GAuV: A Graph-Based Automated Verification Framework for Perfect Semi-Honest Security of Multiparty Computation Protocols

Xingyu Xie*†‡, Yifei Li*‡§, Wei Zhang*, Tuowei Wang*, Shizhen Xu†, Jun Zhu*†, Yifan Song*✉
*Tsinghua University, †RealAI
Email: {xiexy21,zhangw23,wtw23}@mails.tsinghua.edu.cn, liyifei.411@outlook.com,
shizhen.xu@realai.ai, {dcszj,yfsong}@mail.tsinghua.edu.cn

***Abstract*—Proving the security of a Multiparty Computation (MPC) protocol is a difficult task. Under the current simulation-based definition of MPC, a security proof consists of a simulator, which is usually specific to the concrete protocol and requires to be manually constructed, together with a theoretical analysis of the output distribution of the simulator and corrupted parties' views in the real world. This presents an obstacle in verifying the security of a given MPC protocol. Moreover, an instance of a secure MPC protocol can easily lose its security guarantee due to careless implementation, and such a security issue is hard to detect in practice.**

In this work, we propose a general automated framework to verify the perfect security of instances of MPC protocols against the semi-honest adversary. Our framework has perfect soundness: any protocol that is proven secure under our framework is also secure under the simulation-based definition of MPC. We demonstrate the completeness of our framework by showing that for any instance of the well-known BGW protocol, our framework can prove its security for every corrupted party set with polynomial time. Unlike prior work that only focuses on black-box privacy which requires the outputs of corrupted parties to contain no information about the inputs of the honest parties, our framework may potentially be used to prove the security of arbitrary MPC protocols.

We implement our framework as a prototype. The evaluation shows that our prototype automatically proves the perfect semi-honest security of BGW protocols and B2A (binary to arithmetic) conversion protocols in reasonable durations.

***Index Terms*—MPC protocol, perfect semi-honest security, automated verification, graph transformation**

1. Introduction

The design and implementation of cryptographic mechanisms are difficult to get correct. The security proofs are often complicated and error-prone; the bugs hidden in optimized implementations are often hard to catch by code testing or auditing [1]. For instance, there had been a security issue unnoticed for 17 years, in Needham–Schroeder public-key protocol, before Lowe revealed it [2].

For higher assurance, computer-aided verification is adopted to guarantee the security properties with machine-checkable proofs. To analyze security, two different models have been developed: the symbolic model and the computational model.

In the symbolic model [3], cryptographic primitives such as Enc and Dec are treated as black-box functions from which no information is leaked. The symbolic model is a suitable foundation for automated analysis [4]–[6]. Unfortunately, it drops the probabilistic nature due to abstraction, and considers a limited adversary that can only derive according to several inference rules and perform one of several possible actions.

The computational model is closer to the reality than the symbolic model. In the computational model, adversaries are regarded as probabilistic polynomial-time (PPT) algorithms. Generally, reasoning in the computational model requires writing machine-checkable proofs [7]. However, it needs significant efforts and expertise, and often costs time that is measured in person-years [8], [9]. Recent studies [10]–[14] make it possible to automatically reason in the computational model, but they all focus on game-based proofs. In general, simulation-based proofs are more complicated than game-based proofs [1]. A typical example of simulation-based security is multiparty computation (MPC).

Over the last few years, the technique of MPC has drawn a lot of attention from both academia and industry [15], [16]. In MPC, we consider n parties $P_1, \cdots, P_n$. Each party P_i provides input x_i, receives output y_i and jointly computes a function f, where $(y_1, \cdots, y_n) = f(x_1, \cdots, x_n)$. The function f is usually called the "ideal functionality". In the computation, the parties keep their inputs private: each party P_i must not learn anything about the inputs of the other parties except for what can be deduced from x_i and y_i.

Loosely speaking, the security of MPC is defined as the existence of an algorithm $\mathcal{S}$ (called "simulator"), which can simulate the messages received by the corrupted parties given only the corrupted parties' inputs and outputs. The messages received by the corrupted parties from the honest parties are called the "view". We require that the simulated view should be indistinguishable from the view in a real execution, which roughly means that they almost obey the same distribution.

Pettai and Laud [17] proposed the first method to prove the security of MPC automatically. They focus on a strong

‡ Xingyu Xie and Yifei Li contributed equally.
§ Yifei Li was affiliated with Tsinghua University and once interned at RealAI during this work. Currently, Yifei is affiliated with Alibaba Group.
✉Yifan Song is the corresponding author.

form of security: *black-box privacy*. For black-box privacy, the simulator generates the view only given the inputs of corrupted parties. In particular, the simulator is not given the outputs of the corrupted parties. This implies that any black-box private protocol needs to guarantee that the outputs of corrupted parties contain no information about honest parties' inputs, which is not the case for general MPC.

1.1. Contributions

In this work, we focus on automated verification for secure multiparty computation with perfect semi-honest security. At a high level, the perfect semi-honest security requires there exists an algorithm that can simulate the view of corrupted parties. The automated verification needs to automatically find such an algorithm given an instance of a multiparty computation protocol.

We propose GAuV, an automated verification framework for perfect semi-honest security of instances of MPC protocols. Given a protocol represented as a *data-flow graph* (circuit), the framework can automatically find an algorithm together with a proof that shows the algorithm is indeed a simulator for the given protocol.

Compared with the prior art of Pettai and Laud [17], we focus on the standard security of MPC rather than black-box privacy, where the latter indicates that the outputs of corrupted parties should contain no information about honest parties' inputs. Thus, GAuV can potentially apply to general MPC protocols.

In this work, we assume that:

- The number of parties, n, is a constant, as in the previous work [17]. This might be inherent because an automated verification needs to check all possible corrupted party sets, which can be exponential in the number of parties.
- The protocol instance is *correct*, i.e., given the same inputs, its outputs equal the outputs of the ideal functionality f, and f is deterministic. This makes us draw out the *functional correctness* from the task of security analysis. From the community of probabilistic program verification, there have been works [10], [18] aiming to verify this functional correctness.

We prove the perfect soundness of our framework:

Theorem 1.1 (soundness)**.** *Given an acyclic data-flow graph G describing an n-party protocol π that correctly computes an ideal functionality, if for each corrupted party set $I \subseteq \{1, \cdots, n\}$ of cardinality at most t, GAuV returns t-*SECURE FOR *I, then π is t-perfect-secure.*

This follows from Theorem 5.1.

We prove the completeness of our framework for the well-known BGW protocol:

Theorem 1.2 (completeness for BGW protocols)**.** *Given a data-flow graph G describing an n-party BGW protocol, for any corrupted party set $I \subseteq \{1, \cdots, n\}$ of cardinality at most t, GAuV can prove the t-perfect-security for I in polynomial time, if it is instantiated with operable* vintage transformations *and a suitable heuristic evaluation function.*

This follows from Theorem 6.1.

To show the applicability and practical potential of our theoretical framework, we implement it as a prototype tool [1], and apply our tool to BGW protocols and a B2A (binary to arithmetic) secret sharing conversion protocol. The evaluation shows that our tool can automatically prove the perfect semi-honest security of protocols in reasonable durations. Specifically, it handles protocols with graph sizes containing up to 4,500 edges in less than 1,500 s.

1.2. Our Techniques

Our starting point is the method by Pettai and Laud [17]. They represent an MPC protocol as a circuit and try to search a series of circuit transformations as proof. However, when extending it from black-box privacy to perfect security, we encounter the following difficulties.

1. How to represent a transformed protocol?

The difficulty comes from the fact that we want to recompute the values deduced by the corrupted parties from their inputs and outputs, which reverses the computation direction to some extent. During the transformation, it is possible that some value can be computed in two ways, one from the inputs of all parties, and the other from the outputs of corrupted parties. For example, in the initial protocol, the outputs of corrupted parties are clearly given while they can also be computed from the inputs of all parties. Thus, we have to represent the transformed protocol in a way that it can tolerate a value being computed in multiple ways while capturing the correctness of the underlying protocol.

To address this issue, we extend the data-flow graph (circuit) to *quasi-data-flow graph* (QDFG) by allowing a node to be pointed to by multiple edges, i.e., allowing a variable to be computed by multiple operations (Definition 3.2). We further analyze QDFG from the perspective of the value *assignment* of the nodes (Definitions 4.1 and 4.2).

In a QDFG, the correctness of MPC can be captured as follows. For each combination of inputs and outputs computed by the ideal functionality f, and each assignment of *random nodes*, there is an assignment of all nodes (Definition 4.3). We point out that, this correctness intuitively guarantees that a QDFG represents a protocol. This correctness should always be preserved during the transformation.

2. How to transform a QDFG?

The goal of the transformations is to remove the dependency on the inputs and outputs of honest parties while preserving the values known to the corrupted parties (i.e., the view of corrupted parties). We refer to the final graph as a *witness graph*. Note that the witness graph can be transformed into an algorithm that consumes the inputs and outputs of corrupted parties and computes the view of corrupted parties.

The challenge is to ensure that the algorithm we find is indeed a simulator of the protocol, i.e., the original graph.

1. https://doi.org/10.5281/zenodo.10277758

This requires that the graph transformation preserves the distribution of the view of corrupted parties. To address this issue, we introduce the concept of *vintage transformation* (Definition 4.4). At a high level, a vintage transformation preserves: (1) the correctness, and (2) the distribution of the view of corrupted parties. We assure the second condition by requiring that there exists a k-surjection from the set of assignments before the transformation to the set of assignments after the transformation. To see why this is sufficient, observe that the probability of each assignment δ of the view is a fraction between the sizes of two sets of assignments, i.e., the number of assignments in which the view equals δ, over the number of all assignments. After a transformation, dividing both the denominator and the numerator by the same constant k, the fraction is preserved.

3. How to find a transformation series?

Regarding QDFGs as states and vintage transformations as transitions, we use a search algorithm to find a transformation series with the aid of a heuristic evaluation function. The task of automated verification is generally hard to be complete and efficient. But by carefully designing the operable vintage transformations and the evaluation function, we can verify BGW protocol instances in polynomial time.

1.3. Worked Example

To illustrate how graphs are transformed, we present a worked example of a transformation series as depicted in Figures 1 to 4. Figure 1 shows an instance of BGW protocol which will be detailed in Section 6.1 and Figure 4 shows a witness QDFG.

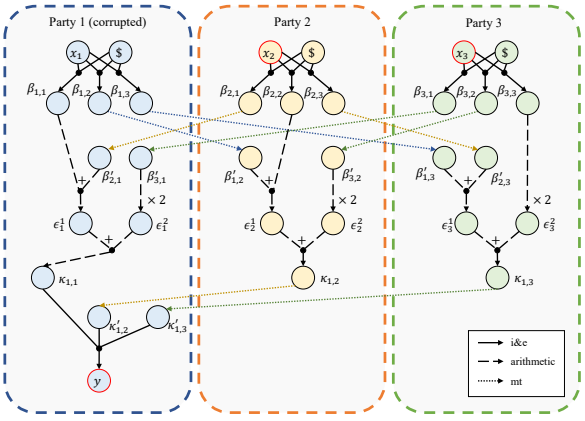

Figure 1. A BGW protocol as (quasi-)data-flow graph

In this example, we consider $t = 1$, which means all the polynomials are of degree 1. Here, we have three types of operations: **i&e** (polynomial interpolation and then evaluation at some point), arithmetic (addition and multiplication), and **mt** (message transit between two parties). See Section B.1 for their precise definitions.

The red nodes are *bubbles* that violate the expectations for a witness graph. Thus, our primary goal of transformation searching is to resolve all bubbles. Since y (the output of corrupted parties) is provided to the simulator, if the in-degree of the node representing y is not zero, we should recognize it as a bubble. Since x_2 and x_3 (the inputs of honest parties) are unknown to the simulator, they are also bubbles.

The ideal functionality of this instance of BGW protocol is $f(x_1, x_2, x_3) = (x_1 + x_2 + 2x_3, \perp, \perp)$, where $\perp$ means no output. As preparation, there are 3 publicly known distinct numbers $\alpha_1, \alpha_2, \alpha_3$. As shown in Figure 1, the protocol runs as follows.

1) Each party P_i randomly samples a polynomial $q_i(x)$ of degree t, computes *shares* $\beta_{i,j} = q_i(\alpha_j)$, and sends $\beta_{i,j}$ to P_j as $\beta'_{i,j}$.
2) Each party P_i simultaneously computes $x_1 + x_2 + 2x_3$ with its own shares.
3) Each party P_i sends the computed share $\kappa_{1,i}$ to P_1, and P_1 interpolates a polynomial from the received shares and evaluates at zero to reveal y.

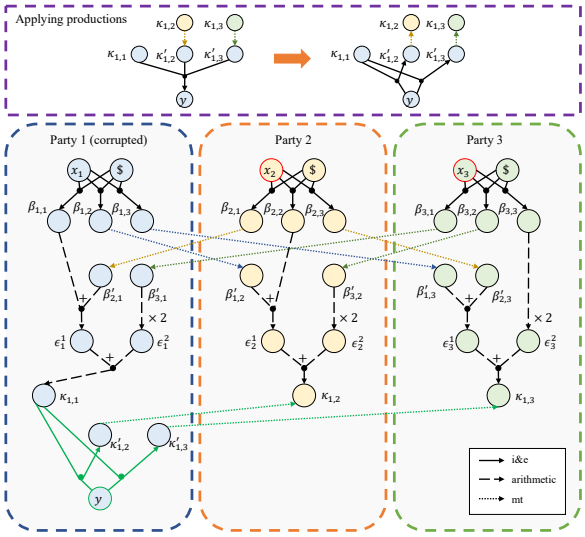

Figure 2. Applying reconstruction production for equivalent rewriting.

In this work, we propose two kinds of operable vintage transformations: equivalent rewriting and tail node elimination.

Equivalent rewriting means reversing the direction of an edge. It is enabled by *production*. If the left side of the production matches a subgraph of the QDFG, we could substitute the subgraph with the right side of the production. This is what happens in Figure 2 and Figure 3. The provided productions are based on the following two facts:

- a polynomial of degree t can be exactly determined by any $t + 1$ points;
- message transit does not change the value.

See Section B.2 for the definitions of reconstruction production (Definition B.2) and sharing production (Definition B.1).

In addition, note that in Figure 3, the nodes providing randomness are changed. This is because we define zero-in-degree nodes as *random nodes*.

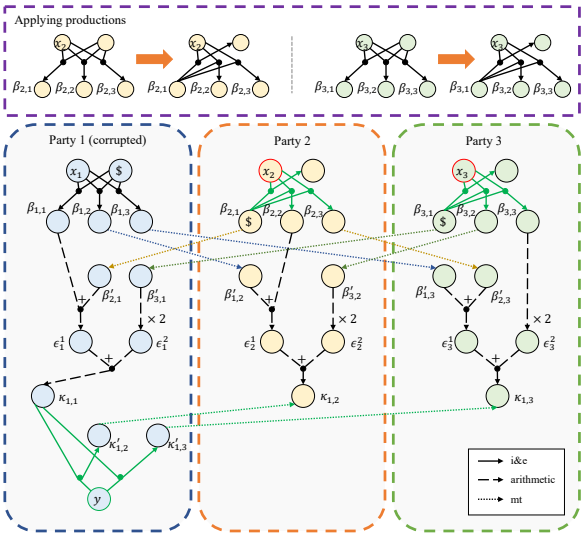

Figure 3. Applying sharing production for equivalent rewriting (twice).

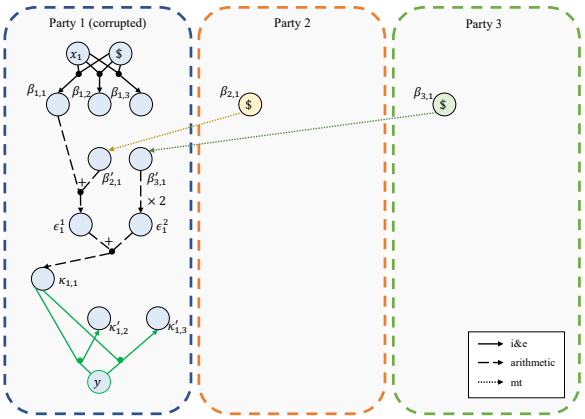

Figure 4. Eliminating tail nodes as many as possible.

Tail node elimination means removing a zero-out-degree node of honest parties. Since we always consider acyclic graphs, the value of a zero-out-degree node can be determined by the other nodes, which indicates it is actually redundant. Note that the view consists of the nodes of corrupted parties, which is our goal in a witness graph. Thus, the nodes of corrupted parties cannot be eliminated. Keep eliminating tail nodes as many as possible: from $\kappa_{1,2}$ and $\kappa_{1,3}$, then ϵ_2^1, ϵ_2^2, ϵ_3^1, ϵ_3^2, etc. Finally, we can eliminate the nodes representing x_2 and x_3. Now, in Figure 4, we resolve all bubbles and find a witness graph, where the input and output nodes of corrupted parties all have zero in-degree.

2. Preliminary: Perfect Security of MPC

The secure multiparty computation is modeled as an n-ary functionality $f : X_1 \times \cdots \times X_n \to Y_1 \times \cdots \times Y_n$, which maps the private input of each party to its desired function output. Specifically, the i-th party provides the i-th input to f and receives the i-th output. An adversary is an attacker to the secure multiparty computation protocol.

In this work, we focus on threshold and semi-honest adversaries, who can corrupt at most t parties for a given $t < n$. The semi-honest security requires that parties corrupted by an adversary should still follow the protocol execution. But the adversary may try to learn additional information about honest parties' inputs from the view of corrupted parties. We consider non-adaptive adversaries who should specify the parties they want to corrupt at the beginning of the computation. For each pair of parties, a secure (private and authentic) synchronous channel is assumed to exist so that they can directly send messages to each other. We follow [19] to define t-perfect-security as follows.

Definition 2.1 (t-perfect-security)**.** *Let $f : X_1 \times \cdots \times X_n \to Y_1 \times \cdots \times Y_n$ be a deterministic n-ary functionality where $X_1, \cdots, X_n, Y_1, \cdots, Y_n$ are finite sets, and $f_i(x_1, \ldots, x_n)$ denotes the functionality output for the i-th party.*

Let π be an n-ary protocol for computing f. During an execution of π on inputs $\vec{x} = (x_1, \ldots, x_n)$, the view *of the i-th party is the tuple containing the* random tape *of the i-th party, intermediate computed results and received messages of party i when all parties follow the protocol to execute, denoted as $\text{VIEW}_i^\pi(\vec{x})$. For the set of corrupted parties $I = \{i_1, \ldots, i_l\} \subseteq [n] := \{1, \ldots, n\}$, let $f_I(\vec{x})$ denote the subsequence $(f_{i_1}(\vec{x}), \ldots, f_{i_l}(\vec{x}))$. Let $\text{VIEW}_I^\pi(\vec{x}) := \big(I, \text{VIEW}_{i_1}^\pi(\vec{x}), \ldots, \text{VIEW}_{i_l}^\pi(\vec{x})\big)$.*

We say that π computes *f with t-perfect-security if there exists a* probabilistic polynomial-time (PPT) algorithm*, denoted S, such that for every $I \subset [n]$ of cardinality $|I| \le t$, and for all $x_1 \in X_1, \ldots, x_n \in X_n$, it holds that*

$$\{S\left(I, \vec{x}_I, f_I(\vec{x})\right)\}_{\vec{x} \in X_1 \times \cdots \times X_n} \equiv \{\text{VIEW}_I^\pi(\vec{x})\}_{\vec{x} \in X_1 \times \cdots \times X_n}, \tag{2.1}$$

where $\equiv$ means two probability ensembles *are identically distributed.*

The PPT algorithm S is also known as the *simulator* and the term *simulation-based proof* refers to the proof technique by constructing such simulators.

Note that this definition is equivalent to the one that defines the view as received messages only, since the simulator knows $\vec{x}_I$ and could simply follow the corrupted parties' parts of the protocol to compute the intermediate results. Here we regard the corrupted parties' intermediate computed results as a part of the view to facilitate automated analysis.

3. Protocols as Quasi-Data-Flow Graphs

At a high level, our verification framework starts with a data-flow graph (DFG) representing the underlying MPC protocol. DFG describes how the values are computed and transmitted. Our idea is to apply a series of graph transformations to gradually remove the dependency on honest parties' inputs while keeping the distribution of the views of corrupted parties unchanged. If we finally obtain a "witness" graph without any dependency on honest parties' inputs,

then such a graph would correspond to a simulator that satisfies Definition 2.1.

However, vanilla DFG is not sufficient since some nodes may be computable in multiple ways. For example, the outputs of corrupted parties can either be received from the ideal functionality or computed following the protocol. To address this issue, we extend the notion of DFG to *quasi-data-flow graph (QDFG)*. In this section, we first review the concept of DFG, and then define QDFG.

3.1. Data-Flow Graph

A *data-flow graph (DFG)* is a graphical representation of a program. In cryptographic terms, it can be viewed as a circuit with nodes as gates and edges as wires. Taking the party into account, DFG can represent an MPC protocol.

In classic DFGs [20], nodes denote operations and their results, and arcs denote data dependencies between operations. To enable later analysis, we choose a slightly different but conceptually equivalent design: nodes denote variables and edges denote operations. In other words, both operands and results are nodes, and the operation is an edge linking operand nodes to the result node. Note that operator could be non-unary — each of our edges is actually a hyper-edge, grouping arcs of an operation.

Operators in this paper are formally defined as a partial function as follows. Here we consider the types in the programs as finite sets, which are the value domains of the variables. Let $\mathcal{T}$ be the set of all types in our scope.

Definition 3.1 (operator)**.** *An m-ary* operator $op : X_1 \times \cdots \times X_m \rightharpoonup Y$ *is a partial function, that can be computed by a deterministic polynomial-time algorithm, where $X_1, \cdots, X_m, Y \in \mathcal{T}$.*

Remark 3.1. *We use partial functions to accommodate operators that require inputs to have certain structures. For example, for polynomial interpolations, the operator requires the inputs to lie on a polynomial.*

In MPC, there is an elementary operator, "message transit" ($\mathbf{mt}$). $\mathbf{mt}$ transfers a variable from one party to another party. With the help of $\mathbf{mt}$, we can always represent a protocol as a DFG so that any edge other than $\mathbf{mt}$ only connects nodes within the same party, as shown in Section 1.3. This reflects the practical requirement that computations are carried out by individual parties. If an operation involves different parties, $\mathbf{mt}$ can be used to link inter-party nodes by introducing intermediary nodes at the executing party, e.g., $\beta_{2,1}$ and $\beta'_{2,1}$ in Figure 1. In this way, we can always convert such an edge into an edge only connecting nodes at the same party and several $\mathbf{mt}$ edges. In high-level words, we explicitly separate computation and communication when representing an MPC protocol as a DFG. This will benefit the convenience of recognizing views: in DFG, a party's view is the values of the nodes at that party.

In our context, the *in-degree* of a node in a DFG means the number of hyper-edges pointing to that node rather than the number of arcs. In a proper DFG, each node with a non-zero in-degree results from a single operation, i.e., its in-degree should be exactly one. This guarantees DFG's *consistency*, allowing unique determination of all node values in a topological order based on zero-in-degree nodes.

In such DFGs, *input nodes* representing protocol inputs and *random nodes* representing random values[2] are both of zero in-degree. Thus as per the nature of the DFG, values of all the other nodes, including *output nodes* representing protocol outputs and other *intermediate nodes* representing intermediate variables produced during the protocol execution, can be uniquely computed in a topological order given values of inputs and random nodes.

3.2. Quasi-Data-Flow Graph

DFG is not sufficient for automated verification based on graph transformations, because:

1) in the beginning, the outputs of corrupted parties are given to the simulator, which means that the output nodes of the DFG are also provided with initial values;
2) after some transformations, it is possible that a node violates the consistency requirement for DFG, becoming the destination of multiple hyper-edges.

To address this issue, we introduce the *quasi-data-flow graph (QDFG)*, which retains all vanilla DFG properties except consistency. All DFGs are inherently QDFGs.

Definition 3.2 (quasi-data-flow graph, QDFG)**.** *A positive integer n denotes the number of parties. An n-party* quasi-data-flow-graph *G is defined as (V, V^{const}, E, ρ).*

- *V is the set of nodes, among which* const nodes*, denoted $V^{const} \subseteq V$, referring to nodes corresponding to the original inputs and outputs.*
- *E is the set of hyper-edges, where each $e \in E$ is a tuple $(op^e, (src^e_1, \ldots, src^e_{m_e}), dst^e)$ representing the operation of e, among which the operator op^e takes m_e operands; and for convenience we also use $\overrightarrow{src}^e$ to denote the tuple $(src^e_1, \ldots, src^e_{m_e})$.*
- *$\Gamma : V \to \mathcal{T}$ is a typing context, which maps a node to its value domain;*
- *$\rho : V \to [n]$ is a label mapping such that $\rho(v)$ is the party to which $v \in V$ belongs.*

Let us make it clearer about so-called *const nodes*. In the simulation-based proof technique, correctness and security are defined over any possible inputs and corresponding outputs computed by the ideal functionality. Thus in terms of QDFG, we should analyze correctness and security given any predetermined set of proper values of input and output nodes. That is why we call them *const nodes*.

Remark 3.2. *When referring to a certain component of QDFG G, we may use G as the subscript of this component to avoid ambiguity. For example, we use V^{const}_G to denote the const nodes of the QDFG G.*

2. A non-trivial cryptographic protocol should contain some randomness, and we use *random nodes* to represent random values sampled from *random tapes* in the execution of the protocol.

When there is a need to further restrict nodes to belong to some parties, we add the party set as a left superscript onto the notation. For example, ${}^{I}V_G^{const} \subseteq V_G^{const}$ denotes the subset of G's const nodes located at a party set I.

The concepts of in-degree, out-degree, and acyclicity in DFGs are extended onto QDFGs as follows.

Definition 3.3 (in-degree and out-degree)**.** *Given a QDFG G and a node $u \in V_G$:*

- *the* in-degree *of u is* $\deg_G^-(u) \coloneqq |\{e \mid u = dst^e\}|$;
- *the* out-degree *of u is* $\deg_G^+(u) \coloneqq |\{e \mid u \in src^e\}|$.

Definition 3.4 (acyclicity)**.** *We say that a QDFG G is* acyclic*, if there is NO node sequence $v_1, \cdots, v_l$ in G such that:*

- *for each $1 \leq i < l$, there exists $e \in E_G, v_i \in src_e$ and $v_{i+1} = dst^e$; and*
- *there exists $e \in E_G$ s.t. $v_l \in src^e$ and $v_1 = dst^e$.*

Let us focus on the zero-in-degree nodes. In the QDFG of the original protocol, the zero-in-degree nodes represent the variables that are not intermediate computed results, whose values should either come from the inputs or the random tapes. When analyzing simulation-based security, the ideal functionality can also provide the outputs. Thus, in a QDFG, we recognize the non-const zero-in-degree nodes as random nodes.

Definition 3.5 (random node)**.** *Given a QDFG G, the set of its* random nodes *is $V_G^{rand} \coloneqq \{u \mid \deg_G^-(u) = 0\} \setminus V_G^{const}$.*

4. Transformation of QDFG

This section establishes the theoretical foundation for transforming QDFG to prove perfect security in semi-honest MPC protocols using simulation-based proof and QDFG transformations. We demonstrate that an algorithm generating the view of corrupted parties in a transformed QDFG can also replicate this view in the original QDFG. The verification process involves a series of transformations to construct a "witness" QDFG representing a simulator.

We will introduce key definitions (Section 4.1), propose the "vintage transformation" concept (Section 4.2), prove our Main Theorem for framework soundness (Section 4.3), and present two specific vintage transformation classes (Sections 4.4.1 and 4.4.2).

4.1. Preparative Concepts

Since the simulator will receive inputs and outputs of corrupted parties, even in the initial QDFG corresponding to the original MPC protocol, we can consider the values of the corrupted parties' input and output nodes as given values. To formally describe this, we define the notion of (partial) assignment of nodes in QDFG as follows.

Definition 4.1 (assignment)**.** *Given a node set V, an* assignment *on V is a function that maps each node $v \in V$ to a value in $\Gamma(v)$.*

In QDFGs, a proper set of node values must meet the operators' semantics implied by their connecting hyperedges. Generally, given any assignment on a subset of nodes in a QDFG, a complementary assignment on the other nodes such that the semantics of all edges are satisfied may not exist. However, given an assignment on const (input and output) nodes that align the outputs computed by the ideal functionality from the inputs, a QDFG representing an MPC protocol should allow a proper assignment on the whole graph. This leads to importing MPC "correctness" into the QDFG context.

The correctness of an MPC protocol is defined over all possible inputs and corresponding outputs calculated by the ideal functionality. The correctness says that, given any valid inputs, the outputs produced by a protocol are aligned with the outputs produced by the ideal functionality.

To formally define the correctness of QDFG, we first present the definition of *total assignment*, which is intuitively an assignment on all nodes, obeying the semantic requirements of all edges, based on a valid set of const node values.

Definition 4.2 (total assignment)**.** *Given a QDFG G and an assignment γ on V_G^{const}, we say that an assignment α on V_G is a* total assignment *of G w.r.t. γ, if*

- *for every $e \in E_G$, $\alpha(\overrightarrow{src^e}) \in \mathrm{dom}(op^e)$ and $\alpha(dst^e) = op^e(\alpha(\overrightarrow{src^e}))$; and*
- $\alpha|_{V_G^{\mathrm{const}}} = \gamma$.

We use TAS_G^{γ} to denote the set of all total assignments of G w.r.t. γ.

For an MPC protocol of a deterministic ideal functionality, *correctness* means that for each input, the output produced by the execution of the protocol is exactly the same as the output calculated by the ideal functionality. In terms of QDFG, we adapt this definition of correctness by capturing the execution of the protocol with a total assignment that takes the inputs and outputs (i.e., an assignment on const nodes) as parameters.

Definition 4.3 (correctness of QDFG)**.** *Given a QDFG G, we say that G is* correct *w.r.t. assignment γ on V_G^{const}, if for all assignment α on V_G^{rand}, there exists a total assignment β of G w.r.t. γ s.t. $\beta|_{V_G^{rand}} = \alpha$.*

Since we consider an acyclic QDFG G, given an assignment on V_G^{rand} and V_G^{const}, the assignment on all the other nodes can be calculated in topological order, which immediately indicates the uniqueness of the total assignment.

Fact 4.1 (uniqueness implied by correctness)**.** *If an acyclic QDFG G is correct w.r.t. assignment γ on V_G^{const}, for each assignment α on V_G^{rand}, there exists* only one *total assignment β of G s.t. $\beta|_{V_G^{rand}} = \alpha$ and $\beta|_{V_G^{const}} = \gamma$.*

This uniqueness provides a mechanized way to compute the number of total assignments, indicating the following fact.

Fact 4.2 (amount of total assignments)**.** *Let G be an acyclic QDFG that is correct w.r.t. assignment γ on V_G^{const}, then $|TAS_G^\gamma| = \prod_{r \in V_G^{rand}} |\Gamma(r)|$.*

4.2. Vintage Transformation

Now that we have defined assignments and correctness, we can discuss the graph transformation. Each transformation should preserve the distribution of the view of corrupted parties as well as the correctness.

We assure the distribution preservation by requiring that there exists a k-surjection from the set of total assignments before the transformation to the set of total assignments after the transformation. This is sufficient because we observe that the probability of each assignment δ of the view of corrupted parties is a fraction between two amounts of assignments, i.e., the number of total assignments where the view of corrupted parties equals δ, over the number of all total assignments. If the denominator and numerator are divided by the same constant k after a transformation, the fraction remains equal.

To characterize all requirements for a graph transformation, we propose the *vintage transformation*. To be mentioned, when verifying the security, the corrupted party set I should always be taken into account, so our vintage transformation is defined on a fixed set $I \subseteq [n]$.

Definition 4.4 (vintage transformation)**.** *Given a positive integer n denoting the number of parties with a fixed corrupted party set $I \subseteq [n]$, let G and H be two acyclic n-party QDFGs. We say that the transformation from G to H is a* vintage transformation *w.r.t. I, denoted by $G \overset{I}{\rightsquigarrow} H$, if there exists an injection $h_v : V_H \to V_G$, so that the following conditions are satisfied for every assignment γ on V_G^{const} s.t. G is correct w.r.t. γ:*

(i) all nodes of G belonging to corrupted parties fall in the range of h_v, i.e., $\rho_G^{-1}(I) \subseteq \mathrm{range}(h_v)$;

(ii) h_v preserves the typing context, i.e., $\Gamma_H = \Gamma_G \circ h_v$;

(iii) h_v preserves the party label, i.e., $\rho_H = \rho_G \circ h_v$;

(iv) each const node of H is mapped onto a const node of G via h_v, i.e., $h_v(V_H^{const}) \subseteq V_G^{const}$;

(v)

$$\begin{aligned} \psi : \ & TAS_G^\gamma \to TAS_H^\gamma \\ & \alpha \mapsto \alpha \circ h_v \end{aligned}$$

*is a k-*surjection*, i.e., there is a positive integer k s.t. for each $\beta \in TAS_H^\gamma$, $|\psi^{-1}(\beta)| = k$;*

(vi) H is correct w.r.t. $\gamma \circ h_v$.

Intuitively, we can recognize each pair of nodes mapped by h_v as the same node. The requirement "h_v is an injection from V_H to V_G" means the nodes cannot be inserted or merged, but only be removed during vintage transformation.

Now we explain the intuitive necessity of conditions (i-vi). (i): The aim of transformation is to build a simulator that produces the view of corrupted parties, which are all nodes of corrupted parties in the initial DFG. Thus, those nodes should not be removed during vintage transformation. (ii-iv): These three conditions mean that h_v preserves the type of a node, to which party a node belongs, and whether a node is const. We need these two conditions because we wish to recognize each pair of nodes mapped by h_v as the same node. (v): As explained before. (vi): With the premise that the initial QDFG (actually a DFG) is correct, the correctness should be preserved during vintage transformations. The corollary of this correctness, Fact 4.2, will be used later to prove the perfect security in Theorem 4.1.

4.3. Main Theorem for Soundness

Now, we are going to prove the main theorem of this paper, which ensures the soundness of our framework.

The target of our graph transformation is a witness QDFG representing a simulator of the MPC protocol, where the set of zero-in-degree nodes exactly contains random nodes, input nodes, and output nodes of corrupted parties. The input and output nodes of corrupted parties would be inputs of the simulator, and the random nodes would provide randomness for the simulator (a.k.a. the random tapes in the simulator). All input and output nodes of honest parties would have been removed, and all non-zero-in-degree nodes can be computed in topological order, describing a simulator of the witness QDFG. The simulator would be able to produce an assignment of the nodes of the corrupted parties, i.e., the messages received by the corrupted parties.

To better describe how far away a QDFG is from the final witness, some nodes are worth special concerns, called *bubbles*, which can guide searching transformations by setting the goal of resolving (removing) all bubbles. Bubbles contain the nodes that violate the expectations above for a witness QDFG. Thus, when all bubbles are resolved, we have found a witness QDFG and the search could terminate.

Definition 4.5 (bubble)**.** *Given an n-party QDFG G with a corrupted party set $I \subseteq [n]$, the bubbles of a QDFG G is a subset of V_G, denoted ${}^I\mathcal{B}_G$, which consists of the following two types of nodes:*

(i) the input and output nodes of honest parties, i.e., ${}_{[n]\setminus I}V_G^{const}$;

(ii) the input and output nodes of corrupted parties whose in-degree is not 0, i.e.,

$${}^I V_G^{const} \setminus \left\{ u \in V_G \mid \deg^-(u) = 0 \right\}.$$

As Definition 4.5 has provided the rigorous condition for the expected termination, we can now prove our main theorem, indicating the soundness of our verification algorithm.

Theorem 4.1 (soundness)**.** *Let G_0 be an n-party acyclic QDFG, which describes a protocol π that computes an ideal functionality f. The correctness of π is premised, that is, for each assignment γ on $V_{G_0}^{const}$ s.t. $f(\gamma(\overrightarrow{V_{G_0}^{in}})) = \gamma(\overrightarrow{V_{G_0}^{out}})$, G_0 is correct w.r.t. γ, where $\overrightarrow{V_{G_0}^{in}}$ denotes the tuple consisting of the nodes in $V_{G_0}^{in}$, and $\overrightarrow{V_{G_0}^{out}}$ denotes the tuple consisting of the nodes in $V_{G_0}^{out}$.*

If for each corrupted party set $I \subset [n]$ of cardinality at most t, there exists a vintage transformaion series $G_0 \overset{I}{\rightsquigarrow} G_1 \overset{I}{\rightsquigarrow} \cdots \overset{I}{\rightsquigarrow} G_m$ s.t. G_m has no bubble, i.e., ${}^I\mathcal{B}_{G_m} = \emptyset$, then π is t-perfectly-secure.

Proof. This proof can be roughly divided into two parts: (1) to construct a probabilistic polynomial-time algorithm $\mathcal{S}'$; (2) to prove that $\mathcal{S}'$ can simulate the view with the help of the vintage transformation series $G_0 \overset{I}{\rightsquigarrow} G_1 \overset{I}{\rightsquigarrow} \cdots \overset{I}{\rightsquigarrow} G_m$. Throughout the proof, we consider an arbitrary corrupted party set $I \subset [n]$ of cardinality at most t. Let h_v^i denote the vintage node injection of $G_{i-1} \overset{I}{\rightsquigarrow} G_i$ $(1 \leq i \leq m)$.

To describe the distribution of the nodes of corrupted parties, consider random variables $X_i : \beta \mapsto \beta|_{{}^I V_{G_i}}$ of type

$$TAS_{G_i}^{\gamma \circ h_v^1 \circ \cdots \circ h_v^i} \to \left\{ \beta|_{{}^I V_{G_i}} : \beta \in TAS_{G_i}^{\gamma \circ h_v^1 \circ \cdots \circ h_v^i} \right\}.$$

The randomness in QDFG is totally provided by the unique sampling assignment of random nodes. And, Fact 4.1 tells us each sample belongs to exactly one total assignment. Thus, the probability of the random variable on an assignment δ is a fraction of two amounts of total assignments, i.e., the number of total assignments where the view of corrupted parties equals δ, over the number of all total assignments.

First, we will prove that we can construct a probabilistic polynomial-time algorithm that computes X_m, from the edge structure of G_m. For each assignment γ on $V_{G_0}^{const}$ s.t. $f(\gamma(\overrightarrow{V_{G_0}^{in}})) = \gamma(\overrightarrow{V_{G_0}^{out}})$, we know that G_0 is correct w.r.t. γ, so according to the condition (vi) of vintage transformation (Definition 4.4), $G_0 \overset{I}{\rightsquigarrow} G_1$ indicates that G_1 is correct w.r.t. $\gamma \circ h_v^1$, and G_2 is correct w.r.t. $\gamma \circ h_v^1 \circ h_v^2$, etc. Finally, we have that G_m is correct w.r.t. $\gamma \circ h_v^1 \circ \cdots \circ h_v^m$.

Since G_m has no bubble, we have:

- ${}^{[n]\setminus I}\overrightarrow{V_{G_m}^{in}} \cup {}^{[n]\setminus I}\overrightarrow{V_{G_m}^{out}} = \emptyset$, and
- $\{u | \deg^-(u) = 0\} = {}^I\overrightarrow{V_{G_m}^{in}} \cup {}^I\overrightarrow{V_{G_m}^{out}} \cup V_{G_m}^{rand}$,

which indicates $V_{G_m}^{const} = {}^I\overrightarrow{V_{G_m}^{in}} \cup {}^I\overrightarrow{V_{G_m}^{out}}$. Let γ be an arbitrary assignment on $V_{G_0}^{const}$ s.t. $f(\gamma({}^I\overrightarrow{V_{G_0}^{in}})) = \gamma({}^I\overrightarrow{V_{G_0}^{out}})$, and let α be an arbitrary assignment on $V_{G_m}^{rand}$. As G_m is correct w.r.t $\gamma \circ h_v^1 \circ \cdots h_v^m$, there exists a total assignment β of G_m w.r.t. $\gamma \circ h_v^1 \circ \cdots h_v^m$ s.t. $\beta|_{V_{G_m}^{rand}} = \alpha$. Thus, since G_m is acyclic, we can calculate this total assignment β in topological order. As each operator can be calculated by a deterministic polynomial-time algorithm (Definition 3.1), all operations in a topological order compose a deterministic polynomial-time algorithm $\mathcal{S}''$ of the whole QDFG. Note that the algorithm is constructed according to the edges of G_m, which is independent of the assignment on G_m^{const} and G_m^{rand}, i.e., independent of the choice of γ and α.

By uniformly sampling the assignment on $V_{G_m}^{rand}$ for $\mathcal{S}''$, we construct a probabilistic polynomial-time algorithm $\mathcal{S}'$, which computes X_m.

Second, we will prove that the distribution of the nodes of corrupted parties in G_0 is equal to the distribution of corresponding nodes in G_1, and then equal to the distribution in G_2, ..., until G_m.

Consider arbitrary i and assignment γ on $V_{G_0}^{const}$ s.t. $f(\gamma(\overrightarrow{V_{G_0}^{in}})) = \gamma(\overrightarrow{V_{G_0}^{out}})$. To prove the equality between two discrete distribution of random variables X_{i-1} and X_i, we can enumerate all their values. A value of X_{i-1} is an assignment on ${}^I V_{G_{i-1}}$, while a value of X_i is an assignment on ${}^I V_{G_i}$. By condition (i) and (iii) of vintage transformation $G_{i-1} \overset{I}{\rightsquigarrow} G_i$, $h_v^i({}^I V_{G_i}) = {}^I V_{G_{i-1}}$. Thus, to prove the equality between the distributions of X_{i-1} and X_i, it is sufficient to show that for all assignment δ on ${}^I V_{G_{i-1}}$, $\Pr[X_{i-1} = \delta] = \Pr\left[X_i = \delta \circ h_v^i\right]$.

Recall that the probability of the random variable is the fraction between two amounts of total assignments. This turns equation (4.1) to (4.2), and (4.3) to (4.4). Equation (4.2) is deduced to (4.3) by the condition (v) of vintage transformation $G_{i-1} \overset{I}{\rightsquigarrow} G_i$, which divides both the denominator and numerator by the same constant k.

$$\Pr[X_{i-1} = \delta] \tag{4.1}$$

$$= \frac{\left|\left\{\beta_{i-1} \in TAS_{G_{i-1}}^{\gamma \circ h_v^1 \circ \cdots \circ h_v^{i-1}} \;\middle|\; \beta_{i-1}|_{{}^I V_{G_{i-1}}} = \delta\right\}\right|}{\left|TAS_{G_{i-1}}^{\gamma \circ h_v^1 \circ \cdots \circ h_v^{i-1}}\right|} \tag{4.2}$$

$$= \frac{\left|\left\{\beta_i \in TAS_{G_i}^{\gamma \circ h_v^1 \circ \cdots \circ h_v^i} \;\middle|\; \beta_i|_{{}^I V_{G_i}} = \delta \circ h_v^i\right\}\right|}{\left|TAS_{G_i}^{\gamma \circ h_v^1 \circ \cdots \circ h_v^i}\right|} \tag{4.3}$$

$$= \Pr\left[X_i = \delta \circ h_v^i\right] \tag{4.4}$$

This holds for all i, so for each assignment δ on ${}^I V_{G_0}$, we have $\Pr[X_0 = \delta] = \Pr\left[X_m = \delta \circ h_v^1 \circ \cdots \circ h_v^m\right]$. That is to say, the distributions of X_0 and X_m are equal up to $h_v^1 \circ \cdots \circ h_m^v$. Eventually, we can construct a probabilistic polynomial-time algorithm $\mathcal{S}$, which

1) accepts assignment γ on $V_{G_0}^{const}$,
2) calls sub-procedure $\mathcal{S}'$ that computes X_m with $\gamma \circ h_v^1 \circ \cdots \circ h_v^m$ (recall that $\mathcal{S}'$ is a probabilistic polynomial-time algorithm that computes X_m),
3) receives an assignment ϵ on ${}^I V_{G_m}$ from $\mathcal{S}'$, and
4) produces $\epsilon \circ (h_v^m|_{{}^I V_{G_m}})^{-1} \circ \cdots \circ (h_v^1|_{{}^I V_{G_1}})^{-1}$ (note that $h_v^i|_{{}^I V_{G_m}}$ is invertible because of the condition (i) of Definition 4.4).

We have shown what $\mathcal{S}$ produces is equivalent to X_0, but why does X_0 represent the corrupted parties' view in the real world? Note that γ is an assignment on V_G^{const} s.t. $f(\gamma(\overrightarrow{V_{G_0}^{in}})) = \gamma(\overrightarrow{V_{G_0}^{out}})$, adhering the ideal functionality f, and G_0 is correct w.r.t. this γ. For these two reasons, by sampling values for random nodes, we can calculate a total assignment for all nodes in G_0 in topological order. This calculation can be regarded as "executing" the protocol π, which indicates that the distribution of X_0 equals that of the view of corrupted parties in the real world. Therefore, $\mathcal{S}$ indeed simulates the distribution of the view of the corrupted parties in the real world.

Since the argument above works for all assignment γ, we have that, for all possible input, the distribution produced by $\mathcal{S}$ is equivalent to the distribution of the real view of the

corrupted parties. Finally, since I is arbitrarily chosen of cardinality at most t at the very beginning of this proof, we have proved that the protocol π described by G_0 is t-perfectly-secure (Definition 2.1). □

4.4. Operable Vintage Transformation

In this subsection, we construct two classes of operable vintage transformation which will be used in the automated verification algorithm. More operable vintage transformations could be constructed, but these two are sufficient for BGW protocols. For each class of operable vintage transformation, we will first give its intuition and definition, and then show that it is indeed a vintage transformation.

4.4.1. Equivalent Rewriting. Recall that when a simulator accepts the inputs and outputs of corrupted parties and produces some intermediate results, the direction of computation needs to be somehow changed. In terms of QDFG, we need to "reverse" the direction of some edges from output nodes, without modifying the overall requirements of the edge set.

Here we give an example to make this intuition clearer. Considering an equation with three variables $c = a + b$, we can transform it into another equivalent $a = c - b$. In terms of QDFG, we transform a three-node and one-edge QDFG where node c is the destination, into another QDFG where node a is the destination.

To formalize and generalize this idea, inspired by the graph rewriting system [21], we propose a production-based method. A production is a rewriting rule specifying a subgraph substitution that can be performed to generate new graphs. We give the formal definition of *equivlent production* as follows, which intuitively means that in any graph, L can be substituted with R, while the semantic requirements of edges remain.

Definition 4.6 (equivalent production)**.** *We say that a QDFG pair $p = (L, R)$ is an* equivalent production*, if*

- $V_L = V_R, V_L^{const} = V_R^{const}$*, and* $\rho_L = \rho_R$*; and*
- *for any assignment γ on $V_L^{const} = V_R^{const}$, $TAS_L^\gamma = TAS_R^\gamma$.*

Remark 4.1. *Note that the only component of R which may differ from that of L is the edge set.*

The substitution can be performed on G when L matches a subgraph of G. To rigorously describe what a match is, we borrow the concept of morphism into the context of QDFG as *QDFG morphism*, which is a function preserves the structure of QDFG.

Definition 4.7 (QDFG morphism)**.** *Given two QDFGs G_1 and G_2, a* QDFG morphism *$f : G_1 \to G_2$, $f = (f_v, f_e)$ consists of two injective functions $f_v : V_{G_1} \to V_{G_2}$ and $f_e : E_{G_1} \to E_{G_2}$ preserving the structure of QDFG. To be precise, f_v and f_e*

(i) preserve the source and destination structure of hyperedges, i.e., for any $e \in E_{G_1}$,
- *$dst_{f_e(e)} = f_v(dst^e)$,*
- *$|\overrightarrow{src}^e| = |\overrightarrow{src}^{f_e(e)}|$, and*
- *for each $src_i^e \in \overrightarrow{src}^e$, $src_i^{f_e(e)} = f_v(src_i^e)$;*

(ii) preserve operations[3], i.e., for any $e \in E_{G_1}$, $op^{f_e(e)} = op^e$;

(iii) preserve whether a node is const, i.e., for any $v \in V_{G_1}$, $[v \in V_{G_1}^{const}] = [f_v(v) \in V_{G_2}^{const}]$; and

(iv) preserve party label, i.e., for any $v \in V_{G_1}$, $\rho_{G_2}(f_v(v)) = \rho_{G_1}(v)$.

Now we are ready to define the *equivalent rewriting*: when the pattern L matches a subgraph of G, we can substitute that subgraph with R. To keep the correctness, we further check two conditions after the substitution.

Definition 4.8 (equivalent rewriting)**.** *Given an equivalent production $p = (L, R)$, a QDFG G, and a QDFG morphism $f : L \to G$ called the* match*, we can apply the equivalent production p to construct a QDFG $H \coloneqq (V_G, V_G^{const}, E_H, \rho_G)$, where*

$$E_H \coloneqq (E_G \setminus f_e(E_L)) \cup \{(op^e, (f_v(src_1^e), \cdots, f_v(src_{m_e}^e)), f_v(dst^e)) \mid e \in E_R\}.$$

We say H is constructed from G by equivalent rewriting *with equivalent production p on match morphism m, if*

- *both G and H are acyclic; and*
- *for any $t \in \mathcal{T}$, the number of random nodes of type t is preserved, i.e.,*

$$\sum_{r \in V_G^{rand}} [\Gamma_G(r) = t] = \sum_{r \in V_H^{rand}} [\Gamma_H(r) = t].$$

To better understand equivalent rewriting, we point out that it is naturally symmetric, which means that we can also substitute R with L to generate G from H.

Fact 4.3 (symmetry)**.** *Given an equivalent production $p = (L, R)$, a QDFG G, a QDFG morphism f, and a QDFG H constructed from G by equivalent rewriting with equivalent production p on match f, there exists $g : R \to H$ (where $g_v = f_v$) s.t. G is constructed from H by equivalent rewriting with equivalent production $p' = (R, L)$ on match g. In other words,*

$$E_G = (E_H \setminus g_e(E_R)) \cup f_e(E_L), \text{ and}$$
$$E_H = (E_G \setminus f_e(E_L)) \cup g_e(E_R).$$

Observing this symmetry, with the help of the two symmetric morphisms f and g, we can build a bijection between the total assignments of G and H from the bijection between the total assignment of L and R. This means that the semantic requirements of E_G and E_H are the same, and that the equivalent rewriting is indeed a vintage transformation.

Theorem 4.2. *Given the number of parties n and a corrupted party set $I \subseteq [n]$, considering an equivalent production p, a QDFG G, a QDFG morphism f, and a QDFG H*

3. Note that condition (i) also implies $op^{f_e(e)}$ takes as many source operands as op^e, and the order of operands is also preserved.

constructed from G by equivalent rewriting with equivalent production p on match f, the equivalent rewriting is a vintage transformation w.r.t. I.

The full proof is given in Appendix A.1.

4.4.2. Tail Node Elimination. The correctness of QDFG guarantees that we can choose an arbitrary topological order to compute a total assignment. Considering a zero-out-degree node at honest parties, there is a topological order where the node is placed at the tail. If we compute a total assignment along this topological order, we can simply ignore this tail node, keeping the computation of the other nodes unaffected. To formalize this idea, we define the *tail node elimination* and state that it is a vintage transformation as follows, with Figure 4 as an example.

Definition 4.9 (tail node elimination)**.** *Given a QDFG G and a node $u \in V_G$ s.t. $\deg^+_G(u) = 0$, we can construct a QDFG H by eliminating u and in-edges of u from G, i.e.,*

$$H \coloneqq (V_G \backslash \{u\}, V_G^{const} \backslash \{u\}, \{e \in E_G | dst_e \neq u\}, \rho_G|_{V_G \backslash \{u\}}).$$

We say that H is constructed from G by tail node elimination *of node u.*

Theorem 4.3. *Given the number of parties n and a corrupted party set $I \subseteq [n]$, considering a QDFG G, a node $u \in V_G \setminus {}^I V_G$, and a QDFG H constructed from G by tail node elimination of node u, the tail node elimination is a vintage transformation w.r.t. I.*

The proof is given in Appendix A.2.

5. Automated Verification Algorithm

Based on Theorem 4.1, the problem of proving the perfect security can be reduced to looking for a series of vintage transformations. This makes it possible to design an automated verification algorithm.

Recall that a bubble consists of nodes violating the expectations, the goal of our algorithm is to resolve all bubbles by a series of vintage transformations. This goal can be considered as a search problem: the QDFGs are states, and vintage transformations between QDFGs are transitions. From this perspective, we can design a search algorithm guided by a heuristic evaluation function.

Algorithm 1 shows the details. Given an n-party QDFG G_0 representing a correct MPC protocol, a corrupted party set $I \subseteq [n]$ of cardinality at most t, and a heuristic evaluation function Φ, the algorithm can tell whether the protocol is t-perfectly-secure for I or it does not know if the protocol is secure. To fully prove the t-perfect-security of a protocol, we need to enumerate all possible corrupted party $I \subseteq [n]$ of cardinality at most t to call this algorithm. Let Δ be the states (QDFGs) to visit (line 2), and, let $\Delta_{visited}$ be the states (QDFGs) visited (line 3). In each step of the search loop (line 4-17), we choose the most promising state G in Δ, i.e., the state that is minimum evaluated by Φ (line 5). We try all possible vintage transformations on it to discover more states to visit (line 11-16). Line 9-10 are necessary maintenance for Δ and $\Delta_{visited}$. If G is a witness QDFG, we know that there is a simulator for I and the searching can stop (line 6-8). If the searching is exhausted and has never been stopped by a witness, we report UNKNOWN – we are unsure if the protocol is secure for I (line 18).

Algorithm 1 Automated verification algorithm

```
1: procedure TryProving(G_0: n-party QDFG, I ⊆ [n]: corrupted party set, Φ: heuristic evaluation function)
2:     Δ ← {G_0}
3:     Δ_visited ← ∅
4:     while Δ ≠ ∅ do
5:         G ← arg min_Δ Φ
6:         if ^I B_G = ∅ then
7:             return SECURE FOR I                ▷ witness is found
8:         end if
9:         Δ ← Δ \ {G}
10:        Δ_visited ← Δ_visited ∪ {G}
11:        H ← {H | G ⇝^I H}
12:        for all H ∈ H do
13:            if H ∉ Δ_visited then
14:                Δ ← Δ ∪ {H}
15:            end if
16:        end for
17:    end while
18:    return UNKNOWN                             ▷ unsure if secure for I
19: end procedure
```

Generally, the time complexity of this searching algorithm is exponential. However, by smartly and precisely evaluate how promising a state (QDFG) is, it is possible to prove a class of protocols in polynomial time, as we will show in Section 6.1.

If the algorithm returns SECURE for I, we have found a witness graph and a vintage transformation series starting at G_0 and ending at the witness graph, which can be proved by induction on the number of iterations of the while-loop. Thus, Theorem 4.1 ensures the soundness of Algorithm 1.

Theorem 5.1 (soundness of the algorithm)**.** *Let G_0 be an n-party acyclic QDFG, which describes a protocol π that correctly computes an ideal functionality.*

If for each corrupted party set $I \subset [n]$ of cardinality at most t, Algorithm 1 returns t-SECURE FOR I, then π is t-perfectly-secure.

6. Applications of Our Framework

In this section, we will demonstrate how to apply our verification framework, GAuV, to two types of protocols: the BGW protocol [22], which is a classic representative to compute any functionality consisting of addition and multiplication with perfect security, and the B2A (binary domain to arithmetic domain) sharing conversion protocol via daBit [23], which is a typical ingredient in the protocols computed in several different domains.

For each case, we will first show how the protocol runs. Then, we will illustrate how to verify its perfect security by vintage transformations. To utilize equivalent rewriting, an

equivalent production set $\mathcal{P}$ is needed. We will show how to mechanize the intuitive insight of the protocol security into equivalent productions.

Finally, we will further discuss the generalization of our framework beyond these two applications in Section 6.3.

6.1. The BGW Protocol

6.1.1. Protocol Design. Now we briefly introduce the BGW protocol [24]. For simpler presentation, let us assume here the input of each party P_i is only one number x_i. The ideal functionality f can be described by an arithmetic circuit C, consisting of three types of gates: addition, multiplication, and multiplication-by-a-constant. Let $\mathbb{F}$ be a finite field of size greater than n, which accommodates all computations.

Shamir secret sharing. Shamir secret sharing scheme is a basic tool in the BGW protocol. A secret sharing scheme takes a secret number s and produces n shares, with the property that it is possible to reconstruct s from all shares, but any t shares reveal nothing about s. Shamir's secret sharing scheme works as follows. For sharing, a polynomial $q(x)$ of degree t is randomly selected such that its constant term is s. The shares are defined as $q(\alpha_i)$ for every $i \in [n]$, where $\alpha_1, \cdots, \alpha_n$ are n distinct non-zero predetermined values. For reconstructing, the n shares are collected to interpolate the polynomial $q(x) = \sum_{j=1}^{n} \ell^x_{\alpha_j}(\alpha_1, \cdots, \alpha_n) q(\alpha_j)$, where $\ell^{\delta'}_{\delta}(\beta_1, \cdots, \beta_p) \coloneqq \prod_{1 \le j \le p} \frac{\delta' - \beta_j}{\delta - \beta_j}$ is the Lagrange interpolating base. Then, the secret s can be computed as $q(0)$. In the following text, we use $[s]$ to denote a polynomial q where $q(0) = s$, and $[s]^{[1]}, \cdots, [s]^{[n]}$ to denote the shares $q(\alpha_1), \cdots, q(\alpha_n)$.

Phases. The BGW protocol works by having the parties simultaneously emulate the circuit upon Shamir shares rather than the private secrets. The protocol has three phases:

1) **Input sharing stage**: Each party P_i randomly generates a Shamir sharing $[x_i]$ and sends $[x_i]^{[j]}$ to P_j.
2) **Circuit emulation stage**: The parties jointly emulate the circuit gate by gate in topological order. Each gate has one or two input wires and an output wire.
   - *Addition gate*: Let $[x]^{[i]}$ and $[y]^{[i]}$ be the shares of input wires held by party P_i. P_i defines its share of the output wire to be $[z]^{[i]} \coloneqq [x]^{[i]} + [y]^{[i]}$.
   - *Multiplication-by-a-constant gate with a constant c*: Let $[x]^{[i]}$ be the shares of input wires held by party P_i. P_i defines its share of the output wire to be $[z]^{[i]} \coloneqq c \cdot [x]^{[i]}$.
   - *Multiplication gate*: Let $[x]^{[i]}$ and $[y]^{[i]}$ be the shares of input wires held by party P_i. P_i computes $([z]')^{[i]} \coloneqq [x]^{[i]} \cdot [y]^{[i]}$, where $[z]'$ can be seen as a degree-$2t$ polynomial which is a multiplication of two degree-t polynomial $[x]$ and $[y]$.
     To reduce the degree of $[z]'$ from $2t$ to t, P_i randomly generates a Shamir sharing $[([z]')^{[i]}]$ and sends $[([z]')^{[i]}]^{[j]}$ to party P_j, as in the input sharing stage. Upon receiving $[([z]')^{[1]}]^{[i]}, \cdots, [([z]')^{[n]}]^{[i]}$, P_i defines its share of the output wire to be $[z]^{[i]} \coloneqq \sum_{j=1}^{n} [([z]')^{[j]}]^{[i]} \ell^0_{\alpha_j}(\alpha_1, \cdots, \alpha_{j-1}, \alpha_{j+1}, \cdots, \alpha_n)$.
3) **Output reconstruction stage**: Each P_i sends P_k the share $[y_k]^{[i]}$ for the output wire o_k of party P_k. Upon receiving all shares $[y_k]^{[1]}, \cdots, [y_k]^{[n]}$, P_k reconstructs the whole Shamir sharing $[y_k]$ and outputs y_k.

6.1.2. Automated Verification. We first give an intuition of the key to the security of BGW protocols, and then we describe how to mechanize the intuitive insight to provide equivalent productions for GAuV.

Let us analyze how to simulate the views of the corrupted parties, which consists of two parts:

1) the corrupted parties' shares sent from the honest parties at the input sharing stage and the multiplication gates of the circuit emulation stage, and
2) the honest parties' shares sent from the honest parties at the output reconstruction stage.

These two parts of the views of the corrupted parties can be simulated essentially due to the properties of Shamir secret sharing scheme, respectively:

1) any at most t shares can be seen as uniformly random, which indicates that the first part can be regarded as uniformly random;
2) the whole secret sharing can be reconstructed from the secret and any t shares, which indicates that the second part needs to consider the information provided by the corrupted parties' outputs.

The two properties above can be mechanized as two kinds of equivalent productions (see Section B.2 for formal definitions):

1) **sharing production** (see Figure 3): the subgraph that describes a sharing generation procedure can be rewritten as determining the Shamir sharing by the secret and t shares, including the corrupted parties' shares. After rewriting by this production, due to the demand of t random nodes for generating a secret sharing and the randomness amount preservation condition (Definition 4.8), the chosen t shares will become random nodes.
2) **reconstruction production** (see Figure 2): the subgraph that describes a secret reconstruction procedure can be rewritten as determining the Shamir sharing by the secret and t shares, including the corrupted parties' shares.

With the help of the two kinds of equivalent productions above, we can automatically verify the security of all BGW protocols for any functionality. Also, using a well-designed heuristic evaluation function, the search of Algorithm 1 degenerates into a greedy procedure: in each iteration, we go to the next state from the current state through the best vintage transformation until a witness graph is found. Also, the upper bound of the heuristic evaluation function is polynomial size, which indicates the completeness in polynomial time, as formalized in the following theorem.

Theorem 6.1 (completeness for BGW protocols)**.** *Let QDFG G be an n-party BGW protocol as constructed in Figure 7. Consider vintage transformations containing*

equivalent rewriting of equivalent production set $\mathcal{P}$ and tail node elimination. For any corrupted party set $I \subseteq [n]$ of cardinality at most t, there exists a heuristic evaluation function so that Algorithm 1 can prove the t-perfect-security for I in polynomial time.

Theorem 6.1 is a direct corollary of Theorem B.2, whose full proof will be present in Appendix B.

6.2. B2A Conversion via daBit

6.2.1. Protocol Design. This protocol computes an ideal functionality that accepts a secret sharing $[b]_2$ and outputs $[b]_M$. Here, we also consider Shamir secret sharing as the underlying secret sharing scheme, which has been introduced in Section 6.1.1. $[b]_2$ is a Shamir sharing in an extension field of $\mathbb{Z}_2$, and $[b]_M$ is a sharing in a field $\mathbb{Z}_M$, where the sizes of these two fields are both larger than n.

The high-level idea of this protocol is as follows. Let P_1 reconstruct the binary sharing and share in the arithmetic domain. To prevent P_1 from knowing b, we prepare a pair of random sharing in both domains as a mask, add the random mask in the binary domain, and subtract the mask in the arithmetic domain. This mask is called daBit, a random bit r shared as $([r]_2, [r]_M)$.

Protocol 5 shows the details. Additionally, we explain the notation "$\oplus$": $x \oplus y$ means x xor y, where x and y are two bits. Since $x \oplus y = x + y - 2xy$, we can compute $[x \oplus y]_M$ as $[x]_M + [y]_M - 2[xy]_M$, in a way same as the circuit emulation stage of BGW protocols in Section 6.1.1.

Note that the original ideal functionality is randomized since the output is a random Shamir secret sharing $[b]_M$. Unfortunately, randomized functionalities are out of the reach of GAuV. However, we can force the first t parties to provide their desired shares for $[b]_M$. Together with the secret b, these t shares uniquely determine a secret sharing $[b]_M$ via Lagrange interpolation. Also, b can be reconstructed from any $[b]_M$ via Lagrange interpolation. Since the secret should be kept private, we can let all parties perform this linear combination wrapped in the secret sharing scheme. This is what Protocol 5 does in steps 5-8. In this way, we provide a general strategy to adapt the protocols, that output secret sharings, into a deterministic functionality version.

6.2.2. Automated Verification. The security the secret sharing scheme provides is essentially the same as analyzed in Section 6.1.2. Besides, the key to the security is that the reconstructed secret of P_1, $c = b \oplus r$, is fairly random for the adversary, which results from the fact that r is fairly random. To mechanize this insight, we need to provide another equivalent production to rewrite the equation as $r = c \oplus b$. After applying this production, due to the randomness amount preservation condition (Definition 4.8), c will become a random node.

6.3. Discussion on Generalization

Intuitively, our framework is tied with the *hybrid argument* [25], a common proof technique in cryptography to show the indistinguishability between two distributions. The hybrid argument gives a sequence of intermediate distributions, called *hybrids*, and builds up the indistinguishability between the first and last from a chain of indistinguishability between every two adjacent hybrids. In MPC, hybrid arguments are widely applied to show the indistinguishability between the real and ideal worlds.

The search for vintage transformation series is to find a sequence of hybrids that generate the views of corrupted parties and gradually remove the dependency on honest parties' inputs. Therefore, if such a sequence is found, then this sequence can be directly translated to a hybrid argument, showing the security of the protocol. The set of vintage transformations defines a set of possible new hybrids we can move to from the current hybrid. From this perspective, what this work does is to manually provide equivalence rules (how a hybrid can be transformed while preserving the distribution) and automatically search for an equivalence chain of hybrids.

Furthermore, for most MPC protocols with perfect and semi-honest security, which includes the BGW protocol [22], DN protocol [26], or recent new process such as [27] and [28], the transformation between two adjacent hybrids is either an equivalent computation of the same set of values or removing values that are not needed to generate the views of corrupted parties. Thus, such a transformation corresponds to a vintage transformation defined in our framework. Once the needed operable vintage transformations have been identified, our framework is able to find such a proof with sufficient (possibly exponential) running time.

Limitations. *a*) Our framework cannot handle a protocol whose security cannot be proved by any hybrid sequence where the transformation between every two adjacent hybrids corresponds to a vintage transformation. That being said, we do not know whether this kind of protocols exist or not. *b*) Although two protocols are analyzed as references, finding the operable vintage transformations and heuristic evaluation functions for new protocols may still require manual work.

7. Implementation and Evaluation

To show the applicability and scalability of our framework, we implement it as a prototype tool and apply it to the protocols in Section 6. The overall workload involved in the implementation can be approximately gauged by the amount of code: over 2,700 lines of C++ code. To improve efficiency, we employ thread parallelism, with each searching task for every corrupted party set assigned to a separate thread.

Our evaluation assesses all possible settings of the number of parties, n less than 10, and the threshold of corrupted parties, t no more than $\lfloor n/2 \rfloor$, imposing a 1,500 s time limit. Specifically, for the BGW protocols, we generate random circuits of various sizes up to 50 gates. The evaluation is conducted on a workstation with AMD EPYC 7H12 CPU @ 3.3 GHz with 64 cores .

Input: Each party P_i provides a Shamir share $[b]_2^{[i]}$. $P_1, \cdots, P_t$ also provide t values $\beta_1, \cdots, \beta_t$, respectively, to determine the first t shares of the converted Shamir sharing $[b]_M$.
Output: $P_{t+1}, \cdots, P_n$ outputs $\beta_{t+1}, \cdots, \beta_n$, respectively, where $\beta_1, \cdots, \beta_n$ constitutes a Shamir sharing $[b]_M$. where for the first t shares of $[b]_M$ we have $[b]_M^{[1]} = \beta_1, \cdots, [b]_M^{[t]} = \beta_t$.

1. Each party P_i chooses a random bit r_i and shares $[r_i]_2$ and $[r_i]_M$.
2. Each party P_i computes $[r]_2^{[i]} = \sum_{j=1}^{n} [r_j]_2^{[i]}$, $[b \oplus r]_2^{[i]} = [r]_2^{[i]} + [b]_2^{[i]}$ and sends it to P_1.
3. P_1 reconstructs $c = b \oplus r$ from $[b \oplus r]_2$ and shares $[c]_M$.
4. All parties compute $[r]_M = \left[\bigoplus_{j=1}^{n} r_j\right]_M$ and $[b]_M = [c \oplus r]_M$.
5. Each party P_i shares $[b]_M^{[i]}$, i.e., randomly generates a Shamir sharing $[[b]_M^{[i]}]_M$ and sends $[[b]_M^{[i]}]_M^{[j]}$ to party P_j.
6. Each party P_i $(1 \le i \le t)$ shares β_i.
7. Each party P_i computes $[\beta_k]_M^{[i]} = \sum_{j=1}^{t} \ell_{\alpha_j}^{\alpha_k}(0, \alpha_1, \cdots, \alpha_{j-1}, \alpha_{j+1}, \cdots, \alpha_t)[\beta_j]_M^{[i]} + \ell_0^{\alpha_k}(\alpha_1, \cdots, \alpha_t) \sum_{j=1}^{n} \ell_{\alpha_j}^{0}(\alpha_1, \cdots, \alpha_{j-1}, \alpha_{j+1}, \cdots, \alpha_n)[[b]_M^{[j]}]_M^{[i]}$ and sends it to P_k for each $t < k \le n$.
8. Each party P_i $(t < i \le n)$ reconstructs β_i from $[\beta_i]_M$ and outputs β_i.

Protocol 5. Binary to arithmetic sharing conversion via the daBit (deterministic functionality version)

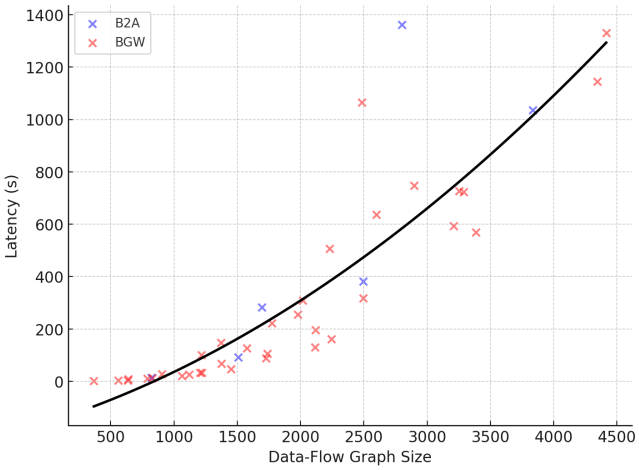

Figure 6. Time cost over data-flow graph size (the number of edges) of multiparty computation protocols, fitted by a quadratic curve.

The evaluation results, as depicted in Figure 6, show that within 1,500 s time limit, our tool can prove the security of protocols cantaining 4,500 edges. Figure 6 also illustrates that the runtime of our tool approximates a quadratic function of the protocol size, aligning with the prediction in Theorem 6.1. This data-flow graph size can be construed as an approximation of the actual execution time of the protocol, under the assumption that all operations could be efficiently carried out. Our evaluation enumerates various configurations across three dimensions: n, the number of parties; t, the threshold of corrupted parties; and C, specifically for BGW protocols, the circuit size. Both n and C are captured by the graph size, which approximately scales as $O(n^2C)$ for the protocols analyzed. On the other hand, t generally has little influence on the time cost due to the thread parallelism, as long as sufficient processor cores are provided.

8. Related Works

We mainly focus on related work in the field of computer-aided verification of the security of MPC protocols. As far as we know, Pettai and Laud [17] propose the first method to automatically verify the security of MPC protocols. However, as discussed in Section 1, they only focus on black-box privacy, a stronger form of security, while our work focuses on perfect security.

There are attempts to build machine-checkable proofs of MPC protocols with interactive provers. Most of them use EASYCRYPT [7]: Stoughton and Varia [29] prove Private Count Retrieval protocol; Almeida et al. [30] prove Yao's protocol; Haagh et al. [31] prove Maurer's protocol; Eldefrawy and Pereira [32] prove the BGW protocol. Besides, Butler et al. [33] prove Oblivious Transfer in CRYPTHOL [34]. Apart from EASYCRYPT and CRYPTHOL, a simpler prover, IPDL [35], is developed atop Coq [36].

9. Conclusion

We propose a sound framework for automated verification of the perfect security of instances of MPC protocols against semi-honest adversaries. We demonstrate the completeness of our framework for BGW protocols. We implement our theoretical framework as a prototype tool, GAuV, and evaluate it on BGW protocols and a B2A conversion protocol. One direction for future work is to evaluate our framework on more protocols and extend it to weaker forms of security. Another direction is to formalize our approach in logic and produce proofs checkable in an interactive prover, e.g., EASYCRYPT. It seems interesting to embed our technique into EASYCRYPT to further improve its automation and ability for simulation-based proofs.

Acknowledgment

The authors thank Elaine Shi for the helpful discussions and suggestions in the early stages.

References

[1] M. Barbosa, G. Barthe, K. Bhargavan, B. Blanchet, C. Cremers, K. Liao, and B. Parno, "Sok: Computer-aided cryptography," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 777–795.

[2] G. Lowe, "Breaking and fixing the Needham-Schroeder Public-Key Protocol using FDR," in *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, ser. TACAs '96. Berlin, Heidelberg: Springer-Verlag, 1996, p. 147–166.

[3] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[4] B. Blanchet, "An efficient cryptographic protocol verifier based on Prolog rules," in *14th IEEE Computer Security Foundations Workshop (CSFW-14)*. Cape Breton, Nova Scotia, Canada: IEEE Computer Society, Jun. 2001, pp. 82–96.

[5] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, P. C. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron, "The AVISPA tool for the automated validation of internet security protocols and applications," in *Computer Aided Verification*, K. Etessami and S. K. Rajamani, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 281–285.

[6] C. J. F. Cremers, "The Scyther tool: Verification, falsification, and analysis of security protocols," in *Computer Aided Verification*, A. Gupta and S. Malik, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 414–418.

[7] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin, "Computer-aided security proofs for the working cryptographer," in *Proceedings of the 31st Annual Conference on Advances in Cryptology*, ser. CRYPTO'11. Berlin, Heidelberg: Springer-Verlag, 2011, p. 71–90.

[8] G. Klein, J. Andronick, K. Elphinstone, T. Murray, T. Sewell, R. Kolanski, and G. Heiser, "Comprehensive formal verification of an OS microkernel," *ACM Trans. Comput. Syst.*, vol. 32, no. 1, feb 2014.

[9] Q. Cao, L. Beringer, S. Gruetter, J. Dodds, and A. W. Appel, "VST-Floyd: A separation logic tool to verify correctness of C programs," *J. Autom. Reason.*, vol. 61, no. 1–4, p. 367–422, jun 2018.

[10] G. Barthe, X. Fan, J. Gancher, B. Grégoire, C. Jacomme, and E. Shi, "Symbolic proofs for lattice-based cryptography," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 538–555.

[11] B. Blanchet and D. Pointcheval, "Automated security proofs with sequences of games," in *Advances in Cryptology - CRYPTO 2006*, C. Dwork, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 537–554.

[12] G. Barthe, J. M. Crespo, B. Grégoire, C. Kunz, Y. Lakhnech, B. Schmidt, and S. Zanella-Béguelin, "Fully automated analysis of padding-based encryption in the computational model," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 1247–1260.

[13] G. Barthe, B. Grégoire, and B. Schmidt, "Automated proofs of pairing-based cryptography," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1156–1168.

[14] G. Bana and H. Comon-Lundh, "A computationally complete symbolic attacker for equivalence properties," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 609–620.

[15] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1175–1191.

[16] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. D'Oliveira, H. Eichner, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konečný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, M. Raykova, H. Qi, D. Ramage, R. Raskar, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, "Advances and open problems in federated learning," *Foundations and Trends in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[17] M. Pettai and P. Laud, "Automatic proofs of privacy of secure multi-party computation protocols against active adversaries," in *IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*, C. Fournet, M. W. Hicks, and L. Viganò, Eds. IEEE Computer Society, 2015, pp. 75–89.

[18] C. Smith, J. Hsu, and A. Albarghouthi, "Trace abstraction modulo probability," *Proc. ACM Program. Lang.*, vol. 3, no. POPL, jan 2019.

[19] O. Goldreich, *Foundations of Cryptography*. Cambridge University Press, 2004, vol. 2, ch. 7, p. 599–764.

[20] Arvind and D. E. Culler, "Dataflow architectures," *Annual Review of Computer Science*, vol. 1, no. 1, pp. 225–253, 1986.

[21] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer, *Fundamentals of Algebraic Graph Transformation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, ch. 2–3, pp. 21–71.

[22] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, ser. STOC '88. Association for Computing Machinery, 1988, p. 1–10.

[23] D. Rotaru and T. Wood, "Marbled circuits: Mixing arithmetic and boolean circuits with active security," in *Progress in Cryptology – INDOCRYPT 2019*, F. Hao, S. Ruj, and S. Sen Gupta, Eds. Cham: Springer International Publishing, 2019, pp. 227–249.

[24] R. Gennaro, M. O. Rabin, and T. Rabin, "Simplified VSS and fast-track multiparty computations with applications to threshold cryptography," in *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 101–111.

[25] M. Fischlin and A. Mittelbach, "An overview of the hybrid argument," Cryptology ePrint Archive, Paper 2021/088, 2021.

[26] I. Damgård and J. B. Nielsen, "Scalable and unconditionally secure multiparty computation," in *Advances in Cryptology - CRYPTO 2007*, A. Menezes, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 572–590.

[27] V. Goyal, H. Li, R. Ostrovsky, A. Polychroniadou, and Y. Song, "ATLAS: Efficient and scalable MPC in the honest majority setting," in *Advances in Cryptology – CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part II*. Berlin, Heidelberg: Springer-Verlag, 2021, p. 244–274.

[28] D. Escudero, V. Goyal, A. Polychroniadou, and Y. Song, "TurboPack: Honest majority MPC with constant online communication," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 951–964.

[29] A. Stoughton and M. Varia, "Mechanizing the proof of adaptive, information-theoretic security of cryptographic protocols in the random oracle model," in *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, 2017, pp. 83–99.

[30] J. B. Almeida, M. Barbosa, G. Barthe, F. Dupressoir, B. Grégoire, V. Laporte, and V. Pereira, "A fast and verified software stack for secure function evaluation," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1989–2006.

[31] H. Haagh, A. Karbyshev, S. Oechsner, B. Spitters, and P.-Y. Strub, "Computer-aided proofs for multiparty computation with active security," in *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, 2018, pp. 119–131.

[32] K. Eldefrawy and V. Pereira, "A high-assurance evaluator for machine-checked secure multiparty computation," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 851–868.

[33] D. Butler, D. Aspinall, and A. Gascón, "Formalising oblivious transfer in the semi-honest and malicious model in CryptHOL," in *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, ser. CPP 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 229–243.

[34] D. A. Basin, A. Lochbihler, and S. R. Sefidgar, "CryptHOL: Game-based proofs in higher-order logic," *Journal of Cryptology*, vol. 33, no. 2, pp. 494–566, 2020.

[35] J. Gancher, K. Sojakova, X. Fan, E. Shi, and G. Morrisett, "A core calculus for equational proofs of cryptographic protocols," *Proc. ACM Program. Lang.*, vol. 7, no. POPL, jan 2023.

[36] The Coq Development Team. (2023, Jul.) The Coq proof assistant. [Online]. Available: https://doi.org/10.5281/zenodo.8161141

Appendix A. Proofs for Operable Vintage Transformation

A.1. Equivalent Rewriting

Theorem A.1. *Given the number of parties n and a corrupted party set $I \subseteq [n]$, considering an equivalent production p, a QDFG G, a QDFG morphism f and a QDFG H constructed from G by equivalent rewriting with equivalent production p on match m, the equivalent rewriting is a vintage transformation w.r.t. I.*

Proof. G and H are acyclic by definition of equivalent rewriting.

Define $h_v \coloneqq \mathrm{id}_{V_G} = \mathrm{id}_{V_H}$. We will prove that h_v satisfies the conditions (i-vi) of Definition 4.4, with an arbitrary assignment γ on V_G^{const} s.t. G is correct w.r.t. γ.

(i-iiii) are naturally ensured since h_v is an identity function.

(v): here, we will prove a stronger statement that $\psi : \alpha \mapsto \alpha \circ h_v$ is a bijection of type $TAS_G^\gamma \to TAS_H^{\gamma \circ h_v}$. Since $h_v = \mathrm{id}_{V_G}$, $\psi = \mathrm{id}_{TAS_G^\gamma}$. It is sufficient to show that $TAS_G^\gamma = TAS_H^\gamma$. In the following, we will simplify this goal step by step until it can be directly proved.

Consider each assignment α on V_G s.t. $\alpha|_{V_G^{const}} = \gamma$. $\alpha \in TAS_G^\gamma$ iff for every $e \in E_G$ we have $\alpha(\overrightarrow{src}^e) \in \mathrm{dom}(op^e) \wedge op^e(\alpha(\overrightarrow{src}^e)) = \alpha(dst^e)$. We abbreviate this property to $P(e, \alpha)$. Then, what we need to prove now is that "$\forall e \in E_G, P(e, \alpha)$" iff "$\forall e \in E_H, P(e, \alpha)$".

By Fact 4.3, there exists a QDFG morphism $g : R \to H$ which is symmetric to f, and, $E_G \setminus f_e(E_L) \subseteq E_G \cap E_H$ and $E_H \setminus g_v(E_R) \subseteq E_G \cap E_H$. Thus, we can simplify our goal as: "$\forall e \in f_e(E_L), P(e, \alpha|_{f_v(V_L)})$" iff "$\forall e \in g_e(E_R), P(e, \alpha|_{g_v(V_R)})$".

Since QDFG morphisms preserve source and destination structure and operations of hyper-edges (Definition 4.7), it is sufficent to prove that "$\forall e \in E_L, P(e, \alpha \circ f_v)$" iff "$\forall e \in E_R, P(e, \alpha \circ g_v)$".

Since QDFG morphisms preserve whether a node is const (Definition 4.7), we have that

$$(\alpha \circ f_v)|_{V_L^{const}} = (\alpha|_{V_L^{const}}) \circ f_v = \gamma \circ f_v.$$

Thus, our goal can be simlified as: "$\alpha \circ f_v \in TAS_L^{\gamma \circ f_v}$" iff "$\alpha \circ g_v \in TAS_R^{\gamma \circ g_v}$". This equivalence is directly indicated by the definition of equivalent production (Definition 4.6).

(vi): As we have proved for (iv), $|TAS_H^\gamma| = |TAS_G^\gamma|$. Since G is correct w.r.t. γ, by Fact 4.2, $|TAS_G^\gamma| = \prod_{r \in V_G^{rand}} |\Gamma_G(r)|$. From the condition of equivalent rewriting (Definition 4.8) that $\sum_{r \in V_G^{rand}} [\Gamma_G(r) = t] = \sum_{r \in V_H^{rand}} [\Gamma_H(r) = t]$ for any $t \in \mathcal{T}$, we have $\prod_{r \in V_G^{rand}} |\Gamma_G(r)| = \prod_{r \in V_H^{rand}} |\Gamma_H(r)|$. Therefore, $|TAS_H^\gamma| = \prod_{r \in V_H^{rand}} |\Gamma_H(r)|$.

And, the number of possible assignments on V_H^{rand} is $\prod_{r \in V_H^{rand}} |\Gamma_H(r)|$. Thus, for each assignment α on V_H^{rand}, if there does not exist a total assignment $\beta \in TAS_H^\gamma$ s.t. $\beta|_{V_H^{rand}} = \alpha$, we will find a contradiction: $|TAS_H^\gamma| < \prod_{r \in V_H^{rand}} |\Gamma_H(r)|$. This contradiction indicates that the non-existence hypothesis is wrong. Thus, we can conclude that H is correct w.r.t. $\gamma \circ h_v$. □

A.2. Tail Node Elimination

Theorem A.2. *Given the number of parties n and a corrupted party set $I \subseteq [n]$, considering a QDFG G, a node $u \in V_G \setminus {}^I V_G$, and a QDFG H constructed from G by tail node elimination of node u, the tail node elimination is a vintage transformation w.r.t. I.*

Proof. As $E_H \subseteq E_G$ and G is acyclic, H is also acyclic.

Define $h_v \coloneqq \mathrm{id}_{V_H}$. We will prove that h_v satisfies the conditions (i-vi) of Definition 4.4, with each assignment γ on V_G^{const} s.t. G is correct w.r.t. γ.

(i-iv) are naturally ensured as h_v is an identity function and $u \notin {}^I V_G$.

The construction of H could give us the following observations:

1. Because $\deg^+(u) = 0$, removing u does not change the indegree of any other nodes. Thus, the only node of different indegrees in G and H is u, which further indicates that $V_H^{rand} = V_G^{rand} \setminus \{u\}$.
2. Because $E_H \subseteq E_G$, i.e., all semantic requirements of a total assignment of H are semantic requirements of a total assignment of G, we have that for each $\beta \in$

TAS_G^γ, $\beta \circ h_v \in TAS_H^{\gamma \circ h_v}$, that is, ψ is a function of type $TAS_G^\gamma \to TAS_H^{\gamma \circ h_v}$.

Now we can prove (vi): consider any assignment α on V_G^{rand}, which means, by observation 1, each assignment on $V_H^{rand} \subseteq V_G^{rand}$ is also considered. According to the correctness of G w.r.t. γ, there exists a total assignment β of G w.r.t. γ s.t. $\beta|_{V_G^{rand}} = \alpha$. According to observation 2, $\beta|_{V_H} \in TAS_H^{\gamma \circ h_v}$ and $\beta|_{V_H^{rand}} = \alpha|_{V_H^{rand}}$. Since this holds for each α, H is correct w.r.t. $\gamma \circ h_v$.

For (v), we consider two cases to prove that $\psi : \alpha \mapsto \alpha \circ h_v$ is k-surjective where k is 1 or $|\Gamma_G(u)|$.

- $u \notin V_G^{rand}$: From observation 2, we have known that $\psi(TAS_G^\gamma) \subseteq TAS_H^{\gamma \circ h_v}$. Now we need to prove that ψ is bijective i.e. 1-surjective.
  (injectivity) Consider any $\alpha_1, \alpha_2 \in TAS_G^\gamma$ s.t. $\psi(\alpha_1) = \psi(\alpha_1)$, i.e., $\alpha_1|_{V_H} = \alpha_2|_{V_H}$. As $u \notin V_G^{rand}$, i.e., $\deg_G^-(u) \neq 0$, there is an $e \in E_G$ s.t. $dst^e = u$, and, $\overrightarrow{src}^e \subseteq V_H$, since G is acyclic. Thus,
  $$\alpha_1(u) = op^e(\alpha_1(\overrightarrow{src}^e)) = op^e(\alpha_2(\overrightarrow{src}^e)) = \alpha_2(u).$$
  Due to $V_G = V_H \cup \{u\}$, we have that $\alpha_1 = \alpha_2$.
  (surjectivity) The correctness of G w.r.t. γ implies Fact 4.2 for G, which is also ensured by (v) for H. So, we can find the equivalence between the amount of total assignments of G and H:
  $$|TAS_G^\gamma| = \prod_{r \in V_G^{rand}} |\Gamma_G(r)| = \prod_{r \in V_H^{rand}} |\Gamma_H(r)| = |TAS_H^{\gamma \circ h_v}|.$$
  Thus, $\psi(TAS_G^\gamma) = TAS_H^{\gamma \circ h_v}$.
- $u \in V_G^{rand}$: u is an isolated node ($\deg_G^+(u) = \deg_G^-(u) = 0$). Thus, for each assignment $\beta \in TAS_H^{\gamma \circ h_v}$, we can construct $\alpha \in TAS_G^\gamma$ s.t. $\alpha|_{V_H} = \beta$, $\alpha(u)$ is arbitrarily chosen from $\Gamma_G(u)$, and $\psi(\alpha) = \beta$. In this way, we can construct $\prod_{r \in V_G^{rand}} |\Gamma_G(r)|$ preimages in all, which are exactly as many as all total assignments in TAS_G^γ. Therefore, $|\psi^{-1}(\beta)| = |\Gamma_G(u)|$, i.e., ψ is $|\Gamma_G(u)|$-surjective.

□

Appendix B. Proofs for Completeness for BGW Protocols

In this section, we give the details about the completeness of Algorithm 1 on BGW protocols. We formalize BGW protocols as QDFGs in Section B.1, provide equivalent productions in Section B.2, and finally show the completeness in Section B.3.

B.1. BGW Protocol as QDFG

Note that BGW protocols are actually a class of protocols (QDFGs), which can be constructed in the same way. What an instance of protocol (QDFG) looks like is dependent on threshold t, algorithmic circuit C and distinct non-zero values $\alpha_1, \cdots, \alpha_n$.

Before formulating the graph structures, we formulate the operators we need:

- *addition* $\mathbf{add} : \mathbb{F}^2 \to \mathbb{F} : (x, y) \mapsto x + y$
- *multiplication-by-a-constant* $\mathbf{mul}_c : \mathbb{F} \to \mathbb{F} : x \mapsto cx$
- *multiplication* $\mathbf{mul} : \mathbb{F}^2 \to \mathbb{F} : (x, y) \mapsto xy$
- *message transit* $\mathbf{mt} : \mathbb{F} \to \mathbb{F} : x \mapsto x$
- *polynomial interpolation and evaluation*
  $$\mathbf{i\&e}^\gamma_{\beta_1,\cdots,\beta_m} : \mathbb{F}^m \rightharpoonup \mathbb{F}$$
  $$: (y_1, \cdots, y_m) \mapsto \sum_{i=1}^{m} y_i \ell^\gamma_{\beta_j}(\beta_1, \cdots, \beta_{j-1}, \beta_{j+1}, \beta_m),$$
  where $\ell^\gamma_\delta(\beta_1, \cdots, \beta_p) := \prod_{1 \le j \le p} \frac{\gamma - \beta_j}{\delta - \beta_j}$ and points $(\beta_1, y_1), \cdots, (\beta_m, y_m)$ fall in a polynomial of degree t.

Now we can formulate BGW protocols as QDFGs in Figure 7. Here, for the sake of clarity of exposition, we focus on the circuit C containing exactly n input wires and n output wires. The modifications to the general case are straightforward.

B.2. Equivalent Productions for Transformation

The equivalent production set $\mathcal{P}$ consists of all equivalent productions of two kinds: sharing productions (see Figure 3) and reconstruction productions (see Figure 2). These productions are mainly about operator $\mathbf{i\&e}$ (polynomial interpolation and evaluation): all participant (source and destination) nodes could be reorganized so that $t+1$ nodes are source nodes and at least one node is the destination.

Remark B.1. *WLOG, the corrupted party set I is assumed as a set of consecutive number starting from* 1.

Definition B.1 (sharing production)**.** *A QDFG pair $p = (L, R)$ is a* sharing production *if*

- $|V_L| = |V_R| = n + t + 1$ *(we denote $V_L = V_R$ by $\{u_0, \cdots, u_t, v_1, \cdots, v_n\}$);*
- $E_L = \bigcup_{i=1}^n \{(\mathbf{i\&e}^{\alpha_i}_{0,\cdots,t}, (u_0, \cdots, u_t), v_i)\}$;
- $E_R = \bigcup_{i=1}^t \{(\mathbf{i\&e}^i_{0,\alpha_1,\cdots,\alpha_t}, (u_0, v_1, \cdots, v_t), u_i)\} \cup \bigcup_{i=t+1}^n \{(\mathbf{i\&e}^i_{0,\alpha_1,\cdots,\alpha_t}, (u_0, v_1, \cdots, v_t), v_i\}$;
- $V_L^{const} = V_R^{const}$ *and* $\rho_L = \rho_R = j \in [n] \setminus I$.

Fact B.1. *A sharing production is an equivalent production.*

Definition B.2 (reconstruction production)**.** *A QDFG pair $p = (L, R)$ is a* reconstruction production *if*

- $|V_L| = |V_R| = 2n + 1$ *(we denote $V_L = V_R$ by $\{u_0, \cdots, u_n, v_1, \cdots, v_n\}$);*
- $E_L = \{(\mathbf{i\&e}^0_{\alpha_1,\cdots,\alpha_n}, (u_1, \cdots, u_n), u_0)\} \cup \bigcup_{i=1}^n \{(\mathbf{mt}, (v_i), u_i)\}$;
- $E_R = \bigcup_{i=t+1}^n \{(\mathbf{i\&e}^{\alpha_i}_{0,\alpha_1,\cdots,\alpha_t}, (u_0, \cdots, u_t), u_i)\} \cup \bigcup_{i=1}^t \{(\mathbf{mt}, (v_i), u_i)\} \cup \bigcup_{i=t+1}^n \{(\mathbf{mt}, (u_i), v_i)\}$;
- $V_L^{const} = V_R^{const}$;
- $\rho_L = \rho_R$ *and* $\rho_L(u_0) = \rho_R(u_0) \in I$ *and* $\rho_L(v_i) = \rho_R(v_i) = i$ *for each* $i \in [n]$.

Fact B.2. *A reconstruction production is an equivalent production.*

1) **The input sharing stage**: Each party P_i contains an input node of x_i and t random nodes for uniformly choosing a polynomial, also, P_i contains n nodes $\beta_{i,1}, \cdots, \beta_{i,n}$ representing n shares of x_i. For each $j \in [n]$, there is an edge of operator $\mathbf{i\&e}^{\alpha_j}_{0,1,\cdots,t}$ taking the input node and random nodes as source and $\beta_{i,j}$ as destination at party P_i.
For each $\beta_{i,j} (i \neq j)$, there is also a node $\beta'_{i,j}$ at party P_j and an edge of operator $\mathbf{mt}$ taking $\beta_{i,j}$ as source and $\beta'_{i,j}$ as destination.
2) **The circuit emulation stage**: Let $g_1, \cdots, g_l$ be a predetermined topological ordering of the gates of circuit C.
For $k = 1, \cdots, l$, party P_i has a node ϵ_i^k to denote the share of output wire of g_k:
   - *Case 1 g_k is an addition gate*: Let γ_i^k and δ_i^k be the nodes of the input wires of g_k held by party P_i. There is an $\mathbf{add}$ edge taking γ_i^k and δ_i^k as source and ϵ_i^k as destination.
   - *Case 2 g_k is a multiplication-by-a-constant gate with constant c*: Let γ_i^k be the node of the input wire of g_k held by party P_i. There is a $\mathbf{mul}_c$ edge taking γ_i^k as source and ϵ_i^k as destination.
   - *Case 3 g_k is a multiplication gate*: Let γ_i^k and δ_i^k be the nodes of the input wires of g_k held by party P_i. For each P_i, there is a node ζ_i^k, and, a $\mathbf{mul}$ edge taking γ_i^k and δ_i^k as source and ζ_i as destination.
   $\zeta_1^k, \cdots, \zeta_n^k$ fall in a polynomial of degree $2t$. To reduce the degree, we need a re-sharing similar to the input sharing stage. Each P_i contains t random nodes and n nodes $\eta_{i,1}^k, \cdots, \eta_{i,n}^k$. For each P_i and $j \in [n]$, there is an edge of operator $\mathbf{i\&e}^{\alpha_j}_{0,1,\cdots,t}$ taking ζ_i^k and t random nodes as source and $\eta_{i,j}^k$ as destination.
   Then we will send the shares. Let $\eta_{i,j}^{k\ '}$ be the nodes of sent shares at party P_j, especially, $\eta_{i,i}^{k\ '} = \eta_{i,i}^k$. For each $i \neq j \in [n]$, there is an edge of operator $\mathbf{mt}$ taking $\eta_{i,j}^k$ as source and $\eta_{i,j}^{k\ '}$ as destination.
   Now we perform a linear combination to reduce the degree of polynomial from $2t$ to t. There are n nodes $\theta_1, \cdots, \theta_n$ and $n-1$ nodes $\iota_1^k, \cdots, \iota_{n-1}^k$, which stores the results of multiplications and additions, respectively, when evaluating the linear combination. Each party P_i contains an edge of operator $\mathbf{mul}_{\ell^0_{\alpha_j}(\alpha_1,\cdots,\alpha_{j-1},\alpha_{j+1},\cdots,\alpha_n)}$ taking $\eta_{j,i}^{k\ '}$ as source and θ_j^k as destination for each $j \in [n]$. Also, party P_i contains an edge of operator $\mathbf{add}$ taking θ_j^k and θ_{j+1}^k as source and ι_j^k as destination for each $1 \leq j < n$.
   Let the node of output wire $\epsilon_i^k = \iota_{n-1}^k$.
3) **The output reconstruction stage**: Let $o_1, \cdots, o_n$ be the output wires of circuit C, where party P_i's output is the value on wire o_i. For every $j \in [n]$ denote by $\kappa_{i,j}$ the node of shares that parties hold for o_i, and, we use $\kappa'_{i,j}$ to denote the node of sent shares. Thus, there are $\mathbf{mt}$ edges taking $\kappa_{i,j}$ as source and $\kappa'_{i,j}$ as destination ($i \neq j$). And we simply let $\kappa'_{i,i} = \kappa_{i,i}$.
Each party P_i contains an output node ω_i. The edge to produce ω_i has an operator $\mathbf{i\&e}^0_{\alpha_1,\cdots,\alpha_n}$ and source nodes $\kappa_{i,1}, \cdots, \kappa_{i,n}$.

Figure 7. BGW Protocol as QDFG

B.3. Completeness for BGW Protocols

First, we give a heuristic evaluation function by measuring the distance of a QDFG to a witness graph. Formally, we define the *BGW potential function* Φ on QDFGs to be the tuple of:

1) The number of bubbles, i.e., $\left|{}^I\mathcal{B}_G\right|$.
2) The number of nodes at honest parties that can reach corrupted parties. A node reaching corrupted parties means that there exists a path from this node to some node at corrupted parties.
3) The number of nodes in G, i.e., $|V_G|$.

Φ functions will be compared in the lexicographical order.

Using the Φ as the heuristic evaluation function, we can prove that, for a BGW protocol, Algorithm 1 degenerates into a steepest descent optimization procedure: in each iteration, we choose the best state transformed from the last iteration until a witness graph is found. Furthermore, in Algorithm 1 the chosen graphs in all iterations consist of a normal transformation series. We formalize and prove the above intuitional idea in the following theorem.

Theorem B.1. *Let QDFG G_0 be an n-party BGW protocol as constructed in Figure 7, and, let $I \subseteq [n]$ be a set of corrupted parties s.t. $|I| \leq t$. Consider vintage transformations containing equivalent rewriting of equivalent production set $\mathcal{P}$ and tail node elimination. For any vintage transformation series $G_0 \overset{I}{\rightsquigarrow} G_1 \cdots \overset{I}{\rightsquigarrow} G_m$ s.t. for each $1 \leq i \leq m$,*

a) $\Phi(G_i) < \Phi(G_{i-1})$, and

b) for any G'_i s.t. $G_{i-1} \overset{I}{\rightsquigarrow} G'_i$, $\Phi(G_i) \leq \Phi(G'_i)$,

one of the followings is satisfied:

1) ${}^I\mathcal{B}_{G_m} = \emptyset$,

2) there exists a graph G_{m+1} s.t. $G_m \overset{I}{\rightsquigarrow} G_{m+1}$ and $\Phi(G_{m+1}) < \Phi(G_m)$.

Proof. This proof could be roughly divided into two parts: (1) construct a class of normal transformation series from G_0 to a QDFG without bubble; (2) prove that given $G_0 \overset{I}{\rightsquigarrow} \cdots \overset{I}{\rightsquigarrow} G_m$ must be a prefix of normal transformation series.

First, notice that there is a vintage transformation series starting at G_0 consisting of the following fragments:

1) equivalent rewritings of reconstruction production for the output reconstruction stage at corrupted parties I;
2) equivalent rewritings of sharing production, for the input sharing stage and all multiplication gates in the circuit emulation stage, at honest parties $[n] \setminus I$;
3) tail node eliminations of
   - output node ω_i and all output shares $\kappa'_{i,j}$ $(i \neq j)$ at honest parties $i \in [n] \setminus I$,
   - the nodes of gates that cannot reach output wires $o_1, \cdots, o_{|I|}$, at honest parties $[n] \setminus I$,

- the input nodes at honest parties, and
- all the rest nodes at parties $P_{t+1}, \cdots, P_n$.

After the constructed vintage transformation, the bubbles are all resolved. Because

- in fragment 1, the reconstruction production has been applied for the output node ω_i at each corrupted party, which indicates the in-degree of ω_i has become zero,
- there is no equivalent production to provide any in-edge for the input node at corrupted parties, and,
- in fragment 3, all the input nodes at honest parties have been eliminated.

We can also show that our constructed vintage transformation series satisfies (a) and (b), i.e., along each vintage transformation in the series, the evaluation function Φ decreases with the largest possible difference.

1) In G_0, at each corrupted party $i \in I$, an equivalent rewriting of the reconstruction production for bubble node ω_i sets its in-degree to zero and resolves it. This action reduces the bubble count, the first entry of Φ, and is the only transformation that decreases this first entry until all ω_i bubbles are resolved.
2) Once every ω_i $(i \in I)$ bubble is resolved, only honest parties' input nodes remain as bubbles. But before we eliminate any node, there is no vintage transformation to resolve them. Therefore, in the second fragment, the first entry of Φ cannot be decreased by any vintage transformation. Now it is sufficient to show applying sharing production decreases the second entry of Φ.
   As depicted in Figure 7, applying sharing production converts t random nodes into zero-out-degree nodes, which could previously reach corrupted parties via share nodes and $\mathbf{mt}$ edges, decreasing the second entry of Φ by at least t.
   Note that sharing production is limited to input sharing and multiplication gates in the circuit emulation stage, and tail node elimination, which removes zero-out-degree nodes at honest parties, cannot impact corrupted parties. Consequently, no other vintage transformations affect the second entry of Φ.
3) After fragment 2, no equivalent rewriting can decrease Φ, as reconstruction and sharing productions can be applied only once, which reverses $\mathbf{i}\&\mathbf{e}$ edges. Node elimination merely shrinks the graph without introducing new patterns, so no equivalent rewriting can be performed to decrease Φ in fragment 3.
   We have shown that the only possible vintage transformation in fragment 3 is tail node elimination, which indeed decreases the Φ tuple: it lowers the first entry if a bubble node is eliminated (otherwise unchanged); it doesn't affect the second entry since the eliminated node's out-degree is zero and doesn't reach corrupted parties; and it always decreases the third entry by one.

Note that the order of transformations in a fragment is not precisely specified. What we construct is indeed a class of vintage transformation series. We call them *normal transformation series*.

Second, assume the opposite, that $G_0 \overset{I}{\rightsquigarrow} \cdots \overset{I}{\rightsquigarrow} G_m$ is not a prefix of any normal transformation series. Let $k \leq m$ be the minimal number s.t. $G_0 \overset{I}{\rightsquigarrow} \cdots \overset{I}{\rightsquigarrow} G_k$ is not a prefix of any normal transformation series, which indicates $G_0 \overset{I}{\rightsquigarrow} \cdots \overset{I}{\rightsquigarrow} G_{k-1}$ is a prefix of some normal transformation series. Let G_k^* be the next graph after G_{k-1} in this normal transformation series. We will prove that $\Phi(G_k^*) < \Phi(G_k)$, which contradicts the requirement (b) of $G_0 \overset{I}{\rightsquigarrow} \cdots \overset{I}{\rightsquigarrow} G_m$. Consider which fragment $G_{k-1} \overset{I}{\rightsquigarrow} G_k^*$ belongs to:

1) In fragment 1, the input nodes of honest parties have non-zero out-degree, which indicates they cannot be removed by tail node elimination. Thus, the only possibly resolved bubbles are the output nodes of corrupted parties, which can only be resolved by applying reconstruction production. Therefore, $G_{k-1} \overset{I}{\rightsquigarrow} G_k$ cannot decrease the amount of the bubbles, while $G_{k-1} \overset{I}{\rightsquigarrow} G_k^*$ resolves a bubble.
2) In fragment 2, observe that no reconstruction production could be applied. If $G_{k-1} \overset{I}{\rightsquigarrow} G_k$ is not an equivalent rewriting of sharing production, it must be a tail node elimination. Since the input nodes of honest parties in fragment 2 have non-zero out-degree, they cannot be removed by tail node elimination. Thus, $G_{k-1} \overset{I}{\rightsquigarrow} G_k$ cannot decrease the first entry of Φ, but can only decrease the third entry of Φ, while $G_{k-1} \overset{I}{\rightsquigarrow} G_k^*$ decreases the second entry of Φ.
3) In fragment 3, recall that no equivalent rewriting could be performed. This fragment in normal transformation series contains all possible tail node eliminations. Thus, it is impossible that $G_0 \overset{I}{\rightsquigarrow} \cdots \overset{I}{\rightsquigarrow} G_k$ is not a prefix of any normal transformation series.

We have proved that $G_0 \overset{I}{\rightsquigarrow} \cdots \overset{I}{\rightsquigarrow} G_m$ is a prefix of some normal transformation series. Thus, if G_m is not the end of the series, there exists a graph G_{m+1} in the series s.t. $G_m \overset{I}{\rightsquigarrow} G_{m+1}$; otherwise, we have shown that all bubbles must have been resolved after a normal transformation series, i.e., ${}^I\mathcal{B}_{G_m} = \emptyset$. □

Note that each entry of $\Phi(G)$ is $O(|G|)$. Thus, Theorem B.1 indicates that the length of a normal transformation series is also $O(|G|)$. Induction on the length of the normal transformation series, with the help of Theorem B.1, one can finally prove the completeness of Algorithm 1 for BGW protocols with polynomial time.

Theorem B.2 (completeness for BGW protocols)**.** *Let QDFG G be an n-party BGW protocol as constructed in Figure 7. Consider vintage transformations containing equivalent rewriting of equivalent production set $\mathcal{P}$ and tail node elimination. For any corrupted party set $I \subseteq [n]$ of cardinality at most t, With the BGW potential function as the heuristic evaluation function, Algorithm 1 can prove the t-perfect-security for I in polynomial time.*

Appendix C.
Meta-Review

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

C.1. Summary

The paper describes a technique for automatically verifying secure MPC protocols in the perfect security model, assuming a semi-honest attacker. The idea is to view the protocol as a (quasi-)data-flow graph, which is rewritten iteratively using graph transformations until it is in a form that corresponds to a PPT simulator of the protocol in the idealized model. The paper provides a prototype of the framework for BGW protocols and another protocol, B2A.

C.2. Scientific Contributions

- Provides a Valuable Step Forward in an Established Field

C.3. Reasons for Acceptance

1) The paper presents a new approach to verifying MPC protocols.
2) The approach is fully automated in a tool for BGW and B2A protocols.
3) The approach also promises to generalize to other MPC protocols.