Round Efficient Secure Multiparty Quantum Computation with Identifiable Abort

Bar Alon[1⋆], Hao Chung[2⋆⋆], Kai-Min Chung[3⋆⋆⋆], Mi-Ying Huang[3,4†], Yi Lee[3‡], and Yu-Ching Shen[3⋆⋆]

[1] Department of Computer Science, Ariel University, Israel
[2] Department of Electrical and Computer Engineering, Carnegie Mellon University, USA
[3] Institute of Information Science, Academia Sinica, Taiwan
[4] Department of Computer Science and Information Engineering, National Taiwan University, Taiwan

Abstract. A recent result by Dulek et al. (EUROCRYPT 2020) showed a secure protocol for computing any quantum circuit even without the presence of an honest majority. Their protocol, however, is susceptible to a "denial of service" attack and allows even a single corrupted party to force an abort. We propose the first quantum protocol that admits *security-with-identifiable-abort*, which allows the honest parties to agree on the identity of a corrupted party in case of an abort. Additionally, our protocol is the first to have the property that the number of rounds where quantum communication is required is *independent of the circuit complexity*. Furthermore, if there exists a post-quantum secure classical protocol whose round complexity is independent of the circuit complexity, then our protocol has this property as well. Our protocol is secure under the assumption that classical quantum-resistant fully homomorphic encryption schemes with decryption circuit of logarithmic depth exist. Interestingly, our construction also admits a reduction from quantum fair secure computation to classical fair secure computation.

⋆ E-mail: alonbar08@gmail.com. This work was supported by ISF grant 152/17 and by the Ariel Cyber Innovation Center in conjunction with the Israel National Cyber directorate in the Prime Minister's Office. Part of the work was done while visiting Academia Sinica.

⋆⋆ E-mail: haochung@andrew.cmu.edu/yuching@iis.sinica.edu.tw. This research is partially supported by the Young Scholar Fellowship (Einstein Program) of the Ministry of Science and Technology (MOST) in Taiwan, under grant number MOST 108-2636-E-002-014 and Executive Yuan Data Safety and Talent Cultivation Project (ASKPQ-109-DSTCP).

⋆⋆⋆ E-mail: kmchung@iis.sinica.edu.tw. This research is partially supported by the Air Force Office of Scientific Research under award number FA2386-20-1-4066, and MOST, Taiwan, under Grant no. MOST 109-2223-E-001-001-MY3.

† E-mail: miyinghuangtw@gmail.com. This work is supported by the Young Scholar Fellowship (Einstein Program) of the Ministry of Science and Technology (MOST) in Taiwan, under grant number MOST 109-2636-E-002-025.

‡ E-mail: ethanlee515@gmail.com. This work was done in part while the author was affiliated to National Taiwan University.

1 Introduction

In the setting of secure multiparty computation (MPC), the goal is to allow a set of mutually distrustful parties to compute some function of their private inputs in a way that preserves some security properties, even in the face of adversarial behavior by some of the parties. Some of the desired properties of a secure protocol include correctness (cheating parties can only affect the output by choosing their inputs), privacy (nothing but the specified output is learned), fairness (all parties receive an output or none do), and even guaranteed output delivery (meaning that all honestly behaving parties always learn an output). Informally speaking, a protocol π computes a functionality f with full-security if it provides all of the above security properties.

It is well-known that, assuming an honest majority and a broadcast channel, any functionality can be computed with full-security [RBO89]. However, achieving fairness, and hence full-security, is impossible in general assuming no honest majority [Cle86]. Instead, one usually settles on a weaker notion called *security-with-abort*, which completely disregards fairness. Roughly, security-with-abort guarantees that either the protocol terminates successfully, in which case the honest parties receive their outputs, or the protocol aborts, in which case all honest parties learn that there was an attack. Note that since fairness is not guaranteed, it might be the case where the adversary learns the output of the corrupted parties. In many setting, however, security-with-abort is not enough, as an adversary can cause a *denial-of-service* attack by repeatedly aborting the protocol. Thus, it is highly desirable to consider the stronger security notion called *security-with-identifiable-abort* (SWIA) [IOZ14]. Here, if the protocol is aborted, then all honest parties additionally agree on an identity of a corrupted party. It is well-known that there are protocols admitting SWIA for any number of corrupted parties, e.g., the GMW protocol [GMW87].

In this work we consider the quantum version of MPC. In the *fully quantum* setting, the functionality – including the inputs and outputs – is quantum. As such, the parties, as well as the adversary attacking the protocol, are quantum. Secure multiparty quantum computation (MPQC) in the fully quantum setting, was first studied by [CGS02], who constructed a fully secure n-party protocol tolerating strictly less than $n/6$. The threshold $n/6$ was subsequently improved the more general honest majority setting [BOCG+06], assuming the availability of a classical broadcast channel. Similarly to the classical setting, if there is no honest majority, then full-security is impossible to achieve in general [ABDR04, Kit].[5] Moreover, [DNS12] presented a secure-with-abort protocol in the two-party case, and recently [DGJ+20] extended it to the multiparty case, tolerating any number of corrupted parties.

The protocol of [DGJ+20], however, does not admit *identifiable abort*. This follows from the fact that it is impossible to broadcast a quantum state. Therefore a corrupted party can accuse an honest party of not sending it a message, thus,

[5] The impossibility proof is in the information theoretic setting, where the adversary is unbounded. However, even though Cleve's impossibility result is stated for classical protocols, the proof can still be applied for quantum protocols.

not only is the quantum state lost, but the other parties cannot identify the corrupted party. When compared to the classical setting, this raises the following natural question.

> *Can any multiparty quantum circuit be computed with security-with-identifiable-abort, tolerating any number of corrupted parties?*

1.1 Our Results

In this paper, we answer the above question affirmatively. Additionally, our protocol is the first to have the property that the number of rounds where quantum communication is required is independent of the circuit complexity. Furthermore, if there exists a post-quantum secure classical protocol whose round complexity is independent of the circuit complexity, then our protocol has this property as well.

Similarly to [DGJ+20, DNS12], we present the results and the protocol, assuming the availability of a reactive trusted party, called cMPC, that is able to compute any *classical* multiparty functionality. We refer to this as the cMPC-*hybrid model*. Furthermore, we assume that the parties are able to broadcast classical messages. The implementation of cMPC can be done by first removing the reactive assumption using standard techniques, and then implement each call using a post-quantum secure-with-identifiable-abort protocol. We refer the reader to Section 3.2 for more details. We prove the following.

Theorem 1 (Informal). *Assume the existence of a classical quantum-resistant fully homomorphic encryption scheme with decryption circuit of logarithmic depth. Then any multiparty quantum circuit can be computed with security-with-identifiable-abort tolerating any number of corrupted parties in the* cMPC*-hybrid model. Moreover, the round complexity of the quantum communication of the protocol is independent of the circuit complexity.*

The formal statement of the theorem appears in Section 4. A few notes are in place. First, Brakerski and Vaikuntanathan [BV11] showed that the existence of a fully-homomorphic encryption satisfying the conditions stated in Theorem 1 can be reduced to the *learning with errors* assumption.

Second, we note that the protocol can be split into an online phase and an offline phase, where the parties have yet to receive their inputs. In the offline phase, the parties prepare auxiliary magic states in order to compute quantum gates later in the online phase. In fact, it suffices that the parties know only an upper bound on the number of gates in the circuit before interacting in the offline phase.

Third, although the number of rounds requiring quantum information to be sent is independent of the circuit complexity (i.e., independent of the number of gates), it still depends on the number of parties, the number of input-qubits and output-qubits of the circuit, and the security parameter. Specifically, the offline phase consists of $O(n^4 \cdot \kappa)$ rounds, and the online phase consists of $O(n^3 \cdot (\ell_{\mathsf{in}} + \ell_{\mathsf{out}}))$ rounds, where n is the number of parties, κ is the security parameter, and

ℓ_{in} and ℓ_{out} upper-bound the number of input-qubits and output-qubits of the circuit, respectively.

Fourth, although our protocol admits security-with-identifiable-abort, any single corrupted party can cause it to abort. It is arguably more desirable and an interesting open problem to have a protocol that requires the adversary to corrupt more parties to cause an abort.

Finally, an interesting consequence of our construction is that quantum fair secure computation can be implemented assuming the hybrid functionality cMPC is fair.[6]

In the following sections, we present the ideas behind the construction.

1.2 Our Techniques

In this section, we present the main ideas behind the construction of our protocol.

A Warm-Up: Reliable Transmission of Quantum States

Before presenting the general construction, let us consider the following simple task. Suppose that there are n parties $\mathrm{P}_1, \ldots, \mathrm{P}_n$, where P_1 – called the sender – holds a quantum state ρ. The goal of the parties is to send ρ to P_n – called the receiver – such that if either the sender or the receiver is corrupted and deviate from the protocol, then the other parties can identify which of them is corrupted. Moreover, this should hold even the corrupted party collude with some of the other parties in $\{\mathrm{P}_2, \ldots, \mathrm{P}_{n-1}\}$.

As stated before, simply having P_1 send ρ to P_n, and have P_n broadcast a complaint in case it did not receive a message, does not work. Indeed, it could be the case where the receiver is corrupted, and falsely accuse the sender of not sending ρ. Since broadcasting a quantum state is impossible, to the other parties, this scenario is identical to the case where a corrupted sender did not send ρ. Thus, the desired security property is not met. Moreover, due to the no-cloning theorem, the state ρ is now permanently lost, making it unclear as to how to proceed the protocol.

Dealing with false accusations. As such "packet loss" seems unavoidable, our first idea is to not send ρ directly, but rather to encode ρ using a *quantum error-correcting code* (QECC), that can tolerate d deletions, where d will be determined below. This generates an q-qubit codeword $(\sigma_k)_{k=1}^{q}$, for some q, which will then be transmitted qubit-by-qubit as explained below.[7] By doing so, P_n can still recover ρ as long as it receives enough qubits of the codeword.

We next explain how the parties can transmit the codeword's qubits in such a way that will allow them to identify the corrupted party, if such exists. For simplicity of the current discussion, let us assume that the adversary can perform one of the following two attacks. Either it does not send a message, or it can falsely accuse a party of not sending a message. Below we will explain how to

[6] Intuitively, fair computation means that either all parties receive their respective outputs, or none of them do.

[7] Here we abuse the notation that we denote the k^{th} qubit of the codeword σ_k, while these q qubits may be entangled.

remove this assumption and how to resist general malicious attackers. Under these simplifying assumptions, we can make the following observation. If P_n accused P_1 of not sending a message, then all parties know that at least one of them is corrupted. Therefore, they can agree to remove the channel between them, and have P_1 send the next qubit of the codeword via a different path. The parties continue in this fashion until either enough σ_k's where successfully transmitted to the receiver, or until there is no path from the sender to the receiver. Formally, the parties keep track of a simple and undirected graph G, which represents *trust* between parties, i.e., an edge between two vertices exists if and only if there was no accusation between the two parties that the vertices represent. Observe that in the above protocol, all honest parties form a clique in G. Thus, if G becomes disconnected, the honest parties can agree on a corrupted party *not* connected to them. Therefore, using a QECC that can tolerate $d = \Theta(n^2)$ deletions results in a secure protocol.

Dealing with general malicious behavior. Next, we show to remove the simplifying assumption of the behavior of the adversary, and allow it to tamper with the messages arbitrarily. Here, we utilize *quantum authentication codes* [BCG+02], that allow a party to verify if a quantum state was tampered with. However, in our protocol the parties must know where on the path the message had been tampered with (if any tampered occurred), in order to later remove the corresponding edge. To achieve this, we define a new primitive, which we call *sequential authentication* (SA), that allows the sender to transmit a qubit to the receiver along some path, so that if the qubit was tampered with, all parties know where on the path the tampering occurred. We then combine SA with the previous protocol that dealt with false accusations, to construct a secure-with-identifiable-abort protocol for the transmission of a quantum state. One subtlety in the final construction, is that any path from P_1 to P_n must go through all parties, so as to ensure that at least one honest party can verify the integrity of the message.

We now describe the construction of a protocol for sequential authentication. The construction is inspired by the swaddling notion from [DNS12] and the public authentication test from [DGJ+20], which are both based on *Clifford authentication codes*. Let us first recall Clifford codes [ABOE10]. Given a m-qubit state ρ and a security parameter κ, the Clifford encryption[8] appends an auxiliary register $|0^\kappa\rangle\langle 0^\kappa|$, called *traps*. Then, a random Clifford operator E is sampled from the Clifford group acting on $m+\kappa$ qubits. Finally, the encryption outputs the ciphertext $E(\rho \otimes |0^\kappa\rangle\langle 0^\kappa|)E^\dagger$, where E serves as the secret key. The decryption of a Clifford ciphertext σ, simply applies $E^\dagger$ to σ and measures the last κ trap qubits. If the measurement outcome is all-zero, then the decoding algorithm outputs the resulting state of the first m qubits. Otherwise, it rejects. The security of Clifford codes stems from the fact that any operation that is

[8] It is more common to use the term Clifford encoding. However, in the quantum setting authentication implies encryption. Thus, we refer to these as encryptions to remove confusion with the QECC encoding.

applied to the ciphertext, will flip each qubit in the trap with noticeable probability upon measurement. Moreover, the secret key of the Clifford code can be sampled efficiently by a classical algorithm [DLT02].

Constructing a sequential authentication protocol. We utilize these property to build a protocol for SA. Suppose that a message ρ is going to be transmitted through ℓ parties. Let us first present a naïve solution. The first party on the path will append $\ell\kappa$ qubits of $|0\rangle$ to ρ. Then, using the classical MPC functionality cMPC, the parties will securely sample for P_1 a Clifford key E_1 to encrypt its state. It then sends the encrypted message to P_2. To verify the authenticity of the state, the parties will again use cMPC for sampling a Clifford $V_2 = E_2 E_1^\dagger$, where E_2 acts only on the first $(\ell-1)\kappa$ qubits. We then let P_2 receive V_2 and apply it to the encrypted message it received from P_1. This allows P_2 to measure the last κ qubits and compare them to zero. For each party P_i on the transmitting path, P_i measures κ qubits of traps. The parties can then continue in this fashion. Notice, however, that a corrupt P_1 might only append the last κ qubits honestly, which will not be immediately detected by P_2. This could later result in an honest party accusing another honest party. To overcome this issue, we use a similar trick to the public authentication test [DGJ+20], and have the Clifford V_2 that cMPC sampled include a random invertible linear transformation over $\mathbb{F}_2$ acting on all traps. Specifically, we let $V_2 = E_2 G_2 E_1^\dagger$, where we abuse notations and let $G_2 |x\rangle = |G_2(x)\rangle$. Observe that if P_1 did not prepare the traps correctly, then upon measurement with high probability P_2 will not obtain all-zero.

Security With Packet Drops

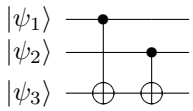
With the above technique, it is natural to incorporate it into the construction of [DGJ+20]. This naïve solution, however, does not work. Towards explaining the issue, let us first briefly describe the protocol of [DGJ+20]. Roughly, their protocol starts with an input encoding phase, such that at the end of the phase each party's input is encrypted under a Clifford code with cMPC holding the secret Clifford key. This is done similarly to the sequential authentication protocol described earlier. The parties then proceed to perform computation over the encrypted inputs. Computation over single-qubit Clifford gates can be done by simply letting cMPC update its key, while CNOT gates require communication since the inputs to CNOT gates are encrypted separately under different Clifford keys.

While the input encoding phase can be modified to admit security-with-identifiable-abort, it is unclear how to modify the computation phase of the protocol. This follows from the fact that the parties are required to use QECC over their inputs in the input encoding phase, thus at the end of this phase, each party will hold a Clifford encoding of each qubit of its input's codeword. As a result, the parties have to either perform the computation over QECC codewords in some way, or decode the Clifford encrypted codewords and perform computation similarly to [DGJ+20]. We next give an intuitive explanation as to why both solutions fail.

Let us first argue why the second solution fails. That is, suppose the parties decode all QECC encodings before starting to perform any computation. The issue here is that once the parties decode the QECC they lose its protection, hence the protocol cannot tolerate losing quantum states after this step. Since the protocol of [DGJ+20] requires communicating quantum messages to compute CNOT gates, this causes inevitable packet drops *during computation*, causing the honest parties to output incorrect values.

The former solution fails due to the fact that a corrupted party might not encode its qubit correctly using the QECC. Observe that our sequential authentication protocol will not be able to detect such error, since it is able to detect an attack only after a Clifford had been applied. Furthermore, this error might propagate into the evaluation. Indeed, consider the following example.

Suppose that the parties use repetition code as an implementation of the QECC.[9] In repetition code, a logical zero $|\bar{0}\rangle$ is encoded as $|000\rangle$ and a logical one $|\bar{1}\rangle$ is encoded as $|111\rangle$. The decoding is done by taking the majority, e.g., $|000\rangle$, $|001\rangle$, $|010\rangle$ and $|100\rangle$ are all decoded to $|\bar{0}\rangle$. Suppose three parties wish to compute the following circuit, where the CNOTs are applied transversally, and where the inputs $|\psi_i\rangle$ are repetition codes of logical $|\bar{0}\rangle$.



Clearly, in an honest execution the value of $|\psi_3\rangle$ becomes $|000\rangle$ which decodes to $|0\rangle$. Now, suppose the two parties holding $|\psi_1\rangle$ and $|\psi_2\rangle$ are corrupted and prepares $|\psi_1\rangle = |001\rangle$ and $|\psi_2\rangle = |010\rangle$. Then the value of $|\psi_3\rangle$ under such an attack becomes $|011\rangle$. Consequently, even if all codewords are of logical 0 at the beginning, the decoding would result in a logical 1.

A possible way to try and fix this issue, would be to try to *correct* the QECC codewords. However, this in particular would require the parties to compute a multi-qubit gate (e.g., CNOT), which as stated before, cannot be done without losing the quantum states due to a potential attack.

With this state of affairs, we aim to construct a protocol that has the property that no adversary can cause qubits to be "dropped" during the computation of the circuit. Thus, we first propose an abstraction of a security notion that allows the adversary to "drop" some of the input-states and output-states. We call this security notion secure-with-identifiable-abort-and-packet-drop (IDPD-security). We then show how to reduce the problem of constructing a secure-with-identifiable-abort protocol to the problem of constructing an IDPD-secure protocol.

Defining IDPD-security. Let us now define IDPD-security. Similarly to other notions of security in multiparty computation, here we follow the standard ideal vs. real paradigm. Roughly, the ideal-world follows similar instructions to

[9] Repetition codes only resist bit-flip error (i.e., Pauli X attack). However, it is sufficient for the purposes of demonstration here.

that of the security-with-identifiable-abort ideal-world, with the following two additions. First, when the parties send their inputs to the trusted party, the adversary additionally sends it a bounded-sized set, representing which input-qubits are to be replaced with $|0\rangle$ (modelling "packet drop"). Note that it might be the case where a single party holds several qubits as inputs, and the adversary changes only a subset of them to the 0 state. The second change we make is done after the adversary receives its output from the trusted party. Here, the adversary either instructs the trusted party to abort while revealing the identity of a corrupted party, or it instructs the trusted party to continue and drop some qubits from the output.[10] In case the adversary instructed to continue, the trusted party then sends to all other parties their respective outputs that remained. Additionally, the trusted party reveals which input-qubits and which output-qubits were dropped. The formal definition of IDPD-security can be found in Section 3.1.

Reducing SWIA to IDPD-security. We now show a simple reduction from SWIA to IDPD-security. The reduction makes use of a QECC. Let C be the circuit that the parties wish to compute. First, each party encodes its input using the QECC. The parties then use an IDPD-secure protocol in order to compute the circuit C' that first decodes its inputs using the QECC, then applies C, and finally re-encodes each output using the QECC. Upon receiving their encoded outputs, each party locally decodes it to obtain their output. To see why this reduction works, observe that the adversary can only drop some of the qubits in the input to C' and some of the qubits in the output. Therefore, by the properties of the QECC and IDPD-security, either the original state can be reconstructed, or the adversary has revealed the identity of a corrupted party.

Securely Computing A General Circuit

We next explain how to achieve a secure protocol for computing a general circuit. With the above reduction, it suffices to construct an IDPD-secure protocol. Unfortunately, previous approaches, such as that of [DGJ+20], for constructing secure protocols fail to achieve IDPD-security. Indeed, as stated before, the protocol of [DGJ+20] requires communicating quantum messages to compute CNOT gates, which causes inevitable packet drops during computation and thus fails to achieve IDPD-security.

Our approach. To circumvent the aforementioned issue, the parties need a way to perform computation *without* quantum communication. To do so, our main idea is to delegate the computation to some designated party, say P_1, and let it perform computation under *verifiable quantum fully homomorphic encryption* (VQFHE) [ADSS17]. More precisely, the first step of our protocol will encrypt all parties' input using the VQFHE scheme of [ADSS17], called TrapTP, send their encrypted inputs to P_1, and store the VQFHE classical secret key sk in cMPC. We refer to this step as the *pre-computation* step. This allows us to let P_1

[10] Formally, the ideal-world is parametrized by two polynomial in the security parameter that bound the number input-qubits and number of output-qubits that can be dropped.

perform the computation homomorphically to obtain encrypted output without any quantum communication. Furthermore, the verification of the evaluation can be done using the help of cMPC holding sk. If the verification passes, P_1 delivers the output to each party. Note that an additional advantage of our approach is that the round complexity of our protocol is independent of the circuit complexity.

VQFHE scheme TrapTP. We first review some useful facts about the TrapTP scheme. In TrapTP, the encryption of a 1-qubit state $|\psi\rangle$ consists of a quantum part and a classical part. The quantum part is a *trap code* encryption of $|\psi\rangle$

$$\Pi X^x Z^z(\mathsf{QECC.Enc}(|\psi\rangle) \otimes |0\rangle^{\otimes\kappa} \otimes |+\rangle^{\otimes\kappa}),$$

where Π is a random permutation over 3κ qubits (which is part of the secret key sk) and $x, z \leftarrow \{0,1\}^{3\kappa}$ are sampled independent and uniformly at random. The classical part is a classical FHE encryption of the Pauli key x, z. Homomorphic evaluation requires a quantum evaluation key ρ_{evk}, which consists of multiple TrapTP encryptions of magic states, including ancilla zero states, phase (P) states $|P\rangle := P|+\rangle$, Hadamard (H) states $|H\rangle := (H \otimes I)\mathsf{CNOT}(|+\rangle \otimes |0\rangle)$, T states $|T\rangle := T|+\rangle$, and a special gadget state $|\gamma\rangle$ (see Section 7.3 in the full version [ACC+20] for a more detailed definition of $|\gamma\rangle$). These (encrypted) states are used to perform computation homomorphically over the underlying trap codes.

The pre-computation step. Recall that the goal is to send TrapTP encrypted inputs to P_1, with the secret key stored in cMPC. The first step is to let each party send their input to P_1 using the technique we developed in Section 1.2. Namely, we let each party to send Clifford encryptions of their input qubits using sequential authentication protocol through paths determined by a trust graph G. We formalize this as an *authenticated routing* (AR) protocol that achieves the following functionality with IDPD-security.

> **Authenticated Routing (AR):** As input, each sender P_i holds multiple quantum messages $\rho_1, \ldots, \rho_\ell$ (the "packets") to send to P_1. As output, the receiver P_1 receives Clifford ciphertexts $\sigma_j = E_j(\rho_j \otimes |0^t\rangle\langle 0^t|)E_j^\dagger$ with trap size t and cMPC receives the Clifford keys E_j for $j \in [\ell]$ with at most n^2 packet drop.

We note that in AR, a packet ρ_j can consist of multiple qubits and the trap size can be set arbitrarily; these properties will be useful later. Here, we let each P_i send their input qubit-by-qubit to P_1 using AR with trap size $3\kappa - 1$. After that, P_1 holds Clifford encodings of all parties' input (with certain packet drops). Note that AR allows to drop at most n^2 input states, while it is acceptable in IDPD-security.

However, in TrapTP, the quantum messages are encrypted under trap code instead of Clifford code. We next use the following simple *re-encrypt* protocol to turn Clifford codes into trap codes: Let $\sigma = E(\rho \otimes |0^{3\kappa-1}\rangle\langle 0^{3\kappa-1}|)E^\dagger$ be a

Clifford encoding of ρ held by P_1 with the corresponding Clifford key E held by cMPC. We simply let cMPC send to P_1 the Clifford operator

$$V = X^x Z^z \Pi (U_{\mathsf{Enc}} \otimes I^{\otimes\kappa} \otimes H^{\otimes\kappa}) E^\dagger,$$

where U_{Enc} is an unitary operator maps $\rho \otimes |0^{\kappa-1}\rangle\langle 0^{\kappa-1}|$ into an QECC codeword. Observe that if P_1 applies V to σ, the result would be a trap-code encryption of ρ, which is also the quantum part of the TrapTP encryption of ρ. Also note that since the Clifford key E is uniformly random to P_1, it serves as a one-time pad, hence P_1 learns nothing about the trap code secret Π, x, z from V. After that, we can let cMPC generate and send the classical part of the TrapTP encryption of ρ to P_1 so that it obtains a complete TrapTP encryption of ρ.

It is worth mentioning that a natural alternative is to use trap code to construct SA in AR to avoid using two different codes with re-encryption. However, this does not provide a secure protocol since, unlike Clifford codes, in trap codes each qubit is encrypted individually. If only one qubit has been tampered with, then there is no guarantee that the adversary would be immediately caught.

To conclude the pre-computation step, it is left to prepare the evaluation key ρ_{evk} for P_1, which consists of multiple TrapTP encryptions of auxiliary magic states and a special gadget state. Preparing such states turns out to be involved, which we discuss next.

Magic state preparation (except T). We first note that it suffices to generate Clifford encryption of these states, and we can apply the above re-encryption protocol to turn them into TrapTP encryption.

Let us start with the simplest case of ancilla zero state $|0\rangle$. For this, we can use the AR protocol to send the the empty state, denoted ε, with trap size 3κ to prepare it. Indeed, the Clifford encoding outputs

$$E(\varepsilon \otimes |0^{3\kappa}\rangle\langle 0^{3\kappa}|)E^\dagger = E(|0\rangle\langle 0| \otimes |0^{3\kappa-1}\rangle\langle 0^{3\kappa-1}|)E^\dagger,$$

as required. Note that AR protocol takes as input a list of "packets," where n^2 packets may be dropped. Since magic state preparation is independent to parties' private states, the parties actually call AR protocol with n^2+1 packets to make sure that at least one packet can be delivered. Then, the server and cMPC keep the lexicographically first remaining packet. For simplicity, we omit the number of initial packets.

Next, consider preparing a $|P\rangle$ magic state. Since a P gate is a Clifford, we can generate it by preparing encoding of $|0\rangle$ and update the Clifford key held by cMPC. Specifically, if cMPC updates its Clifford E to $E(PH)^\dagger$ (where PH is applied only to the first qubit of the codeword), then decrypting the ciphertext with the updated key would result in

$$(E(PH)^\dagger)^\dagger E(|0\rangle \otimes |0^{3\kappa-1}\rangle) = PH(|0\rangle \otimes |0^{3\kappa-1}\rangle) = |P\rangle \otimes |0^{3\kappa-1}\rangle\,.$$

The $|H\rangle$ magic state, is also generated by a Clifford, but consists of two qubits. To generate this, we first use AR to send the the empty state with trap

size 6κ and view it as

$$E(|0\rangle^{M_1} \otimes |0\rangle^{M_2} \otimes |0^{3\kappa-1}\rangle^{T_1} \otimes |0^{3\kappa-1}\rangle^{T_2}),$$

where the gray superscript denote the registers the qubits are stored in. Then, we let cMPC send to P_1 the Clifford operator

$$V = (E_1^{M_1T_1} \otimes E_2^{M_2T_2})(H \otimes I)\mathsf{CNOT}(H \otimes I)^{M_1M_2}E^\dagger,$$

where E_1 and E_2 are two Clifford sampled uniformly at random and independently, and where the gray superscript denote the registers on which each operator acts. Observe that upon applying V to its codeword, P_1 will obtain an encrypted H state. Additionally, as V is distributed like a uniform random Clifford operator, it follows that a corrupted P_1 will gain no new information.

More generally, the above examples suggest that we can prepare any ℓ-qubit state in the Clifford group by first preparing Clifford encoding of $3\ell\kappa$ qubits $E|0^{3\ell\kappa}\rangle$ using AR, and letting cMPC send Clifford operator V to instruct P_1 to prepare the Clifford state and split it into ℓ Clifford encodings of each qubit. We note that the special gadget state $|\gamma\rangle$ is of this type and therefore can be prepared in this way.

T magic state preparation. Among all magic states, the preparation of $T := T|+\rangle$ magic state is the most difficult, since T is not a Clifford operator. We follow a similar approach to that of [DGJ+20], but with modifications to achieve security-with-identifiable-abort. Here, we give a brief overview of their construction and discuss the required modifications.

At a high-level, the protocol asks a party, say P_1, to prepare a large number N of (supposedly) $|T\rangle$ states under Clifford encoding with Clifford keys stored in cMPC. This can be done by, e.g., letting P_1 send these states using AR in our context. Then, the parties randomly distribute these encoded states among themselves, and have $\mathrm{P}_2, \ldots, \mathrm{P}_n$ verify that they are indeed $|T\rangle$ states. This is done by sending the Clifford keys to P_i, and having P_i measure the decoded states in the $\{|T\rangle, |T^\perp\rangle\}$-basis. If any $|T^\perp\rangle$ outcome is detected, the protocol aborts. If not, then we know that the states held in P_1 contains only a small number of errors with high probability. The protocol then apply a T state distillation circuit (over the encoded states) to distill the desired T magic states.

To achieve security-with-identifiable-abort, we cannot let the protocol be aborted when an error is detected, since the parties cannot distinguish the case where the error was due to a malicious P_1 preparing incorrect states, or a malicious party P_i falsely reporting the error. Thus, to identify the malicious party, we let each party P_i report its *error rate* ϵ_i, i.e., the fraction of $|T^\perp\rangle$ outcomes it obtained, to cMPC with ϵ_1 set to 0. cMPC then sort these numbers, and check if there are two consecutive numbers with difference greater than a certain threshold δ that is larger than expected sampling errors. If so, cMPC finds the smallest such pairs, say, they are $\epsilon_i < \epsilon_j$ reported by P_i and P_j, respectively, and publish the result. The parties then abort, with an honest party P_k identifying P_i (resp., P_j) as the malicious party if $\epsilon_k \geq \epsilon_j$ (resp., $\epsilon_k \leq \epsilon_i$). Intuitively, this

works since all honest parties should obtain roughly the same error rate up to a small sampling error, and hence they will belong to the same side and accuse the same party being the malicious party. Also, if the protocol does not abort, it means that all reported error rates are small, since $\epsilon_1 = 0$ and we still have the guarantee that the error rate of the states held in P_1 is small.

The second issue is that we need to be able to apply the T state distillation circuit to the (Clifford encrypted) states held by P_1, which is a classically-controlled Clifford circuit (A circuit consists of Clifford gates and measurements, and which Clifford gates should be applied depends on all previous measurement outcomes.). If these states are encrypted separately, then we do not know how to compute the distillation circuit without quantum communication, as this is the problem we want to solve to begin with. Fortunately, as discussed above, if these states are encrypted as a single Clifford ciphertext of a multi-qubit message, then we can perform Clifford operation on the underlying message and split it into multiple Clifford ciphertexts of smaller messages by letting cMPC sending proper Clifford instruction to P_1. We can further extend it to evaluate classically-controlled Clifford circuit. Based on this observation, we let P_1 to prepare the N copies of $|T\rangle$ states and send it as a N-qubit quantum message $\rho = |T\rangle^{\otimes N}$ in AR (with a sufficiently large trap size). This allows us to distribute the states to all parties (by splitting the ciphertexts) and apply the T state distillation circuit to the states held by P_1 later.

Final issue: re-encryption to Clifford codes. The computation step is rather straightforward, so we do not discuss the details here but just state that as a result, P_1 holds trap code encoding of the output. All that is left is to show how it can distribute each output to its corresponding party. The idea is to reverse the operations done until now. That is, to first re-encrypt the trap codes back to Clifford codes, and then use AR to distribute the outputs. The final issue is that re-encrypting trap code to Clifford cannot be done in the same way as it was done in the other direction. This is because before, we use the randomness of the Clifford key as one-time pad to protect the trap code key, but now the randomness in the trap code key is not enough to protect the Clifford key.

To resolve the issue, we again use AR. Let us say σ is a trap code that P_1 needs to send to a party P_i. We let P_1 send ρ as a 3κ-qubit message to itself using AR. As a result, P_1 will receive a Clifford encoding $\sigma = E(\rho \otimes |0^t\rangle\langle 0^t|)E^\dagger$ (with a sufficiently large trap size t) for which we can let P_1 perform Clifford operation on the underlying message ρ. Note that if P_1 is malicious, the underlying message ρ of σ may not be a valid trap code. Thus, we let P_1 and cMPC verify and decode the supposedly trap code ρ. Specifically, cMPC will check the classical parts of the computation. If the verification rejects, we abort and identify P_1 as the malicious party. If it passes, then we obtain a Clifford encoding of the qubit underlying the trap code as desired. Finally, we remind the reader that some of the trap codes in ρ may be dropped by AR, but this is allowed since IDPD-security allows to drop part of the output qubits.

1.3 Roadmap

In Section 2 we provide the required preliminaries. In Section 3 we explain in detail the model of our computation. Then, in Section 4 we state our main theorem and show the reduction to IDPD-security. In Section 5 we give the construction of sequential authentication, and in Section 6 we use it to construct authenticated routing. These constructions admits information theoretic security. Following that, in Section 7 we show how to prepare all required magic states. In Section 8 we show how to securely compute the pre-computation protocol, Section 9 is dedicated to performing the computation of the circuit, and finally, in Section 10 we show how the parties can distribute the output securely. We note that only the computation protocol from Section 9 has computational security.

2 Preliminaries

For space considerations, most standard definitions and notations are deferred to the full version [ACC+20]. We next provide the definitions that we find more essential for the readability of the bulk of the paper.

For $n \in \mathbb{N}$, let $[n] = \{1, 2 \dots n\}$. We also let Sym_n to denote the symmetric group over n symbols. Given a binary string x, we write $|x|$ to denote the length of x, and $w(x)$ to denote the *relative Hamming weight* of x which equals to Hamming weight of x divided by $|x|$. For a string x and a subset $\mathcal{S} \subseteq [|x|]$, we use $x_{\mathcal{S}}$ to denote the substring of x indicated by $\mathcal{S}$.

Given a set $\mathcal{S}$, we write $s \leftarrow \mathcal{S}$ to indicate that s is selected uniformly at random from $\mathcal{S}$. Similarly, given a random variable (or a distribution) X, we write $x \leftarrow X$ to indicate that x is selected according to X. A function $\mu \colon \mathbb{N} \to [0, 1]$ is called negligible, if for every positive polynomial $p(\cdot)$ and all sufficiently large n, it holds that $\mu(n) < 1/p(n)$. We use $\mathrm{neg}(\cdot)$ to denote an unspecified negligible function.

For $n \in \mathbb{N}$, we use $\mathcal{H}^n$ to denote the Hilbert space of n qubits. We write $D(\mathcal{H})$ to denote the set of density matrices over the Hilbert space $\mathcal{H}$, and let $\mathcal{D}^n := D(\mathcal{H}^n)$. We define $\mathcal{D}^* := \bigcup_{n=0}^{\infty} D(\mathcal{H}^n)$ to denote the set of the density matrices acting on the Hilbert space of arbitrary number of qubits. We use lowercase Greek alphabets, e.g., ρ, σ, τ, to denote quantum state. We use capital Latin alphabets, e.g., A, B, M, T, to denote quantum registers. For a quantum register A, we write $|A|$ to denote the number of qubits in it. We denote the Hilbert space of a quantum register A by $\mathcal{H}_A$. The Hilbert space $\mathcal{H}_{AB}$ of a joint quantum register AB is the tensor product of the Hilbert spaces of each subsystems, that is, $\mathcal{H}_{AB} = \mathcal{H}_A \otimes \mathcal{H}_B$. It will be convenient to denote by $\varepsilon \in \mathcal{D}^0$ the empty state.

The trace distance between two quantum states ρ and σ, denoted as $\Delta(\rho, \sigma)$, is define by $\Delta(\rho, \sigma) = \frac{1}{2}\|\rho - \sigma\|_1$, where $\|M\|_1 = \mathrm{tr}\left(\sqrt{M^\dagger M}\right)$ is the trace norm of a matrix. Let $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. An EPR pair is the two-qubit state $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.

A *state ensemble* $\rho = \{\rho_{a,\kappa}\}_{a \in \mathcal{D}_\kappa, \kappa \in \mathbb{N}}$ is an infinite sequence of quantum states indexed by $a \in \mathcal{D}_\kappa$ and $\kappa \in \mathbb{N}$, where $\mathcal{D}_\kappa$ is a domain that might depend

on κ. When the domains of a and κ are clear from context, we remove them for brevity. We write $\rho \approx_{\mathrm{neg}(\kappa)} \sigma$ if there exists a negligible function μ, such that for all $\kappa \in \mathbb{N}$ and $a \in \mathcal{D}_\kappa$, it holds that $\Delta(\rho_{a,\kappa}, \sigma_{a,\kappa}) \leq \mu(\kappa)$. We sometimes abuse notations and write $\rho_{a,\kappa} \approx_{\mathrm{neg}(\kappa)} \sigma_{a,\kappa}$.

Let QPT stand for quantum polynomial time. Computational indistinguishability is defined as follows.

Definition 1 *Let $\rho = \{\rho_{a,\kappa}\}_{a\in\mathcal{D}_\kappa,\kappa\in\mathbb{N}}$ and $\sigma = \{\sigma_{a,\kappa}\}_{a\in\mathcal{D}_\kappa,\kappa\in\mathbb{N}}$ be two ensembles. We say that ρ and σ are* computationally indistinguishable*, denoted $\rho \overset{c}{\equiv} \sigma$, if for every non-uniform* QPT *distinguisher* D*, there exists a negligible function $\mu(\cdot)$, such that for all $\kappa \in \mathbb{N}$ and $a \in \mathcal{D}_\kappa$, it holds that*

$$\left|\Pr\left[\mathsf{D}(\rho_{a,\kappa}) = 1\right] - \Pr\left[\mathsf{D}(\sigma_{a,\kappa}) = 1\right]\right| \leq \mu(\kappa).$$

For a quantum operator U, we write U^A to specify that the quantum operator U acts on register A. Similarly, we write ρ^A to specify that the quantum state ρ lies in register A. Here, the register written in gray on the superscript is only for reminder, and whether it is written does not change the meaning of the operator or the state. That is, $U^A = U$ and $\rho^A = \rho$. We write χ^A to denote the maximally mixed state $I^A/|A|$ of register A. For $n \in \mathbb{N}$ we let $\mathcal{C}_n$ denote the set of Clifford operators acting on n qubits.

In this paper we make use of quantum error-correcting codes (QECC) and quantum authentication scheme (QAS) [ABOE10]. A QAS is a way to ensure that quantum state was not tampered with. We refer the reader to Section 2 of the full version [ACC+20] for a formal definition of a QAS. To remove confusion with the encoding and decoding of QECC, we view QAS as an encryption scheme.

We make use of Clifford codes [ABOE10]. Roughly, they are defined as follows. The encryption procedure, denoted $\mathsf{CAuth.Enc}_E$, encrypt a quantum message ρ using a Clifford E as its key (sampled uniformly at random). The procedure first append to ρ a *trap* $|0^t\rangle\langle 0^t|$, where t is considered the security parameter, and then applies the Clifford E to $\rho \otimes |0^t\rangle\langle 0^t|$. The decoding procedure, denoted $\mathsf{CAuth.Dec}_E$, accepts if and only if after applying $E^\dagger$ to its input and measure the traps, the value of the trap is $|0^t\rangle\langle 0^t|$. We refer the reader to Section 2 of the full version [ACC+20] for a formal definition alongside the security properties of Clifford encodings.

We also use trapcodes [BGS13]. A trapcode encrypts a single qubit ρ^M held in register M as follows. First, apply $\mathsf{QECC.Enc}$ on register M, and append t-qubits from register T_X in the state $(|0\rangle\langle 0|)^{\otimes t}$, and append t-qubits from register T_Z in the states $(|+\rangle\langle +|)^{\otimes t}$. Then permute the qubits of $\tilde{M}T_XT_Z$ according to Π, where $\tilde{M}$ is the register holding the encoding of the qubit ρ. Finally, apply X^xZ^z on register $\tilde{M}T_XT_Z$. That is,

$$\mathsf{TAuth.Enc}_{\Pi,x,z}(\rho) := X^xZ^z\Pi\left(\mathsf{QECC.Enc}(\rho) \otimes (|0\rangle\langle 0|)^{\otimes t} \otimes (|+\rangle\langle +|)^{\otimes t}\right)(X^xZ^z\Pi)^\dagger.$$

The decryption applies $(X^xZ^z\Pi)^\dagger$ onto register $\tilde{M}T_XT_Z$, and measure the register T_X in computational basis, and measure the register T_Z in Hadamard $\{|+\rangle, |-\rangle\}$-basis. If the outcome of T_X is all zeros and the outcome of T_Z is all $+$, then apply $\mathsf{QECC.Dec}$ on register $\tilde{M}$ and set $|\mathsf{Acc}\rangle\langle\mathsf{Acc}|$ in F. Otherwise,

replace the state in M with $|\bot\rangle\langle\bot|$ and set $|\mathsf{Rej}\rangle\langle\mathsf{Rej}|$ in F. Since X^xZ^z is a Pauli operator up to a phase ± 1 or $\pm i$, we sometimes write $\mathsf{TAuth.Enc}_{\Pi,P}$ and $\mathsf{TAuth.Dec}_{\Pi,P}$, where P is a Pauli operator. We refer the reader to Section 2 of the full version [ACC+20] for a formal definition alongside the security and homomorphic properties of trapcodes.

We define the trap code *partial decryption* operation $\mathsf{TAuth.PDec}$, as the unitary part of $\mathsf{TAuth.Dec}$. That is, it decodes the permutation and quantum one time pad, perform the (unitary part of) QECC decoding on the first t qubits, and then apply Hadamards on the last t qubits to map $|+\rangle$ to $|0\rangle$. Formally, we define it as follows.

Definition 2 *Let $x, z \in \{0,1\}^m$ and let $\Pi \in \mathrm{Sym}_m$. Then,*

$$\mathsf{TAuth.PDec}_{\Pi,x,z} := (U_{\mathsf{Dec}} \otimes I^{\otimes 2t})(I^{\otimes 2t} \otimes H^{\otimes t})(X^xZ^z\Pi)^\dagger,$$

where U_{Dec} is the unitary operator corresponding to the $\mathsf{QECC.Dec}$ *circuit.*

Notice that when applied to a $\mathsf{TAuth.Enc}$ encoding of ρ using the same keys, the result is $\rho \otimes |0^{2t}\rangle\langle 0^{2t}|$. Similarly, we define the trap code *partial encryption* operation $\mathsf{TAuth.PEnc}$ as the unitary part of $\mathsf{TAuth.Enc}$.

Definition 3 *Let $x, z \in \{0,1\}^m$ and let $\Pi \in \mathrm{Sym}_m$. Then,*

$$\mathsf{TAuth.PEnc}_{\Pi,x,z} := X^xZ^z\Pi(U_{\mathsf{Enc}} \otimes I^{\otimes t} \otimes H^{\otimes t}),$$

where U_{Enc} is the unitary operator corresponding to the $\mathsf{QECC.Enc}$ *circuit.*

3 The Model of Computation

The security of multiparty computation protocols is defined using the real vs. ideal paradigm. In this paradigm, we consider the real-world model, in which protocols are executed. Here, an n-party quantum protocol π for computing a quantum circuit family $C = \{C_\kappa\}_{\kappa\in\mathbb{N}}$ is defined by a set of n interactive uniform QPT circuits $\mathcal{P} = \{\mathrm{P}_1, \ldots, \mathrm{P}_n\}$. To alleviate notation, we simply write C for the circuit. We then formulate an ideal model for executing the task. This ideal model involves a trusted party whose functionality captures the security requirements of the task. Finally, we show that the real-world protocol "emulates" the ideal-world protocol, i.e., for any real-world adversary $\mathcal{A}$ there exists an ideal-world adversary Sim (called the simulator) such that the global output of an execution of the protocol with $\mathcal{A}$ in the real-world is distributed similarly to the global output of running Sim in the ideal model.

In this work we are mainly interested in the security notion called *security-with-identifiable-abort*. Due to space considerations, the formal definition is deferred to Section 3 of the full version [ACC+20].

3.1 Security With Packet Drops

We now introduce a relaxed security notion of security-with-identifiable-abort that allows the adversary to drop some of the input-qubits and some of the output-qubits. We call this security notion *IDPD-security*. This security notion is

parameterized with two polynomials $d_{\mathsf{in}} = d_{\mathsf{in}}(\kappa)$ and $d_{\mathsf{out}} = d_{\mathsf{out}}(\kappa)$ representing an upper bound on the number of input-qubits and output-qubits, respectively, the adversary is allowed to drop from the computation. The definition follows the standard ideal vs. real paradigm.

Informally, in the ideal world, in addition to sending inputs, the adversary also instructs the trusted party which single qubits are to be replaced with 0. Then, upon receiving the output, the adversary can decide to either abort the protocol while revealing the identity of a corrupted party, or to instruct the trusted party to discard some of the qubits in the output and distribute it.

We now formally describe the $(d_{\mathsf{in}}, d_{\mathsf{out}})$-IDPD ideal model, which specifies the requirements for an IDPD-secure computation of a circuit C with security parameter κ. Unlike the informal discussion from Section 1.2, it will be more convenient to have the adversary send to the trusted party the set of *remaining* qubits. Let $\mathcal{A}$ be an adversary in the ideal-world, which is given an auxiliary quantum state ρ_{aux} and corrupts a subset $\mathcal{I}$ of the parties.

Security with identifiable abort and packet drops

Inputs: Each honest party P_i holds the security parameter 1^κ where $\kappa \in \mathbb{N}$ and an input $\rho_i = (\rho_{ij})_{j=1}^{\ell_{\mathsf{in}}}$ where each $\rho_{ij} \in \mathcal{D}^1$ is single-qubit. The adversary is given 1^κ, input ρ_i of every corrupted party $\mathrm{P}_i \in \mathcal{I}$, and an auxiliary input ρ_{aux}. Finally, the trusted party T is given the security parameter 1^κ.

Parties send inputs: Each honest P_i sends ρ_i to T. For every corrupted party P_i, the adversary sends a state ρ_i^* to T as the input of P_i.

The adversary instructs T to drop some input-qubits: The adversary sends to T a set $\mathcal{R}_{\mathsf{in}} \subseteq \left\{(i,j) \in \mathbb{N}^2 \mid i \in [n], j \in [\ell_{\mathsf{in}}]\right\}$ of size $|\mathcal{R}_{\mathsf{in}}| \geq n\ell_{\mathsf{in}} - d_{\mathsf{in}}$ (note that it could be the case that $n\ell_{\mathsf{in}} < d_{\mathsf{in}}$, in which case no restriction are imposed on $\mathcal{R}_{\mathsf{in}}$). Denote

$$\rho'_{ij} = \begin{cases} |0\rangle\langle 0| & \text{if } (i,j) \notin \mathcal{R}_{\mathsf{in}} \\ \rho_{ij} & \text{if } (i,j) \in \mathcal{R}_{\mathsf{in}} \text{ and } i \notin \mathcal{I} \\ \rho^*_{ij} & \text{if } (i,j) \in \mathcal{R}_{\mathsf{in}} \text{ and } i \in \mathcal{I} \end{cases}$$

and let $\rho' = (\hat{\rho}_{ij})_{i \in [n], j \in [\ell_{\mathsf{in}}]}$.

The trusted party performs the computation: The trusted party T prepares ancilla zero states ρ_0 and computes $C(\rho', \rho_0)$. Let $(\sigma_1, \ldots, \sigma_n, \sigma_{\mathsf{discard}})$ be the resulting output-states, where σ_i is the output associated with party P_i. The trusted party sends $\sigma_{\mathcal{I}}$ to $\mathcal{A}$.

Adversary instructs T to drop some output-qubits or halt: For every $i \in [n]$ write $\sigma_i = (\sigma_{ij})_{j=1}^{\ell_{\mathsf{out}}}$, where each $\sigma_{ij} \in \mathcal{D}^1$ is single-qubit. The adversary $\mathcal{A}$ sends to T either $(\texttt{continue}, \mathcal{R}_{\mathsf{out}})$ where $\mathcal{R}_{\mathsf{out}} \subseteq \left\{(i,j) \in \mathbb{N}^2 \mid i \in [n], j \in [\ell_{\mathsf{out}}]\right\}$ is of size $|\mathcal{R}_{\mathsf{out}}| \geq \ell_{\mathsf{out}} - d_{\mathsf{out}}$, or $(\texttt{abort}, \mathrm{P}_i)$ for some $\mathrm{P}_i \in \mathcal{I}$. If the adversary sent $(\texttt{continue}, \mathcal{R}_{\mathsf{out}})$, then for every honest party $\mathrm{P}_i \notin \mathcal{I}$, the trusted party sends it $(\mathcal{R}_{\mathsf{in}}, \mathcal{R}_{\mathsf{out}}, \sigma'_i)$, where

$\sigma'_i = \left(\sigma'_{ij}\right)_{j=1}^{\ell_{\mathsf{out}}}$ are defined as

$$\sigma'_{ij} = \begin{cases} \sigma_{ij} & \text{if } (i,j) \in \mathcal{R}_{\mathsf{out}} \\ \bot & \text{if } (i,j) \notin \mathcal{R}_{\mathsf{out}} \end{cases}$$

Otherwise, if $\mathcal{A}$ sent $(\mathtt{abort}, \mathrm{P}_i)$, then T sends $(\mathtt{abort}, \mathrm{P}_i)$ to all honest parties.

Outputs: Each honest party outputs whatever it received from the trusted party, the parties in $\mathcal{I}$ output nothing, and the adversary outputs some function of its view.

Observe that if $d_{\mathsf{in}} = d_{\mathsf{out}} = 0$ then the above process is identical to the security-with-identifiable-abort process. We denote by $\mathrm{IDEAL}^{(d_{\mathsf{in}},d_{\mathsf{out}})\text{-}\mathsf{IDPD}}_{C,\mathcal{A}(\rho_{\mathsf{aux}})}(\kappa, (\rho_i)_{i=1}^n)$ the joint output of the adversary $\mathcal{A}$ and the honest parties in an execution of the above ideal-world computation of C, on security parameter κ, inputs $(\rho_i)_{i=1}^n$, auxiliary input ρ_{aux}, and packet-drop bounds d_{in} and d_{out}. When d_{in} and d_{out} are clear from context, we remove them from the notations.

We next give the definition of IDPD-security.

Definition 4 (IDPD-security) *Let π be a protocol for computing a circuit C, and let $d_{\mathsf{in}} = d_{\mathsf{in}}(\cdot)$ and $d_{\mathsf{out}} = d_{\mathsf{out}}(\cdot)$ be two polynomials. We say that π computes C with computational $(d_{\mathsf{in}}, d_{\mathsf{out}})$-*IDPD-security*, if the following holds. For every non-uniform* QPT *adversary $\mathcal{A}$, controlling a set $\mathcal{I} \subset \mathcal{P}$ in the real-world, there exists a non-uniform* QPT *adversary $\mathsf{Sim}_{\mathcal{A}}$, controlling $\mathcal{I}$ in the IDPD ideal-world, such that*

$$\left\{\mathrm{IDEAL}^{(d_{\mathsf{in}},d_{\mathsf{out}})\text{-}\mathsf{IDPD}}_{C,\mathsf{Sim}_{\mathcal{A}}(\rho_{\mathsf{aux}})}(\kappa, (\rho_i)_{i=1}^n)\right\}_{\kappa\in\mathbb{N},\rho_1,\ldots,\rho_n,\rho_{\mathsf{aux}}\in\mathcal{D}^*} \overset{C}{\equiv} \left\{\mathrm{REAL}_{\pi,\mathcal{A}(\rho_{\mathsf{aux}})}(\kappa, (\rho_i)_{i=1}^n)\right\}_{\kappa\in\mathbb{N},\rho_1,\ldots,\rho_n,\rho_{\mathsf{aux}}\in\mathcal{D}^*} \quad (1)$$

Statistical and perfect security are defined similarly by replacing $\overset{C}{\equiv}$ with $\approx_{\mathrm{neg}(\kappa)}$ and $=$, respectively, and assuming unbounded adversaries and simulators.

In Section 4, we reduce the problem of constructing a secure-with-identifiable-abort protocol, to the problem of constructing an IDPD-secure protocol.

3.2 The Hybrid Model

The *hybrid model* is a model that extends the real model with a trusted party that provides ideal computation for specific circuits. The parties communicate with this trusted party as specified by the ideal model.

Let C be a quantum circuit. Then, an execution of a protocol π computing a circuit C' in the C-hybrid model involves the parties sending normal messages to each other (as in the real model) and in addition, having access to a trusted party computing C. It is essential that the invocations of C are done sequentially, meaning that before an invocation of C begins, the preceding invocation must

finish. In particular, there is at most a single call to C per round, and no other messages are sent during any round in which C is called.

Let type be an ideal world. Let $\mathcal{A}$ be a non-uniform QPT machine with auxiliary input ρ_{aux} controlling a subset $\mathcal{I} \subset \mathcal{P}$ of the parties. We denote by $\text{HYBRID}^{C,\mathsf{type}}_{\pi,\mathcal{A}(\rho_{\mathsf{aux}})}(\kappa, \rho_1, \ldots, \rho_n)$ the joint output of the adversary and of the honest parties, following an execution of π with ideal calls to a trusted party computing C according to the ideal model "type," on inputs $\rho_1, \ldots, \rho_n$, auxiliary input ρ_{aux} given to $\mathcal{A}$, and security parameter κ. We call this the (C, type)-hybrid model. When type is clear from context we remove it for brevity.

The Classical MPC Hybrid Model

Following [DNS12, DGJ+20], throughout the paper, we assume the availability of a *trusted party*, denoted cMPC, that is able to compute any efficiently computable *classical* multiparty functionality. Furthermore, we assume cMPC is a *reactive* functionality, i.e., it is allowed to have an internal state that may be taken into account the next time it is invoked. One particular classical functionality we employ is the broadcast functionality. Thus, we implicitly assume that each party can broadcast a classical message at any given round of the protocol.

Similarly to [DGJ+20], we can implement cMPC using a post-quantum secure protocol. Specifically, we first remove the assumption that cMPC is reactive via standard techniques. To maintain security-with-identifiable-abort, this is done as follows. At the end of each call to cMPC, its state s will be shared in an additive n-out-of-n secret sharing scheme. Let s_i denote the i^{th} share. The functionality then uses a post-quantum secure signature scheme to sign each share. Let σ_i denote the signature of s_i. The output of P_i will now additionally include s_i, σ_i, and the verification key of the signature scheme (which is the same for all parties). Note that the parties do not keep the signing key. In the next call to cMPC, the parties will additionally send their signed shares and keys to cMPC. If the keys are not all equal, then cMPC sends to party P_i the output $(\texttt{abort}, \mathrm{P})$, where P is the lexicographically smallest party whose key differs from the key of P_i. Otherwise, if all the keys are the same, cMPC verifies all shares, sending the identity of a party whose verification failed if such a party exists, and reconstruct the state s and continue with the computation otherwise. Note that since the honest parties forward the output they received from the previous call, in case of abort they all agree on the identity of a corrupted party.

Finally, we can implement each call to cMPC assuming a correlated randomness setup, using the information theoretic UC-secure protocol of [IOZ14] and apply [Unr10]'s lifting theorem, to obtain post-quantum security. Furthermore, pre-computing the randomness in an off-line phase yields a protocol in the pre-processing model [DPSZ12]. Such protocols have an off-line phase which admits computational security, however, assuming no attack was successful during this phase, their online-phase admit information theoretic security.

Furthermore, for the sake of presentation, we sometimes abuse the existence of cMPC, and construct some of the ideal worlds with the ability to interact with it. Although this cannot happen in the standalone model, such an assumption

can be removed using the techniques described above, i.e., each party will hold a signed share of cMPC's input and receive a signed share of its output.

We denote by $\text{HYBRID}^{\mathsf{cMPC}}_{\pi,\mathcal{A}(\rho_{\mathsf{aux}})}(\kappa,(\rho_i)_{i=1}^n)$ the joint output of the adversary $\mathcal{A}$, cMPC, and of the honest parties in a random execution of π in the cMPC-hybrid model on security parameter $\kappa \in \mathbb{N}$, inputs $\rho_1,\ldots,\rho_n$, and an auxiliary input ρ_{aux}.

4 Statement of Our Main Result

In this section we present the main theorem of the paper, namely that any multiparty quantum functionality can be computed with security-with-identifiable-abort against any number of corrupted parties.

Theorem 2. *Assume the existence of a classical quantum-resistant fully homomorphic encryption scheme with decryption circuit of logarithmic depth. Let C be an n-ary quantum circuit. Then C can be computed with computational security-with-identifiable-abort in the cMPC-hybrid model. Moreover, the round complexity of the protocol is independent of the circuit depth.*

Toward proving Theorem 2 we first show how to reduce the problem to the problem of constructing an IDPD-secure protocol for a related circuit. The following lemma states the existence of such an IDPD-secure protocol.

Lemma 1. *Assume the existence of a classical quantum-resistant fully homomorphic encryption scheme with decryption circuit of logarithmic depth. Let C be an n-ary quantum circuit. Then C can be computed with computational $(n^2,2n^2)$-IDPD security in the cMPC-hybrid model. Moreover, the round complexity of the protocol is independent of the circuit depth.*

The proof of Lemma 1 is given in Section 10. Toward proving it, in the following sections we construct several building blocks used in the construction of the final protocol. We now use it to prove Theorem 2. It suffices to prove the following claim, asserting that security-with-identifiable-abort can be reduced to IDPD-security.

Claim 3 *Let C be an n-ary quantum circuit and let $d_{\mathsf{in}} = d_{\mathsf{in}}(\kappa)$ and $d_{\mathsf{out}} = d_{\mathsf{out}}(\kappa)$ be two polynomials. Additionally, let QECC denote a quantum error-correcting code that can tolerate $\max\{d_{\mathsf{in}},d_{\mathsf{out}}\}$ errors. Then C can be computed with perfect security-with-identifiable-abort in the $(C',(d_{\mathsf{in}},d_{\mathsf{out}})\text{-}\mathsf{IDPD})$-hybrid model, where*

$$C' = \mathsf{QECC.Enc}^{\otimes n\ell_{\mathsf{out}}} \circ C \circ \mathsf{QECC.Dec}^{\otimes n\ell_{\mathsf{in}}}.$$

That is, C' transversely decodes each of its inputs using the QECC, computes C, and then re-encode each output.

Due to space considerations, the formal proof is deferred to Section 4 of the full version [ACC+20].

5 Sequential Authentication

In this section, we present a protocol, called *sequential authentication* (SA), that allows a party – called the sender – to send an encryption of its input along a predetermined path known to everyone, to a designated party called the receiver (not necessarily different from the sender). The security achieved by this protocol roughly guarantees that in case the protocol is aborted, the parties will identify two parties, one of which is guaranteed to be corrupted. Later, in Section 6, using sequential calls to SA we show how to use it in order to augment the security to obtain an IDPD-secure protocol.

Let us first formally define the ideal world of SA. To simplify the presentation, we assume that cMPC is an additional party that will receive an output. Additionally, the parties have two common inputs, in addition to the security parameter. These are a path $\texttt{PATH}$ and number of traps t. The domain of $\texttt{PATH}$ is the set of all (non-simple) paths that goes through all parties, whose length is exactly[11] $\ell := n^2$. To remove confusion with the parties themselves, we call the parties along the path *relays* and denote by Q_i the i^{th} party along the path (note that a single party may be multiple relays on the path). Furthermore, we call Q_1 the sender, and call Q_ℓ the receiver.

Ideal world of sequential authentication

Inputs: Each party P_i and cMPC holds the security parameter 1^κ, a path description $\texttt{PATH} = (\mathsf{Q}_1, \cdots, \mathsf{Q}_\ell)$ that goes through every party at least once, and the number of traps $t = t(\kappa)$ required for the output ciphertext. The sender Q_1 holds an m-qubit input state ρ. The adversary $\mathcal{A}$ is given an auxiliary quantum state ρ_{aux}.

The sender sends input: If $\mathsf{Q}_1 \notin \mathcal{I}$, then it sends ρ as its input to T. Otherwise, the adversary chooses an input ρ^* to be given to T. Let ρ' be the input received by the trusted party.

The trusted party encodes the state and sends $\mathcal{A}$ its output: T samples a Clifford $E \leftarrow \mathcal{C}_{m+t}$ and encode ρ' to obtain $\sigma \leftarrow \mathsf{CAuth.Enc}_E(\rho')$. If $\mathsf{Q}_\ell \in \mathcal{I}$, then T sends σ to $\mathcal{A}$.

The adversary instructs trusted party to continue or to abort: The adversary $\mathcal{A}$ sends to T either $\texttt{continue}$ or $(\texttt{abort}, \mathsf{Q}_i, \mathsf{Q}_{i+1})$ where $1 \leq i < \ell$ and where either Q_i or Q_{i+1} is corrupted. If $\mathcal{A}$ sent $\texttt{continue}$, then T sends E to cMPC. Additionally, if the receiver $\mathsf{Q}_\ell \notin \mathcal{I}$ is honest, then T sends it σ. Otherwise, if $\mathcal{A}$ sends $(\texttt{abort}, \mathsf{Q}_i, \mathsf{Q}_{i+1})$, then T forwards it to the cMPC and all honest parties.

Outputs: The honest parties output whatever they received from T, the corrupted parties in $\mathcal{I}$ output nothing, and the adversary outputs some function of its view.

[11] The reason for the fixed length is due to a technicality that follows from the way SA is used.

We denote by $\mathsf{SA}_{\mathcal{A}(\rho_{\mathsf{aux}})}(\kappa, \mathtt{PATH}, t, \rho)$ the joint output of $\mathcal{A}$, the honest parties, and cMPC, in an execution of the above ideal world, on security parameter κ, input ρ, auxiliary input ρ_{aux}, path $\mathtt{PATH}$, and the number of traps t.

As mentioned in Section 1.2, our construction is similar to swaddling from [DNS12] and the public authentication test from [DGJ+20], both of which are based on Clifford code. Due to space limitations the construction alongside its proof of security is deferred to Section 5 of the full version [ACC+20].

We also make use of a variant of SA, where the input – instead of an arbitrary state – is encrypted under Clifford encryption, with the key being held by cMPC. We call this variant *input-ciphertext* SA (CTSA). The protocol follows the same lines as the protocol for computing SA and is presented in Section 5.2 of the full version [ACC+20]. We stress that unlike the protocol for SA, this protocol is *not* secure and will only be used as a subroutine in other protocols.

6 Authenticated Routing

In this section, we present a protocol, called *authenticated routing* (AR), that allows the parties to securely send an encryption of their inputs to a designated party. The security achieved by this protocol is *IDPD-security* (i.e., security-with-identifiable-abort-and-packet-drops), as was defined in Section 3.1. We extensively use AR as a building block in order to construct a secure-with-identifiable-abort protocol for a general circuit.

We next define the AR functionality. For a polynomial $t = t(\kappa) \geq \kappa$, representing *trap-size*, denote by $\mathsf{AR} = \mathsf{AR}_t$ the following mapping. Each party P_i holds ℓ_{in} packets $\rho_i = (\rho_{ij})_{j=1}^{\ell_{\mathsf{in}}}$, where each $\rho_{ij} \in \mathcal{D}^m$. An output is given only to P_1 – called the receiver – and to cMPC. Specifically, cMPC receives a collection of Cliffords $(E_{ij})_{i\in[n],j\in[\ell_{\mathsf{in}}]}$, where $E_{ij} \leftarrow \mathcal{C}_{m+t}$ are sampled independently and uniformly at random, and the receiver P_1 receives the Clifford encoding of each packet ρ_{ij} under E_{ij}; that is, P_1 receives $(\mathsf{CAuth.Enc}_{E_{ij}}(\rho_{ij}))_{i\in[n],j\in[\ell_{\mathsf{in}}]}$. In the following section we present a protocol that computes AR with $(n^2, 0)$-IDPD-security in the cMPC-hybrid model, i.e., at most n^2 input-packets can be dropped by the adversary while no output-packets can be dropped.

6.1 The Authenticated Routing Protocol

In this section, we present our protocol for authenticated routing. Roughly, the idea is as follows. Throughout the entire interaction, we let cMPC maintain a graph G that represents *trustfulness*. In more details, each vertex in G corresponds to a party and the graph is initialized as the complete graph. Then, whenever a party accuses another party, the corresponding edge will be removed from G.[12] Following the initialization, each party P_i tries to send its packets (ρ_{ij}), one by one, to P_1 along a path that goes through all parties. Such a path can be computed, and thus agreed upon, by having cMPC repeatedly applying BFS/DFS to find a path from P_i to P_2, then to P_3 until it reaches the last party

[12] We note that this technique, of using the graph to allow honest parties to unanimously agree on the identity of a corrupted party, was independently used in another recent paper by [BMMMQ20].

P_n, from which it finds a path to P_1. The parties will send the packets along the path using the SA functionality.[13] If SA aborted, then the parties now hold two identities P_a and P_b given to them by SA, one of which is guaranteed to be corrupted. cMPC will then remove the edge ab from the graph G. Now, party P_i will try to send the rest of its packets using a *different* route that does not pass through the edge ab. The parties continue in this fashion until either most qubits were sent successfully, or until G becomes disconnected, in which case all honest parties are in the same connected component. Therefore, they can agree to identify a party *not* connected to them as malicious.

Let us now consider the case where a call SA ended in abort. Here, a single packet had been dropped, and so by the ideal-world definition of IDPD-security, the parties must agree to replace this packet with the 0 state. To do this, the parties will call SA again with the *empty state* $\varepsilon \in \mathcal{D}^0$ and $m + \kappa$ traps. To see why this work, notice that the Clifford encoding of the empty state with $m+\kappa$ traps, is equivalent to a Clifford encoding of $|0^m\rangle$ with κ traps. Moreover, by the security of Clifford encoding, if the adversary changes the traps from 0 then SA will abort again, which will remove another edge from the graph. Thus, this can be done repeatedly until either G becomes disconnected or the parties successfully encode the 0 state.

The IDPD-security of the protocol described so far can still be breached by a malicious adversary, due to the following difficulty one would encounter while constructing a simulator. Suppose that the adversary corrupted the receiver P_1, and consider a call to SA, for a packet $(1, j)$ for some $j \in [\ell_{\mathsf{in}}]$, i.e., the receiver sends to itself an encoding of the packet. To generate the corresponding transcript, the simulator in the ideal-world must query the adversary for its input ρ to the SA functionality and must send to $\mathcal{A}$ an output in return. Observe that $\mathcal{A}$ expects to receive $\sigma = \mathsf{CAuth.Enc}_E(\rho)$ where the key E is held by the cMPC. Since the simulator does not know the key E, it cannot generate a-priori a value σ that is consistent with the output of cMPC. On the other hand, the simulator may not be ready to send inputs to the trusted party either, as it must hold *all* input packets from the adversary. Since rewinding the adversary can help the environment to distinguish between the real world and ideal world, it seems to be the case where there are no good ways to generate this output of SA. One possible solution is to modify the AR ideal functionality so it will immediately encode and output each received packet before receiving the next packet. Unfortunately, the resulting ideal functionality would be too weak for our purposes later.

It is possible to overcome this challenge by tweaking the protocol. A simple solution to this issue would be to have the receiver, after it has received all of the packets, to send them to itself again using CTSA (that is, the ciphertext-input version of SA). With this modification, the simulator can send authentication of 0s to the receiver as the outputs of SA. To see why this works, recall that Clifford authentication ciphertexts are identical to maximally mixed states due

[13] Recall that SA requires the path to be of length n^2. Note that the way cMPC computes the path always generates a path of length at most n^2. If the path is shorter, then cMPC can just add the last party repeatedly.

to Clifford twirling. Thus, the simulator can then collect all input packets and interact with the trusted party to receive the correct outputs. When the receiver sends the dummy ciphertexts to itself, the simulator collects and verifies them, before replacing them with the correct outputs.

Although this approach works, as CTSA is not secure as a standalone protocol, it hard to formally argue the security of the above protocol. Instead, we slightly modify the protocol and also construct a slightly different simulator. The idea is to have the simulator send halves of EPR pairs as the output of SA, instead of authentications of 0s. Since halves of EPR pairs are indistinguishable from Clifford authentication ciphertexts, the adversary will reply with the same messages in both the real and the ideal world. After the simulator collects all packets and interact with the trusted party, it then replaces these halves of EPR pairs with the correct output via quantum teleportation. To complete the teleportation, the simulator must send a Pauli operator for the adversary to apply. Thus, we correspondingly add a key-update step at the end of the protocol to provide an opportunity for this.

Protocol 1 Authenticated Routing protocol π_{AR}

Inputs: Each party P_i holds private input $\rho_i = (\rho_{ij})_{j=1}^{\ell_{\mathsf{in}}}$ where $\rho_{ij} \in \mathcal{D}^m$.
Common input: The security parameter 1^κ and the packet-size m.

1. cMPC initializes G as the complete graph with n vertices where each vertex represents a party, and initializes a set $\mathcal{R}_{\mathsf{in}} = \emptyset$, which will keep track of all packets that were sent successfully.
2. For each packet $(i, j) \in [n] \times [\ell_{\mathsf{in}}]$:
   (a) cMPC computes a path $\mathtt{PATH}_{ij}$ in G from P_i to P_1 that goes through all parties of size exactly n^2, and send it to all parties.
   (b) The parties call SA with P_i's input being ρ_{ij}, with κ as the common trap-size, and $\mathtt{PATH}$ as the common path.
      – If SA outputs $(\mathtt{abort}, \mathrm{P}_a, \mathrm{P}_b)$ for some $a, b \in [n]$, then cMPC removes the edge ab from the graph G. If G becomes disconnected, then cMPC sends ab to all parties. Each party then outputs $(\mathtt{abort}, \mathrm{P})$, where $\mathrm{P} \in \{\mathrm{P}_a, \mathrm{P}_b\}$ is the party on the edge *not* connected to it on the graph, and halts.
      – Otherwise SA terminates successfully, sending a uniform random Clifford $F_{ij} \leftarrow \mathcal{C}_{m+\kappa}$ to cMPC, sending σ_{ij} to P_1, where $\sigma_{ij} = \mathsf{CAuth.Enc}_{F_{ij}}(\rho_{ij})$, and sending $\mathtt{continue}$ to all other parties. In this case, cMPC adds (i, j) to $\mathcal{R}_{\mathsf{in}}$.
3. For each dropped packet $(i, j) \in ([n] \times [\ell_{\mathsf{in}}]) \setminus \mathcal{R}_{\mathsf{in}}$:
   (a) cMPC finds a path $\mathtt{PATH}'_{ij}$ in G from P_1 to itself that passes through every party and send it to all parties.
   (b) The parties call SA, trap-size $m + \kappa$, no private inputs, and $\mathtt{PATH}'_{ij}$ as the common input.
      – If SA outputs $(\mathtt{abort}, \mathrm{P}_a, \mathrm{P}_b)$ for some $a, b \in [n]$, then, similarly to Step 2b, cMPC removes the edge ab from the graph G. If G becomes disconnected, then cMPC sends ab to all parties. Each party P then

outputs $(\texttt{abort}, \mathrm{P}')$, where $\mathrm{P}' \in \{\mathrm{P}_a, \mathrm{P}_b\}$ is the party on the edge *not* connected to P, and halt. Otherwise, if the graph is still connected, then the parties go back to Step 3a.
  - Otherwise SA terminates successfully, sending a uniform random Clifford $F_{ij} \leftarrow \mathcal{C}_{m+\kappa}$ to cMPC, sending σ_{ij} to P_1, where $\sigma_{ij} \leftarrow \mathsf{CAuth.Enc}_{F_{ij}}(\varepsilon)$, and sending $\texttt{continue}$ to all other parties.
4. For all packets $(i, j) \in [n] \times [\ell_{\mathsf{in}}]$:
   (a) cMPC samples a Pauli independently and uniformly at random $P_{ij} \leftarrow \mathcal{P}_{m+\kappa}$ and sends it to P_1.
   (b) P_1 applies P_{ij} to σ_{ij}, obtaining τ_{ij}.
   (c) cMPC updates its Clifford key to be $E_{ij} = P_{ij}F_{ij}$.
5. cMPC sends $\mathcal{R}_{\mathsf{in}}$ to all parties.
6. Each party outputs $(\texttt{continue}, \mathcal{R}_{\mathsf{in}})$, cMPC additionally outputs the Cliffords $\{E_{ij}\}_{ij \in S_{\mathsf{input}}}$, and P_1 additionally outputs $\{\tau_{ij}\}_{ij \in S_{\mathsf{input}}}$.

We next state the security of the protocol.

Lemma 2. *Protocol π_{AR} computes the functionality AR with perfect $(n^2, 0)$-IDPD-security in the $\{\mathsf{cMPC}, \mathsf{SA}\}$-hybrid model.*

The proof of security is deferred to Section 6 of the full version [ACC+20]. We also make use of the *input-ciphertext* variant of AR, denoted CTAR. Similarly to CTSA, this is also insecure in general, so we only describe the protocol. Here, unlike in AR, there is only a single sender, denoted P_1, and multiple receivers. The goal of this variant is to send each *plaintext* from the P_1 to its designated receiver, encrypted under a *new* key. The protocol follows similar ideas to π_{AR} and is given in Section 6.2 of the full version [ACC+20]. We let π_{CTAR} be the protocol for computing the input-ciphertext variant of the AR functionality.

7 Magic State Preparation

Recall that we aim to have a single designated party to homomorphically evaluate a universal circuit over encrypted values. Towards achieving this, the parties require five kinds of magic states. These include ancilla zero states, P magic states, T magic states, H magic state and gadgets γ. In all magic state preparation protocols, an output will be given to only two parties: the server P_1 and cMPC. Furthermore, since the preparation is independent of the parties' inputs, these can be prepared in advance in an offline phase (in fact, the parties only require to know an upper bound on the number of gates in the circuit).

For space considerations, we will only present a rough overview of the construction of the protocol for preparing T magic states. The rest of the magic states can be prepared by using simpler protocols. These protocols can be found in the full version [ACC+20]. To simplify the presentation, the protocol and the functionality will prepare a single magic state. This, however, can be easily generalized to create more magic T states using the same number of rounds (see the full version for more details).

We next define the functionality MSP_T for preparing a single T magic state, $|T\rangle := T|+\rangle$. The other functionalities are denoted $\mathsf{MSP}_{\mathsf{ms}}$, where $\mathsf{ms} \in \{Z, P, H, \gamma\}$, are defined similarly (here ms represents either the ancilla zero state, $|P\rangle := P|+\rangle$ state, $|H\rangle := (H \otimes I)|\Phi^+\rangle$ state, or a gadget state $|\gamma\rangle$ defined in Section 7.3 of the full version [ACC+20] , respectively). MSP_T is a no-input functionality whose output is defined as follows. Let $E \leftarrow \mathcal{C}_{3\kappa}$ be a uniform random Clifford. Then the mapping outputs to P_1 a Clifford encryption of the T magic state $\mathsf{CAuth.Enc}_E(|T\rangle\langle T|)$ and outputs to cMPC the corresponding key E.

7.1 T Magic State Preparation Protocol

Our protocol is a modification of the protocol of [DGJ+20] for preparing T magic states, which achieves only security-with-abort. Let us first give an overview of the protocol of [DGJ+20]. First, recall the result of magic state distillation [BK05] (or see the discussion in Section 2.4 of the full version [ACC+20]), that given $\mathrm{poly}(\log(1/\delta_0))$ copies of noisy T magic states with a constant fraction error, using the T distillation circuit we can obtain δ_0-close $|T\rangle$ state. Now, in [DGJ+20], the server P_1 prepares κn copies of $|T\rangle$. Then, the parties execute the *secure-with-abort* input-encoding protocol of [DGJ+20], which generates a Clifford encoding of each $|T\rangle$ state, i.e., $(\mathsf{CAuth.Enc}_{E_j}(|T\rangle\langle T|))_{j=1}^{\kappa n}$, and outputs to cMPC the corresponding Clifford keys. Following this, cMPC samples disjoint sets $S_1, \cdots, S_n \subseteq [\kappa n]$, each of size κ, uniformly at random, and sends them to all parties. For each subset S_i, the server P_1 sends $\{\mathsf{CAuth.Enc}_{E_j}(|T\rangle\langle T|)\}_{j\in S_i}$ to P_i and cMPC sends $\{E_j\}_{j\in S_i}$ to P_i. Then, party P_i decrypts and measures the received states in the $\{|T\rangle, |T^\perp\rangle\}$-basis, and broadcast an `abort` if it gets $|T^\perp\rangle$ in any of the measurements. We refer to this step as the *random sampling test*. Upon receiving an `abort` from a party, all parties abort and halt. Otherwise, [DGJ+20] showed that the remaining copies the server holds differ by a constant fraction from $\{\mathsf{CAuth.Enc}_{E_j}(|T\rangle\langle T|)\}_{j\in S_1}$ with respect to the trace distance. The server can then get a state of negligible trace distance from $\mathsf{CAuth.Enc}_E(|T\rangle\langle T|)$ with respect to a new key E, by running the T distillation circuit (guaranteed to exist by Theorem 2.8 in the full version [ACC+20]).

Clearly, the above protocol does not admits security-with-identifiable-abort. Indeed, not only is the input-encoding protocol of [DGJ+20] does not admit security-with-identifiable-abort, it further holds that a corrupted party may accuse an honest server by lying about the states that it measured. To overcome this two issues, we perform the following modifications to the protocol of [DGJ+20].

First, we make the following observations. Regardless of what the server prepares, assuming all states were successfully sent during the random sampling test, with overwhelming probability all honest parties will have roughly the same number of $|T^\perp\rangle$ states upon measurement. Therefore, instead of broadcasting `abort`, we consider each party's *error rate*, defined to be number of $|T^\perp\rangle$ it holds divided by κ. These will be sent to cMPC, who compares them. If there are two parties whose error rates are significantly far apart, then cMPC can publish them, and the honest parties can agree on which is corrupted.

Second, note that the previous observation holds only if the states were faithfully distributed by during the random sampling test. Indeed, the corrupted parties can bias the error rates by dropping the packets. Consider the following example. Suppose that P_1 is corrupted and $\mathrm{P}_2, \ldots, \mathrm{P}_n$ are honest. The server P_1 prepares each state to be $|T\rangle$ with probability $1/2$ and $|T^{\perp}\rangle$ with probability $1/2$. Then, after cMPC sends to P_1 the set S_2, the adversary drops the all $|T\rangle$ states that belong to S_2. As for the the states that belong to S_3, the adversary will drop the $|T^{\perp}\rangle$ states. Consequently, the error rate ϵ_2 will be much higher than $1/2$ while ϵ_3 will be much lower than $1/2$. To solve this issue, we would like that for any state that was dropped, the error rate will not include it. We achieve this as follows. The server first prepares $N_0 := 3n^2 + 1$ copies of $|T\rangle\langle T|^{\otimes \kappa n}$. Then, for each copy of $|T\rangle\langle T|^{\otimes \kappa n}$ a random sampling test is applied, where each party receives κ of the qubits among $|T\rangle\langle T|^{\otimes \kappa n}$. Moreover, the qubits will be transmitted to their destination using the AR functionality, to ensure that the adversary cannot drop too many qubits. If there is a state that was lost during the transmission, all parties start the random sampling test for another copy of $|T\rangle\langle T|^{\otimes \kappa n}$.

Third, the parties need to be able to sample a subset of the qubits held by P_1, even if $|T\rangle\langle T|^{\otimes \kappa n}$ are encrypted under a Clifford code. The idea is to have cMPC update the encryption keys so that they will randomly permute the qubits and re-encrypt using a new key of the form $(E_1 \otimes \cdots \otimes E_n)$, where each $E_i \leftarrow \mathcal{C}_{\kappa+t}$, where t is the number of traps. Additionally, the permutation must ensure that each κ of the T states have t trap states $|0\rangle$ appended to. Observe that decrypting using such key would result in n pieces of T states, each of size κ and encrypted under a different key with t traps. The parties can then use input-ciphertext authenticated routing (formally defined as Protocol 5 in the full version [ACC+20]) to distribute the states.

Finally, we encounter the following difficulty when constructing the simulator. The T distillation circuit is not deterministic in the sense that which gates should be applied depend on previous measurement outcomes. To simplify the security proof, after performing the T distillation, the parties execute π_{CTAR} to have the server P_1 route all ciphertexts generated by the T distillation circuit *to itself*. Now, similarly to π_{AR}, we let cMPC updating its key using a random Pauli. Then, similarly to the simulation for AR, the simulator can send halves of EPR pairs as the output during the execution of π_{CTAR}. Among the remaining packets after π_{CTAR}, the simulator can teleport the output it got from the trusted party to P_1.

The formal description of the protocol alongside its proof of security are deferred to Section 7.4 of the full version [ACC+20].

8 Secure Delegation of The Computation – Preparation

In this section, we present a protocol, which we call *pre-computation*, that allows the parties to securely delegate the computation to a designated party, called the server. More concretely, at the end of the protocol, the server will hold an encryption of each of the parties' inputs, while cMPC will hold the keys used for the encryption. Later, in Section 9, we show how the server and

cMPC can homomorphically evaluate a quantum circuit over the encrypted inputs. Thus, we require that the encryption used by the parties to be the TrapTP VQFHE scheme of [ADSS17]. Recall that a VQFHE is four-tuple of QPT algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$, that generate keys, encrypt plaintext, evaluate a circuit of an encrypted message, and decrypt a ciphertext while verifying the evaluation was performed honestly. Our pre-computation protocol can thus be viewed as a way to implement TrapTP.KeyGen and TrapTP.Enc – the key generation and encryption algorithms of TrapTP, respectively – in a distributed manner. We next present a formal definition of the functionality we wish to compute.

Let $L = L(\kappa)$ be a polynomial (this will later represent an upper-bound on the size of a circuit to be evaluated and the number of ancilla 0 states it requires). Denote by $\mathsf{PreComp} = \mathsf{PreComp}_L$ the following mapping. Party P_i holds an ℓ_{in}-qubit input $\rho_i = (\rho_{ij})_{j=1}^{\ell_{\mathsf{in}}}$, where $\ell_{\mathsf{in}} = \ell_{\mathsf{in}}(\kappa) \in \mathbb{N}$ is some polynomial. Only the server P_1 and cMPC are given outputs, defined as follows. Let $(\mathsf{sk}, \rho_{\mathsf{evk}}) \leftarrow \mathsf{TrapTP.KeyGen}(1^\kappa, 1^L)$ (recall that $\mathsf{sk} \in \{0,1\}^*$ is a classical string and ρ_{evk} is a quantum state). Then cMPC receives sk and P_1 receives the encryptions of each input-qubit, encryptions of 0 states, and the evaluation key ρ_{evk}, i.e., it receives $(\hat{\rho}, \hat{\rho}_0, \rho_{\mathsf{evk}})$ where $\hat{\rho} = (\mathsf{TrapTP.Enc}_{\mathsf{sk}}(\rho_{ij}))_{(i,j)\in[n]\times[\ell_{\mathsf{in}}]}$, and $\hat{\rho}_0 = (\mathsf{TrapTP.Enc}_{\mathsf{sk}}(|0\rangle\langle 0|))_{i=1}^{L}$.

We now present a rough overview of our protocol for computing PreComp with $(n^2, 0)$-IDPD-security that outputs ciphertexts of size 3κ. The formal description of the protocol alongside its proof of security are deferred to Section 8 of the full version [ACC+20]. Conceptually, the protocol consists of three main steps. First, the private inputs of each party are routed to P_1 by a call to AR. This results in P_1 holding the Clifford ciphertexts of all private inputs, and cMPC holding the keys. Second, the parties call the magic state preparation functionalities. This include MSP_Z that prepares ancilla 0 states, MSP_P that prepares magic P states, MSP_H that prepares magic H states, MSP_T that prepares magic T states, and MSP_γ that prepares gadget states. At the end of the call, the server P_1 holds the Clifford ciphertexts of all these magic states, while cMPC holds the Clifford keys. Finally, as we use the TrapTP scheme, the homomorphic evaluation can only be applied to trap-code ciphertexts. Thus, we show how the server and cMPC can *re-encrypt* all Clifford ciphertexts to trap-code ciphertext.

9 Secure Delegation of The Computation – Computation

In the previous section we introduced the PreComp protocol for generating a key of TrapTP and outputs to the server P_1 encryptions of all inputs. In this section, we present a protocol that securely implements evaluation and verified decryption. This in turn, allows P_1 to perform the circuit evaluation, while ensuring to the other parties that the evaluation was done correctly. Looking ahead, as the server would need to distribute the outputs, unlike in PreComp, the resulting encrypted value held by the server would be under Clifford code. In Section 10 below, we will show how to securely distribute these Clifford ciphertexts.

Formally, let C be an n-ary circuit. Define the functionality Comp as follows. Let ℓ_{in} and ℓ_{out} be the number of input-qubits and output-qubits, respectively,

of each party P_i. Denote the input of P_i as ρ_i. An output is given only to the server P_1 and cMPC, as follows. cMPC receives a uniform random Clifford for each output-qubit, namely it receives $(E_{ij})_{(i,j)\in[n]\times[\ell_{\mathsf{out}}]}$ where $E_{ij} \leftarrow \mathcal{C}_{1+n^2\kappa}$ are sampled independently and uniformly at random. The server P_1 receives the Clifford encryptions of each of the output-qubits encrypted with the Cliffords given to cMPC. That is, P_1 receives $\mathsf{CAuth.Enc}_{E_{ij}}(\sigma_{ij})$, where for all $i \in [n]$ and $j \in [\ell_{\mathsf{out}}]$ it holds that $\sigma_{ij} \in \mathcal{D}^1$ is a single qubit, and these are defined as $(\sigma_{ij})_{(i,j)\in[n]\times[\ell_{\mathsf{out}}]} = C(\rho_1, \ldots, \rho_n)$.

Roughly, our protocol works as follows. First, the parties prepare the values required to run $\mathsf{TrapTP.Eval}$. That is, they generate keys and ciphertexts of their inputs under TrapTP using $\mathsf{PreComp}$. The server P_1 can now run $\mathsf{TrapTP.Eval}$ on the ciphertexts to homomorphically evaluate the circuit, obtaining the outcome $\hat{\sigma}_{ij}$ for every party i and qubit j (possibly after some qubits where dropped). As $\hat{\sigma}_{ij}$ is encrypted using TrapTP, the parties now need to re-encrypt it to a Clifford ciphertext. To do this, the parties call AR, to have the server route the encrypted results *to itself*. This results in a Clifford ciphertext τ_{ij} of a trap code ciphertext of the outputs.

We now explain how cMPC can verify the computation. Let us first recall two important properties of $\mathsf{TrapTP.Eval}$ and $\mathsf{TrapTP.VerDec}$. Recall that $\mathsf{TrapTP.Eval}$, in addition to performing a computation over trap codes, if further produces a log of the computation, that includes all the classical messages including randomness, computation steps, and all intermediate results during evaluation. Next recall, that although $\mathsf{TrapTP.VerDec}$ is a quantum procedure, it includes a classical subroutine that verifies these logs. We denote this subroutine by $\mathsf{CheckLogs}$. It is given a secret key sk (generated from $\mathsf{TrapTP.KeyGen}$ used in $\mathsf{PreComp}$) and a log to be checked, and outputs updated Pauli keys and a flag to indicate whether the computation was performed faithfully. We refer the reader to [ADSS17] for a detailed construction of the classical algorithm.

Now, P_1 sends log to cMPC who applies $\mathsf{CheckLogs}$ to check validity. Then, the server split the trap registers into Z_1 and Z_2 for each ciphertext τ_{ij}, and cMPC sends it the Clifford

$$V_{ij} = (E_{ij}^{MZ_2} \otimes R_{ij}^{S} \otimes \mathsf{TAuth.PEnc}_{\Pi_{ij},Q_{ij}}^{TZ_1})\mathsf{TAuth.PDec}_{\Pi_0,P_{ij}}^{MST} F_{ij}^{\dagger}.$$

Here, the first term $F_{ij}^{\dagger}$ would remove the Clifford encryption added by AR, resulting in a trap code ciphertext. Then the partial decryption of the trap code $\mathsf{TAuth.PDec}$ is applied (see Definition 2), using the global permutation Π_0, and the Pauli P_{ij} it got from checking the logs using $\mathsf{CheckLogs}$. This converts the trap code ciphertext into a plaintext in register M, traps to be verified in register T, and the syndrome in register S (recall that trap codes use QECC in their construction). The term E_{ij} is a new Clifford that re-encrypts the plaintexts under a Clifford code, the term R_{ij} is a random Pauli that overwrites the syndromes to prevent leak of information, and $\mathsf{TAuth.PEnc}_{\Pi_{ij},Q_{ij}}$ perform partial encryption of the trap code (see Definition 3), to re-encrypt the traps in register T under trap code with a newly sampled key. The server is then asked to homomorphically measure these traps and send the measurement results to cMPC to verify,

who aborts if the verification failed. Specifically, the verification compares the measured traps to 0's. If the protocol does not abort, the server outputs the computation results that is now under a Clifford code, and cMPC outputs the corresponding keys.

The proof of security is done by reducing it to the security of the TrapTP scheme. The formal description of the protocol alongside its proof of security are deferred to Section 9 of the full version [ACC+20].

10 Secure Computation of a Quantum Circuit With Packet Drops

In this section, we are finally ready to present our protocol for computing an arbitrary quantum circuit C with IDPD-security. That is, we prove Lemma 1. We do so by constructing a protocol in the $\{\mathsf{cMPC}, \mathsf{Comp}\}$-hybrid model. Given the functionality Comp from the previous section, the protocol is rather simple. The parties first call Comp, which ensures that P_1 will hold a Clifford encoding of the output of each party, and cMPC will hold the keys. The parties then execute π_{CTAR} to route each output held by P_1 to the correct party. Finally, if the protocol did not yet abort, then cMPC will send the keys to each output to the corresponding party.

The formal description of the protocol and its proof of security are deferred to Section 10 of the full version [ACC+20].

Remark 1. Interestingly, assuming that cMPC sends all Clifford keys to the parties *simultaneously*, it follows that identifiable fair[14] classical MPC is sufficient for identifiable fair quantum MPC (see Appendix A of the full version [ACC+20] for a formal definition). Indeed, observe that until cMPC sends the Clifford keys, all quantum states held by each party is encrypted. Therefore, an adversary that causes the protocol to abort gains no information on the output. Thus, if the last call to cMPC is fair, then all parties will receive the keys to their respective encrypted output simultaneously. This results in a fair MPQC protocol in the cMPC-hybrid model.

Acknowledgements

The authors would like to thank the anonymous reviewers for their useful comments and suggestions, and in particular for pointing out the existence of NC^1 decryption of classical fully homomorphic encryption schemes [BV11]. We would also like to thank Eran Omri for many useful conversations.

[14] In addition to fairness, identifiable fair computation have the added property that in case the protocol aborts, the honest parties agree on the identity of at least one corrupted party.

Bibliography

[ABDR04] Andris Ambainis, Harry Buhrman, Yevgeniy Dodis, and Hein Rohrig. Multiparty quantum coin flipping. In *Proceedings. 19th IEEE Annual Conference on Computational Complexity, 2004.*, pages 250–259. IEEE, 2004.

[ABOE10] Dorit Aharonov, Michael Ben-Or, and Elad Eban. Interactive proofs for quantum computations. In Andrew Chi-Chih Yao, editor, *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*, pages 453–469. Tsinghua University Press, 01 2010.

[ACC+20] Bar Alon, Hao Chung, Kai-Min Chung, Mi-Ying Huang, Yi Lee, and Yu-Ching Shen. Round efficient secure multiparty quantum computation with identifiable abort. Cryptology ePrint Archive, Report 2020/1464, 2020. https://eprint.iacr.org/2020/1464.

[ADSS17] G. Alagic, Y. Dulek, C. Schaffner, and F. Speelman. Quantum fully homomorphic encryption with verification. 2017.

[BCG+02] H. Barnum, C. Crepeau, D. Gottesman, A. Smith, and A. Tapp. Authentication of quantum messages. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.* IEEE Comput. Soc, 2002.

[BGS13] Anne Broadbent, Gus Gutoski, and Douglas Stebila. Quantum one-time programs. In *Advances in Cryptology – CRYPTO 2013*, pages 344–360. Springer Berlin Heidelberg, 2013.

[BK05] Sergei Bravyi and Alexei Kitaev. Universal quantum computation with ideal clifford gates and noisy ancillas. *Physical Review A*, 2005.

[BMMMQ20] Nicholas-Philip Brandt, Sven Maier, Tobias Müller, and Jörn Müller-Quade. Constructing secure multi-party computation with identifiable abort. *IACR Cryptol. ePrint Arch.*, 2020:153, 2020.

[BOCG+06] Michael Ben-Or, Claude Crepeau, Daniel Gottesman, Avinatan Hassidim, and Adam Smith. Secure multiparty quantum computation with (only) a strict honest majority. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE, 2006.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 97–106. IEEE Computer Society, 2011.

[CGS02] Claude Crépeau, Daniel Gottesman, and Adam Smith. Secure multi-party quantum computation. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing - STOC '02*. ACM Press, 2002.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 364–369, 1986.

[DGJ+20] Yfke Dulek, Alex B Grilo, Stacey Jeffery, Christian Majenz, and Christian Schaffner. Secure multi-party quantum computation with a dishonest majority. *Advances in Cryptology - EUROCRYPT 2020.*, 2020.

[DLT02] D. P. DiVincenzo, D. W. Leung, and B. M. Terhal. Quantum data hiding. *IEEE Transactions on Information Theory*, 48(3):580–598, 2002.

[DNS12] Frédéric Dupuis, Jesper Buus Nielsen, and Louis Salvail. Actively secure two-party evaluation of any quantum operation. In *Annual Cryptology Conference*, pages 794–811. Springer, 2012.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *Proceedings of the nineteenth annual ACM conference on Theory of computing - STOC '87*. ACM Press, 1987.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multiparty computation with identifiable abort. In *Advances in Cryptology – CRYPTO 2014*, pages 369–386. Springer Berlin Heidelberg, 2014.

[Kit] Alexei Kitaev. Quantum coin-flipping. Talk at QIP 2003 (slides and video at MSRI), December 2002.

[RBO89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85, 1989.

[Unr10] Dominique Unruh. Universally composable quantum multi-party computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 486–505. Springer, 2010.