathsf{BC}_k^{r-1}\right\}_{k\in[n]}$.
- Compute and broadcast $\mathsf{BC}_i^r \leftarrow \mathsf{nxt\text{-}msg}_i^r(x_i, \rho_i, T_i^{<r})$.

– **Output:** Each party $\mathcal{P}_i$ does the following:
  - Update the transcript $T_i^{<R+1} = T_i^{<R} \cup \left\{\mathsf{BC}_k^R\right\}_{k\in[n]}$.
  - execute the output function$y_i \leftarrow \mathsf{output}_i(x_i, \rho_i, T_i^{<R+1})$

Fig. 2: Round-Preserving IA Compiler

5.1 Security

Theorem 3. *Let Π be any R-round computationally secure n-party protocol with identifiable abort tolerating at most t static corruptions. Without loss of generality, assume Π uses only broadcast communication from (and including) round 2. Then $\Pi^{\mathsf{sr}}_{\mathsf{IA}}$ is computationally secure with* identifiable abort *against super-rushing adversaries tolerating at most t static corruptions, assuming a setup with CRS, a computationally secure non-interactive key exchange scheme and a non-interactive zero-knowledge proof system. That is, for every* PPT *super-rushing adversary $\mathcal{A}_{\mathsf{sr}}$ there exists an expected polynomial-time simulator $\mathcal{S}_{\mathsf{sr}}$ such that for all $\mathcal{I} \subseteq [n]$ with $|\mathcal{I}| \leq t$, the following two distributions are computationally indistinguishable:*

$$\left\{\textsc{Real}_{\Pi^{\mathsf{sr}}_{\mathsf{IA}},\mathcal{I},\mathcal{A}_{\mathsf{sr}}(z)}(\mathbf{x},\lambda)\right\}_{\mathbf{x}\in(\{0,1\}^*)^n,\,\lambda\in\mathbb{N}} \approx_c \left\{\textsc{Ideal}^{\mathsf{IA}}_{f,\mathcal{I},\mathcal{S}_{\mathsf{sr}}(z)}(\mathbf{x},\lambda)\right\}_{\mathbf{x}\in(\{0,1\}^*)^n,\,\lambda\in\mathbb{N}}$$

where $\textsc{Ideal}^{\mathsf{IA}}$ *denotes the ideal execution with the identifiable abort.*

Proof. Let $\mathcal{A}_{\mathsf{sr}}$ be any PPT super-rushing adversary attacking $\Pi^{\mathsf{sr}}_{\mathsf{IA}}$. We describe a super-rushing simulator $\mathcal{S}_{\mathsf{sr}}$ for $\Pi^{\mathsf{sr}}_{\mathsf{IA}}$ via a machine $\mathcal{M}$, defined in Figure 3. The machine $\mathcal{M}$ internally runs $\mathcal{A}_{\mathsf{sr}}$ and exposes a *left session* interface through which the honest parties' messages of Π are exchanged, regardless of whether the left session is connected to a real execution or a simulator. In a nutshell, $\mathcal{M}$ acts as a rushing adversary for Π and simulates the execution of the compiled protocol $\Pi^{\mathsf{sr}}_{\mathsf{IA}}$ internally for $\mathcal{A}_{\mathsf{sr}}$.

Since Π is computationally secure with identifiable abort against rushing adversaries, for every PPT rushing adversary $\mathcal{A}_{\mathsf{r}}$ with $|\mathcal{I}| \leq t$, there exists a PPT simulator $\mathcal{S}_{\mathsf{r}}$ such that

$$\textsc{Real}_{\Pi,\mathcal{I},\mathcal{A}_{\mathsf{r}}(z)}(\mathbf{x},\lambda) \approx_c \textsc{Ideal}^{\mathsf{IA}}_{f,\mathcal{I},\mathcal{S}_{\mathsf{r}}(z)}(\mathbf{x},\lambda).$$

In particular, such a $\mathcal{S}_{\mathsf{r}}$ exists for the rushing adversary $\mathcal{M}(\mathcal{A}_{\mathsf{sr}},\cdot)$. Our simulator $\mathcal{S}_{\mathsf{sr}}$ (Figure 4) runs $\mathcal{S}_{\mathsf{r}}$ internally.

Machine $\mathcal{M}(\mathcal{A}_{\mathsf{sr}}, \rho_{\mathsf{sr}})$

1. For every $i \in [n] \setminus \mathcal{I}$, initialize the virtual honest party $\bar{\mathcal{P}}_i$ with arbitrary input and randomness.
2. Run $\mathcal{A}_{\mathsf{sr}}$ with coins ρ_{sr}.
3. Set up the common reference string $(\mathtt{crs}, \tau) \leftarrow \mathsf{simSetup}(1^\lambda, \mathcal{R})$.

Left-session routing: Broadcast messages BC_i^r from the left session are forwarded over the broadcast channel; point-to-point messages $\mathsf{msg}^r_{i\to j}$ are sent from $\bar{\mathcal{P}}_i$ to $\mathcal{P}_j$.

– **For Round 1 messages:** Each party $\bar{\mathcal{P}}_i \in [n] \setminus \mathcal{I}$ does the following:
  - For each $j \in [n] \setminus \{i\}$, run $(pk_{i,j}, sk_{i,j}) \leftarrow \mathsf{NIKE.KeyGen}(1^\lambda; r_{i,j})$. Broadcast $\{pk_{i,j}\}_{j\in[n]\setminus\{i\}}$.
  - Interact with the left session to receive $\left(\mathsf{BC}_i^1, \{\mathsf{msg}^1_{i\to j}\}_{j\in\mathcal{I}}\right)$,
  - Broadcast BC_i^1, and send $\mathsf{msg}^1_{i\to j}$ via point-to-point channel to $\mathcal{P}_j$ for all $j \in \mathcal{I}$.

– **For Round 2 messages:**
  - $\mathcal{M}$ maintains peek set $\mathcal{L} \subseteq [n] \setminus \mathcal{I}$ (initially $\mathcal{L} = \emptyset$), which records the indices of the honest parties that the adversary requests for its round-2 messages. When $\mathcal{A}_{\mathsf{sr}}$ requests the round-2 messages from an honest $\mathcal{P}_i$ ($i \in [n]\setminus\mathcal{I}$) to the corrupted parties, call such a request $\mathsf{Peek}(\mathcal{P}_i, 2)$. $\mathcal{M}$ checks:

1. $i \notin \mathcal{L}$ (i.e., the adversary has not peeked $\mathcal{P}_i$'s round-2 messages before), and
2. all round-1 messages from every $\mathcal{P}_j \in \mathcal{I}$ have been delivered to $\bar{\mathcal{P}}_i$.

If any condition fails, call this $\mathsf{NotSuperRushing}$ and $\mathcal{M}$ aborts.[a] Otherwise, $\mathcal{M}$ proceeds as follows.

- For each party $\bar{\mathcal{P}}_i \in [n] \setminus \mathcal{I}$, upon a peek request $\mathsf{Peek}(\mathcal{P}_i, 2)$ with $|\mathcal{L}| < |[n] \setminus \mathcal{I}| - 1$,: $\bar{\mathcal{P}}_i$ does the following.
  - For all $k \in [n] \setminus \{i\}$: compute $(s_{i\to k},\, s_{k\to i}) \leftarrow \mathsf{SharedKey}(pk_{k,i}, sk_{i,k})$, compute $\pi_{k,i} \leftarrow \mathsf{Prove}\big(\mathtt{crs},\, (pk_{k,i}, pk_{i,k}, s_{k\to i}),\, (sk_{i,k}, r_{i,k})\big)$ and broadcast $(s_{k\to i},\, \pi_{k,i})$.
  - Sample $\widetilde{\mathsf{BC}}_i^2 \leftarrow \mathbb{F}$ uniformly and broadcast $\widetilde{\mathsf{BC}}_i^2$.
  - Add i to the list $\mathcal{L}$.
- Upon a peek request $\mathsf{Peek}(\mathcal{P}_i, 2)$ with $|\mathcal{L}| = |[n] \setminus \mathcal{I}| - 1$, or if the protocol has completed round $r-1$ and $|\mathcal{L}| < |[n] \setminus \mathcal{I}| - 1$, $\mathcal{M}$ does the following:
  - Forward $\left\{\mathsf{BC}_j^2\right\}_{j\in\mathcal{I}}$ and $\left\{\mathsf{msg}_{j\to\ell}^1\right\}_{j\in\mathcal{I},\,\ell\in[n]\setminus\mathcal{I}}$ from the adversary to the left session; receive back $\left\{\mathsf{BC}_\ell^2\right\}_{\ell\in[n]\setminus\mathcal{I}}$ from the left session. There are two possible cases:
    - if $|\mathcal{L}| = |[n] \setminus \mathcal{I}| - 1$, then there is only one honest party $\bar{\mathcal{P}}_i$ for whom the adversary has not made the peek request for its Round 2 messages. We proceed to correct its shares as described below.
    - if $|\mathcal{L}| < |[n] \setminus \mathcal{I}| - 1$, this is the case where the adversary has not issued peek requests for all honest parties, but the protocol has completed round $r-1$. In this case, $\mathcal{M}$ designates an arbitrary honest party for whom the adversary has not yet issued a peek request as $\bar{\mathcal{P}}_i$ (the party whose shares would be corrected). For each of the remaining honest parties for whom the peek request has not been issued i.e. $\bar{\mathcal{P}}_s$ ($s \notin \{\mathcal{L}^r \cup \mathcal{I}\}$ and $s \neq i$): compute $(s_{s\to k},\, s_{k\to s}) \leftarrow \mathsf{SharedKey}(pk_{k,s}, sk_{s,k})$ for all $k \in [n] \setminus \{s\}$. Then compute $\pi_{k,s} \leftarrow \mathsf{Prove}\big(\mathtt{crs},\, (pk_{k,s}, pk_{s,k}, s_{k\to s}),\, (sk_{s,k}, r_{s,k})\big)$ and broadcast $\{(s_{k\to s},\, \pi_{k,s})\}_{k\in[n]\setminus\{s\}}$. Compute $M_s \leftarrow \sum_{k\in[n]} s_{s\to k}$. and send the real masked message $\widetilde{\mathsf{BC}}_s = \mathsf{BC}_s + M_s$ over broadcast. Lastly, $\mathcal{M}$ corrects $\bar{\mathcal{P}}_i$'s shares as follows.

    The **correction** for $\bar{\mathcal{P}}_i$'s shares:
    - Sample $\widetilde{\mathsf{BC}}_i^2 \leftarrow \mathbb{F}$ uniformly.
    - For each $\ell \in [n] \setminus \mathcal{I}$: compute the *correction share*

$$s_{\ell\to i} \;=\; \widetilde{\mathsf{BC}}_\ell^2 - \mathsf{BC}_\ell^2 \;-\; \sum_{k\in[n]\setminus\{i\}} s_{\ell\to k}$$

      and then compute $\pi_{\ell,i} \leftarrow \mathsf{simP}(\mathtt{crs},\, \tau,\, (pk_{\ell,i}, pk_{i,\ell}, s_{\ell\to i}))$.
    - For each $j \in \mathcal{I}$: compute $(s_{i\to j}, s_{j\to i}) \leftarrow \mathsf{SharedKey}(pk_{j,i}, sk_{i,j})$, compute $\pi_{j,i} \leftarrow \mathsf{Prove}(\mathtt{crs}, (pk_{j,i}, pk_{i,j}, s_{j\to i}), (sk_{i,j}, r_{i,j}))$
    - $\bar{\mathcal{P}}_i$ broadcast $(s_{k\to i}, \pi_{k,i})$ for all $k \in [n]$.
  - $\bar{\mathcal{P}}_i$ Broadcast $\widetilde{\mathsf{BC}}_i^2$.

– **For Rounds r messages ($r \geq 3$):** If $r = 3$, $\bar{\mathcal{P}}_i$ does the following two steps (let $(\hat{s}_{a\to b}, \hat{\pi}_{a,b})$ denote the actual share and proof broadcasted by $\mathcal{P}_b$):
- For each $j \in \mathcal{I}$ and $j^* \in \mathcal{I}$: if $\mathsf{Verify}\big(\mathtt{crs},\, (pk_{j,j^*}, pk_{j^*,j}, s_{j\to j^*}),\, \pi_{j,j^*}\big) \neq 1$, identify $\mathcal{P}_{j^*}$ as a cheater.
- For each $\ell \in [n] \setminus \mathcal{I}$ and $j \in \mathcal{I}$: if $\hat{s}_{\ell\to j} \neq s_{\ell\to j}$, where $s_{\ell\to j}$ is computed by $\bar{\mathcal{P}}_\ell$ in the generation of round-2 messages: identify $\mathcal{P}_j$ as a cheater.
- For each $j \in \mathcal{I}$: compute $M_j \leftarrow \sum_{k\in[n]} s_{j\to k}$ and recover the broadcast message $\mathsf{BC}_j^2 \leftarrow \widetilde{\mathsf{BC}}_j^2 - M_j$.

Forward $\{\mathsf{BC}^2_j\}_{j\in\mathcal{I}}$ from the adversary to the left session; receive back $\{\mathsf{BC}^2_\ell\}_{\ell\in[n]\setminus\mathcal{I}}$ from the left session.

– Once the simulator's execution is complete, output whatever $\mathcal{A}_{\mathsf{sr}}$ outputs and halt.

[a] Since $\mathcal{A}_{\mathsf{sr}}$ is a valid super-rushing adversary, NotSuperRushing never occurs; it is listed for completeness.

Fig. 3: Machine $\mathcal{M}$ for $\Pi^{\mathsf{sr}}_{\mathsf{IA}}$

Simulator $\mathcal{S}_{\mathsf{sr}}$ for $\Pi^{\mathsf{sr}}_{\mathsf{IA}}$

– Sample randomness ρ_{sr} and run $\mathcal{S}_{\mathsf{r}}$ for the adversary $\mathcal{M}(\mathcal{A}_{\mathsf{sr}}, \rho_{\mathsf{sr}})$.
– For the query $\{x_j\}_{j\in\mathcal{I}}$ to the ideal functionality issued by $\mathcal{S}_{\mathsf{r}}$, forward it and return the response.
– For the query $(\mathsf{abort}, \hat{\mathcal{I}})$ to the ideal functionality, append $\hat{\mathcal{I}}$ with the identified cheaters by $\mathcal{M}$ and forward $(\mathsf{abort}, \hat{\mathcal{I}})$ to the ideal functionality.
– Retrieve the abort message abortmsg from the ideal functionality.
– Output whatever $\mathcal{S}_{\mathsf{r}}$ outputs and halt.

Fig. 4: Simulator $\mathcal{S}_{\mathsf{sr}}$ for $\Pi^{\mathsf{sr}}_{\mathsf{IA}}$

We now define hybrid experiments to prove the theorem. Let i^* denote the honest party whose shares are corrected in $\mathcal{M}$.

Hybrid H_0: This is the real execution of $\Pi^{\mathsf{sr}}_{\mathsf{IA}}$ with the super-rushing adversary $\mathcal{A}_{\mathsf{sr}}$.

Hybrid H_1: Identical to H_0, except that the common reference string is now generated as $(\mathtt{crs}, \tau) \leftarrow \mathsf{simSetup}(1^\lambda, \mathcal{R})$ instead of $\mathtt{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, \mathcal{R})$, and when $\mathcal{P}_{i^*}$ broadcasts its shares, for each honest $\ell \in [n] \setminus \mathcal{I}$ the proof π_{ℓ,i^*} is replaced by $\mathsf{simP}(\mathtt{crs}, \tau, (pk_{\ell,i^*}, pk_{i^*,\ell}, s_{\ell\to i^*}))$.

Hybrid H_2: Identical to H_1, except that cheater detection is switched from NIZK verification to direct comparison: for each $\ell \in [n] \setminus \mathcal{I}$ and $j \in \mathcal{I}$, if the share $\hat{s}_{\ell\to j}$ broadcast by $\mathcal{P}_j$ satisfies $\hat{s}_{\ell\to j} \neq s_{\ell\to j}$ (where $s_{\ell\to j}$ is the value computed by $\bar{\mathcal{P}}_\ell$), then $\mathcal{P}_j$ is identified as a cheater.

Hybrid H_3: Machine $\mathcal{M}$ (Figure 3) runs $\mathcal{A}_{\mathsf{sr}}$ internally and interacts with the real execution of Π in the left session.

Hybrid H_4: Same as H_3 but the real execution of Π in the left session is replaced by $\mathcal{S}_{\mathsf{r}}$. Note that H_4 and $\mathcal{S}_{\mathsf{sr}}$ produce identical output distributions.

Lemma 1. *Under the zero-knowledge property of NIZK (Definition 8), the output distributions of H_0 and H_1 are computationally indistinguishable.*

Proof. The difference between H_0 and H_1 is that

1. the common reference string is now generated as $(\mathtt{crs}, \tau) \leftarrow \mathsf{simSetup}(1^\lambda, \mathcal{R})$ instead of $\mathtt{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, \mathcal{R})$.
2. the proofs $\{\pi_{\ell,i^*}\}_{\ell\in[n]\setminus\mathcal{I}}$, broadcast by $\mathcal{P}_{i^*}$ at the last peek, are replaced by simulated proofs $\mathsf{simP}(\mathtt{crs}, \tau, (pk_{\ell,i^*}, pk_{i^*,\ell}, s_{\ell\to i^*}))$.

The corresponding share values are identical in both hybrids. By the zero-knowledge property (Definition 8), for every statement $\phi = (pk_{\ell,i^*}, pk_{i^*,\ell}, s_{\ell\to i^*}) \in \mathcal{L}$, the distribution $\mathsf{Prove}(\mathtt{crs}, \phi, sk_{i^*,\ell})$ is computationally indistinguishable from $\mathsf{simP}(\mathtt{crs}, \tau, \phi)$. Applying this argument over all $|[n] \setminus \mathcal{I}|$ proofs via union bound, we get $\mathsf{H}_0 \approx_c \mathsf{H}_1$.

Lemma 2. *Under the simulation soundness of NIZK (Definition 8), the output distributions of H_1 and H_2 are computationally indistinguishable.*

Proof. The only difference between H_1 and H_2 is how cheater detection is performed: in H_1, honest parties run $\mathsf{Verify}(\mathtt{crs}, (pk_{k,j}, pk_{j,k}, s_{k\to j}), \pi_{k,j})$ for all $k \in [n]$ and $j \in \mathcal{I}$; in H_2, they instead directly check whether $\hat{s}_{\ell\to j} = s_{\ell\to j}$ (for $\ell \in [n]\setminus\mathcal{I}$ and $j \in \mathcal{I}$). We argue these two approaches differ only with negligible probability.

By contradiction, suppose a corrupted party $\mathcal{P}_j$ broadcasts a share $\hat{s}_{\ell\to j} \neq s_{\ell\to j}$ for some honest $\ell \in [n]\setminus\mathcal{I}$, and the accompanying proof $\hat{\pi}_{\ell,j}$ passes Verify. Then we can use $\mathcal{A}_{\mathsf{sr}}$ to construct a PPT adversary $\mathcal{A}$ that breaks the simulation soundness of the NIZK proof system. $\mathcal{A}_{\mathsf{sr}}$ observes polynomial number of queries of simP under different statements $\{\phi_{\ell,j} = (pk_{\ell,j}, pk_{j,\ell}, s_{\ell\to j})\}_{\ell\in[n]\setminus\mathcal{I}, j\in\mathcal{I}}$ and produces a valid proof $\hat{\pi}_{\ell,j}$ for the statement $\hat{\phi} = (pk_{\ell,j}, pk_{j,\ell}, \hat{s}_{\ell\to j})$ with $\hat{s}_{\ell\to j} \neq s_{\ell\to j}$. $\mathcal{A}$ can submit these queries and the final forgery relation and proof to the challenger and win the simulation soundness game with non-negligible probability, a contradiction.

Therefore, a corrupted party can broadcast $\hat{s}_{\ell\to j} \neq s_{\ell\to j}$ with a valid proof only with negligible probability, i.e., $\mathsf{H}_1 \approx_c \mathsf{H}_2$.

Lemma 3. *Under the computational security of NIKE (Definition 7), the output distributions of* H_2 *and* H_3 *are computationally indistinguishable.*

Proof. We argue that the changes introduced in H_3 are computationally indistinguishable from H_2. Concretely, H_3 is identical to H_2 with the following additional changes:

- For each non-last-peek honest party $\bar{\mathcal{P}}_i$: the masked message $\widetilde{\mathsf{BC}_i^2}$ is sampled uniformly from $\mathbb{F}$.
- For the last-peek party $\bar{\mathcal{P}}_{i^*}$: the masked message $\widetilde{\mathsf{BC}_{i^*}^2}$ is sampled uniformly from $\mathbb{F}$; for each honest $\ell \in [n]\setminus\mathcal{I}$, the share $s_{\ell\to i^*}$ is replaced by the correction value

$$s_{\ell\to i^*} \;=\; \widetilde{\mathsf{BC}_\ell^2} - \mathsf{BC}_\ell^2 - \sum_{k\in[n]\setminus\{i^*\}} s_{\ell\to k},$$

.

Masked messages before last peek. For each non-last-peek honest party $\bar{\mathcal{P}}_i$, the masked message in H_2 is $\widetilde{\mathsf{BC}_i^2} = \mathsf{BC}_i^2 + M_i$ where $M_i = \sum_{k\in[n]} s_{i\to k}$. Since $\mathcal{A}_{\mathsf{sr}}$ holds only corrupted parties' secret keys, by NIKE security each honest-to-honest share $s_{i\to k}$ (for $k \in [n]\setminus\mathcal{I}$) is computationally indistinguishable from uniform given $pk_{i,k}, pk_{k,i}$. Hence M_i is computationally indistinguishable from uniform, and so $\widetilde{\mathsf{BC}_i^2}$ is computationally indistinguishable from a uniformly random element of $\mathbb{F}$.

Last-peek masked message and correction share. Let $c_\ell = \sum_{k\in[n]\setminus\{i^*\}} s_{\ell\to k}$ be the sum of ℓ's shares to all parties other than i^*. In H_2, the adversary sees the tuple

$$\big(s_{\ell\to i^*},\ \mathsf{simP}(\mathtt{crs}, \tau, (pk_{\ell,i^*}, pk_{i^*,\ell}, s_{\ell\to i^*})),\ \mathsf{BC}_\ell^2 + c_\ell + s_{\ell\to i^*}\big),$$

where $s_{\ell\to i^*}$ is the real NIKE share. In H_3, the masked message $\widetilde{\mathsf{BC}_\ell^2}$ is sampled uniformly from $\mathbb{F}$ and the correction share is set to $s'_{\ell\to i^*} = \widetilde{\mathsf{BC}_\ell^2} - \mathsf{BC}_\ell^2 - c_\ell$, yielding

$$\big(s'_{\ell\to i^*},\ \mathsf{simP}(\mathtt{crs}, \tau, (pk_{\ell,i^*}, pk_{i^*,\ell}, s'_{\ell\to i^*})),\ \widetilde{\mathsf{BC}_\ell^2}\big).$$

Since the map $s \mapsto (s,\ \mathsf{BC}_\ell^2 + c_\ell + s)$ is a bijection, when $s_{\ell\to i^*}$ is truly random, so is $\mathsf{BC}_\ell^2 + c_\ell + s_{\ell\to i^*}$, and vice versa. Since $s_{\ell\to i^*}$ is only pseudorandom (by NIKE), the two distributions are computationally indistinguishable, i.e., $\mathsf{H}_2 \approx_c \mathsf{H}_3$.

Lemma 4. *Let* Π *be computationally secure with identifiable abort against rushing adversaries. Then the output distributions of* H_3 *and* H_4 *are computationally indistinguishable.*

Proof. Suppose a PPT distinguisher $\mathcal{A}$ separates H_3 from H_4 with non-negligible probability. We construct a rushing adversary $\mathcal{A}_{\mathsf{r}} := \mathcal{M}(\mathcal{A}_{\mathsf{sr}}, \cdot)$ breaking the security of Π. By construction (Figure 3), $\mathcal{M}$ collects all corrupted parties' round-$(r-1)$ messages from $\mathcal{A}_{\mathsf{sr}}$ before forwarding them to the left session, and observes all honest parties' round-r messages of Π before any corrupted round-r message is committed, which precisely

matches the rushing model. By the computational IA security of Π against rushing adversaries, there exists $\mathcal{S}_{\mathsf{r}}$ such that:

$$\textsc{Real}_{\Pi^{\mathsf{sr}}_{\mathsf{IA}},\mathcal{I},\mathcal{M}(\mathcal{A}_{\mathsf{sr}},\cdot)(z)}(\mathbf{x},\lambda) \approx_c \textsc{Ideal}^{\mathsf{IA}}_{f,\mathcal{I},\mathcal{S}_{\mathsf{r}}(z)}(\mathbf{x},\lambda).$$

The only difference between H_3 and H_4 is that the real Π execution in the left session is replaced by $\mathcal{S}_{\mathsf{r}}$. Hence any distinguisher between H_3 and H_4 yields a distinguisher for the computational IA security of Π, a contradiction.

Combining Lemmas 1, 2, 3, and 4, we have $\mathsf{H}_0 \approx_c \mathsf{H}_4$. Since H_0 is the real execution of $\Pi^{\mathsf{sr}}_{\mathsf{IA}}$ and H_4 corresponds to the ideal execution with $\mathcal{S}_{\mathsf{sr}}$, we conclude that $\Pi^{\mathsf{sr}}_{\mathsf{IA}}$ is computationally secure with identifiable abort against super-rushing adversaries.

Finally, we note that in an honest-majority setting, the CRS assumption can be eliminated by instead using multi-string NIZK [GO07], where each party broadcasts its own CRS in the first round, and the NIZK guarantees hold provided that a majority of the CRS values are generated honestly. We state the corresponding corollary below.

Corollary 2. *Let Π be any R-round computationally secure n-party protocol with identifiable abort tolerating at most $t < n/2$ static corruptions. Without loss of generality, assume Π uses only broadcast communication from (and including) round 2. Then $\Pi^{\mathsf{sr}-hm}_{\mathsf{IA}}$ is computationally secure with* identifiable abort *against super-rushing adversaries tolerating at most t static corruptions, assuming a computationally secure non-interactive key exchange scheme and a multi-string non-interactive zero-knowledge proof system. That is, for every PPT super-rushing adversary $\mathcal{A}_{\mathsf{sr}}$ there exists an expected polynomial-time simulator $\mathcal{S}_{\mathsf{sr}}$ such that for all $\mathcal{I} \subseteq [n]$ with $|\mathcal{I}| \leq t$, the following two distributions are computationally indistinguishable:*

$$\left\{\textsc{Real}_{\Pi^{\mathsf{sr}-hm}_{\mathsf{IA}},\mathcal{I},\mathcal{A}_{\mathsf{sr}}(z)}(\mathbf{x},\lambda)\right\}_{\mathbf{x}\in(\{0,1\}^*)^n,\lambda\in\mathbb{N}}, \left\{\textsc{Ideal}^{\mathsf{IA}}_{f,\mathcal{I},\mathcal{S}_{\mathsf{sr}}(z)}(\mathbf{x},\lambda)\right\}_{\mathbf{x}\in(\{0,1\}^*)^n,\lambda\in\mathbb{N}}$$

where $\textsc{Ideal}^{\mathsf{IA}}$ *denotes the ideal execution with the identifiable abort.*

5.2 Communication Complexity

Let CC'_{BC} and CC denote the broadcast communication complexity in round 2 and total communication complexity of the underlying rushing protocol Π respectively. The total communication complexity of $\Pi^{\mathsf{sr}}_{\mathsf{IA}}$ is $CC + \mathcal{O}(n \cdot CC'_{\mathsf{BC}}) + \kappa(n \cdot CC'_{\mathsf{BC}})$, where $\kappa(m)$ denotes the NIZK proof size on m statements (each of constant size). Recall that Round 1 involves broadcast of $\mathcal{O}(n \cdot CC'_{\mathsf{BC}})$ NIKE public keys. Next, for each of the derived shared keys, a NIZK proof is generated, resulting in $\kappa(n \cdot CC'_{\mathsf{BC}})$ additional communication. The concrete proof size depends on the chosen NIZK instantiation [Gro16,GLS+23,DP26].

Acknowledgments. This publication is part of the gravitation project Challenges in Cyber Security (CiCS) with file number 024.006.037 which is financed by the Dutch Research Council (NWO). Michele Ciampi is partially supported by Input Output Research Hub (IORH) of the University of Edinburgh, and by Sunday Group Inc. Divya Ravi was supported by NWO (Dutch Research Council) grant with number VI.Veni.242.295.

References

ABT19. Benny Applebaum, Zvika Brakerski, and Rotem Tsabary. Degree 2 is complete for the round-complexity of malicious MPC. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 504–531, Darmstadt, Germany, May 19–23, 2019.

ACCI25. Gilad Asharov, Anirudh Chandramouli, Ran Cohen, and Yuval Ishai. Peeking into the future: MPC resilient to super-rushing adversaries. Lecture Notes in Computer Science, pages 390–420, June 2025.

ACGJ18. Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Round-optimal secure multiparty computation with honest majority. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 395–424, Santa Barbara, CA, USA, August 19–23, 2018.

ACGJ19. Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Two round information-theoretic MPC with malicious security. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 532–561, Darmstadt, Germany, May 19–23, 2019.

ACJ17. Prabhanjan Ananth, Arka Rai Choudhuri, and Abhishek Jain. A new approach to round-optimal secure multiparty computation. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 468–499, Santa Barbara, CA, USA, August 20–24, 2017.

ADLS91. Hagit Attiya, Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. In *23rd Annual ACM Symposium on Theory of Computing*, pages 359–369, New Orleans, LA, USA, May 6–8, 1991. ACM Press.

AES26. Gilad Asharov, Fatima Elsheimy, and Gilad Stern. Fast and efficient perfectly secure network-agnostic secure computation. In Joan Daemen and Emmanuel Thomé, editors, *Advances in Cryptology – EUROCRYPT 2026*, pages 185–213, Cham, 2026. Springer Nature Switzerland.

AKP20a. Benny Applebaum, Eliran Kachlon, and Arpita Patra. The resiliency of MPC with low interaction: The benefit of making errors (extended abstract). In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 562–594, Durham, NC, USA, November 16–19, 2020.

AKP20b. Benny Applebaum, Eliran Kachlon, and Arpita Patra. The round complexity of perfect MPC with active security and optimal resiliency. In *61st Annual Symposium on Foundations of Computer Science*, pages 1277–1284, Durham, NC, USA, November 16–19, 2020. IEEE Computer Society Press.

AKP22. Benny Applebaum, Eliran Kachlon, and Arpita Patra. Round-optimal honest-majority MPC in minicrypt and with everlasting security - (extended abstract). In *TCC 2022: 20th Theory of Cryptography Conference, Part II*, Lecture Notes in Computer Science, pages 103–120, November 2022.

AKP23. Benny Applebaum, Eliran Kachlon, and Arpita Patra. The round complexity of statistical MPC with optimal resiliency. pages 1527–1536. ACM Press, 2023.

BCB+24. Hendrik Ballhausen, Stefanie Corradini, Claus Belka, Dan Bogdanov, Luca Boldrini, Francesco Bono, Christian Goelz, Guillaume Landry, Giulia Panza, Katia Parodi, Riivo Talviste, Huong Elena Tran, Maria Antonietta Gambacorta, and Sebastian Marschner. Privacy-friendly evaluation of patient data with secure multiparty computation in a european pilot study. *npj Digit. Medicine*, 7(1), 2024.

BCD+09. Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography and Data Security, 13th International Conference, FC 2009, Accra Beach, Barbados, February 23-26, 2009. Revised Selected Papers*, Lecture Notes in Computer Science, pages 325–343. Springer, 2009.

BCLZL23. Renas Bacho, Daniel Collins, Chen-Da Liu-Zhang, and Julian Loss. Network-agnostic security comes (almost) for free in DKG and MPC. In *Advances in Cryptology – CRYPTO 2023, Part I*, Lecture Notes in Computer Science, pages 71–106, Santa Barbara, CA, USA, August 2023.

BGJ+18. Saikrishna Badrinarayanan, Vipul Goyal, Abhishek Jain, Yael Tauman Kalai, Dakshita Khurana, and Amit Sahai. Promise zero knowledge and its applications to round optimal MPC. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 459–487, Santa Barbara, CA, USA, August 19–23, 2018.

BGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 1–10, Chicago, IL, USA, May 2–4, 1988. ACM Press.

BKL19. Erica Blum, Jonathan Katz, and Julian Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 131–150, Nuremberg, Germany, December 1–5, 2019.

BL18. Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 500–532, Tel Aviv, Israel, April 29 – May 3, 2018.

BLV17. Azer Bestavros, Andrei Lapets, and Mayank Varia. User-centric distributed solutions for privacy-preserving analytics. *Commun. ACM*, 60(2):37–39, 2017.

BLW08. Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Computer Security – ESORICS 2008*, volume 5283 of *LNCS*, pages 192–206. Springer, 2008.

BOCG93. Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, page 52–61, New York, NY, USA, 1993. Association for Computing Machinery.

BOKR94. Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '94, page 183–192, New York, NY, USA, 1994. Association for Computing Machinery.

BTHN10. Zuzana Beerliová-Trubíniová, Martin Hirt, and Jesper Buus Nielsen. On the theoretical gap between synchronous and asynchronous mpc protocols. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '10, page 211–218, New York, NY, USA, 2010. Association for Computing Machinery.

Can03. Ran Canetti. Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239, 2003.

CCG+20. Arka Rai Choudhuri, Michele Ciampi, Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. Round optimal secure multiparty computation from minimal assumptions. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 291–319, Durham, NC, USA, November 16–19, 2020.

CDR+23. Michele Ciampi, Ivan Damgård, Divya Ravi, Luisa Siniscalchi, Yu Xia, and Sophia Yakoubov. Broadcast-optimal four-round MPC in the plain model. In *TCC 2023: 21st Theory of Cryptography Conference, Part II*, Lecture Notes in Computer Science, pages 3–32, November 2023.

CDvdG88. David Chaum, Ivan Damgård, and Jeroen van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 87–119, Santa Barbara, CA, USA, August 16–20, 1988.

CGZ20. Ran Cohen, Juan A. Garay, and Vassilis Zikas. Broadcast-optimal two-round MPC. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 828–858, Zagreb, Croatia, May 10–14, 2020.

Cle86. Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th Annual ACM Symposium on Theory of Computing*, pages 364–369, Berkeley, CA, USA, May 28–30, 1986. ACM Press.

CRSW22. Michele Ciampi, Divya Ravi, Luisa Siniscalchi, and Hendrik Waldner. Round-optimal multi-party computation with identifiable abort. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part I*, volume 13275 of *Lecture Notes in Computer Science*, pages 335–364, Trondheim, Norway, May 30 – June 3, 2022.

DGKN09. Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In *Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography: PKC '09*, Irvine, page 160–179, Berlin, Heidelberg, 2009. Springer-Verlag.

DMR+21. Ivan Damgård, Bernardo Magri, Divya Ravi, Luisa Siniscalchi, and Sophia Yakoubov. Broadcast-optimal two round MPC with an honest majority. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 155–184, Virtual Event, August 16–20, 2021.

DP26. Benjamin E. Diamond and Jim Posen. Polylogarithmic proofs for multilinears over Binary towers. In *Advances in Cryptology – EUROCRYPT 2026*, Lecture Notes in Computer Science. Springer, 2026.

FHKP13. Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. Non-interactive key exchange. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013: 16th International Conference*

on Theory and Practice of Public Key Cryptography, volume 7778 of *Lecture Notes in Computer Science*, pages 254–271, Nara, Japan, February 26 – March 1, 2013.

GIKR02. Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. On 2-round secure multiparty computation. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 178–193, Santa Barbara, CA, USA, August 18–22, 2002.

GJPR21. Aarushi Goel, Abhishek Jain, Manoj Prabhakaran, and Rajeev Raghunath. On communication models and best-achievable security in two-round MPC. In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part II*, volume 13043 of *Lecture Notes in Computer Science*, pages 97–128, Raleigh, NC, USA, November 8–11, 2021.

GLS15. S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 63–82, Santa Barbara, CA, USA, August 16–20, 2015.

GLS$^+$23. Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In *Advances in Cryptology – CRYPTO 2023, Part II*, Lecture Notes in Computer Science, pages 193–226, Santa Barbara, CA, USA, August 2023.

GM17. Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 581–612, Santa Barbara, CA, USA, August 20–24, 2017.

GMPP16. Sanjam Garg, Pratyay Mukherjee, Omkant Pandey, and Antigoni Polychroniadou. The exact round complexity of secure computation. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 448–476, Vienna, Austria, May 8–12, 2016.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.

GO07. Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 323–341, Santa Barbara, CA, USA, August 19–23, 2007.

Gol04. Oded Goldreich. *Foundations of cryptography, volume 2*. Cambridge university press Cambridge, 2004.

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326, Vienna, Austria, May 8–12, 2016.

GS18. Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 468–499, Tel Aviv, Israel, April 29 – May 3, 2018.

HHPV18. Shai Halevi, Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Round-optimal secure multi-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 488–520, Santa Barbara, CA, USA, August 19–23, 2018.

HLP11. Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 132–150, Santa Barbara, CA, USA, August 14–18, 2011.

JLZS26. Xiaoyu Ji, Chen-Da Liu-Zhang, and Yifan Song. Perfectly secure network-agnostic mpc comes for free. In Joan Daemen and Emmanuel Thomé, editors, *Advances in Cryptology – EUROCRYPT 2026*, pages 337–365, Cham, 2026. Springer Nature Switzerland.

Kat07. Jonathan Katz. On achieving the "best of both worlds" in secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th Annual ACM Symposium on Theory of Computing*, pages 11–20, San Diego, CA, USA, June 11–13, 2007. ACM Press.

Lin20. Yehuda Lindell. Secure multiparty computation. *Commun. ACM*, 64(1):86–96, December 2020.

LZLM$^+$20. Chen-Da Liu-Zhang, Julian Loss, Ueli Maurer, Tal Moran, and Daniel Tschudi. MPC with synchronous security and asynchronous responsiveness. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 92–119, Daejeon, South Korea, December 7–11, 2020.

MW16. Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 735–763, Vienna, Austria, May 8–12, 2016.

PR18a. Arpita Patra and Divya Ravi. On the exact round complexity of secure three-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 425–458, Santa Barbara, CA, USA, August 19–23, 2018.

PR18b. Arpita Patra and Divya Ravi. On the power of hybrid networks in multi-party computation. *IEEE Transactions on Information Theory*, 64(6):4207–4227, 2018.

PS18. Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 3–33, Tel Aviv, Israel, April 29 – May 3, 2018.

SARN20. Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. On the optimality of optimistic responsiveness. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 839–857, Virtual Event, USA, November 9–13, 2020. ACM Press.

SvHA+19. Alex Sangers, Maran van Heesch, Thomas Attema, Thijs Veugen, Mark Wiggerman, Jan Veldsink, Oscar Bloemen, and Daniël Worm. Secure multiparty pagerank algorithm for collaborative fraud detection. In *Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers*, page 605–623, Berlin, Heidelberg, 2019. Springer-Verlag.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

A Building Blocks

Definition 9 (Digital Signature Scheme [Can03]). *A* digital signature scheme *for message space $\mathcal{M}$ is a triple of algorithms* $(\mathsf{Sig.KeyGen}, \mathsf{Sign}, \mathsf{Ver})$*:*

- $\mathsf{Sig.KeyGen}(1^\lambda)$*: on input the security parameter 1^λ, outputs a key pair* $(\mathsf{sk}, \mathsf{vk})$*.*
- $\mathsf{Sign}(\mathsf{sk}, m)$*: on input a signing key* sk *and a message $m \in \mathcal{M}$, outputs a signature σ.*
- $\mathsf{Ver}(\mathsf{vk}, m, \sigma)$*: on input a verification key* vk*, a message $m \in \mathcal{M}$, and a signature σ, outputs a bit $b \in \{0, 1\}$.*

Correctness. *For all $\lambda \in \mathbb{N}$, $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$, and $m \in \mathcal{M}$:*

$$\Pr[\mathsf{Ver}(\mathsf{vk},\, m,\, \mathsf{Sign}(\mathsf{sk}, m)) = 1] = 1.$$

Definition 10 (Unforgeability of Digital Signature). *Let* $(\mathsf{Sig.KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ *be a signature scheme and $\mathcal{A}$ a* PPT *adversary. Define the* signature forgery experiment $\mathsf{EUF\text{-}CMA}^{\mathsf{Sig}}(\lambda, \mathcal{A})$*:*

1. *The challenger samples $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$, sends* vk *to $\mathcal{A}$, and initialises $Q \leftarrow \emptyset$.*
2. *$\mathcal{A}$ adaptively submits signing queries: on query m, the challenger returns $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$ and records $Q \leftarrow Q \cup \{m\}$.*
3. *$\mathcal{A}$ outputs (m^*, σ^*).*
4. *$\mathcal{A}$* wins *if $\mathsf{Ver}(\mathsf{vk}, m^*, \sigma^*) = 1$ and $m^* \notin Q$.*

The signature scheme is EUF-CMA secure *if for every* PPT *$\mathcal{A}$:*

$$\Pr\Big[\mathsf{EUF\text{-}CMA}^{\mathsf{Sig}}(\lambda, \mathcal{A}) = 1\Big] \leq \mathrm{negl}(\lambda).$$

Definition 11 (Additive Secret Sharing). *Let $\mathbb{G}$ be a finite abelian group written additively. An n-out-of-n* additive secret sharing scheme *over $\mathbb{G}$ is a pair of algorithms* $(\mathsf{Share}, \mathsf{Rec})$*:*

- $\mathsf{Share}(s)$*: on input $s \in \mathbb{G}$, sample $s_1, \ldots, s_{n-1} \leftarrow \mathbb{G}$ uniformly at random, set $s_n = s - \sum_{i=1}^{n-1} s_i$, and output $(s_1, \ldots, s_n)$.*
- $\mathsf{Rec}(s_1, \ldots, s_n)$*: output $\sum_{i=1}^{n} s_i$.*

that satisfies the following properties:
Correctness. *For all $s \in \mathbb{G}$ and $(s_1, \ldots, s_n) \leftarrow \mathsf{Share}(s)$, $\mathsf{Rec}(s_1, \ldots, s_n) = s$.*
Perfect Privacy. *For any strict subset $T \subsetneq [n]$, any two secrets $s, s' \in \mathbb{G}$, and any $\mathbf{y} \in \mathbb{G}^{|T|}$:*

$$\Pr_{\mathsf{Share}(s)}[\{s_i\}_{i \in T} = \mathbf{y}] \;=\; \Pr_{\mathsf{Share}(s')}[\{s_i\}_{i \in T} = \mathbf{y}] \;=\; \frac{1}{|\mathbb{G}|^{|T|}}.$$

B Security Proof for Broadcast-only Protocols

Proof. Let $\mathcal{A}_{\mathsf{sr}}$ be any super-rushing adversary for the protocol Π. To prove security, we need to construct a simulator $\mathcal{S}_{\mathsf{sr}}$ against $\mathcal{A}_{\mathsf{sr}}$. To construct $\mathcal{S}_{\mathsf{sr}}$, we first construct a machine $\mathcal{M}(\mathcal{A}_{\mathsf{sr}}, \cdot)$ that internally runs $\mathcal{A}_{\mathsf{sr}}$ and exposes an external interface, which we call the *left session*, through which honest parties' messages of Π are exchanged regardless of whether the left session is connected to a real execution or a simulator. We show that $\mathcal{M}(\mathcal{A}_{\mathsf{sr}}, \cdot)$ is a valid rushing adversary for the rushing protocol Π, and then invoke the rushing security of Π to obtain the desired simulator $\mathcal{S}_{\mathsf{sr}}$.

The machine $\mathcal{M}(\mathcal{A}_{\mathsf{sr}}, \rho_{\mathsf{sr}})$

$\mathcal{M}$ runs $\mathcal{A}_{\mathsf{sr}}$ as a subroutine with randomness ρ_{sr} and relays all messages between $\mathcal{A}_{\mathsf{sr}}$ and the left session without modification. Whenever $\mathcal{A}_{\mathsf{sr}}$ asks to receive a message $\mathsf{msg}^{r}_{i^* \to j^*}$ for $j^* \in \mathcal{I}$ and $i^* \in [n] \setminus \mathcal{I}$, $\mathcal{M}$ does the following:

- verify that for every $i \in \mathcal{I}$, $\mathcal{M}$ has already received the round-$r-1$ message $\mathsf{msg}^{r-1}_{i \to j^*}$. If this does not hold, call this event $\mathsf{NotSuperRushing}$ and abort.[a]

It then forwards the messages received from the left session to $\mathcal{A}_{\mathsf{sr}}$. When the execution ends, $\mathcal{M}$ outputs whatever $\mathcal{A}_{\mathsf{sr}}$ outputs and halts.

[a] Note that since $\mathcal{M}$ interacts with a valid super-rushing adversary $\mathcal{A}_{\mathsf{sr}}$, $\mathsf{NotSuperRushing}$ never happens. It is included here just for completeness.

Fig. 5: Machine $\mathcal{M}$ for Theorem 1.

Simulator $\mathcal{S}_{\mathsf{sr}}$

$\mathcal{S}_{\mathsf{sr}}$ internally runs $\mathcal{S}_{\mathsf{r}}$ for the rushing adversary $\mathcal{A}_{\mathsf{r}} = \mathcal{M}(\mathcal{A}_{\mathsf{sr}}, \rho_{\mathsf{sr}})$:

- Sample ρ_{sr} and instantiate $\mathcal{S}_{\mathsf{r}}$ with adversary $\mathcal{M}(\mathcal{A}_{\mathsf{sr}}, \rho_{\mathsf{sr}})$.
- Forward all messages between $\mathcal{S}_{\mathsf{r}}$ and the ideal functionality without modification. (This includes all messages to the ideal functionality allowed as per the relevant output delivery guarantee of Π, as defined in Definition 2)
- Output whatever $\mathcal{S}_{\mathsf{r}}$ outputs and halt.

Fig. 6: Simulator $\mathcal{S}_{\mathsf{sr}}$ for Theorem 1.

We now describe a sequence of hybrids below. **Hybrid** H_0: The real execution against $\mathcal{A}_{\mathsf{sr}}$.

Hybrid H_1: The execution $\mathcal{M}(\mathcal{A}_{\mathsf{sr}}, \rho_{\mathsf{sr}})$, where $\mathcal{M}$ interacts with the real execution of Π via the left session.

Hybrid H_2: Same as H_1, except the left session of $\mathcal{M}$ is replaced by $\mathcal{S}_{\mathsf{r}}$. Note that H_2 is exactly $\mathcal{S}_{\mathsf{sr}}$.

Lemma 5. H_0 *and* H_1 *are identically distributed.*

Proof. By construction, $\mathcal{M}$ relays every message between $\mathcal{A}_{\mathsf{sr}}$ and the left session without modification. Therefore, the two distributions are identical.

Lemma 6. *Let Π be a perfectly (resp. statistically / computationally) secure rushing protocol. Then the output distributions of* H_1 *and* H_2 *are perfectly (resp. statistically / computationally) indistinguishable.*

Proof. We claim that $\mathcal{M}(\mathcal{A}_{\mathsf{sr}}, \cdot)$ is a valid rushing adversary for Π. The key observation is that $\mathcal{A}_{\mathsf{sr}}$'s super-rushing request for round-$(r{+}1)$ messages of an honest party $\mathcal{P}_{i^*}$ can only occur after all corrupted parties' round-r messages have been sent to $\mathcal{P}_{i^*}$. Since Π uses only broadcast, once the adversarial round-r messages are sent, the same messages will be delivered to every honest party and cannot be modified. In particular, they are fully determined before any honest party's round-$(r{+}1)$ messages are generated or delivered.

Therefore, $\mathcal{M}$'s interaction with the left session is consistent with the rushing model: the adversary may choose its round-$(r+1)$ messages after seeing the honest parties' round-$(r+1)$ messages, but it must have sent all corrupted parties' round-r messages before receiving any honest party's round-$(r+1)$ message.

Set $\mathcal{A}_{\mathsf{r}} := \mathcal{M}(\mathcal{A}_{\mathsf{sr}}, \cdot)$. The only difference between H_1 and H_2 is that the left session, which in H_1 is connected to the real execution of Π, is replaced by $\mathcal{S}_{\mathsf{r}}$ in H_2. By assumption, since Π must be secure against the rushing adversary $\mathcal{A}_{\mathsf{r}}$, there exists a simulator $\mathcal{S}_{\mathsf{r}}$ such that the following distributions are perfectly (resp. statistically / computationally) indistinguishable:

$$\textsc{Real}_{\Pi,\mathcal{I},\mathcal{A}_{\mathsf{r}}(z)}(\mathbf{x},\lambda)\ ,\ \textsc{Ideal}_{f,\mathcal{I},\mathcal{S}_{\mathsf{r}}(z)}(\mathbf{x},\lambda),$$

Suppose there exists an unbounded (resp. PPT) distinguisher $\mathcal{A}$ that distinguishes H_1 from H_2 with non-negligible probability. We construct a distinguisher $\mathcal{A}'$ that contradicts the rushing security of Π as follows: $\mathcal{A}'$ runs $\mathcal{M}(\mathcal{A}_{\mathsf{sr}}, \rho_{\mathsf{sr}})$ as its internal rushing adversary, feeds the resulting transcript to $\mathcal{A}$, and outputs $\mathcal{A}$'s decision. Since $\mathcal{A}'$ interacts with the real execution of Π exactly as H_1 and with the ideal execution exactly as H_2, any non-negligible advantage of $\mathcal{A}$ yields a non-negligible advantage for $\mathcal{A}'$, contradicting rushing security of Π.

Combining Lemmas 5 and 6 completes the proof.

C Round and Communication-Pattern Preserving Compiler for Selective Abort Security

In this section, we present a round-preserving compiler that upgrades a rushing protocol to super-rushing security, while also preserving the type of communication channels that were used in the underlying rushing protocol. More specifically, unlike in Section 4, we do not assume that the underlying rushing protocol uses only broadcast communication from Round 2 onwards.

The main modification we make to the protocol $\Pi^{\mathsf{sr}}_{\mathsf{SA}}$ is as follows: Instead of generating masks only for the broadcast messages in Round 2, each party $\mathcal{P}_i$ now, in every round t, generates a mask $M_i^{\mathsf{BC},t}$ for its broadcast message in Round t, as well as a mask $M_{i\to j}^{\mathsf{P2P},t}$ for its point-to-point message to each $\mathcal{P}_j$ in Round t. Each of these masks is additively shared, where $\{m_{i,k}^{\mathsf{BC},t}\}_{k\in[n]}$ denote the additive shares for the masks $M_i^{\mathsf{BC},t}$ and $\{m_{i\to j,k}^{\mathsf{P2P},t}\}_{k\in[n]}$ denotes the additive shares for the masks $M_{i\to j}^{\mathsf{P2P},t}$. For each additive share $m_{i,k}^{\mathsf{BC},t}$ corresponding to a broadcast-message mask, n one-time MACs are generated (similar to in $\Pi^{\mathsf{sr}}_{\mathsf{SA}}$). However, for each additive share corresponding to a point-to-point message, say $m_{i\to j,k}^{\mathsf{P2P},t}$, only one MAC needs to be generated: the recipient of the additive share (i.e., $\mathcal{P}_k$) receives the MAC tag, while the recipient of the point-to-point message (i.e., $\mathcal{P}_j$) receives the MAC key. In Round t, $\mathcal{P}_j$ would receive the masked point-to-point Round t message from $\mathcal{P}_i$ together with the relevant additive shares from the other parties, together with the MAC tags. $\mathcal{P}_j$ can now verify these shares using the corresponding MAC keys that it holds. Upon successful verification $\mathcal{P}_j$ proceeds to reconstruct the mask $M_{i\to j}^{\mathsf{P2P},t}$, which can now be used to recover the point-to-point message of the underlying protocol. The formal description of the communication-pattern preserving compiler appears below.

We give a formal description of our compiler below. The compiler takes the following building blocks:

- An R-round rushing protocol Π, expressed as a set of algorithms
$$\{(\mathsf{fst\text{-}msg}_i, \mathsf{nxt\text{-}msg}_i^r, \mathsf{output}_i)\}_{i\in[n],\ r\in\{2,\ldots,R\}}$$
  - $\left\{\left(\mathsf{BC}_i^1, \{\mathsf{msg}_{i\to j}^1\}_{j\in[n]}\right) \leftarrow \mathsf{fst\text{-}msg}_i(x_i,\rho_i)\right\}_{i\in[n]}$
  - $\left\{\left(\mathsf{BC}_i^r, \{\mathsf{msg}_{i\to j}^r\}_{j\in[n]}\right) \leftarrow \mathsf{nxt\text{-}msg}_i^r(x_i,\rho_i,T_i^{<r})\right\}_{i\in[n],r\in\{2,\ldots,R\}}$
  - $\left\{y_i \leftarrow \mathsf{output}_i(x_i,\rho_i,T_i^{<R+1})\right\}_{i\in[n]}$

  where r represents the current round number, BC_i^r represents the broadcast messages of party $\mathcal{P}_i$ in round r, $\mathsf{msg}_{i\to j}^r$ represents the point-to-point messages from $\mathcal{P}_i$ to $\mathcal{P}_j$, y_i represents the output of $\mathcal{P}_i$ and $T_i^{<r}$ represents its transcript that includes messages until (but not including) round r.
- An n-out-of-n additive secret sharing scheme $(\mathsf{Share}, \mathsf{Rec})$ over $(\mathbb{F}, +)$, where $\mathbb{F}$ is a field into which all messages of Π are embedded (see Definition 11).
- A one-time $\mathsf{EUF\text{-}1\text{-}CMA}$-secure MAC scheme $(\mathsf{Setup}, \mathsf{Auth}, \mathsf{Vrfy})$ with message space $\mathbb{F}$ (see Definition 6).

Protocol $\Pi_{\mathsf{SA}}^{cp,\mathsf{sr}}$

Input: Each party $\mathcal{P}_i$ has input x_i and randomness ρ_i.

Construction: The R-round super-rushing protocol proceeds as follows:

– **Round 1:** Each party $\mathcal{P}_i$ does the following:

- Run $\mathsf{Setup}(1^\lambda)$ to generate a set of verification keys $\left\{\left(\kappa_{i,k}^{\mathsf{BC},j,t}, \kappa_{i\to j,k}^{\mathsf{P2P},t}\right)\right\}_{j,k\in[n],t\in\{2,\ldots,R\}}$, and send the verification keys $\left\{\left(\kappa_{i,k}^{\mathsf{BC},j,t}, \kappa_{i\to j,k}^{\mathsf{P2P},t}\right)\right\}_{k\in[n],t\in\{2,\ldots,R\}}$ to party $\mathcal{P}_j$ through point-to-point channel for all $j\in[n]$.
- Sample masks $\left\{\left(M_i^{\mathsf{BC},t}, \left\{M_{i\to j}^{\mathsf{P2P},t}\right\}_{j\in[n]}\right)\right\}_{t\in\{2,\ldots,R\}}$ uniformly from $\mathbb{F}$.
- For each broadcast mask $M_i^{\mathsf{BC},t}$, compute $(m_{i,1}^{\mathsf{BC},t},\ldots,m_{i,n}^{\mathsf{BC},t}) \leftarrow \mathsf{Share}(M_i^{\mathsf{BC},t})$. For each share $m_{i,k}^{\mathsf{BC},t}$, compute $\tau_{i,k}^{\mathsf{BC},j,t} \leftarrow \mathsf{Auth}(\kappa_{i,k}^{\mathsf{BC},j,t}, m_{i,k}^{\mathsf{BC},t})$ for all $j\in[n]$. Send $\left(m_{i,k}^{\mathsf{BC},t}, \left\{\tau_{i,k}^{\mathsf{BC},j,t}\right\}_{j\in[n]}\right)$ over point-to-point channel to $\mathcal{P}_k$.
- For each point-to-point mask $M_{i\to j}^{\mathsf{P2P},t}$, compute $(m_{i\to j,1}^{\mathsf{P2P},t},\ldots,m_{i\to j,n}^{\mathsf{P2P},t}) \leftarrow \mathsf{Share}(M_{i\to j}^{\mathsf{P2P},t})$. For each share $m_{i\to j,k}^{\mathsf{P2P},t}$, compute $\tau_{i\to j,k}^{\mathsf{P2P},t} \leftarrow \mathsf{Auth}(\kappa_{i\to j,k}^{\mathsf{P2P},t}, m_{i\to j,k}^{\mathsf{P2P},t})$ for all $j\in[n]$. Send $\left\{(m_{i\to j,k}^{\mathsf{P2P},t}, \tau_{i\to j,k}^{\mathsf{P2P},t})\right\}_{j\in[n]}$ over point-to-point channel to $\mathcal{P}_k$.
- Compute $\left(\mathsf{BC}_i^1, \{\mathsf{msg}_{i\to j}^1\}_{j\in[n]}\right) \leftarrow \mathsf{fst\text{-}msg}_i(x_i,\rho_i)$.
- Send BC_i^1 over broadcast channel.
- Send $\mathsf{msg}_{i\to j}^1$ over point-to-point channel to $\mathcal{P}_j$ for all $j\in[n]$.

– **Round** r **($2\le r\le R$):** Each party $\mathcal{P}_i$ proceeds as follows:

- If $\mathcal{P}_i$ is missing any share from round $r-1$, it outputs $\perp$.
- If $r\neq 2$ do the following, otherwise skip to the next step:
  - For every $k,\ell\in[n]$, run the verification algorithm to check $\mathsf{Vrfy}(\kappa_{k,\ell}^{\mathsf{BC},i,r-1}, m_{k,\ell}^{\mathsf{BC},r-1}, \tau_{k,\ell}^{\mathsf{BC},i,r-1})$ and $\mathsf{Vrfy}(\kappa_{k\to i,\ell}^{\mathsf{P2P},r-1}, m_{k\to i,\ell}^{\mathsf{P2P},r-1}, \tau_{k\to i,\ell}^{\mathsf{P2P},r-1})$. If any share fails the verification, $\mathcal{P}_i$ outputs $\perp$.
  - Reconstruct the masks $\left\{\left(M_k^{\mathsf{BC},r-1}, M_{k\to i}^{\mathsf{P2P},r-1}\right)\right\}_{k\in[n]}$: $M_k^{\mathsf{BC},r-1} \leftarrow \mathsf{Rec}(m_{k,1}^{\mathsf{BC},r-1},\ldots,m_{k,n}^{\mathsf{BC},r-1})$ and $M_{k\to i}^{\mathsf{P2P},r-1} \leftarrow \mathsf{Rec}(m_{k\to i,1}^{\mathsf{P2P},r-1},\ldots,m_{k\to i,n}^{\mathsf{P2P},r-1})$.
  - For every masked message $\widetilde{\mathsf{BC}_k^{r-1}}$ and $\widetilde{\mathsf{msg}_{k\to i}^{r-1}}$ received in the previous round, recover $\mathsf{BC}_k^{r-1} \leftarrow \widetilde{\mathsf{BC}_k^{r-1}} - M_k^{\mathsf{BC},r-1}$ and $\mathsf{msg}_{k\to i}^{r-1} \leftarrow \widetilde{\mathsf{msg}_{k\to i}^{r-1}} - M_{k\to i}^{\mathsf{P2P},r-1}$, obtaining $\left\{\left(\mathsf{BC}_k^{r-1}, \mathsf{msg}_{k\to i}^{r-1}\right)\right\}_{k\in[n]}$.
- Update the transcript $T_i^{<r} = T_i^{<r-1} \cup \left\{\left(\mathsf{BC}_j^{r-1}, \mathsf{msg}_{j\to i}^{r-1}\right)\right\}_{j\in[n]}$. Note that for completeness, we define $T_i^{<1} = \emptyset$.
- Broadcast the authenticated shares for $\mathcal{P}_j$ for all $j\in[n]$: $\left(m_{j,i}^{\mathsf{BC},r}, \left\{\tau_{j,i}^{\mathsf{BC},k,r}\right\}_{k\in[n]}\right)$
- Send to $\mathcal{P}_j$ over point-to-point channels, the authenticated shares $(m_{k\to j,i}^{\mathsf{P2P},r}, \tau_{k\to j,i}^{\mathsf{P2P},r})$ for all $k\in[n]$.
- Compute the Round r messages $(\mathsf{BC}_i^r, \{\mathsf{msg}_{i\to j}^r\}_{j\in[n]}) \leftarrow \mathsf{nxt\text{-}msg}_i^r(x_i,\rho_i,T_i^{<r})$.
- Send $\widetilde{\mathsf{BC}_i^r} = \mathsf{BC}_i^r + M_i^{\mathsf{BC},r}$ over broadcast channel (arithmetic in $\mathbb{F}$).

- Send $\widetilde{\mathsf{msg}^r_{i\to j}} = \mathsf{msg}^r_{i\to j} + M^{\mathsf{P2P},r}_{i\to j}$ over point-to-point channel to $\mathcal{P}_j$ for all $j \in [n]$.

– **Output:** Each party $\mathcal{P}_i$ does the following:
- If $\mathcal{P}_i$ is missing any share from round R, it outputs $\perp$.
- For every $k, \ell \in [n]$, run the verification algorithm to check $\mathsf{Vrfy}(\kappa^{\mathsf{BC},i,R}_{k,\ell}, m^{\mathsf{BC},R}_{k,\ell}, \tau^{\mathsf{BC},i,R}_{k,\ell})$ and $\mathsf{Vrfy}(\kappa^{\mathsf{P2P},R}_{k\to i,\ell}, m^{\mathsf{P2P},R}_{k\to i,\ell}, \tau^{\mathsf{P2P},R}_{k\to i,\ell})$. If any share fails the verification, $\mathcal{P}_i$ outputs $\perp$.
- Reconstruct the masks $\left\{\left(M^{\mathsf{BC},R}_k, M^{\mathsf{P2P},R}_{k\to i}\right)\right\}_{k\in[n]}$: $M^{\mathsf{BC},R}_k \leftarrow \mathsf{Rec}(m^{\mathsf{BC},R}_{k,1}, \ldots, m^{\mathsf{BC},R}_{k,n})$ and $M^{\mathsf{P2P},R}_{k\to i} \leftarrow \mathsf{Rec}(m^{\mathsf{P2P},R}_{k\to i,1}, \ldots, m^{\mathsf{P2P},R}_{k\to i,n})$.
- For every masked message $\widetilde{\mathsf{BC}^R_k}$ and $\widetilde{\mathsf{msg}^R_{k\to i}}$ received in the previous round, recover $\mathsf{BC}^R_k \leftarrow \widetilde{\mathsf{BC}^R_k} - M^{\mathsf{BC},R}_k$ and $\mathsf{msg}^R_{k\to i} \leftarrow \widetilde{\mathsf{msg}^R_{k\to i}} - M^{\mathsf{P2P},R}_{k\to i}$, obtaining $\left\{\left(\mathsf{BC}^R_k, \mathsf{msg}^R_{k\to i}\right)\right\}_{k\in[n]}$.
- Update the transcript $T_i^{<R+1} = T_i^{<R} \cup \left\{\left(\mathsf{BC}^R_j, \mathsf{msg}^R_{j\to i}\right)\right\}_{j\in[n]}$.
- Compute the output as $y_i \leftarrow \mathsf{output}_i(x_i, \rho_i, T_i^{<R+1})$..

Fig. 7: Round and communication-channel Preserving Compiler from Rushing to Super-Rushing Security

C.1 Extension to Unanimous Abort

As described earlier in Section 4.4, using digital signatures lets us upgrade the security to unanimous abort. With respect to the communication-pattern preserving compiler, this involves the following modification: When the shares are associated with masks for point-to-point messages, their signed versions are transmitted over the corresponding point-to-point channels (and over the broadcast channel for broadcast shares, respectively). However, in the first and last round, additional broadcast communication is incurred in order to guarantee unanimity: (a) In the first round, parties broadcast the verification keys of the digital signature scheme, as described above. (b) In the last round, if the underlying protocol involves point-to-point communication, the parties must additionally agree on the validity of the corresponding point-to-point additive shares. This is because the underlying protocol guarantees UA only when corrupted point-to-point messages are dropped, whereas in the compiled protocol the adversary may also interfere with the recovery of honest messages. Accordingly, we assume that the final round of the underlying UA protocol is over broadcast, without loss of generality.

D Security Proof for Communication-Pattern Preserving Selective Abort Compiler

Below, we present the proof of the communication-pattern preserving compiler with selective abort. Since this compiler is more general than the one in Section 4, the proof also establishes Theorem 2.

Theorem 4. *Let Π be any R-round synchronous statistically (resp. computationally) secure n-party protocol computing a functionality f over broadcast and point-to-point channels with selective abort, tolerating at most t static corruptions. Then, the compiled protocol $\Pi^{cp,\mathsf{sr}}_{\mathsf{SA}}$ (Figure 7) is statistical (resp. computational) secure with selective abort against super-rushing adversaries tolerating at most t static corruptions. That is, for every unbounded (resp. PPT) super-rushing adversary $\mathcal{A}_{\mathsf{sr}}$ there exists an expected polynomial-time simulator $\mathcal{S}_{\mathsf{sr}}$ such that for all $\mathcal{I} \subseteq [n]$ with $|\mathcal{I}| \leq t$, the following two distributions are statistical (resp. computationally) indistinguishable:*

$$\left\{\textsc{Real}_{\Pi^{cp,\mathsf{sr}}_{\mathsf{SA}},\mathcal{I},\mathcal{A}_{\mathsf{sr}}(z)}(\mathbf{x},\lambda)\right\}_{\mathbf{x}\in(\{0,1\}^*)^n,\,\lambda\in\mathbb{N}}, \left\{\textsc{Ideal}^{\mathsf{SA}}_{f,\mathcal{I},\mathcal{S}_{\mathsf{sr}}(z)}(\mathbf{x},\lambda)\right\}_{\mathbf{x}\in(\{0,1\}^*)^n,\,\lambda\in\mathbb{N}}$$

where $\textsc{Ideal}^{\mathsf{SA}}$ denotes the ideal execution with the selective-abort.

Remark 2. Although Theorem 2 is stated for an underlying protocol with selective-abort security, it applies more broadly: since unanimous abort, identifiable abort, fairness, and guaranteed output delivery all imply selective abort, the compiler upgrades any of these stronger guarantees to super-rushing security (with selective abort).

Proof. Let $\mathcal{A}_{\mathsf{sr}}$ be the super-rushing adversary that violates security of $\Pi_{\mathsf{SA}}^{cp,\mathsf{sr}}$. To prove security, we describe a simulator $\mathcal{S}_{\mathsf{sr}}$ for such an adversary $\mathcal{A}_{\mathsf{sr}}$. Before doing that, we define a machine $\mathcal{M}$ as described in Figure 8. $\mathcal{M}$ internally runs the adversary $\mathcal{A}_{\mathsf{sr}}$. It also exposes an external interface, which we call the *left session*, through which all honest-parties' messages of Π are exchanged, regardless of whether the left session is connected to a real execution or a simulator. In a nutshell, $\mathcal{M}$ acts as a rushing adversary for Π and simulates the execution of the compiled protocol $\Pi_{\mathsf{SA}}^{cp,\mathsf{sr}}$ internally for $\mathcal{A}_{\mathsf{sr}}$.

By the assumption that Π is secure against rushing adversaries, for every PPT rushing adversary $\mathcal{A}_{\mathsf{r}}$ for $|\mathcal{I}| \leq t$, there exists a rushing simulator $\mathcal{S}_{\mathsf{r}}$ for Π such that:

$$\textsc{Real}_{\Pi,\mathcal{I},\mathcal{A}_{\mathsf{r}}(z)}(\mathbf{x},\lambda) \approx \textsc{Ideal}_{f,\mathcal{I},\mathcal{S}_{\mathsf{r}}(z)}(\mathbf{x},\lambda).$$

Hence, there exists a simulator $\mathcal{S}_{\mathsf{r}}$ for the adversary $\mathcal{M}(\mathcal{A}_{\mathsf{sr}},\cdot)$. Our simulator $\mathcal{S}_{\mathsf{sr}}$ internally runs $\mathcal{S}_{\mathsf{r}}$ and works as described in Figure 9.

Machine $\mathcal{M}(\mathcal{A}_{\mathsf{sr}},\rho_{\mathsf{sr}})$

1. For every $i \in [n] \setminus \mathcal{I}$, initialize the virtual honest party $\bar{\mathcal{P}}_i$ with arbitrary input and randomness.
2. Run $\mathcal{A}_{\mathsf{sr}}$ with randomness ρ_{sr}.
3. Initialize abort set $\mathcal{J} \leftarrow \emptyset$, which will track which honest parties abort during the execution of $\Pi_{\mathsf{SA}}^{cp,\mathsf{sr}}$.

Left-session routing: Messages received from the left session are routed according to their type: broadcast messages BC_i^r are sent over the broadcast channel, and point-to-point messages $\mathsf{msg}_{i\to j}^r$ are sent over the point-to-point channel from $\bar{\mathcal{P}}_i$ to the designated recipient $\mathcal{P}_j$.

- **For Round 1 messages:** Each party $\bar{\mathcal{P}}_i \in [n] \setminus \mathcal{I}$ does the following:
  - Run $\mathsf{Setup}(1^\lambda)$ to generate a set of verification keys $\left\{\left(\kappa_{i,k}^{\mathsf{BC},j,t}, \kappa_{i\to j,k}^{\mathsf{P2P},t}\right)\right\}_{j\in\mathcal{I},k\in[n],t\in\{2,\ldots,R\}}$, and send the verification keys $\left\{\left(\kappa_{i,k}^{\mathsf{BC},j,t}, \kappa_{i\to j,k}^{\mathsf{P2P},t}\right)\right\}_{k\in[n],t\in\{2,\ldots,R\}}$ to party $\mathcal{P}_j$ through point-to-point channel for all $\mathcal{P}_j \in \mathcal{I}$.
  - Sample broadcast shares $\{m_{i,k}^{\mathsf{BC},t}\}_{k\in\mathcal{I}}$ uniformly from $\mathbb{F}$ for each $t \in \{2,\ldots,R\}$. Compute $\tau_{i,k}^{\mathsf{BC},j,t} \leftarrow \mathsf{Auth}(\kappa_{i,k}^{\mathsf{BC},j,t}, m_{i,k}^{\mathsf{BC},t})$ for all $j \in \mathcal{I}$. Send $\left(m_{i,k}^{\mathsf{BC},t}, \left\{\tau_{i,k}^{\mathsf{BC},j,t}\right\}_{j\in\mathcal{I}}\right)$ over point-to-point channel to $\mathcal{P}_k$.
  - Sample point-to-point shares $\{m_{i\to j,k}^{\mathsf{P2P},t}\}_{j\in[n],k\in\mathcal{I}}$ uniformly from $\mathbb{F}$ for each $t \in \{2,\ldots,R\}$. Compute $\tau_{i\to j,k}^{\mathsf{P2P},t} \leftarrow \mathsf{Auth}(\kappa_{i\to j,k}^{\mathsf{P2P},t}, m_{i\to j,k}^{\mathsf{P2P},t})$ for all $j \in [n]$. Send $\left\{(m_{i\to j,k}^{\mathsf{P2P},t}, \tau_{i\to j,k}^{\mathsf{P2P},t})\right\}_{j\in[n]}$ over point-to-point channel to $\mathcal{P}_k$ for all $k \in \mathcal{I}$.
  - Interact with the left session and receive $\left(\mathsf{BC}_i^1, \{\mathsf{msg}_{i\to j}^1\}_{j\in\mathcal{I}}\right)$
  - Send BC_i^1 over broadcast channel.
  - Send $\mathsf{msg}_{i\to j}^1$ over point-to-point channel to $\mathcal{P}_j$ for all $j \in \mathcal{I}$.
- **For Round r messages ($2 \leq r \leq R$):**
  - $\mathcal{M}$ maintains, for each round $r \in \{2,\ldots,R\}$, a peek set $\mathcal{L}^r \subseteq [n] \setminus \mathcal{I}$ (initially $\mathcal{L}^r = \emptyset$), which records the indices of the honest parties that the adversary requests for its round r messages. When $\mathcal{A}_{\mathsf{sr}}$ requests the round-r messages from an honest $\mathcal{P}_i$ ($i \in [n]\setminus\mathcal{I}$) to the corrupted parties, call such a request $\mathsf{Peek}(\mathcal{P}_i, r)$. $\mathcal{M}$ checks:
    1. $i \notin \mathcal{L}^r$ (i.e., the adversary has not made a peek request for $\mathcal{P}_i$'s round-r messages before), and

2. all round-$(r-1)$ messages from every $\mathcal{P}_j \in \mathcal{I}$ have been delivered to $\bar{\mathcal{P}}_i$.

If any condition fails, call this event $\mathsf{NotSuperRushing}$ and $\mathcal{M}$ aborts.[a] Otherwise, $\mathcal{M}$ processes the request as described below.

- For each $\bar{\mathcal{P}}_i \in [n] \setminus \mathcal{I}$, if $\bar{\mathcal{P}}_i$ is missing any share for round $r-1$ from any corrupted party in $\mathcal{I}$, add i to $\mathcal{J}$ ($\mathcal{J} \leftarrow \mathcal{J} \cup \{i\}$).
- If $r \neq 2$, each party $\bar{\mathcal{P}}_i$ does the following, otherwise skip to the next step:
  - For each $k \in \mathcal{I}$ and $j \in \mathcal{I}$, let $(\hat{m}^{\mathsf{BC},r-1}_{k,j}, \hat{\tau}^{\mathsf{BC},i,r-1}_{k,j})$ and $(\hat{m}^{\mathsf{P2P},r-1}_{k\to i,j}, \hat{\tau}^{\mathsf{P2P},r-1}_{k\to i,j})$ denote the shares received from $\mathcal{P}_j$. Run $\mathsf{Vrfy}(\kappa^{\mathsf{BC},i,r-1}_{k,j}, \hat{m}^{\mathsf{BC},r-1}_{k,j}, \hat{\tau}^{\mathsf{BC},i,r-1}_{k,j})$ and $\mathsf{Vrfy}(\kappa^{\mathsf{P2P},r-1}_{k\to i,j}, \hat{m}^{\mathsf{P2P},r-1}_{k\to i,j}, \hat{\tau}^{\mathsf{P2P},r-1}_{k\to i,j})$. If any check fails, add i to $\mathcal{J}$ ($\mathcal{J} \leftarrow \mathcal{J} \cup \{i\}$).
  - **(Tamper check)** For each $\ell \in [n] \setminus \mathcal{I}$ and $j \in \mathcal{I}$, check that $\hat{m}^{\mathsf{BC},r-1}_{\ell,j} = m^{\mathsf{BC},r-1}_{\ell,j}$ and $\hat{m}^{\mathsf{P2P},r-1}_{\ell\to i,j} = m^{\mathsf{P2P},r-1}_{\ell\to i,j}$, where $m^{\mathsf{BC},r-1}_{\ell,j}$ and $m^{\mathsf{P2P},r-1}_{\ell\to i,j}$ are the shares originally distributed by $\bar{\mathcal{P}}_\ell$ in round 1. If any check fails, add i to $\mathcal{J}$ ($\mathcal{J} \leftarrow \mathcal{J} \cup \{i\}$).
- For each $i \in [n] \setminus \mathcal{I}$ and $j \in \mathcal{I}$, broadcast the authenticated shares $\left(m^{\mathsf{BC},r}_{j,i}, \left\{\tau^{\mathsf{BC},r}_{j\to k,i}\right\}_{k\in[n]}\right)$.
- For each $i \in [n] \setminus \mathcal{I}$ and $j \in \mathcal{I}$, send to $\mathcal{P}_j$ through point-to-point channel the authenticated shares of masks from the corrupted parties: $\left\{(m^{\mathsf{P2P},r}_{j^*\to j,i}, \tau^{\mathsf{P2P},r}_{j^*\to j,i})\right\}_{j^*\in[n]\setminus\mathcal{I}}$.
- For each honest party $\bar{\mathcal{P}}_i$, upon a peek request $\mathsf{Peek}(\mathcal{P}_i, r)$ with $|\mathcal{L}^r| < |[n] \setminus \mathcal{I}| - 1$, $\bar{\mathcal{P}}_i$ does the following:
  - For each $\ell \in [n] \setminus \mathcal{I}$, sample random shares $m^{\mathsf{BC},r}_{\ell,i}$ and compute the corresponding MACs $\tau^{\mathsf{BC},k,r}_{\ell,i} \leftarrow \mathsf{Auth}(\kappa^{\mathsf{BC},k,r}_{\ell,i}, m^{\mathsf{BC},r}_{\ell,i})$ for all $k \in [n]$. Broadcast the authenticated shares $\left(m^{\mathsf{BC},r}_{\ell,i}, \left\{\tau^{\mathsf{BC},k,r}_{\ell,i}\right\}_{k\in[n]}\right)$ for all $\ell \in [n] \setminus \mathcal{I}$.
  - For each $\ell \in [n] \setminus \mathcal{I}$ and $j \in \mathcal{I}$, sample random shares $m^{\mathsf{P2P},r}_{\ell\to j,i}$ and compute the corresponding MACs $\tau^{\mathsf{P2P},r}_{\ell\to j,i} \leftarrow \mathsf{Auth}(\kappa^{\mathsf{P2P},r}_{\ell\to j,i}, m^{\mathsf{P2P},r}_{\ell\to j,i})$ for all $\ell \in [n] \setminus \mathcal{I}$ and $j \in \mathcal{I}$. Send $\left\{(m^{\mathsf{P2P},r}_{\ell\to j,i}, \tau^{\mathsf{P2P},r}_{\ell\to j,i})\right\}_{\ell\in[n]\setminus\mathcal{I}}$ over point-to-point channel to $\mathcal{P}_j$ for all $j \in \mathcal{I}$.
  - Sample $\widetilde{\mathsf{BC}}^r_i \leftarrow \mathbb{F}$ uniformly and send $\widetilde{\mathsf{BC}}^r_i$ over broadcast channel.
  - Sample $\widetilde{\mathsf{msg}}^r_{i\to j} \leftarrow \mathbb{F}$ uniformly for each $j \in \mathcal{I}$ and send $\widetilde{\mathsf{msg}}^r_{i\to j}$ over point-to-point to $\mathcal{P}_j$ for all $j \in \mathcal{I}$.
  - Add i to the list $\mathcal{L}^r$.
- Upon a peek request $\mathsf{Peek}(\mathcal{P}_i, r)$ with $|\mathcal{L}^r| = |[n] \setminus \mathcal{I}| - 1$, or if the protocol has completed round $r-1$ and $|\mathcal{L}^r| < |[n] \setminus \mathcal{I}| - 1$, $\mathcal{M}$ does the following:
  - If $r \neq 2$ do the following, otherwise skip to the next step:
    - Reconstruct the masks $\left\{\left(M^{\mathsf{BC},r-1}_j, M^{\mathsf{P2P},r-1}_{j\to\ell}\right)\right\}_{j\in\mathcal{I},\ell\in[n]\setminus\mathcal{I}}$: $M^{\mathsf{BC},r-1}_j \leftarrow \mathsf{Rec}(m^{\mathsf{BC},r-1}_{j,1}, \ldots, m^{\mathsf{BC},r-1}_{j,n})$ and $M^{\mathsf{P2P},r-1}_{j\to\ell} \leftarrow \mathsf{Rec}(m^{\mathsf{P2P},r-1}_{j\to\ell,1}, \ldots, m^{\mathsf{P2P},r-1}_{j\to\ell,n})$.
    - For every masked message $\widetilde{\mathsf{BC}^{r-1}_j}$ and $\widetilde{\mathsf{msg}^{r-1}_{j\to\ell}}$ ($j \in \mathcal{I}, \ell \in [n] \setminus \mathcal{I}$) received from the adversary, recover $\mathsf{BC}^{r-1}_j \leftarrow \widetilde{\mathsf{BC}^{r-1}_j} - M^{\mathsf{BC},r-1}_j$ and $\mathsf{msg}^{r-1}_{j\to\ell} \leftarrow \widetilde{\mathsf{msg}^{r-1}_{j\to\ell}} - M^{\mathsf{P2P},r-1}_{j\to\ell}$, obtaining $\left\{\left(\mathsf{BC}^{r-1}_j, \mathsf{msg}^{r-1}_{j\to\ell}\right)\right\}_{j\in\mathcal{I},\ell\in[n]\setminus\mathcal{I}}$.

* Forward the broadcast messages $\left\{\mathsf{BC}_j^{r-1}\right\}_{j\in\mathcal{I}}$ and point-to-point messages $\left\{\mathsf{msg}_{j\to i}^{r-1}\right\}_{j\in\mathcal{I},i\in[n]\setminus\mathcal{I}}$ from the adversary $\mathcal{A}_{\mathsf{sr}}$ to the left session. It receives back from the left session the messages $\left\{\mathsf{BC}_i^r, \{\mathsf{msg}_{i\to j}^r\}_{j\in\mathcal{I}}\right\}_{i\in[n]\setminus\mathcal{I}}$. There can be two possible cases:
  - if $|\mathcal{L}^r| = |[n]\setminus\mathcal{I}| - 1$, then there is only one honest party $\bar{\mathcal{P}}_i$ for whom the adversary has not made the peek request for its Round r messages. We proceed to correct its shares as described below.
  - if $|\mathcal{L}^r| < |[n]\setminus\mathcal{I}| - 1$, this is the case where the adversary has not issued peek requests for all honest parties, but the protocol has completed round $r-1$. In this case, $\mathcal{M}$ designates an arbitrary honest party for whom the adversary has not yet issued a peek request as $\bar{\mathcal{P}}_i$ (the party whose shares would be corrected). For each of the remaining honest parties for whom the peek request has not been issued i.e. $\bar{\mathcal{P}}_k$ ($k \notin \{\mathcal{L}^r \cup \mathcal{I}\}$ and $k \neq i$): sample $m_{\ell,k}^{\mathsf{BC},r}$ uniformly at random for each $\ell \in [n]\setminus\mathcal{I}$, compute the corresponding MACs $\tau_{\ell,k}^{\mathsf{BC},j,r} \leftarrow \mathsf{Auth}(\kappa_{\ell,k}^{\mathsf{BC},j,r}, m_{\ell,k}^{\mathsf{BC},r})$ for all $j \in [n]$, broadcast the authenticated shares $\left(m_{\ell,k}^{\mathsf{BC},r}, \left\{\tau_{\ell,k}^{\mathsf{BC},j,r}\right\}_{j\in[n]}\right)$ for all $\ell \in [n]\setminus\mathcal{I}$, and send the real masked message $\widetilde{\mathsf{BC}}_k^r = \mathsf{BC}_k^r + M_k^{\mathsf{BC},r}$ over broadcast (and analogously for point-to-point shares and masked messages). Lastly, $\mathcal{M}$ corrects $\bar{\mathcal{P}}_i$'s shares as follows.

  The **correction** for $\bar{\mathcal{P}}_i$'s shares:
  - Sample $\widetilde{\mathsf{BC}}_i^r \leftarrow \mathbb{F}$ and $\widetilde{\mathsf{msg}}_{i\to j}^r \leftarrow \mathbb{F}$ uniformly for each $j \in \mathcal{I}$.
  - Compute the corrected share for each $\ell \in [n]\setminus\mathcal{I}$ and $j \in \mathcal{I}$:

$$m_{\ell,i}^{\mathsf{BC},r} = \widetilde{\mathsf{BC}}_\ell^r - \mathsf{BC}_\ell^r - \sum_{i^*\neq i} m_{\ell,i^*}^{\mathsf{BC},r},$$

$$m_{\ell\to j,i}^{\mathsf{P2P},r} = \widetilde{\mathsf{msg}}_{\ell\to j}^r - \mathsf{msg}_{\ell\to j}^r - \sum_{i^*\neq i} m_{\ell\to j,i^*}^{\mathsf{P2P},r}$$

* For each $\ell \in [n]\setminus\mathcal{I}$, $\bar{\mathcal{P}}_i$ computes the MAC for the corrected broadcast shares $\tau_{\ell,i}^{\mathsf{BC},k,r} \leftarrow \mathsf{Auth}(\kappa_{\ell,i}^{\mathsf{BC},k,r}, m_{\ell,i}^{\mathsf{BC},r})$ for each $k \in [n]$. Broadcast the authenticated shares $\left(m_{\ell,i}^{\mathsf{BC},r}, \left\{\tau_{\ell,i}^{\mathsf{BC},k,r}\right\}_{k\in[n]}\right)$ for all $\ell \in [n]\setminus\mathcal{I}$.
* For each $\ell \in [n]\setminus\mathcal{I}$ and $j \in \mathcal{I}$, $\bar{\mathcal{P}}_i$ computes the MAC for the corrected point-to-point shares $\tau_{\ell\to j,i}^{\mathsf{P2P},r} \leftarrow \mathsf{Auth}(\kappa_{\ell\to j,i}^{\mathsf{P2P},r}, m_{\ell\to j,i}^{\mathsf{P2P},r})$ and send $\left\{(m_{\ell\to j,i}^{\mathsf{P2P},r}, \tau_{\ell\to j,i}^{\mathsf{P2P},r})\right\}_{\ell\in[n]\setminus\mathcal{I}}$ over point-to-point channel to $\mathcal{P}_j$ for all $j \in \mathcal{I}$.
* $\bar{\mathcal{P}}_i$ sends $\widetilde{\mathsf{BC}}_i^r$ over broadcast channel.
* $\bar{\mathcal{P}}_i$ sends $\widetilde{\mathsf{msg}}_{i\to j}^r$ over point-to-point to $\mathcal{P}_j$ for all $j \in \mathcal{I}$.

– Once the simulator's execution stops, it outputs whatever $\mathcal{A}_{\mathsf{sr}}$ outputs and halts.

[a]Note that since $\mathcal{M}$ interacts with a valid super-rushing adversary $\mathcal{A}_{\mathsf{sr}}$, NotSuperRushing never happens. It is included here just for completeness.

Fig. 8: Machine $\mathcal{M}$ which emulates the adversary for Π

Simulator $\mathcal{S}_{\mathsf{sr}}$

- Sample randomness ρ_{sr} and run $\mathcal{S}_{\mathsf{r}}$ for the adversary $\mathcal{M}(\mathcal{A}_{\mathsf{sr}}, \rho_{\mathsf{sr}})$.
- For the query $\{x_j\}_{j\in\mathcal{I}}$ to the ideal functionality issued by $\mathcal{S}_{\mathsf{r}}$, forward it and return the response to $\mathcal{S}_{\mathsf{r}}$.
- Retrieve the abort set $\mathcal{J}$ maintained by $\mathcal{M}$ (see Figure 8). Send $(\mathsf{abort}, \mathcal{J})$ to the ideal functionality.
- Output whatever $\mathcal{S}_{\mathsf{r}}$ outputs and halts.

Fig. 9: Simulator $\mathcal{S}_{\mathsf{sr}}$ for $\Pi^{cp,\mathsf{sr}}_{\mathsf{SA}}$

Now, we describe a sequence of hybrids to complete our proof.

Hybrid H_0: This is the real execution of $\Pi^{cp,\mathsf{sr}}_{\mathsf{SA}}$ with the super-rushing adversary $\mathcal{A}_{\mathsf{sr}}$.
Hybrid H_1: Let $\widetilde{\mathcal{M}}$ be the machine that behaves identically to $\mathcal{M}$ (Figure 8), except that the **(tamper check)** step is replaced by running the MAC verification algorithm Vrfy on all shares, as in the real protocol. H_1 is the same as H_0 except the adversary $\mathcal{A}_{\mathsf{sr}}$ is wrapped by $\widetilde{\mathcal{M}}$ (with random coins ρ_{sr}), which interacts with the real execution of Π via the left session.
Hybrid H_2: Same as H_1, but now $\mathcal{M}$ runs as fully described in Figure 8, i.e. it additionally performs the **tamper check** step.
Hybrid H_3: Same as H_2 except the left session is replaced by the simulator $\mathcal{S}_{\mathsf{r}}$. Note that H_3 and $\mathcal{S}_{\mathsf{sr}}$ are equal.

Lemma 7. *Given an n-out-of-n additive secret sharing scheme, the output distributions of H_0 and H_1 are identical.*

Proof. We show that the joint view of $\mathcal{A}_{\mathsf{sr}}$ is identically distributed in H_0 and H_1.

Round 1. MAC keys are sampled via $\mathsf{Setup}(1^\lambda)$ in both experiments. Shares sent to corrupted parties are sampled uniformly from $\mathbb{F}$, independent of honest parties' inputs. Hence the view of $\mathcal{A}_{\mathsf{sr}}$ in round 1 is identically distributed.

Non-last peek ($|\mathcal{L}^r| < |[n] \setminus \mathcal{I}| - 1$). $\mathcal{A}_{\mathsf{sr}}$ holds exactly the $|\mathcal{I}| < n$ shares sent to corrupted parties, a strict subset $T \subsetneq [n]$. By perfect privacy of $(\mathsf{Share}, \mathsf{Rec})$ (Definition 11), the shares $\{m^{\mathsf{BC},r}_{\ell,k}\}_{k\in T}$ are uniform over $\mathbb{F}^{|T|}$ and independent of $M^{\mathsf{BC},r}_\ell$. Since $M^{\mathsf{BC},r}_\ell$ is uniform, the masked message $\widetilde{\mathsf{BC}}^r_\ell = \mathsf{BC}^r_\ell + M^{\mathsf{BC},r}_\ell$ is uniform over $\mathbb{F}$ from $\mathcal{A}_{\mathsf{sr}}$'s view, matching the distribution in H_1 where $\mathcal{M}$ samples $\widetilde{\mathsf{BC}}^r_\ell \leftarrow \mathbb{F}$ directly. An identical argument applies to $\widetilde{\mathsf{msg}}^r_{\ell\to j}$.

Last peek ($|\mathcal{L}^r| = |[n] \setminus \mathcal{I}| - 1$). $\mathcal{M}$ correctly reconstructs $M^{\mathsf{BC},r-1}_j$ for each $j \in \mathcal{I}$ and recovers the plaintext messages $\{\mathsf{BC}^{r-1}_j, \mathsf{msg}^{r-1}_{j\to\ell}\}_{j\in\mathcal{I}}$. These are forwarded to the left session as in H_0, so the response $(\mathsf{BC}^r_i, \{\mathsf{msg}^r_{i\to j}\}_{j\in\mathcal{I}})$ is identically distributed. The correction share

$$m^{\mathsf{BC},r}_{\ell,i} = \widetilde{\mathsf{BC}}^r_\ell - \mathsf{BC}^r_\ell - \sum_{i^*\neq i} m^{\mathsf{BC},r}_{\ell,i^*}, \quad \widetilde{\mathsf{BC}}^r_\ell \leftarrow \mathbb{F},$$

ensures Rec yields the correct mask $M^{\mathsf{BC},r}_\ell$, so the masked message $\widetilde{\mathsf{BC}}^r_\ell$ seen by $\mathcal{A}_{\mathsf{sr}}$ matches H_0 exactly. The same holds for $m^{\mathsf{P2P},r}_{\ell\to j,i}$.

Combining over all rounds, the joint view of $\mathcal{A}_{\mathsf{sr}}$ is identically distributed in H_0 and H_1.

Lemma 8. *Given a one-time $\mathsf{EUF\text{-}1\text{-}CMA}$-secure MAC scheme, the output distributions of H_1 and H_2 are statistically indistinguishable.*

Proof. H_1 and H_2 are identical except in how $\mathcal{M}$ verifies shares attributed to honest parties: in H_1, $\mathcal{M}$ runs $\mathsf{Vrfy}(\kappa, \hat{m}, \hat{\tau})$; in H_2, $\mathcal{M}$ checks $\hat{m} = m$ directly. These two checks agree on every input except when a corrupted party submits a forged share: a value $\hat{m} \neq m$ with a valid tag satisfying $\mathsf{Vrfy}(\kappa, \hat{m}, \hat{\tau}) = 1$. In H_1 such a forgery passes Vrfy and is accepted; in H_2 the equality check detects it and $\mathcal{M}$ aborts.

Suppose by contradiction that the above does not happen, we reduce this event to breaking $\mathsf{EUF\text{-}1\text{-}CMA}$ security. Fix any tuple (ℓ, i, k, t) indexing a particular key $\kappa^{\mathsf{BC},t}_{\ell\to i,k}$. The key is generated by honest $\mathcal{P}_\ell$ and sent point-to-point to honest $\mathcal{P}_i$ only; the corrupted $\mathcal{P}_k$ (who forwards the share) never receives it. We build an adversary $\mathcal{A}_{\mathsf{MAC}}$ to the $\mathsf{EUF\text{-}1\text{-}CMA}$ security game: $\mathcal{A}_{\mathsf{MAC}}$ uses the challenge key κ in place of $\kappa^{\mathsf{BC},t}_{\ell\to i,k}$ and samples all remaining keys honestly. From the protocol execution, $\mathcal{A}_{\mathsf{MAC}}$ queries the MAC oracle once to obtain $\tau \leftarrow \mathsf{Auth}(\kappa, m)$, in this case $\left(m^{\mathsf{BC},t}_{\ell,k},\, \tau^{\mathsf{BC},i,t}_{\ell,k}\right)$. If $\mathcal{A}_{\mathsf{sr}}$ subsequently causes $\mathcal{P}_i$ to receive $(\hat{m}, \hat{\tau})$ with $\hat{m} \neq m$ and $\mathsf{Vrfy}(\kappa, \hat{m}, \hat{\tau}) = 1$, then $\mathcal{A}_{\mathsf{MAC}}$ outputs $(\hat{m}, \hat{\tau})$ as a valid forgery in the $\mathsf{EUF\text{-}1\text{-}CMA}$ game. Since each key authenticates exactly one share, $\mathcal{A}_{\mathsf{MAC}}$ makes exactly one query. By $\mathsf{EUF\text{-}1\text{-}CMA}$ security, the forgery probability is $\mathrm{negl}(\lambda)$ per tuple; a union bound over $\mathsf{poly}(\lambda)$ tuples gives negligible total probability. Hence H_1 and H_2 are statistically indistinguishable.

Lemma 9. *Let Π be a statistically (resp. computationally) secure rushing protocol computing f. Then the output distributions of H_2 and H_3 are statistically (resp. computationally) indistinguishable.*

Proof. Suppose there exists an unbounded (resp. PPT) distinguisher $\mathcal{A}$ that can distinguish H_2 and H_3 with non-negligible probability. We construct a rushing adversary $\mathcal{A}_{\mathsf{r}} := \mathcal{M}(\mathcal{A}, \cdot)$ that breaks the statistical (resp. computational) security of Π.

$\mathcal{M}(\mathcal{A}, \cdot)$ is a valid rushing adversary for Π. By construction (Figure 8), $\mathcal{M}$ waits to receive all the round-r messages from the left session before delivering any round r adversarial messages to honest parties. However, $\mathcal{M}$ receives the honest round-$(r+1)$ messages *only after* all the round r adversarial messages have been sent. This precisely matches the rushing model: $\mathcal{M}$ sees all honest round-r messages of Π before sending its r-round outgoing messages, but it receives the honest round $(r+1)$ messages only after sending its r-round outgoing messages. Hence $\mathcal{M}(\mathcal{A}, \cdot)$ is a valid rushing adversary for Π.

By the statistical (resp. computational) security of Π against rushing adversaries, for the rushing adversary $\mathcal{M}(\mathcal{A}, \cdot)$ there exists a simulator $\mathcal{S}_{\mathsf{r}}$ such that the following distributions are statistically (resp. computationally) indistinguishable:

$$\textsc{Real}_{\Pi^{cp,\mathsf{sr}}_{\mathsf{SA}},\mathcal{I},\mathcal{M}(\mathcal{A},\cdot)(z)}(\mathbf{x}, \lambda)\ ,\ \textsc{Ideal}_{f,\mathcal{I},\mathcal{S}_{\mathsf{r}}(z)}(\mathbf{x}, \lambda).$$

The only difference between H_2 and H_3 is that the left session, which in H_2 is connected to the real execution of Π, is replaced by $\mathcal{S}_{\mathsf{r}}$ in H_3. Hence any distinguisher $\mathcal{A}$ that separates H_2 from H_3 with non-negligible probability would contradict the statistical (resp. computational) security of Π, completing the proof.

Combining Lemmas 7, 8 and 9, we conclude that H_0 and H_3 are statistically (resp. computationally) indistinguishable. Since H_0 is the real execution of $\Pi^{cp,\mathsf{sr}}_{\mathsf{SA}}$ and H_3 is the ideal execution of f with simulator $\mathcal{S}_{\mathsf{sr}}$, we have shown that $\Pi^{cp,\mathsf{sr}}_{\mathsf{SA}}$ securely computes f against super-rushing adversaries with selective abort.