Authenticated Garbling from Simple Correlations

Samuel Dittmer[1][0000−0003−0018−6354], Yuval Ishai[2], Steve Lu[1][0000−0003−1837−8864], and Rafail Ostrovsky[1,3][0000−0002−1501−1330]

[1] Stealth Software Technologies, Inc.
[2] Technion - Israel Institute of Technology
[3] University of California, Los Angeles

Abstract. We revisit the problem of constant-round malicious secure two-party computation by considering the use of *simple correlations*, namely sources of correlated randomness that can be securely generated with sublinear communication complexity and good concrete efficiency.
The current state-of-the-art protocol of Katz et al. (Crypto 2018) achieves malicious security by realizing a variant of the *authenticated garbling* functionality of Wang et al. (CCS 2017). Given oblivious transfer correlations, the communication cost of this protocol (with 40 bits of statistical security) is comparable to roughly 10 garbled circuits (GCs). This protocol inherently requires more than 2 rounds of interaction.
In this work, we use other kinds of simple correlations to realize the authenticated garbling functionality with better efficiency. Concretely, we get the following reduced costs in the random oracle model:

- Using variants of both vector oblivious linear evaluation (VOLE) and multiplication triples (MT), we reduce the cost to 1.31 GCs.
- Using only variants of VOLE, we reduce the cost to 2.25 GCs.
- Using only variants of MT, we obtain a *non-interactive* (i.e., 2-message) protocol with cost comparable to 8 GCs.

Finally, we show that by using recent constructions of pseudorandom correlation generators (Boyle et al., CCS 2018, Crypto 2019, 2020), the simple correlations consumed by our protocols can be securely realized without forming an efficiency bottleneck.

1 Introduction

Practical protocols for low-latency secure 2-party computation typically rely on Garbled Circuits (GC) [23]. Such protocols have constant round complexity, online communication proportional to the input size, total communication proportional to the circuit size, and good computational cost. We revisit the question of concretely efficient GC-based protocols with malicious security, which has been the topic of a long line of work originating from [16,15]. The authenticated garbling approach of Wang et al. [20] and Katz et al. [14] gives the state-of-the-art protocols along this line. This approach relies on oblivious transfers for a cut-and-choose based implementation of a preprocessing functionality made up of a collection of authenticated wire labels.

This work is motivated by recent techniques for securely generating simple forms of correlated randomness [3,5,18,4,6,22,7], which make it feasible to explore practical alternatives to constructions based only on OTs. In this work, we give three new constructions, including a non-interactive secure computation (NISC) protocol [13], which use simple correlations that can be securely generated with sublinear communication complexity and good concrete efficiency.

Protocol	**Correlation**	**Cost (garbled circuits)**	
		Dep. + online	Total
WRK [20]	OT	2.5	11.0
KRRW [14] v1	OT	1.5	7.75
KRRW [14] v2	OT	1	9.7
KRRW [14] with VOLE	$\mathcal{F}_{\mathsf{VOLE}}$	1	2.5
KRRW [14] with SPDZ	MT	1	7
KRRW [14] with SPDZ and cert. VOLE	MT-$\mathcal{F}_{\mathsf{VOLE}}$-$\mathcal{F}_{\mathsf{subVOLE}}$	1	2.9
Ours, v1 (KRRW with $\mathcal{F}_{\mathsf{DAMT}}$ compiler to $\mathcal{F}_{\mathsf{pre}(\kappa)}$)	$\mathcal{F}_{\mathsf{DAMT}}$-$\mathcal{F}_{\mathsf{subVOLE}}$-$\mathcal{F}_{\mathsf{VOLE}}$	**1**	**1.31**
Ours, v2	$\mathcal{F}_{\mathsf{bVOLE}}$-$\mathcal{F}_{\mathsf{subVOLE}}$-$\mathcal{F}_{\mathsf{VOLE}}$	**1.47**	**2.25**
NISC in the single-execution setting			
Ours, v3	$\mathcal{F}_{\mathsf{OLE}}$	**8**	**8**
AMPR14 [1]	CRS	40	40

Table 1. Communication complexity for evaluating a large circuit after a "silent" randomness generation step, as a ratio to the cost of a semi-honest garbled circuit. The bucket size for KRRW is set to $B = 3$, which is a lower bound for circuits of size less than 2^{ρ}. Dep. + online communication refers to the higher of the two party's one-way *circuit-dependent* communication cost, including online and offline phase costs. The total column adds in the cost of circuit-independent offline communication.

Our approach achieves significant savings over the approach of [14], reducing the total communication cost from around 10 semi-honest GCs to 1.31 GCs in our first protocol (comparing to the size of half-gates garbled circuits in both cases). Our second protocol uses a compressed preprocessing functionality that is expensive to generate for small circuits, but outperforms [14] in the large circuit setting, requiring only 2.25 GCs and using only simple "VOLE-type" correlations (see §1.1).

Our third protocol is non-interactive (NISC) and achieves comparable communication complexity (8 GCs) than the variant of [14] with round complexity proportional to the circuit depth, and roughly 5x the communication efficiency of the best NISC protocols [1] in the single execution setting.

Part of our advantage comes from swapping out less efficient ways of generating correlated randomness with recent advantages. For example, a large part of the cost of [14] comes from their methods of generating an *authenticated bits functionality*, which can be realized without any communication given two instances of vector oblivious linear evaluation (VOLE), defined in § 1.1. But our

main advantage comes from novel compilers from new forms of simple correlated randomness to authenticated garbling functionality, including the use of efficient generalizations of *certified VOLE* protocols (see §3.2) that allow verification across more of the verification work to be done under statistical security instead of computational security (see § 4.2). As we show in Table 1, our most efficient protocol still uses roughly 2x less communication than [14] would use, even if we replaced their authenticated bits generation procedure with VOLE.

Alternatively, the SPDZ protocol [10] could be used to realize the preprocessing functionality of [14] with authenticated multiplication triples (MTs) in a black box way. Doing this would require 7 GCs. Applying our certified randomness optimization of § 4.2 to this SPDZ approach would reduce communication to 2.9 GCs, which is still more than both our non-NISC variants.

As we further discuss below, the secure generation of the correlated randomness required by our protocols is typically cheaper than the protocol that consumes it, especially for VOLE-type correlations or when using multiple cores. Moreover, this secure generation is circuit-independent and only involves local computation without any interaction.

1.1 Simple correlations

Our informal definition of a *simple correlation* is one that can be securely generated with sublinear communication complexity and good concrete efficiency. The cost of sending a GC in the semi-honest setting is already linear in the circuit size, and so will dominate the communication cost of setting up the randomness, and any reasonably efficient randomness protocol can be run on multiple cores in the background faster than the communication of the main protocol.

We note that all of the flavors of simple correlations discussed here can be realized with a one-time setup step that generates randomness seeds. These seeds can then be expanded into the full correlated randomness locally by each party. This property facilitates running these protocols in a streaming mode, where the randomness is unpacked as needed. To draw attention to this, and to simplify the presentation, we write $\mathsf{Extend}(\mathcal{F})$ to denote unpacking additional entries from the correlated randomness seeds. Additionally, this one-time setup can be performed non-interactively, which we need to make step 2 of Figure 12 non-interactive for our NISC protocol. We describe these properties more formally as part of an ideal functionality for the correlation calculus in the full version of this paper.

We rely on two main flavors of simple correlations: vector oblivious linear evaluation (VOLE)-type correlations, and multiplication triple (MT)-type correlations. In VOLE, a receiving party learns $\mathbf{v} := \mathbf{a}\beta + \mathbf{c}$ along with the scalar β, while the sending party learns $\mathbf{a}, \mathbf{c}$. VOLE with sublinear communication complexity was introduced by Boyle et al. [3] in 2019 and has been improved since then, see [7] for the most efficient current variant.

In MT, parties learn shares of vectors $\mathbf{x}, \mathbf{y}$ along with shares of the piecewise product $\mathbf{z}$, $z_i = x_i \cdot y_i$. MT have been studied as an important primitive for

Functionality	$\mathcal{F}$-notation	Mathematical relation	Cost comparison
VOLE-type correlations			
Vector OLE	$\mathcal{F}_{\mathsf{VOLE}}$	$\mathbf{v} = \mathbf{a}\beta + \mathbf{c}$, for $\mathbf{a}, \mathbf{c} \in \mathbb{F}_{2^\rho}$	1 VOLE
Subfield Vector OLE	$\mathcal{F}_{\mathsf{subVOLE}}$	$\mathbf{v} = \mathbf{a}\beta + \mathbf{c}$, for $\mathbf{a} \in \mathbb{F}_2, \mathbf{c} \in \mathbb{F}_{2^\rho}$	≈ 0.6 VOLE
Block Vector OLE	$\mathcal{F}_{\mathsf{bVOLE}}$	$\mathbf{v}_i = \mathbf{a}\beta_i + \mathbf{c}_i$, for $i = 1 \ldots, L$	L VOLE
MT-type correlations			
Two-sided authenticated multiplication triples	$\mathcal{F}_{\mathsf{DAMT}}$	Choose $x \cdot y = z$, then share $[x], [y], [z], [\alpha z], [\beta z]$	2 MT
Programmable OLE	$\mathcal{F}_{\mathsf{OLE}}$	$\mathbf{v}_{i,j} = \mathbf{a}_i \cdot \boldsymbol{\beta}_j + \mathbf{c}_{i,j}$ for $(i,j) \in Q$	$\lvert Q \rvert$ MT

Table 2. Correlated randomness used throughout the paper. For programmable OLE, the set Q is an arbitrary set of ordered pairs of indices. Cost comparison is given with reference to the "base" randomness protocol, either VOLE or MT. Generating 1 million entries of VOLE costs roughly 0.05 seconds on standard computers. Generating 1 million entries of MT costs roughly 10 seconds.

years, e.g. [10] but only recently have been able to be generated efficiently and silently [6].

We require several variants of these two types of randomness, as summarized in Table 2. We define all non-standard correlations as functionalities where they arise in the presentation. Crucially, both flavors of randomness generation allow for "progammability" in such a way that each new variant does not require an entirely new protocol, see e.g. [4,6].

Indeed, we can think of VOLE-type and MT-type correlations in terms of simple atomic operations under a "correlation calculus". For VOLE, atomic operations consist of choosing a vector $\mathbf{v} \in F^n$, for some field F, multiplying $\mathbf{v}$ by a scalar β (possibly in an extension field E), sending a vector to a party, and secret-sharing a vector between parties. Taking $F = E = \mathbb{F}_{2^\rho}$ or $\mathbb{F}_{2^\kappa}$ gives standard VOLE, taking $F = \mathbb{F}$ and $E = \mathbb{F}_{2^\rho}$ gives subfield VOLE. Reusing the vector $\mathbf{v}$ with a set of scalars β_i gives block VOLE and block subfield VOLE.

For MT-type correlations, atomic operations consist of picking a random vector $\mathbf{x} \in F^n$, computing the scalar product $\beta\mathbf{x}$, computing the point-wise product $\mathbf{x} \cdot \mathbf{y}$, sending a vector to a party, and sharing a vector between parties. Standard authenticated triples come from computing $\mathbf{z} := \mathbf{x} \cdot \mathbf{y}$ and $\beta\mathbf{z}$ and sharing all four vectors. Our *two-sided authentication triples* come from additionally computing $\alpha\mathbf{z}$, and sharing this as well.

Finally, programmable OLE consists of a family of OLE vectors $\mathbf{v}_{i,j} = \mathbf{a}_i\boldsymbol{\beta}_j + \mathbf{c}_{i,j}$, where the parties agree to re-use certain vectors $\mathbf{a}_i$ and $\boldsymbol{\beta}_j$ on certain entries. The generation time and seed size of programmable OLE scales linearly with the number of pairs (i,j) for which we generate a vector of OLE entries.

The VOLE protocol of [7] can generate a million entries of VOLE correlations in roughly 0.05 seconds, or a million entries of subfield VOLE in roughly 0.03 seconds. The OLE protocol of [6] can generate a million OLE correlations in roughly 10 seconds. For each of these protocols, the dominant cost is the secret sharing of vectors. We therefore expect that block VOLE over L instances costs

roughly L times as much computation as a single VOLE, that standard authenticated triples costs two times as much communication as OLE, and two-sided authenticated triples cost three times as much.

We remark here that the Ring-LPN approach only allows silent generation of authenticated multiplication triples over large fields of characteristic 2 such as $\mathbb{F}_{2^\rho}$. If authenticated triples could be silently generated over $\mathbb{F}_2$, then the preprocessing functionality of [14] could be generated with only 2 bits of communication per gate, via a procedure similar to that given in Lemma 3. It is precisely because there is no simple correlation that can generate the preprocessing functionality directly that the question of the most efficient compiler from simple correlations to that functionality arises.

1.2 Notation

We let f be a function realized by a circuit C, where C is made up of input gates $\mathcal{I}$, boolean gates $\mathcal{G}$, and output gates $\mathcal{O}$. Let the input $\mathcal{I} = \mathcal{I}_A \cup \mathcal{I}_B$ be held by two parties A and B, and define n to be the number of AND gates in $\mathcal{G}$, and $m = |\mathcal{I}| + |\mathcal{G}|$, including all gates in m.

We use κ and ρ as a computational and statistical security parameter, respectively, and take $\kappa = 128$ and $\rho = 40$ for our concrete communication metrics.

During the evaluation of a garbled circuit, we write z_i for the true value of a wire, λ_i for the wire mask, and share λ_i among A and B as $\lambda_i = a_i \oplus b_i$. We use $(\oplus, \wedge)$ for field addition and multiplication over $\mathbb{F}_2$, any of $(\oplus, +, -)$ for field addition over larger fields of characteristic 2, and $\cdot$ or concatenation for field multiplication over larger fields of characteristic 2.

We use α, β for VOLE receiver inputs over $\mathbb{F}_{2^\rho}$ held by A, B respectively, and Δ_A for a VOLE receiver input held by A over $\mathbb{F}_{2^\kappa}$.

When discussing randomness certification in § 3.2, we need to distinguish between an instance of $\mathcal{F}_{\mathsf{VOLE}}$ where party A is the receiver and party B the sender with another instance of $\mathcal{F}_{\mathsf{VOLE}}$ with the roles reversed. In this instance, we refer to the latter functionality as $\mathcal{F}_{\mathsf{ELOV}}$.

1.3 Our contribution

Our first protocol relies on both VOLE-type and MT-type correlations. It employs the same authenticated garbling technique as that in [14], but uses authenticated triples over $\mathbb{F}_{2^\rho}$, rather than cut-and-choose techniques, to generated authenticated wire labels. This construction relies on a new compiler from a special flavor of authenticated triples to the desired preprocessing functionality given in §4.1, as well as a lightweight compiler from preprocessing with statistical security to preprocessing with computational security, given in §4.2.

Theorem 1. *There is a protocol that securely computes f against malicious adversaries in the RO$-\mathcal{F}_{\mathsf{DAMT}}-\mathcal{F}_{\mathsf{VOLE}}-\mathcal{F}_{\mathsf{subVOLE}}$-hybrid model with the following features:*

- **Online Communication:** $O(\kappa(|\mathcal{I}| + |\mathcal{O}|))$.
- **Circuit Dependent Communication:** $(2\kappa + 2)n$ *bits of communication.*
- **Total Communication:** $(2\kappa + 2\rho + 2)n$ *(one-way) or* $(2\kappa + 4\rho + 2)n$ *(two-way) plus terms sublinear in* n.
- **Computation:** $O(\kappa n)$.

Our second protocol relies only on VOLE-type correlations, and a modification of the authenticated garbling protocol that, approximately, uses a garbling approach from [20] to replace the authentication procedure in [14]. We give this modified garbling protocol and prove its correctness in §5.1.

This modified approach increases the communication cost of the online plus circuit dependent step, but allows the use of a simple *block VOLE* functionality instead of one of the more computationally intensive PCGs used to build authenticated triples. As written, the protocol uses quasi-linear work instead of linear work, but this can be reduced to linear work by dividing the gates into blocks of some large fixed size, and running the compressed preprocessing functionality $\mathcal{F}_{\mathsf{cp}}$ on each block in parallel.

This approach is best suited to the large circuit setting, since it requires $L \approx \rho \log |C|$ instances of VOLE (or for sufficiently large N and $|C| > N$, $L = |C| \frac{\rho \log N}{N}$), in order to construct the compressed functionality $\mathcal{F}_{\mathsf{cp}}$. Because VOLE-type correlations are so much more efficient, the computation of the randomness generation for this protocol is roughly comparable to that of the first protocol, but the communication of the VOLE seeds is much larger.

Theorem 2. *There is a protocol that securely computes* f *against malicious adversaries in the* $RO{-}\mathcal{F}_{\mathsf{VOLE}}{-}\mathcal{F}_{\mathsf{subVOLE}}{-}\mathcal{F}_{\mathsf{bVOLE}}$*-hybrid model with the following features:*

- **Online Communication:** $O(\kappa(|\mathcal{I}| + |\mathcal{O}|))$.
- **Circuit Dependent Communication:** $(2\kappa + 3\rho)n$ *bits of communication.*
- **Total Communication:** $(2\kappa + 8\rho + 1)n + o(n)$.
- **Computation:** $O(\kappa n \log n)$ *or* $O(\kappa n)$ *with running* $\mathcal{F}_{\mathsf{cp}}$ *on blocks.*

Our third protocol relies only on MT-type correlations. It uses a similar preprocessing functionality and authenticated garbling protocol as our first protocol, but combines them into a (single-use) NISC protocol. These protocols require certain modifications in order to make them non-interactive. In particular, we require a conditional disclosure of secrets (CDS) functionality to allow the receiver to authenticate their inputs without communication to the prover. We give the details in §6.1.

Theorem 3. *There is a NISC protocol that securely computes* f *against malicious adversaries in the* $RO - \mathcal{F}_{\mathsf{OLE}}$*-hybrid model with the following features:*

- **Online Communication:** $O(\kappa(|\mathcal{I}| + |\mathcal{O}|))$.
- **Circuit Dependent Communication:** $(2\kappa + 3\rho)n$ *bits of communication.*
- **Total Communication:** $16\kappa n + o(n)$ *(one-way) or* $(29\kappa + 3\rho)n + o(1)$ *(two-way).*

– **Computation:** $O(\kappa n)$.

We expect the first and third protocols to be dominant in the secure 2PC and NISC settings, respectively, in the million gate setting and the second protocol to be competetive around ten million gates.

1.4 Structure of paper

In Section 2, we give an overview of the construction of [14], and explain how this construction can be treated as a blueprint pattern for a family of authenticated garbling constructions. We then describe, at a high-level, how each level of the blueprint is modified for each of our three protocols. In Section 3 we describe a series of technical results about certified VOLE, combining correlated randomness functionalities, and conditional disclosure of secrets. Each of these results serve the same general purpose of allowing one party to authenticate that their inputs are well-formed to the other party. We give some additional protocols and proofs in Appendix A. We then give our three protocols $\mathbf{\Pi}_{\mathsf{2pc}}^{\mathsf{DAMT}}$, $\mathbf{\Pi}_{\mathsf{2pc}}^{\mathsf{VOLE}}$ and $\mathbf{\Pi}_{\mathsf{2pc}}^{\mathsf{NISC}}$ in Sections 4, 5, 6, respectively.

2 Authenticated garbling: blueprints and variations

We will present the authenticated garbling protocols in this paper as three different constructions following the same general blueprint design. The protocols can be pictured as a series of structures built side-by-side with the same number of *levels*, and corresponding levels play a similar role in each protocol. We begin by reviewing the approach of [14] through this framework, and then go into more detail about how our approaches differ.

2.1 Review: The authenticated garbling blueprint of KRRW [14]

Authenticated shared bits. The first level of the construction is an authenticated shared bits functionality. In [14], this functionality is presented through the language of IT-MACs. We offer an equivalent definition in the language of simple correlations: The authenticated shared bits functionality is a pair of implementations of $\mathcal{F}_{\mathsf{subVOLE}}$, the first instance is over $\mathbb{F}_{2^\rho}$, with party A acting as sender and B acting as receiver, so that B receives $\beta \in \mathbb{F}_{2^\rho}$, A receives $\mathbf{a} \in \mathbb{F}_2^m$ and $\mathbf{c} \in \mathbb{F}_{2^\rho}^m$, and B receives $\mathbf{v} := \mathbf{a}\beta + \mathbf{c}$. In the second instance, the roles reversed and the $\mathcal{F}_{\mathsf{subVOLE}}$ is given over $\mathbb{F}_{2^\kappa}$, so that A receives $\alpha \in \mathbb{F}_{2^\kappa}$, B receives $\mathbf{b} \in \mathbb{F}_2^m$ and $\mathbf{d} \in \mathbb{F}_{2^\kappa}^m$, and A receives $\mathbf{w} := \mathbf{b}\alpha + \mathbf{d}$.

These shares will play the role of the wire masks in Yao's garbled circuits. For the i-th wire, party B will learn the value $a_i \oplus b_i \oplus z_i$, where z_i is the true wire value under a plaintext evaluation of the circuit. Because the value a_i is unknown to B, B learns nothing from this value. Because the value b_i is unknown to A, A is unable to employ a selective-failure attack to deduce which row of the garbled table B is attempting to read.

Authenticated parallel AND. To make the protocol secure against a malicious A, party B needs to be able to verify that the row of the garbled table B is reading from was constructed correctly. In order to do this, the parties augment the authenticated bit randomness above with authenticated shares of the bits $(a_i \oplus b_i) \wedge (a_j \oplus b_j)$, for every AND gate $\mathcal{G}_k := (i, j, k, \wedge)$, as shown in Figure 1.

This construction requires two stages. The first stage we call *authenticated parallel AND*. Let $\mathsf{PAnd}(n)$ be a circuit consisting of n AND gates executed in parallel, so that the kth gate has input wires $(2k-1, 2k)$ and output wire $2n+k$. To simplify notation, we write $\mathcal{F}_{\mathsf{pre}(\kappa)}$ for $\mathcal{F}_{\mathsf{pre}}^{(\mathsf{PAnd}(n),\kappa,\rho)}$ and $\mathcal{F}_{\mathsf{pre}(\rho)}$ for $\mathcal{F}_{\mathsf{pre}}^{(\mathsf{PAnd}(n),\rho,\rho)}$ where n is clear from context. In [14], the parties realize the preprocessing functionality in the special case of $\mathcal{F}_{\mathsf{pre}(\kappa)}$. Equivalently, they construct authenticated multiplication triples with entries in $\mathbb{F}_2$; as remarked above, there is no simple correlation that can generate these triples silently.

In [14], these triples are generated using cut-and-choose techniques, which makes up the lion's share of the *circuit-independent* communication cost of that protocol.

Remark 1. We note that, as well as translating the language of $\mathcal{F}_{\mathsf{pre}}$ in [14] from IT-MACs to VOLE, we now require that if A holds an input bit, B's share of that input bit's wire mask is 0, and vice versa. This does not alter the security of the protocol but it simplifies some of the proofs.

Fig. 1. Authenticated wire labels

Functionality $\mathcal{F}_{\mathsf{pre}}^{(C,\rho,\kappa)}$: Pre-processing of wire labels for authenticated garbling.

Parametrized by values ρ, κ, and a circuit C consisting of $\mathcal{W}$ wires, $\mathcal{I}$ input wires, $\mathcal{O}$ output wires, and gates $\mathcal{G}$ of the form (i, j, k, T), for $T \in \{\wedge, \oplus\}$, $i, j \in \mathcal{I} \cup \mathcal{W}$, and $k \in \mathcal{W} \cup \mathcal{O}$. Recall that $m := |\mathcal{I}| + |\mathcal{G}|$.

- A chooses $\alpha \in \mathbb{F}_{2^\kappa}$ and wire labels $\mathbf{a} \in \mathbb{F}_2^m$, $\mathbf{c} \in \mathbb{F}_{2^\rho}^m$ and sends them to $\mathcal{F}_{\mathsf{pre}}$.
- B chooses $\beta \in \mathbb{F}_{2^\rho}$ and wire labels $\mathbf{b} \in \mathbb{F}_2^m$, $\mathbf{d} \in \mathbb{F}_{2^\kappa}^m$ and sends them to $\mathcal{F}_{\mathsf{pre}}$.
- For each input wire $i \in \mathcal{I}$, if $i \in \mathcal{I}_A$, set $b_i := 0$, and if $i \in \mathcal{I}_B$, set $a_i := 0$.
- For each gate $\mathcal{G} = (i, j, k, T)$, in topological order:
  - If $T = \oplus$, $\mathcal{F}_{\mathsf{pre}}$ sets the values $a_k = a_i + a_j$, $b_k = b_i + b_j$, $c_k = c_i + c_j$, and $d_k = d_i + d_j$, where the addition is performed in the appropriate field of characteristic 2.
  - If $T = \wedge$, $\mathcal{F}_{\mathsf{pre}}$ chooses values $\hat{a}_k$ uniformly at random from $\mathbb{F}_{2^\rho}$, $\hat{c}_k$ uniformly at random from $\mathbb{F}$, $\hat{d}_k$ uniformly at random from $\mathbb{F}_{2^\kappa}$, and $\hat{b}_k = (a_i + b_i) \cdot (a_j + b_j) + \hat{a}_k$.
- $\mathcal{F}_{\mathsf{pre}}$ computes

$$(\mathbf{v}, \hat{\mathbf{v}}, \mathbf{w}, \hat{\mathbf{w}}) = (\mathbf{a}\beta + \mathbf{c}, \hat{\mathbf{a}}\beta + \hat{\mathbf{c}}, \mathbf{b}\alpha + \mathbf{d}, \hat{\mathbf{b}}\alpha + \hat{\mathbf{d}}).$$

- $\mathcal{F}_{\mathsf{pre}}$ sends $(\mathbf{v}, \hat{\mathbf{v}}, \hat{\mathbf{b}}, \hat{\mathbf{d}})$ to B and $(\mathbf{w}, \hat{\mathbf{w}}, \hat{\mathbf{a}}, \hat{\mathbf{c}})$ to A.

Authenticated circuit wires. The second step is to convert this generic preprocessing $\mathcal{F}_{\mathsf{pre}(\kappa)}$, which serves the parallel AND gate circuit only, to the circuit-dependent preprocessing $\mathcal{F}_{\mathsf{pre}}^{(C,\rho,\kappa)}$. In other words, we now want shares of the bit $(a_i \oplus b_i) \wedge (a_j \oplus b_j)$ for arbitrary pairs of indices (i, j), and $a_i \oplus b_i$, $a_j \oplus b_j$ may in turn represent the XOR of several prior bits.

This conversion is done using standard Beaver triple techniques [2], as we show below in §4.2. In one variant of [14] the triples are instead constructed "in-place", which gives a modified construction with less total communication, but some additional communication in the circuit-dependent phase. The main result of [14] can now be re-stated as follows:

Theorem 4 ([14]). *The KRRW protocol [14] securely computes a functionality f against malicious adversaries in the RO-$\mathcal{F}_{\mathsf{pre}}$-hybrid model, with $2\kappa + 2$ bits of communication per AND gate, $\kappa + 1$ bits of communication per input gate, and 1 bit of communication per output gate.*

Authenticated garbling. The authenticated garbling protocols of both [20] and the follow-up work [14] are both instructive here. After the authenticated circuit wire labels are completed, party A plays the role of the sender in a semi-honest evaluation of Yao's garbled circuit, and some additional interaction allows B to verify the correctness of the opened entry of each AND gate.

For an AND gate $\mathcal{G}_k := (i, j, k, \wedge)$, let $\hat{a}_k, \hat{b}_k$ be the authenticated bit shares of $(a_i \oplus b_i) \wedge (a_j \oplus b_j)$, and let $\lambda_k := a_k \oplus b_k$, with $\hat{\lambda}_k$ defined similarly. If both parties know the value $(\lambda_i \oplus z_i)$, where z_i is the true value of the wire, then they can locally construct authenticated bit shares of

$$z_i \wedge z_j \oplus \lambda_k = \lambda_k \oplus \hat{\lambda}_k \oplus (z_i \oplus \lambda_i)\lambda_j \oplus (z_j \oplus \lambda_j)\lambda_i \oplus (z_i \oplus \lambda_i) \wedge (z_j \oplus \lambda_j).$$

From there, B evaluates the garbled circuit, A securely opens their bit share of $z_i \wedge z_j \oplus \lambda_k$, and B verifies that the value $z_i \wedge z_j \oplus \lambda_k$ is equal to the wire label $z_k \wedge \lambda_k$ computed from garbled circuit evaluation.

The primary distinction between [20] and [14] is how the value of $\lambda_i \oplus z_i$ is computed. In [20], party A computes all four possibilities of $(\lambda_i \oplus z_i, \lambda_j \oplus z_j)$, with the accompanying shares of $z_i \wedge z_j \oplus \lambda_k$. They then construct what are essentially two garbled circuits. The first garbled circuit, used for evaluation, uses computational security to hide gate labels from B. The second garbled circuit, used for authentication, hides only the masked wire labels $z_i \oplus \lambda_i$ and the accompanying share of $z_i \wedge z_j \oplus \lambda_k$, and uses statistical security to stop A from flipping a bit of the masked wire label. In [20], the first garbled circuit requires 3κ communication per gate, and the second requires 4ρ bits of communication.

In the [14] protocol, the first circuit is improved to 2κ bits of communication by applying the half-gate technique of Zahur et al. [24], and the second circuit is replaced with one more round of communication wherein B opens all masked wire labels to A, and A then batches together the proof of correct garbling on the traveled path.

Remark 2. A recent advance due to Rosulek and Roy [17] reduces the cost of semi-honest garbled circuits to $1.5\kappa+5$ bits per AND gate and is compatible with

free XOR. A natural question is whether the approach of [14] can be extended to this new "three-halves" garbled circuit construction. We hope the answer is yes, although there are some obstacles to overcome.

In the [17] construction, the gates and wire labels are "sliced and diced" into half labels, but there is no canonical way for the evaluator to perform a linear combination of these half labels and compute the output wire's half labels. Instead, the desired linear combination is *garbling-dependent*, and randomized and encrypted in such a way that the evaluator learns the desired linear combination without learning anything about the garbling. In the [14] paradigm, the garbler cannot know the garbling, and naturally, it is harder to randomize and encrypt something you do not know. We leave the study of this question to future work.

2.2 New Ideas: Authenticated shared bits

We now go through the levels of this blueprint again, this time explaining the changes that each of our three protocols make to the pattern laid out above. First, for authenticated shared bits, as mentioned above, two instances of $\mathcal{F}_{\mathsf{subVOLE}}$ are sufficient to generate this randomness, and we use exactly this for our first protocol, $\mathbf{\Pi}_{\mathsf{2pc}}^{\mathsf{DAMT}}$.

For the protocol using only VOLE-type correlations, $\mathbf{\Pi}_{\mathsf{2pc}}^{\mathsf{VOLE}}$, we introduce a complication. We now generate all wire tags b_i as a (public) linear combination of entries of a vector $\widetilde{b}$ of wire tags. The length of $\widetilde{b}$ is $O(\rho \log n)$. This allows us to generate shares of values $a_i \wedge b_j$ as a linear combination of values $a_i \wedge \widetilde{b}_{j'}$, which can in turn be represented as entries of VOLE.

To ensure that security against a malicious A remains, we have to verify that we are still protected against selective failure attacks. Following the protocol of [20], we do not allow A to learn the values $z_i \oplus \lambda_i$, and instead send a second garbled circuit that allows B to learn $z_i \oplus \lambda_i$ and the accompanying share of $z_i \wedge z_j \oplus \lambda_k$. If A corrupts only a single gate, then by the randomness of $\widetilde{b}$, A will learn nothing from an abort. However, if A corrupts more gates, the values b_i may be linearly related, and so A could learn something from whether or not B aborts. However, with an appropriate choice of parameters, the values b_i will only be linearly related if A has corrupted so many gates that an abort is inevitable.

We note that a similar approach that generates the vector $\mathbf{a}$ as a linear transformation of a shorter vector $\widetilde{a}$ (i.e. $\mathbf{a} = M_H \widetilde{a}$) would be insecure. Indeed, any vector $\mathbf{w}$ in the (non-empty) left kernel of M_H is orthogonal to $\mathbf{a}$. B must learn the values $z_i \oplus \lambda_i$ in order to evaluate the circuit, and can then subtract their share to obtain $z_i \oplus a_i$. Taking the dot product of $\mathbf{a} \oplus \mathbf{z}$ with $\mathbf{w}$ gives $\mathbf{w} \cdot \mathbf{z}$, and B has broken the zero-knowledge property of the secure computation.

Finally, for the NISC protocol $\mathbf{\Pi}_{\mathsf{2pc}}^{\mathsf{NISC}}$, we can not realize an instance of $\mathcal{F}_{\mathsf{subVOLE}}$ where B is the sender and A is the receiver non-interactively. Instead, we let one of A's inputs to programmable OLE be the vector $\overline{\boldsymbol{\alpha}} := (\alpha, \alpha, \ldots, \alpha)$, and then B's input $\mathbf{b}$ intended for $\mathcal{F}_{\mathsf{subVOLE}}$ can instead be given to $\mathcal{F}_{\mathsf{OLE}}$.

2.3 Authenticated parallel AND

For our first protocol, $\mathbf{\Pi}_{\mathsf{2pc}}^{\mathsf{DAMT}}$, we construct authenticated parallel AND gates from doubly authenticated multiplication triples in two steps. First, we convert from $\mathcal{F}_{\mathsf{DAMT}}^{(\rho,n)}$ to $\mathcal{F}_{\mathsf{pre}(\rho)}$ using a construction inspired by Beaver triples, see § 4.2. This conversion requires 2ρ bits of communication per AND gate.

We then convert from $\mathcal{F}_{\mathsf{pre}(\rho)}$ to $\mathcal{F}_{\mathsf{pre}(\kappa)}$, that is, from preprocessing for parallel AND gates over $\mathbb{F}_{2^\rho}$ to parallel AND gates where bits held by party B are authenticated over $\mathbb{F}_{2^\kappa}$ instead of $\mathbb{F}_{2^\rho}$, using a lightweight protocol that requires only $3 + o(1)$ bits per AND gate. This can be done with semi-honest security using the usual compiler from random to fixed subfield VOLE (see e.g. [3]). To make this secure against malicious B, B must convince A that the bits used for this instance of fixed $\mathcal{F}_{\mathsf{subVOLE}}$ match the authenticated bits generated by $\mathcal{F}_{\mathsf{pre}(\rho)}$. We give a lightweight protocol for this authentication in §4.2.

For our VOLE-only protocol, we instead use the block VOLE construction ($\mathcal{F}_{\mathsf{bVOLE}}$) to obtain bit shares of the product $(a_{2i-1} \oplus b_{2i-1}) \wedge (a_{2i} \oplus b_{2i})$ term by term. Party A holds the bit $a_{2i-1} \wedge a_{2i}$ locally, and can use this value as an entry of its authenticated bits constructed above, and verify its correctness under LPZK. Likewise party B holds the bit $b_{2i-1} \wedge b_{2i}$ locally and can authenticate and verify under LPZK. The cross terms $a_{2i-1} \wedge b_{2i}$ and $a_{2i} \wedge b_{2i-1}$ are linear combinations of terms of the form $a_{2i-1} \wedge \widetilde{b}_j$ and $a_{2i} \wedge \widetilde{b}_j$, respectively, and so bit shares of these terms can be obtained from the block VOLE.

In order to obtain *authenticated* shares, we also need to generate shares of $(a_i \wedge \widetilde{b}_j)\beta$. To do this, we double the size of B's input to the block VOLE, so that B's inputs are $\widetilde{b}_j, \widetilde{b}_j\beta$. (For security reasons, we need to shift all of B's inputs by a random value γ, which is an additional input. We give the details in § 5.2 and Appendix B.1). To verify that B's inputs satisfy the correct relation, B passes their inputs to an instance of $\mathcal{F}_{\mathsf{VOLE}}$, playing the role of Sender, and proves correctness under LPZK.

For technical reasons, our protocol does not guarantee that a cheating A is detected immediately, but instead ensures that, if A cheats, A corrupts their own share of $\hat{b}_i\alpha$, which will then be detected during the evaluation of the garbled circuit with overwhelming probability.

Because of the linear dependence on B's bits, this is no longer a realization of $\mathcal{F}_{\mathsf{pre}(\rho)}$. We define a modified functionality $\mathcal{F}_{\mathsf{cp}}$ and show that the converter from $\mathcal{F}_{\mathsf{pre}(\rho)}$ to $\mathcal{F}_{\mathsf{pre}(\kappa)}$ can likewise convert from $\mathcal{F}_{\mathsf{cp}}^{(\rho)}$ to $\mathcal{F}_{\mathsf{cp}}^{(\kappa)}$.

For our NISC protocol, we follow the same approach as in the VOLE-only protocol to produce shares of $(a_{2i-1} \oplus b_{2i-1}) \wedge (a_{2i} \oplus b_{2i})$ and $(a_{2i-1} \oplus b_{2i-1}) \wedge (a_{2i} \oplus b_{2i})\beta$, term by term. As discussed above, the parties have to generate authenticated bits through a call to $\mathcal{F}_{\mathsf{OLE}}$ instead of $\mathcal{F}_{\mathsf{subVOLE}}$. To generate the pairwise products $b_{2i} \wedge a_{2i-1}$ and $b_{2i-1} \wedge a_{2i}$, and so-on, we re-use A's input $\mathbf{a}$ to the $\mathcal{F}_{\mathsf{OLE}}$ functionality, and pair it with a new vector $\mathbf{b}'$, which reverses the order of every pair (b_{2i-1}, b_{2i}).

Because the protocol is non-interactive, B cannot prove anything about their inputs to A (in the CRS model, this would require a CRS generated by A and

a message from B to A before A's final message from A to B for the secure computation, giving a 3 round protocol). Instead, A and B use a lightweight conditional disclosure of secrets protocol (CDS) which ensures that either B's inputs are well-formed or A's message to B in the NISC protocol appears uniformly random to B. We sketch the protocol briefly here, and describe it in more detail in § 6.1.

For the CDS protocol, parties A and B generate an instance of $\mathcal{F}_{\mathsf{OLE}}$ with A's input the vector $\overline{\boldsymbol{\alpha}} := (\alpha, \alpha, \ldots, \alpha)$, and B's input the vector $\overline{\boldsymbol{\beta}} := (\beta, \beta, \ldots, \beta)$. Call the resulting shares $(\mathbf{v}, \mathbf{c})$, so that if both parties are honest, we have $v_i + c_i = \alpha\beta$ for all i. Then likewise $v_1 - v_i = c_1 - c_i$ for all i if both parties are honest, and are otherwise offset by a term unknown to the cheating party.

Let the vector $\mathbf{s} := (c_i - c_i)$ be held by A and the vector $\mathbf{t} := (v_1 - v_i)$ be held by B. Then A adds $H(\mathbf{s})$ to all future messages, B subtracts $H(\mathbf{t})$ from all future messages. if B cheats, B will be unable to construct $\mathbf{s}$, and so A's messages will appear random.

Similar protocols are used to guarantee that the vector $\mathbf{b}'$ really holds the desired re-ordering of $\mathbf{b}$, and that all necessary polynomial relations on $\mathbf{b}$ hold. We give more detail in § 6.1.

We note that our converters from authenticated gates over ρ to authenticated gates over κ (i.e. the conversion from $\mathcal{F}_{\mathsf{pre}(\rho)}$ to $\mathcal{F}_{\mathsf{pre}(\kappa)}$, and related protocols) can no longer be applied in the NISC setting because this protocol requires opening certain shared values publicly, and thus is interactive. This is one of the reasons that our NISC protocol requires more communication than our other two protocols.

2.4 Authenticated circuit wires

For our first interactive protocol, $\boldsymbol{\Pi}_{\mathsf{2pc}}^{\mathsf{DAMT}}$, the converter from $\mathcal{F}_{\mathsf{pre}(\kappa)}$ to $\mathcal{F}_{\mathsf{pre}}^{(C,\kappa,\rho)}$ follows the approach of [14]. We give the protocol converting from $\mathcal{F}_{\mathsf{pre}(\kappa)}$ to $\mathcal{F}_{\mathsf{pre}}^{(C,\kappa,\rho)}$ in § 4.2. For our VOLE-based protocol $\boldsymbol{\Pi}_{\mathsf{2pc}}^{\mathsf{VOLE}}$, we give instead build $\mathcal{F}_{\mathsf{cp}}^{(C,\rho,\rho)}$ directly and convert from that functionality to $\mathcal{F}_{\mathsf{cp}}^{(C,\kappa,\rho)}$. We describe these conversions in § 5.2.

For our NISC protocol, we define a modified functionality $\mathcal{F}_{\mathsf{pre-wbc}}^{(C,\rho,\kappa)}$ which is similar to the functionality $\mathcal{F}_{\mathsf{pre}}$, but has the property from $\mathcal{F}_{\mathsf{cp}}$ that a cheating A is not immediately detected but corrupts their own shares. We observe that the protocol sketched above for obtaining authenticated parallel AND gates from authenticated bits can be used to obtain authenticated wires for an arbitrary circuit. Instead of swapping b_{2i-1} and b_{2i} in a second input vector to $\mathcal{F}_{\mathsf{OLE}}$, we have one input vector $\mathbf{b}_L$ to the $\mathcal{F}_{\mathsf{OLE}}$ of all left inputs b_i to gates $\mathcal{G}_k = (i, j, k, \wedge)$, and a second input vector $\mathbf{b}_R$ of all right inputs b_j. The same techniques are used to ensure that $\mathbf{b}_L$ and $\mathbf{b}_R$ hold the correct linear transformations of $\mathbf{b}$.

2.5 Authenticated garbling

For our first protocol, we can use the authenticated garbling protocol of [14] directly, once the functionality $\mathcal{F}_{\mathsf{pre}}^{(C,\rho,\kappa)}$ has been realized, with a small modifi-

cation to the step where the initial gate labels are determined to account for our small modification to $\mathcal{F}_{\mathsf{pre}}^{(C,\rho,\kappa)}$ where we allow a party's wire mask zero when the other party knows the true wire value. The protocol still requires, as in [14], $2\kappa + 2$ bits of offline circuit dependent communication per AND gate.

For our VOLE-only protocol, we can no longer use the authentication approach of [14] where B reveals to A the masked wire labels $z_i \oplus \lambda_i = z_i \oplus a_i \oplus b_i$. Of course, A can XOR these shares by the values a_i that A holds, leaving $z_i \oplus b_i$, and, because the values b_i are computed as linear combinations of some shorter vector $\widetilde{b}$, there is some linear combination of the $z_i \oplus b_i$ terms that causes the b_i terms to cancel identically, and A would learn some linear relation on the vector $\mathbf{z}$ of true wire values.

Instead, we combine the techniques of [20] with Zahur's half-gate techniques, so that B can open exactly one authenticated bit, corresponding to $(z_i \wedge z_j) \oplus \lambda_k$, for the k-th multiplication gate. This requires only statistical security, since the output is only used for verification, and does not play the role of a gate label for an output wire. On the other hand, since the output is being used for verification, we can no longer allow a term $H(L_{i,0}, k) \oplus H(L_{j,0}, k)$ to be added to the output, so we need to send an additional element of $\mathbb{F}_{2^\rho}$ as part of the garbled table. In total, the authenticated garbling requires $2\kappa + 3\rho$ bits of offline circuit dependent communication per AND gate.

In our NISC protocol, we also cannot have party B revealing masked wire labels to A, because that would require additional rounds of communication. We use the same approach as in our VOLE-only protocol, but need to show additional care to verify that the protocol can be made non-interactive. We give the details in §6.2 and Appendix B.5.

3 Authenticating correlated randomness

Before we proceed with a technical description of our main protocols, we give an overview of the techniques related to correlated randomness we use throughout the rest of the paper.

3.1 Compilers from "random" to "fixed" randomness variants

There is a standard compiler from random VOLE to fixed VOLE (see e.g. [3]) that allows parties to replace a randomly selected vector $\mathbf{v} := \mathbf{a}\beta + \mathbf{c}$, where all entries are chosen randomly, with a new vector $\mathbf{v}' := \mathbf{a}'\beta' + \mathbf{c}'$, where $\mathbf{a}', \mathbf{c}'$ are chosen by the sender, β' is chosen by the receiver, and the receiver additionally learns $\mathbf{v}'$ given above. The conversion protocol can be stated simply: the receiver sends $\beta' - \beta$ to the sender, the sender sends $\mathbf{a}' - \mathbf{a}$ and $\mathbf{c}' - \mathbf{c} + (\beta' - \beta) \cdot \mathbf{a}'$ to the receiver, and both parties adjust their shares locally. In cases where the sender does not need to control the value of $\mathbf{c}'$, the sender sends only $\mathbf{a}' - \mathbf{a}$, and sets their pair of vectors to $(\mathbf{a}', \mathbf{c} - (\beta' - \beta) \cdot \mathbf{a})$.

We can use this same compiler with block VOLE, where a vector $\mathbf{a}$ is used across several instances of VOLE. To replace a random $\mathbf{a}$ with a fixed vector $\mathbf{a}'$, party A only needs to send the message $\mathbf{a}' - \mathbf{a}$ once across all instances.

A similar compiler exists for a batch of OLE correlations $\mathbf{v} := \mathbf{ab} + \mathbf{c}$, where one party sends $\mathbf{a}' - \mathbf{a}$, the other sends $\mathbf{b}' - \mathbf{b}$, and both parties compute locally to obtain $\mathbf{v}' := \mathbf{a}'\mathbf{b}' + \mathbf{c}'$. As with block VOLE, if the random vector $\mathbf{a}$ is used in multiple instances of programmable OLE, a single message suffices to convert this vector to $\mathbf{a}'$ across all instances.

For a careful accounting of round complexity, we note that, when the value of $\mathbf{c}$ can be chosen randomly, these messages can be sent concurrently or in sequence, in either order. If one party does not require fixed inputs, that party does not need to send a message at all.

3.2 Certification between varieties of correlated randomness

Recall the "correlation calculus" introduced in §1.1, that allows us to express each of our randomness functionalities in terms of a short list of atomic operations. This same "correlation calculus" allows us to re-use vectors and scalars across distinct flavors of correlated randomness as long as they are of the same type (that is, VOLE-type or MT-type).

For example, if we wish to have an instance of $\mathcal{F}_{\mathsf{VOLE}}$ and an instance of $\mathcal{F}_{\mathsf{subVOLE}}$ using the same value β but different vectors $\mathbf{a}, \mathbf{a}'$, then we generate $\mathbf{a}, \mathbf{a}'$ randomly, multiple each vector by β, and share each of the results over the desired field. Similar approaches allow us to use the same vector and different values β, β', and can also be applied to use the same vectors or values between instances of $\mathcal{F}_{\mathsf{subVOLE}}$ or $\mathcal{F}_{\mathsf{VOLE}}$ over different (top-level) fields.

By combining this with the previous observation about compilers from random to fixed VOLE and OLE, we can allow any vector or scalar to be used as an input to any instance of $\mathcal{F}_{\mathsf{VOLE}}$, $\mathcal{F}_{\mathsf{subVOLE}}$, or $\mathcal{F}_{\mathsf{bVOLE}}$.

There are three situations that are not covered by this approach, for which we require bespoke protocols. Each of them work by extending the randomness instances with fresh randomness and evaluating some short polynomial expression on the outputs, which will produce equal outputs for both parties if and only if the desired equality condition holds. A random oracle is applied to the outputs and then the results are compared; any number of certifications of this form can be batched together by applying the random oracle to the collection of outputs.

First, in Section 4 we wish to authenticate that the same value α is used in a call to $\mathcal{F}_{\mathsf{VOLE}}$ and a call to $\mathcal{F}_{\mathsf{DAMT}}$. These are generated by different "correlation calculuses", and it would be a massive efficiency hit to generate $\mathcal{F}_{\mathsf{VOLE}}$ as MT-type randomness. We give a lightweight protocol $\mathbf{\Pi}_{\mathsf{cert}}^{\mathsf{DAMT}\wedge\mathsf{VOLE}}$ in Appendix A.1

Second, in Section 5, we wish to show that, for two calls to VOLE with the parties switching between the role of receiver and sender, the constant value β used by one party in their role as receiver matches another value b used by the same party while playing the role of the sender. We give a lightweight protocol $\mathbf{\Pi}_{\mathsf{cert}}^{\mathsf{VOLE}\wedge\mathsf{ELOV}}$ in Appendix A.2.

Third, in Sections 4 and 5, we wish to certify that two instances of subfield VOLE with different receiver inputs α, Δ_A over different fields $\mathbb{F}_{2^\rho}$, $\mathbb{F}_{2^\kappa}$ have the same vector inputs $\mathbf{b}$, even if one vector is generated via the compiler from

random to fixed VOLE, and another is generated using an unspecified possibly interactive protocol. We give a lightweight protocol $\mathbf{\Pi}_{\mathsf{cert}}^{\rho\wedge\kappa}$ in Appendix A.3.

3.3 Line Point Zero Knowledge

In [11], Dittmer, Ishai and Ostrovsky introduced *Line Point Zero Knowledge*, or LPZK, a protocol for building a NIZK for general circuits using a single instance of VOLE. When working in the random oracle model on circuits corresponding to low degree polynomials, LPZK is especially powerful, because many verifications can be batched together. As shown in [21], any number of polynomials on a total of n inputs of degree at most d can be verified with communication of $(n+d)\kappa$ bits communication. For completeness, and because we use similar arguments elsewhere in this paper, we sketch the argument here.

A prover P wishes to convince a verifier V that P holds inputs $\mathbf{a} = (a_i)$ such that $g(\mathbf{a}) = 0$. Each input a_i becomes the entry of a VOLE $v_i = a_i\beta + c_i$, and V evaluates $g(\mathbf{v})$, which will be a polynomial in β of degree at most $d-1$ if P is telling the truth. After masking these values with an oblivious polynomial evaluation of degree $d-1$, P opens the coefficients and V confirms the desired equality. In the ROM, many such checks can be batched together, with V computing $\sum g(\mathbf{v})H(\mathbf{m};i)$ and P computing the coefficients of $\sum g(\mathbf{a}t+\mathbf{c})H(\mathbf{m};i)$, where $\mathbf{m}$ represents some message transcript committing P to the values $\mathbf{a}$, and i is the index representing the number of times we've evoked this batch check.

This construction includes the cost of the compiler from random VOLE to fixed VOLE. In our case, where we wish to prove relations on an already set fixed VOLE, we can omit the $n\kappa$ bits of communication, and send only $d\kappa$ bits. In this paper, we exclusively apply LPZK to the setting where we wish to prove that already set VOLE inputs satisfy some collection of polynomials of degree d, and take $d \leq 3$ throughout. We write $\mathbf{\Pi}_{\mathsf{LPZK}}(\mathbf{a}, \mathbf{c}, \beta, \mathbf{v}, \mathcal{R})$ for the protocol that proves that $\mathbf{a}$ satisfies the set of relations $\mathcal{R}$, when one party holds $(\mathbf{a}, \mathbf{c})$ and the other party holds β and $\mathbf{v} := \mathbf{a}\beta + \mathbf{c}$.

4 Authenticated garbling from authenticated garbled triples

We follow the blueprint laid out in Section 2, giving the full protocol description and proofs. Recall that in Figure 1, we gave the a preprocessing functionality $\mathcal{F}_{\mathsf{pre}}^{(C,\rho,\kappa)}$ used in the constructions of [20] and [14]. Let $\mathsf{PAnd}(n)$ be a circuit consisting of n AND gates executed in parallel, so that the kth gate has input wires $(2k-1, 2k)$ and output wire $2n+k$. Recall that we write $\mathcal{F}_{\mathsf{pre}(\kappa)}$ for $\mathcal{F}_{\mathsf{pre}}^{(\mathsf{PAnd}(n),\kappa,\rho)}$ and $\mathcal{F}_{\mathsf{pre}(\rho)}$ for $\mathcal{F}_{\mathsf{pre}}^{(\mathsf{PAnd}(n),\rho,\rho)}$.

4.1 From authenticated bits to parallel AND with authenticated triples

The underlying correlated randomness we need for our protocol is subfield VOLE for generating authenticated bits, VOLE, for running proofs of input correctness under LPZK, and doubly authenticated multiplication triples, for converting from authenticated bits to authenticated parallel AND.

Doubly authenticated multiplication triples can be generated from Ring-LPN under the "correlation calculus" discussed in §1.1. This correlated randomness is nonstandard, although it can be viewed as a modified form of the authenticated triples of SPDZ [10]. We give the functionality formally in Figure 2. We then prove the following lemma, which shows how to generate authenticated bits and how to convert these bits to authenticated parallel AND gates.

Fig. 2. Two-sided authenticated triples

Functionality $\mathcal{F}_{\mathsf{DAMT}}^{(\rho,n)}$: Two-sided authenticated triple generation

Parametrized by values $\rho, n \in \mathbb{N}$.

- A chooses $\alpha \in \mathbb{F}_{2^\rho}$ and sends α to $\mathcal{F}_{\mathsf{DAMT}}$.
- B chooses $\beta \in \mathbb{F}_{2^\rho}$ and sends β to $\mathcal{F}_{\mathsf{DAMT}}$.
- $\mathcal{F}_{\mathsf{DAMT}}$ samples vectors $(\mathbf{x}, \mathbf{y})$ uniformly at random from $\mathbb{F}_{2^\rho}^n$.
- $\mathcal{F}_{\mathsf{DAMT}}$ sets $\mathbf{z} := \mathbf{x} \cdot \mathbf{y}$, where the multiplication is done element-wise.
- $\mathcal{F}_{\mathsf{DAMT}}$ generates random shares $(\mathbf{x}_{A,1}, \mathbf{y}_{A,1}, \mathbf{z}_{A,1})$ and $(\mathbf{x}_{B,1}, \mathbf{y}_{B,1}, \mathbf{z}_{B,1})$ of the vectors $(\mathbf{x}, \mathbf{y}, \mathbf{z})$, with random shares chosen in $\mathbb{F}_{2^\rho}$.
- $\mathcal{F}_{\mathsf{DAMT}}$ generates random shares $(\mathbf{x}_{A,2}, \mathbf{y}_{A,2}, \mathbf{z}_{A,2})$ and $(\mathbf{x}_{B,2}, \mathbf{y}_{B,2}, \mathbf{z}_{B,2})$ of the vectors $(\alpha\mathbf{x}, \alpha\mathbf{y}, \alpha\mathbf{z})$, with random shares chosen in $\mathbb{F}_{2^\rho}$.
- $\mathcal{F}_{\mathsf{DAMT}}$ generates random shares $(\mathbf{x}_{A,3}, \mathbf{y}_{A,3}, \mathbf{z}_{A,3})$ and $(\mathbf{x}_{B,3}, \mathbf{y}_{B,3}, \mathbf{z}_{B,3})$ of the vectors $(\beta\mathbf{x}, \beta\mathbf{y}, \beta\mathbf{z})$, with random shares chosen in $\mathbb{F}_{2^\rho}$.
- For $i \in \{1, 2, 3\}$, $\mathcal{F}_{\mathsf{DAMT}}$ sends $(\mathbf{x}_{A,i}, \mathbf{y}_{A,i}, \mathbf{z}_{A,i})$ to A and $(\mathbf{x}_{B,i}, \mathbf{y}_{B,i}, \mathbf{z}_{B,i})$ to B.

Lemma 1. *The protocol in Figure 3 securely computes $\mathcal{F}_{\mathsf{pre}(\rho)}$ against malicious adversaries in the $\mathcal{F}_{\mathsf{DAMT}} - \mathcal{F}_{\mathsf{subVOLE}} - \mathcal{F}_{\mathsf{VOLE}}$-hybrid model with 2ρ bits of communication from B to A and 2ρ bits of communication from A to B per AND gate.*

Completeness. Expanding as in the standard Beaver triple approach, we have

$$\hat{a}_k + \hat{b}_k = ef + ey + fx + z = (a_i + b_i)(a_j + b_j),$$

as desired. Then note that

$$\hat{w}_k + \hat{d}_k = (a_i + b_i)(a_j + b_j)\alpha + \hat{a}_k\alpha = \hat{b}_k\alpha,$$

Fig. 3. Authenticated parallel AND gates from $\mathcal{F}_{\mathsf{DAMT}}$

Protocol $\mathbf{\Pi}^{\mathsf{pre}(\rho)}_{\mathsf{DAMT}}$: Circuit dependent pre-processing of wire labels from authenticated parallel AND gates.

Parametrized by values ρ, κ, and a circuit C consisting of $\mathcal{W}$ wires, $\mathcal{I}$ input wires, $\mathcal{O}$ output wires, and gates $\mathcal{G}$ of the form (i, j, k, T), for $T \in \{\wedge, \oplus\}$, $i, j \in \mathcal{I} \cup \mathcal{W}$, and $k \in \mathcal{W} \cup \mathcal{O}$.

1. A and B invoke $\mathcal{F}_{\mathsf{subVOLE}}$ with A as sender and B as receiver so that A receives $\alpha \in \mathbb{F}_{2^\kappa}$, B receives $\mathbf{b} \in \mathbb{F}_2^m$ and $\mathbf{d} \in \mathbb{F}_{2^\kappa}^m$, and A receives $\mathbf{w} := \mathbf{b}\alpha + \mathbf{d}$.
2. A and B invoke $\mathcal{F}_{\mathsf{subVOLE}}$ with B as sender and A as receiver, so that B receives $\beta \in \mathbb{F}_{2^\rho}$, A receives $\mathbf{a} \in \mathbb{F}_2^m$ and $\mathbf{c} \in \mathbb{F}_{2^\rho}^m$, and B receives $\mathbf{v} := \mathbf{a}\beta + \mathbf{c}$.
3. A and B invoke $\mathcal{F}_{\mathsf{DAMT}}$ with A's input α, B's input β, so that party P receives $(x_{P,\ell,i}, y_{P,\ell,i}, z_{P,\ell,i})$ for $\ell \in \{1, 2, 3\}$ and $1 \leq i \leq n$.
4. A and B compute the authentication messages $(\mathbf{m}_A, \mathbf{m}_B)$ using $\mathbf{\Pi}^{\mathsf{DAMT}\wedge\mathsf{subVOLE}}_{\mathsf{cert}}$. A sends $H(\mathbf{m}_A)$ to B, who verifies that this equals $H(\mathbf{m}_B)$, and otherwise aborts.
5. Initialize a counter $t \leftarrow 1$.
6. For each gate $\mathcal{G} = (i, j, k, T)$, in topological order:
   - If $T = \oplus$:
     - A sets the values $a_k = a_i + a_j$, $c_k = c_i + c_j$, and $w_k = w_i + w_j$.
     - B sets the values $b_k = b_i + b_j$, $d_k = d_i + d_j$ and $v_k = v_i + v_j$.
   - If $T = \wedge$:
     - A sends to B the messages

       $$(m_1^A, m_2^A, m_3^A, m_4^A) := (a_i + x_{A,1,t}, a_j + y_{A,1,t}, c_i + x_{A,3,t}, c_j + y_{A,3,t}).$$

     - B sends to A the messages

       $$(m_1^B, m_2^B, m_3^B, m_4^B) := (b_i + x_{B,1,t}, b_j + y_{B,1,t}, d_i + x_{B,2,t}, d_j + y_{B,2,t}).$$

     - A locally verifies that $(w_i + \alpha x_{A,1,t} + x_{A,2,t} + m_3^B, w_j + y_{A,2,t} + \alpha y_{A,1,t} + m_4^B) = (m_1^B\alpha, m_2^B\alpha)$ and aborts if not.
     - B locally verifies that $(v_i + \beta x_{B,1} + x_{B,3} + m_3^A, v_j + y_{B,3,t} + \beta y_{B,1,t} + m_4^A) = (m_1^A\beta, m_2^A\beta)$ and aborts if not.
     - Both parties locally compute $e := m_1^A + m_1^B$ and $f := m_2^A + m_2^B$.
     - A locally computes

       $$\begin{aligned}\hat{a}_k &= ef + ey_{A,1,t} + fx_{A,1,t} + z_{A,1,t}\\ \hat{c}_k &= ey_{A,3,t} + fx_{A,3,t} + z_{A,3,t}\\ \hat{w}_k &= (ef + \hat{a}_k)\alpha + ey_{A,2,t} + fx_{A,2,t} + z_{A,2,t}.\end{aligned}$$

     - B locally computes

       $$\begin{aligned}\hat{b}_k &= ey_{B,1,t} + fx_{B,1,t} + z_{B,1,t}\\ \hat{d}_k &= ey_{B,2,t} + fx_{B,2,t} + z_{B,2,t}\\ \hat{v}_k &= (ef + \hat{b}_k)\beta + ey_{B,3,t} + fx_{B,3,t} + z_{B,3,t}.\end{aligned}$$

     - $t \leftarrow t + 1$.
7. Party A performs

   $$\hat{\mathbf{w}} \rightarrow \hat{\mathbf{w}} + (\mathbf{a} + \mathsf{lsb}(\hat{\mathbf{a}}))\alpha, \hat{\mathbf{a}} \rightarrow \mathsf{lsb}(\hat{\mathbf{a}})$$

8. Party B performs

   $$\hat{\mathbf{v}} \rightarrow \hat{\mathbf{v}} + (\mathbf{b} + \mathsf{lsb}(\hat{\mathbf{b}}))\beta, \hat{\mathbf{b}} \rightarrow \mathsf{lsb}(\hat{\mathbf{b}}),$$

as desired. Similarly, we have $\hat{a}_k\beta + \hat{c}_k = \hat{v}_k$, as desired.

At the end of the protocol, parties A and B locally adjust these shares so that $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$ become vectors of bits. Since $\hat{\mathbf{a}}+\hat{\mathbf{b}} \in \{0,1\}^n$, we have $(\mathbf{a}+\mathsf{lsb}(\hat{\mathbf{a}})) = (\mathbf{b}+\mathsf{lsb}(\hat{\mathbf{b}}))$, so this adjustment preserves the desired relations.

Security. By the symmetry of the protocol, it is sufficient to consider the case of a malicious A. Let $\mathcal{A}$ be an adversary corrupting A. First, we show that if $\mathcal{A}$ sends incorrect values in a message, B will abort with overwhelming probability. Indeed, if A sends $a_i + x_{A,1} + \phi_1$ instead of $a_i + x_{A,1}$ and $c_i + x_{A,3} + \phi_2$ instead of $c_i + x_{A,3}$, B will verify whether

$$(a_i + x_{A,1} + \phi_1)\beta = (a_i + x_{A,1})\beta + \phi_2,$$

i.e. whether $\beta\phi_1 = \phi_2$.

We can then construct a simple simulator $\mathcal{S}$ that runs $\mathcal{A}$ as a subroutine and plays the role of A in the ideal world. The simulator generates B's last two messages uniformly at random, and the first two messages so that they satisfy the desired check. By the uniform randomness of $y_{B,1}$ and $y_{B,2}$, the distribution of B's messages $d_i+y_{B,1}, d_j+y_{B,2}$ in the real world are identical to the distribution of $\mathcal{S}$'s simulation of B in the ideal world. Since $b_i + x_{B,1}$ and $b_j + x_{B,2}$ can be computed from A's data and the message $d_i + y_{B,1}, d_j + y_{B,2}$, the distribution of these values are identical as well.

$\mathcal{S}$ then sends B's messages to $\mathcal{A}$, and aborts if $\mathcal{A}$ responds with anything besides $(a_i+x_{A,1}, a_j+y_{B,1}, c_i+x_{A,3}, c_j+y_{A,3})$. Otherwise, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs. As discussed above, with overwhelming probability an honest B aborts in the real world whenever $\mathcal{S}$ aborts, so the joint distribution of the outputs of $\mathcal{A}$ and an honest B in the real world are indistinguishable from the joint distribution of the outputs of $\mathcal{A}$ and $\mathcal{S}$ in the ideal world.

4.2 Circuit-dependent preprocessing from parallel AND gates

We now go from authenticated parallel AND gates over ρ to authenticated parallel AND gates over κ, and then to authenicated circuit wires. We begin with the conversion from $\mathcal{F}_{\mathsf{pre}(\rho)}$ to $\mathcal{F}_{\mathsf{pre}(\kappa)}$.

Lemma 2. *The protocol in Figure 4 realizes $\mathcal{F}_{\mathsf{pre}}^{(C,\rho,\kappa)}$ securely in the $\mathcal{F}_{\mathsf{pre}}^{(C,\rho,\rho)} - \mathcal{F}_{\mathsf{subVOLE}}$-RO hybrid model, at the cost of an additional $3n+O(\kappa)$ bits of communication. In particular, $\mathcal{F}_{\mathsf{pre}(\kappa)}$ is securely realizable in the $\mathcal{F}_{\mathsf{pre}(\rho)} - \mathcal{F}_{\mathsf{subVOLE}}$-RO hybrid model.*

Proof. **Completeness.** We have $\mathbf{w}' = \mathbf{b}'\Delta_A + \mathbf{d}'$ and $\hat{\mathbf{w}}' = \hat{\mathbf{b}}'\Delta_A + \hat{\mathbf{d}}'$ both immediately before Step 4 and immediately after Step 5. The desired relations on the vectors $\mathbf{a}+\mathbf{b}$, $\hat{\mathbf{a}}+\hat{\mathbf{b}}$ follow from the correctness of the $\mathcal{F}_{\mathsf{pre}}^{(C,\rho,\rho)}$ functionality.
Security. Security of steps 1,2, and 6 follow from the security of the underlying protocols. Security against a malicious B follows from the correctness of $\mathbf{\Pi}_{\mathsf{cert}}^{\rho\wedge\kappa}$, shown in Lemma 10, which guarantees that A (or a simulator $\mathcal{S}$) will detect an incorrect message with high probability.

For security against a malicious A, note that A sends no message in steps 3 through 5, and that the messages $\mathbf{m}_1$, $\mathbf{m}_2$ can be simulated by sampling uniformly random sequences of bits, by the security of $\mathcal{F}_{\mathsf{subVOLE}}$.

Complexity. We have $|\mathbf{w}| = 2n$ and $|\hat{\mathbf{w}}| = n$, so the messages $\mathbf{m}_1, \mathbf{m}_2$ take $2n + n = 3n$ bits. The certification step calling $\mathbf{\Pi}_{\mathsf{cert}}^{\rho\wedge\kappa}$ costs $O(\kappa)$ bits by Lemma 10. See § 3.2 for an overview of this certified functionality notation.

Fig. 4. Authenticated wire labels over κ from wire labels over ρ

Protocol $\mathbf{\Pi}_{\mathsf{pre}(\rho)}^{\mathsf{pre}(\kappa)}$: Circuit dependent pre-processing of wire labels from authenticated parallel ρ-AND gates.

Parametrized by values ρ, κ, and a circuit C consisting of $\mathcal{W}$ wires, $\mathcal{I}$ input wires, $\mathcal{O}$ output wires, and gates $\mathcal{G}$ of the form (i, j, k, T), for $T \in \{\wedge, \oplus\}$, $i, j \in \mathcal{I} \cup \mathcal{W}$, and $k \in \mathcal{W} \cup \mathcal{O}$.

1. A and B invoke $\mathcal{F}_{\mathsf{pre}}^{(C,\rho,\rho)}$, generating vectors $\mathbf{a}, \mathbf{c}, \mathbf{w}, \hat{\mathbf{a}}, \hat{\mathbf{c}}, \hat{\mathbf{w}}$ and a value α for A and vectors $\mathbf{b}, \mathbf{d}, \mathbf{v}, \hat{\mathbf{b}}, \hat{\mathbf{d}}, \hat{\mathbf{v}}$ and a value β for B.
2. A and B invoke $\mathcal{F}_{\mathsf{subVOLE}}$ with B as sender and A as receiver for the fields $(\mathbb{F}_2, \mathbb{F}_{2^\kappa})$, so that B learns $\mathbf{b}', \mathbf{d}', \hat{\mathbf{b}}', \hat{\mathbf{d}}'$, and A learns $\Delta_A \in \mathbb{F}_{2^\kappa}$ and vectors $\mathbf{w}' := \mathbf{b}'\Delta_A + \mathbf{d}'$ and $\hat{\mathbf{w}}' := \hat{\mathbf{b}}'\Delta_A + \hat{\mathbf{d}}'$.
3. B sends to A the vectors $\mathbf{m}_1 := \mathbf{b} + \mathbf{b}'$ and $\mathbf{m}_2 := \hat{\mathbf{b}} + \hat{\mathbf{b}}'$.
4. A adds to obtain $\mathbf{w}' \leftarrow \mathbf{w}' + \mathbf{m}_1\Delta_A$ and $\hat{\mathbf{w}}' \leftarrow \hat{\mathbf{w}}' + \mathbf{m}_2\Delta_A$.
5. B adds to obtain $\mathbf{b}' \leftarrow \mathbf{b}' + \mathbf{m}_1$, $\hat{\mathbf{b}}' \leftarrow \hat{\mathbf{b}}' + \mathbf{m}_2$.
6. A and B invoke $\mathbf{\Pi}_{\mathsf{cert}}^{\rho\wedge\kappa}$ to certify that the new values of $\mathbf{b}, \hat{\mathbf{b}}$ match their original values.
7. A and B return $\mathbf{a}, \mathbf{c}, \mathbf{w}', \hat{\mathbf{a}}, \hat{\mathbf{c}}, \hat{\mathbf{w}}', \Delta_A$ and $\mathbf{b}', \mathbf{d}', \mathbf{v}, \hat{\mathbf{b}}', \hat{\mathbf{d}}', \hat{\mathbf{v}}, \beta$ respectively.

Next, for completeness, we give a protocol for converting from $\mathcal{F}_{\mathsf{pre}(\kappa)}$ to $\mathcal{F}_{\mathsf{pre}}^{(C,\rho,\kappa)}$. The following result is implicit in [14] and [20].

Lemma 3. *Let C be a circuit with n AND gates. Then the protocol in Figure 5 securely computes $\mathcal{F}_{\mathsf{pre}}^{(C,\kappa,\rho)}$ against malicious adversaries in the RO-subVOLE-$\mathcal{F}_{\mathsf{pre}(\kappa)}$ hybrid model, with an additional $2n$ bits of communication.*

Proof. The security of the first three steps follows from the security of the underlying protocols.

Correctness is immediate, and the proof of security against malicious parties is similar to the proof of Lemma 1.

Remark 3. As discussed in §2.1, Katz et al in [14] realize $\mathcal{F}_{\mathsf{pre}(\kappa)}$ using an optimized version of the TinyOT protocol. Their protocol, in addition to the cost of producing authenticated bits, which could be done with sublinear communication

Fig. 5. Authenticated wire labels from authenticated parallel AND gates

Protocol $\Pi^{\mathsf{pre}(C)}_{\mathsf{pre}(\kappa)}$: Circuit dependent pre-processing of wire labels from authenticated parallel AND gates.

Parametrized by values ρ, κ, and a circuit C consisting of $\mathcal{W}$ wires, $\mathcal{I}$ input wires, $\mathcal{O}$ output wires, and gates $\mathcal{G}$ of the form (i, j, k, T), for $T \in \{\wedge, \oplus\}$, $i, j \in \mathcal{I} \cup \mathcal{W}$, and $k \in \mathcal{W} \cup \mathcal{O}$.

1. A and B invoke $\mathcal{F}_{\mathsf{subVOLE}}$ with A as sender and B as receiver, so that A receives $\alpha \in \mathbb{F}_{2^\kappa}$, B receives $\mathbf{b} \in \mathbb{F}_2^m$ and $\mathbf{d} \in \mathbb{F}_{2^\kappa}^m$, and A receives $\mathbf{w} := \mathbf{b}\alpha + \mathbf{d}$.
2. A and B invoke $\mathcal{F}_{\mathsf{subVOLE}}$ with B as sender and A as receiver, so that B receives $\beta \in \mathbb{F}_{2^\rho}$, A receives $\mathbf{a} \in \mathbb{F}_2^m$ and $\mathbf{c} \in \mathbb{F}_{2^\rho}^m$, and B receives $\mathbf{v} := \mathbf{a}\beta + \mathbf{c}$.
3. A and B invoke $\mathcal{F}^{(\mathsf{PAnd}(n),\kappa,\rho)}_{\mathsf{pre}(\rho)}$ so that A obtains $(\mathbf{w}', \hat{\mathbf{w}}', \hat{\mathbf{a}}', \hat{\mathbf{c}}')$ and B obtains $(\mathbf{v}', \hat{\mathbf{v}}', \hat{\mathbf{a}}', \hat{\mathbf{c}}')$.
4. For each gate $\mathcal{G} = (i, j, k, T)$, in topological order:
   - If $T = \oplus$:
     - A sets the values $a_k = a_i + a_j$, $c_k = c_i + c_j$, and $w_k = w_i + w_j$.

     B sets the values $b_k = b_i + b_j$, $d_k = d_i + d_j$ and $v_k = v_i + v_j$.
   - If $T = \wedge$ is the t-th AND gate:
     - A sends $(a_i + a'_{2t-1}, a_j + a'_{2t})$ to B
     - B sends $(b_i + b'_{2t-1}, b_j + b'_{2t})$ to A
     - A and B locally compute $e_k := a_i + b_i + a'_{2t-1} + b'_{2t-1}$ and $f_k := a_j + b_j + a'_{2t} + b'_{2t}$.
     - A locally computes

$$\begin{aligned}\hat{a}_k &= e_k f_k + e_k a_j + f_k a_i + \hat{a}'_t \\ \hat{c}_k &= e_k c_j + f_k c_i + \hat{c}'_t \\ \hat{w}_k &= e_k w_j + f_k w_i + \hat{w}'_t.\end{aligned}$$

     - B locally computes

$$\begin{aligned}\hat{b}_k &= e_k b_j + f_k b_i + \hat{b}'_t \\ \hat{d}_k &= e_k d_j + f_k d_i + \hat{d}'_t \\ \hat{v}_k &= e_k f_k \beta + e_k v_j + f_k v_i + \hat{v}'_t.\end{aligned}$$

under VOLE, requires $B\kappa$ bits of communication per gate, with $B \approx \rho / \log |C|$. In particular, $B \geq 3$ for $|C| < 2^\rho$. Adding back in the 2κ bits required in the online phase, the cost of [14] is at least 2.5x the cost of a semi-honest garbled circuit for circuits with size $|C| < 2^\rho$. Unfortunately, Lemma 2 does not offer any improvements the approach of [14], since their compiler to $\mathcal{F}_{\mathsf{pre}(\kappa)}$ requires computational security, and so replacing it with a compiler to $\mathcal{F}_{\mathsf{pre}(\rho)}$ would still require $B\kappa$ bits per gate.

An alternative realization of the $\mathcal{F}_{\mathsf{pre}(\kappa)}$ functionality could be accomplished by the SPDZ protocol [10]. This would consume 6 authenticated multiplication triples per AND gate and require 12κ additional communication under a naive implementation. Applying Lemma 2 to the naive SPDZ-style approach gives a compiler to $\mathcal{F}_{\mathsf{pre}(\kappa)}$ by way of $\mathcal{F}_{\mathsf{pre}(\rho)}$ that costs $12\rho+3$ bits of communication per gate, and thus $2\kappa + 12\rho + 3$ bits per gate for the entire protocol, approximately 3x the cost of a semi-honest garbled circuit.

4.3 Authenticated garbling

The only changes we make to the authenticated garbling protocol of [14] are after-effects of our decision to alter the preprocessing functionality so that A does not hold a mask for a wire value that is one of B's inputs, and vice versa. The only steps that change materially therefore are steps 3 and 4. Step 3 in [14], after translating into the language of VOLE, reads:

– For each $i \in \mathcal{I}_B$, A sends a_i to B and invoke $\mathbf{\Pi}_{\mathsf{LPZK}}$ to prove that this a_i matches the value in $\mathcal{F}_{\mathsf{pre}}$. B then sends $y_i \oplus \lambda_i = y_i \oplus a_i \oplus b_i$ to A. Finally, A sends $L_{i,y_i \oplus \lambda_i}$ to B.

We replace this step with the following:

– For each $i \in \mathcal{I}_B$, B sends $y_i \oplus b_i$ to A. Then A sends $L_{i,y_i \oplus \mathbf{b}_i}$ to B.

It is possible to simulate the previous protocol from this version by having B generate A's messages a_i uniformly at random for $i \in \mathcal{I}_A$, and adjusting their value b_i to keep the sum $a_i \oplus b_i$ constant, and having A set $a_i = 0$. These adjustments can occur without any communication, since the values a_i, b_i are never used again by A, B respectively. Therefore the security of one protocol implies the security of the other. We make similar adjustments to Step 4.

Proof of Theorem 1 Combining the three lemmas in this section gives a realization of $\mathcal{F}_{\mathsf{pre}}^{(C,\rho,\kappa)}$ in the $\mathcal{F}_{\mathsf{DAMT}} - \mathcal{F}_{\mathsf{VOLE}} - \mathcal{F}_{\mathsf{subVOLE}}$ model. Applying Theorem 4 and incorporating the minor changes to the authenticated garbling protocol outlined above gives that the desired $\mathbf{\Pi}_{\mathsf{2pc}}^{\mathsf{DAMT}}$ protocol.

5 Authenticated garbling from block VOLE

5.1 Compressed authenticated bits from block VOLE

We begin by stating formally the compressed preprocessing functionality and the block (subfield) VOLE functionality.

The compressed preprocessing functionality *compresses* B's wire labels belonging to AND gates in $\mathbf{b}$ to a much shorter vector $\widetilde{b}$ of length

$$L := \frac{\rho \log n - \rho \log \rho}{\log 2} + 2\rho.$$

Write $\mathbf{b}_{\mathcal{I}}$ for input wires, and $\mathbf{b}'$ for AND gate wires. Then the vector $\mathbf{b}$ is determined from $\mathbf{b}_{\mathcal{I}} \cup \mathbf{b}'$ in the obvious way, and $\mathbf{b}'$ is determined from $\widetilde{b}$ by some public linear transformation M_H. Similarly B's wire masks $\mathbf{d}'$ are computed as $M_H \widetilde{d}$, where $\widetilde{d} \in \mathbb{F}_{2^\kappa}^L$.

Fig. 6. Compressed authenticated wire labels

Functionality $\mathcal{F}_{\mathsf{cp}}^{(C,\rho)}$: Compressed pre-processing of wire labels for authenticated garbling.

Parametrized by the value ρ, and a circuit C consisting of $\mathcal{W}$ wires, $\mathcal{I}$ input wires, $\mathcal{O}$ output wires, and gates $\mathcal{G}$ of the form (i, j, k, T), for $T \in \{\wedge, \oplus\}$, $i, j \in \mathcal{I} \cup \mathcal{W}$, and $k \in \mathcal{W} \cup \mathcal{O}$. Let n be the number of AND gates. Where clear from context, we omit the parameters C, ρ, κ and write $\mathcal{F}_{\mathsf{cp}}$ for $\mathcal{F}_{\mathsf{cp}}^{(C,\rho,\kappa)}$.

- All parties compute
$$L = \frac{\rho \log n - \rho \log \rho}{\log 2} + 2\rho.$$
- A chooses $\alpha \in \mathbb{F}_{2^\rho}$ and wire labels $\mathbf{a} \in \mathbb{F}_2^n$, $\mathbf{c} \in \mathbb{F}_{2^\rho}^n$ and sends them to $\mathcal{F}_{\mathsf{cp}}$.
- B chooses $\beta \in \mathbb{F}_{2^\rho}$ and wire labels $\mathbf{b}_{\mathcal{I}} \in \mathbb{F}^{|\mathcal{I}|}, \widetilde{b} \in \mathbb{F}_2^L$, $\mathbf{d}_{\mathcal{I}} \in \mathbb{F}_{2^\rho}^{|\mathcal{I}|}, \widetilde{d} \in \mathbb{F}_{2^\rho}^L$ and sends them to $\mathcal{F}_{\mathsf{cp}}$.
- $\mathcal{F}_{\mathsf{cp}}$ chooses a random $n \times L$ matrix M_H over $\mathbb{F}_2$ and sends M_H to A and B.
- $\mathcal{F}_{\mathsf{cp}}$ computes the vectors $\mathbf{b}', \mathbf{d}'$ via $\mathbf{b}, = M_H \widetilde{b}$ and $\mathbf{d}' = M_H \widetilde{d}$, and computes $\mathbf{b}, \mathbf{d}$ from $\mathbf{b}', \mathbf{d}'$.
- As a sub-protocol, $\mathcal{F}_{\mathsf{cp}}$ runs a simulation of the interaction of A, B, and $\mathcal{F}_{\mathsf{pre}}^{(C,\rho,\kappa)}$ using $\alpha, \beta, \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ as the various parties' inputs, and stores the output.
- $\mathcal{F}_{\mathsf{cp}}$ sends $(\mathbf{v}, \hat{\mathbf{v}}, \hat{\mathbf{b}}, \hat{\mathbf{d}})$ to B and $(\mathbf{w}, \hat{\mathbf{a}}, \hat{\mathbf{c}})$ to A.
- A sends either Honest or $(\mathsf{Cheat}, \mathbf{m}^*)$ to $\mathcal{F}_{\mathsf{cp}}$.
- If A sent Honest, then $\mathcal{F}_{\mathsf{cp}}$ sends $(\hat{\mathbf{w}})$ to A.
- If A sent $(\mathsf{Cheat}, \mathbf{m}^*)$, then $\mathcal{F}_{\mathsf{cp}}$ sends $(\hat{\mathbf{w}} + \mathbf{m}^* \beta^{-1})$ to A.

The other change made in this pre-processing functionality is that we allow party A to cheat in such a way that is not immediately detected, but corrupts its own output. Specifically, if A sends faulty messages, A can ensure both parties hold shares of $\hat{b}_i \alpha + m^* \beta^{-1}$, rather than $\hat{b}_i \alpha$. Since A does not know β, A cannot use these corrupted shares, and B will discover the error and abort during the execution of the authenticated garbling, as we show in § 5.3.

Fig. 7. Block subfield VOLE

Functionality $\mathcal{F}_{\mathsf{bVOLE}}^{(F,E,k,n)}$: Block VOLE

Parametrized by a pair of fields $F \subseteq E$ and integers k and n. In this paper, we have $F \in \{\mathbb{F}_2, \mathbb{F}_{2^\rho}\}$ and $E = \mathbb{F}_{2^\rho}$. We refer colloquially to the first variant as block subfield VOLE and the second as block VOLE.

- B chooses parameters $\beta_1, \ldots, \beta_k \in E$ and sends them to $\mathcal{F}_{\mathsf{bVOLE}}$.
- $\mathcal{F}_{\mathsf{bVOLE}}$ chooses a collection of vectors $\mathbf{b}_1, \ldots, \mathbf{b}_k \in E^n$ and sends the vectors to A.
- A chooses a vector $\mathbf{a} \in F^n$ and sends $\mathbf{a}$ to $\mathcal{F}_{\mathsf{bVOLE}}$.
- For $i = 1, \ldots, k$, the functionality $\mathcal{F}_{\mathsf{bVOLE}}$ computes $\mathbf{v}_i = \mathbf{a}\beta_i + \mathbf{b}_i$ and sends the result to B.

5.2 From block VOLE to compressed authenticated wire labels

We realize this preprocessing functionality using block VOLE, a collection of VOLE or subfield VOLE instances where one party A uses the same inputs across the VOLE calls. We define this protocol formally in Figure 7, and give the converter from block VOLE to $\mathcal{F}_{\mathsf{cp}}$ in Figure 8. We note that, in Step 12, if $\overline{a}$ is one of A's input to a block VOLE, and $\overline{b} + \gamma$ and γ are two of B's inputs to that block VOLE, then A and B can produce shares of the value $\overline{a}\overline{b}$ by subtracting their respective shares of $\overline{a}(\overline{b}+\gamma)$ and $\overline{a}\gamma$. All monomial terms in Step 12 can be shared in this fashion. We defer the proof of the following lemma to Appendix B.1.

Lemma 4. *The protocol in Figure 8 can securely compute $\mathcal{F}_{\mathsf{cp}}^{(C,\rho,\rho)}$ against malicious adversaries in the $\mathcal{F}_{\mathsf{bVOLE}} - \mathcal{F}_{\mathsf{VOLE}} - \mathcal{F}_{\mathsf{subVOLE}}$ model with $1 + O(\frac{L}{n})$ bits of communication per gate from B to A and $5\rho + 1$ bits of communication per gate from A to B.*

To convert from $\mathcal{F}_{\mathsf{cp}}^{(C,\rho,\rho)}$ to $\mathcal{F}_{\mathsf{cp}}^{(C,\rho,\kappa)}$, we used almost the identical protocol to that used to convert from $\mathcal{F}_{\mathsf{pre}(\rho)}$ to $\mathcal{F}_{\mathsf{pre}(\kappa)}$.

Lemma 5. *The protocol in Figure 4 realizes $\mathcal{F}_{\mathsf{cp}}^{(C,\rho,\kappa)}$ in the $\mathcal{F}_{\mathsf{cp}}^{(C,\rho,\rho)} - \mathcal{F}_{\mathsf{subVOLE}}$-hybrid model, replacing $\mathcal{F}_{\mathsf{pre}}^{(C,\rho,\rho)}$ with $\mathcal{F}_{\mathsf{cp}}^{(C,\rho,\rho)}$ in Step 1.*

Proof. The argument is identical to the argument in Lemma 2. We need only note that the messages $\mathbf{m}_1, \mathbf{m}_2$ are still uniformly random in A's view, in spite of the linear relations on $\mathbf{b}$ allowed by $\mathcal{F}_{\mathsf{cp}}$, because of the masks $\mathbf{b}', \hat{\mathbf{b}}'$.

5.3 Authenticated garbling

In Figure 9, we give our modified authenticated garbled circuit protocol. The wire labels are computed as in [14], but in the authentication step we apply

Fig. 8. Compressed authenticated wire labels from block VOLE

Protocol $\mathbf{\Pi}_{\mathsf{cp}}(C, \rho)$: Compressed pre-processing of wire labels for authenticated garbling.

Parametrized by the value ρ, and a circuit C consisting of $\mathcal{W}$ wires, $\mathcal{I}$ input wires, $\mathcal{O}$ output wires, and gates $\mathcal{G}$ of the form (i, j, k, T), for $T \in \{\wedge, \oplus\}$, $i, j \in \mathcal{I} \cup \mathcal{W}$, and $k \in \mathcal{W} \cup \mathcal{O}$. Let n be the number of AND gates.

1. All parties compute
$$L = \frac{\rho \log n - \rho \log \rho}{\log 2} + 2\rho$$
and choose a public $n \times L$ matrix M_H over $\mathbb{F}_2$.
2. A and B invoke $\mathcal{F}_{\mathsf{subVOLE}}$ with B as sender and A as receiver, so that B learns $(\widetilde{b}, \widetilde{d})$ and A holds $\widetilde{w}$, with length of the VOLE equal to L.
3. The parties extend the VOLE by length n, with additional entries $(w_{i,j}, b_{i,j}, d_{i,j})$ where $b_{i,j}$ is the (i,j)-th entry of $(M_H\widetilde{b})^T \cdot (M_H\widetilde{b})$.
4. Party A locally computes $w = M_H\widetilde{w}$.
5. B constructs the vector $\overline{\mathbf{b}} = \widetilde{b}_i\beta + \gamma, \beta + \gamma, \gamma$ with $\gamma \in \mathbb{F}_{2^\rho}$ chosen randomly.
6. Party A constructs the vector $\overline{\mathbf{a}} := \mathbf{a} \cup (a_i a_j) \cup (\hat{a}_i)$. The first vector is A's input to $\mathcal{F}_{\mathsf{cp}}$, the second vector is the the values $a_i \wedge a_j$, for every multiplication gate $\mathcal{G}_k = (\wedge, i, j)$, and the third vector is a string of random bits which will be part of A's output.
7. The parties call $\mathsf{Extend}(\mathcal{F}_{\mathsf{subVOLE}})$, adding $\overline{\mathbf{b}}$ as an additional $L + 2$ entries.
8. A and B perform $\mathcal{F}_{\mathsf{bVOLE}}^{(\mathbb{F}_{2^\rho}, \mathbb{F}_2, L+2, n)}$, the subfield variant of block VOLE, with B's inputs the vector $\overline{\mathbf{b}}$ and A's inputs the vector $\overline{\mathbf{a}}$.
9. A and B invoke $\mathcal{F}_{\mathsf{bVOLE}}^{(\mathbb{F}_{2^\rho}, \mathbb{F}_{2^\rho}, L+2, n)}$. B's input to the block VOLE is again the vector $\overline{\mathbf{b}}$ with γ as above, and A's input is the vector $\alpha \cdot \overline{\mathbf{a}} \cup (\hat{a}_{i,2}) \cup \{\alpha\}$, that is, A's input above multiplied by α, along with a vector of masks $\hat{a}_{i,2} \in \mathbb{F}_{2^\rho}$ and the additional input α.
10. Both parties call $\mathbf{\Pi}_{\mathsf{LPZK}}$ to prove correctness of the values $a_i \wedge a_j$, $b_{i,j}$, and $\widetilde{b}_i\beta$ under LPZK.
11. B certifies that their inputs to the block VOLE match their inputs to the VOLE with A as receiver, with the $\mathbf{\Pi}_{\mathsf{cert}}^{\mathsf{VOLE}\wedge\mathsf{ELOV}}$ protocol discussed in §3.2.
12. B locally computes:
$$\hat{v}_i := \hat{a}_i\beta + \hat{c}_i$$
$$v_{i,2} := \hat{a}_{i,2}\beta + c_{i,2}$$
$$v_{i,3} := \hat{a}_i\alpha\beta + c_{i,3}$$
$$v_{i,4} := (a_i a_j + a_i b_j + a_j b_i)\beta + c_{i,4}$$
$$v_{i,5} := (a_i a_j + a_i b_j + a_j b_i)\alpha\beta + c_{i,5}$$
where all terms $\hat{c}_i, c_{i,j}$ can be computed locally by A.
13. A sends to B the terms $(m_{i,1}, m_{i,2}) := (\hat{c}_i + c_{i,4}, c_{i,2} + c_{i,3} + c_{i,5})$, and B defines
$$\hat{b}_i := (\hat{v}_i + v_{i,4} + m_{i,1})\,\beta^{-1} + b_i b_j$$
and
$$\hat{d}_i := (v_{i,2} + v_{i,3} + v_{i,5} + m_{i,2})\beta^{-1} + d_{i,j},$$
respectively.
14. A adds locally to hold $\hat{w}_i := \hat{a}_{i,2} + w_{i,j}$.

the half gate technique of Zahur et al. [24] to the secondary garbled circuit approach of [20].We also replace $\mathcal{F}_{\mathsf{pre}}$ with $\mathcal{F}_{\mathsf{cp}}$, and modify Steps 3 and 4 by setting unneeded wire masks to 0 as in §4.3.

Lemma 6. *The protocol given in Figure 9 securely computes a functionality f against malicious adversaries in the RO-$\mathcal{F}_{\mathsf{cp}}^{(C,\rho,\kappa)}$ $-$ $\mathcal{F}_{\mathsf{subVOLE}}$ $-$ $\mathcal{F}_{\mathsf{VOLE}}$-hybrid model, with $2\kappa + 3\rho$ bits of communication per AND gate, $\kappa + 1$ bits of communication per input gate, and 1 bit of communication per output gate.*

The key difficulty is protecting against a selective failure attack by A. Learning whether or not B aborts is equivalent to corrupting some subset of t table entries (by corrupting the messages $G_{i,j}$ or $G'_{i,j}$), and learning whether B opened *any* of those table entries during circuit evaluation. If the t table entries chosen correspond to rows of M_H that are linearly independent, then the labels $M_H\widetilde{b}$ are independent, and the probability of failure is $1 - 2^{-t}$.

We therefore give a simulator that aborts with probability $1-2^{-t}$, and restrict our attention to the case where the t entries correspond to linearly dependent rows of M_H. To treat this case, we recall the notion of (t, k)-independent sets (the concept was first introduced in [12], see [19] for a thorough treatment, and [9,8] for additional discussion). A (t, k)-independent set over $\mathbb{F}_q$ is a subset of $\mathbb{F}_q^k$ such that no $t + 1$ element subset is linearly dependent. For our purposes, it is sufficient to construct a $(\rho - 1, L)$-independent set $\mathbb{B} \subseteq \mathbb{F}_2^L$ such that $|\mathbb{B}| = n$ via a randomized algorithm. Then either the simulator gives the correct abort probability or the protocol aborts almost surely, with probability at least $1-2^{-\rho}$, and either way party A learns nothing. We give the full proof in Appendix B.2.
Proof of Theorem 2 We begin with $\mathcal{F}_{\mathsf{bVOLE}}$. We use Lemma 4 to construct $\mathcal{F}_{\mathsf{cp}}^{(C,\rho,\rho)}$, Lemma 5 to construct $\mathcal{F}_{\mathsf{cp}}^{(C,\rho,\kappa)}$ and prove the correctness of $\mathbf{\Pi}_{\mathsf{2pc}}^{\mathsf{VOLE}}$ in Lemma 6.

6 NISC from garbled circuits

6.1 Conditional Disclosure of Secrets from programmable OLE

We construct a NISC protocol with A as sender and B as receiver. We generate our authenticated bits and the related conversion protocol to authenticated circuit wire labels using the programmable OLE functionality given in Figure 10. This protocol allows us to to the piece-wise product of any pair of vectors selected from a collection of p vectors from A and q vectors from B.

Two obstacles present themselves in the conversion from programmable OLE to authenticated circuit wire labels. First, we can no longer use $\mathbf{\Pi}_{\mathsf{LPZK}}$ to certify B's inputs, since this would violate non-interactivity. Instead, we use a specialized conditional disclosure of secrets (CDS) protocol that ensures that any future messages from A will be uniformly random if B cheats. The second obstacle is

Fig. 9. Authenticated garbling protocol in the $\mathcal{F}_{\mathsf{cp}}$ hybrid model

Protocol $\Pi_{\mathsf{2pc}}^{\mathsf{VOLE}}$

Inputs: Party A holds $x \in \{0,1\}^{|\mathcal{I}_1|}$ and B holds $y \in \{0,1\}^{|\mathcal{I}_2|}$. Both parties hold a circuit C for a function $f : \{0,1\}^{|\mathcal{I}_1|+|\mathcal{I}_2|} \to \{0,1\}^{|\mathcal{O}|}$.

1. A and B call $\mathcal{F}_{\mathsf{cp}}^{(C,\rho,\rho)}$ and then the compiler from $\mathcal{F}_{\mathsf{cp}}^{(C,\rho,\rho)}$ to $\mathcal{F}_{\mathsf{cp}}^{(C,\rho,\kappa)}$, so that A holds $\Delta_A, \mathbf{w}, \hat{\mathbf{w}}, \mathbf{a}, \hat{\mathbf{a}}, \mathbf{c}, \hat{\mathbf{c}}$ and B holds $\beta, , \mathbf{v}, \hat{\mathbf{v}}, \mathbf{b}, \hat{\mathbf{b}}, \mathbf{d}, \hat{\mathbf{d}}$. For each $i \in \mathcal{I}_1 \cup \mathcal{I}_2$, A also picks a uniform κ-bit string $L_{i,0}$. The parties jointly determine keys to hash functions $H : \mathbb{F}_{2^\kappa} \times \{1,\dots,n\} \to \mathbb{F}_{2^\kappa}$ and $H' : \mathbb{F}_{2^\kappa} \times \{1,\dots,n\} \to \mathbb{F}_{2^\rho}$.
2. Following the topological order of the circuit, for each gate $G = (i,j,k,T)$,
   - If $T = \oplus$, A computes $L_{k,0} := L_{i,0} \oplus L_{j,0}$
   - If $T = \wedge$, A computes $L_{i,1} := L_{i,0} \oplus \Delta_A$, $L_{j,1} := L_{j,0} \oplus \Delta_A$, and
     - $G_{k,0} := H(L_{i,0},k) \oplus H(L_{i,1},k) \oplus w_j \oplus a_j \Delta_A$
     - $G_{k,1} := H(L_{j,0},k) \oplus H(L_{j,1},k) \oplus w_i \oplus a_i \Delta_A \oplus L_{i,0}$
     - $L_{k,0} := H(L_{i,0},k) \oplus H(L_{j,0},k) \oplus (w_k \oplus \hat{w}_k) \oplus (a_k \oplus \hat{a}_k) \cdot \Delta_A$
     - $G'_{k,0} := H'(L_{i,0},k) \oplus H'(L_{j,0},k) \oplus c_k \oplus \hat{c}_k$
     - $G'_{k,1} := H'(L_{i,0},k) \oplus H'(L_{i,1},k) \oplus c_j$
     - $G'_{k,2} := H'(L_{j,0},k) \oplus H'(L_{j,1},k) \oplus c_i$

     A sends $G_{k,0}, G_{k,1}, G'_{k,0}, G'_{k,1}, G'_{k,2}$ to B.
3. For each $i \in \mathcal{I}_B$, B sends $y_i \oplus b_i$ to A. Then A sends $L_{i,y_i \oplus \mathbf{b}_i}$ to B.
4. For each $i \in \mathcal{I}_A$, A sends $x_i \oplus a_i$ and $L_{i,x_i \oplus a_i}$ to B.
5. B evaluates the circuit in topological order. For each gate $G = (i,j,k,T)$, B initially holds $(z_i \oplus \lambda_i, L_{i,z_i \oplus \lambda_i})$ and $(z_j \oplus \lambda_j, L_{j,z_j \oplus \lambda_j})$, where z_i, z_j are the underlying values of the wires.
   (a) If $T = \oplus$, B computes $z_k \oplus \lambda_k := (z_i \oplus \lambda_i) \oplus (z_j \oplus \lambda_j)$ and $L_{k,z_k \oplus \lambda_k} := L_{i,z_i \oplus \lambda_i} \oplus L_{j,z_j \oplus \lambda_j}$.
   (b) If $T = \wedge$, B computes $G_0 := G_{k,0} \oplus d_j$, $G_1 := G_{k,1} \oplus d_i$, and evaluates the garbled table (G_0, G_1) to obtain the output label

$$L_{k,z_k \oplus \lambda_k} := H\left((L_{i,z_i \oplus \lambda_i}),k\right) \oplus H\left((L_{j,z_j \oplus \lambda_j},k\right) \oplus (d_k \oplus \hat{d}_k) \\ \oplus (z_i \oplus \lambda_i) G_0 \oplus (z_j \oplus \lambda_j)(G_1 \oplus L_{i,z_i \oplus \lambda_i}).$$

   Then B computes

$$\begin{aligned} & b_k \oplus \hat{b}_k \oplus (z_i \oplus \lambda_i) b_j \oplus (z_j \oplus \lambda_j) b_i \oplus (z_i \oplus \lambda_i) \wedge (z_j \oplus \lambda_j) \\ & \oplus \left((v_k \oplus \hat{v}_k \oplus (z_i \oplus \lambda_i) v_j \oplus (z_j \oplus \lambda_j) v_i\right) \beta^{-1} \\ & \oplus \left(H'(L_{i,z_i \oplus \lambda_i}) \oplus H'(L_{j,z_j \oplus \lambda_j}) \oplus G'_{k,0} \oplus (z_i \oplus \lambda_i) G'_{k,1} \oplus (z_j \oplus \lambda_j) G'_{k,2}\right) \beta^{-1} \\ & = \lambda_k \oplus \hat{\lambda}_k \oplus (z_i \oplus \lambda_i)\lambda_j \oplus (z_j \oplus \lambda_j)\lambda_i \oplus (z_i \oplus \lambda_i) \wedge (z_j \oplus \lambda_j) \\ & = \lambda_k \oplus z_k \end{aligned}$$

6. For each $i \in \mathcal{O}$, A sends a_i to B and calls $\mathbf{\Pi}_{\mathsf{LPZK}}$ to prove these values are correct. B computes $z_i := (\lambda_i \oplus z_i) \oplus a_i \oplus b_i$.

Fig. 10. Programmable OLE

Functionality $\mathcal{F}_{\mathsf{OLE}}^{(n,\kappa,p,q,Q)}$: Programmable OLE over a field $\mathbb{F}_{2^\kappa}$ and relations Q.

Parametrized by integers $n, p, q, \kappa \in \mathbb{N}$ and a set of relations $Q \subseteq \{1,\ldots,p\} \times \{1,\ldots,q\}$, i.e. elements $q \in Q$ are ordered pairs of integers.

- A chooses a collection of vectors $\mathbf{a}^1, \ldots, \mathbf{a}^p$ of length n and sends them to $\mathcal{F}_{\mathsf{OLE}}^{(n,\kappa,p,q,Q)}$.
- B chooses a collection of vectors $\mathbf{b}^1, \ldots, \mathbf{b}^q$ of length n and sends them to $\mathcal{F}_{\mathsf{OLE}}^{(n,\kappa,p,q,Q)}$.
- For each entry $q = (i,j) \in Q$, $\mathcal{F}_{\mathsf{OLE}}^{(n,\kappa,p,q,Q)}$ chooses vectors $\mathbf{v}^q$, $\mathbf{c}^q$ with $\mathbf{v}^q + \mathbf{c}^q = \mathbf{a}^i \cdot \mathbf{b}^j$.
- $\mathcal{F}_{\mathsf{OLE}}^{(n,\kappa,Q)}$ sends $\mathbf{v}^q$ to A and $\mathbf{c}^q$ to B, for all $q \in Q$.

related to the task of minimizing p and q so that the protocol is concretely efficient, and we cover it in § 6.2.

For CDS, informally, B sends a message to A that allows A to learn a secret value $\mathbf{s_B}$ known to B if and only if B's message satisfies a desired set of relations. Otherwise, A will compute a guess $\mathbf{s}_A$, which on at least one entry will appear random to B. Then A appends $H(\mathbf{s}_A)$ to all future messages to B, so that B can recover the underlying message if and only if $\mathbf{s}_A = \mathbf{s}_B$. We give a formal definition of this functionality in Figure 11.

We give a protocol realizing this functionality in Figure 12, and prove its correctness in Appendix B.3. Our protocol works by first proving that A and B each have one input vector that is constant, and using that to realize instances of subfield VOLE over A's input α and each of B's input vectors $\mathbf{b}^i$. Write $Q' := Q \cup \{(p+1, j)\} \cup \{(i, q+1)\}$, for $1 \le j \le q+1$ and $1 \le i \le p$, and $Q'' := Q' \cup \{(p+1, j\}$ for $j = q+2, q+3, q+4$.

It is possible to move Step 6 to Step 2, making the protocol non-interactive, since the values $\hat{c}_i^{(j,k)}$ used in Step 6 can be computed locally by B from the output of the random $\mathcal{F}_{\mathsf{OLE}}$ functionality and B's inputs. Step 6 is separated from Step 2 because the most complicated part of the protocol is in Steps 6 and 7, which are used to verify the relations in $\mathcal{R}_2$. Removing Steps 6 and 7 and using Q' instead of Q'' gives a warm-up CDS protocol for certifying the relations in $\mathcal{R}_1$ only.

Lemma 7. *The protocol in Figure* 12 *realizes the functionality* $\mathcal{F}_{\mathsf{CDS}}^{(n,\kappa,p,q,Q,\mathcal{R})}$ *non-interactively in the RO-*$\mathcal{F}_{\mathsf{OLE}}^{(n,\kappa,p+1,q+4,Q'',\mathcal{R})}$*-hybrid model.*

Fig. 11. Programmable OLE with conditional disclosure of secrets

Functionality $\mathcal{F}_{\mathsf{CDS}}^{(n,\kappa,p,q,Q,\mathcal{R})}$: CDS for $\mathcal{F}_{\mathsf{OLE}}^{(n,\kappa,p,q,Q)}$ over a field $\mathbb{F}_{2^\kappa}$ and relations $\mathcal{R}$.

Parametrized by integers $n, \kappa, p, q, \in \mathbb{N}$, a set of relations $Q \subseteq \{1, \ldots, p\} \times \{1, \ldots, q\}$ as above, and a set of relations $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$, where $\mathcal{R}_1$ is a collection of equality constraints $b_i^j = b_k^\ell$, and $\mathcal{R}_2$ is a collection of quadratic relations of the form $b_i^1 \cdot b_j^1 = b_k^1$. Additionally, let $\mathbf{m}$ be a message that A plans to send to B.

- A and B interact with $\mathcal{F}_{\mathsf{CDS}}$ playing the role of $\mathcal{F}_{\mathsf{OLE}}$ on B's input vectors $\mathbf{b}^i \in \mathbb{F}_2^n$ for $1 \leq i \leq q$ and A's input vectors $\mathbf{a}^i, \mathbf{v}^q \in \mathbb{F}_{2^\kappa}^n$ for $1 \leq i \leq p$ and $q \in Q$.
- If the vectors $\mathbf{b}^i$ satisfies the relations in $\mathcal{R}$, $\mathcal{F}_{\mathsf{CDS}}$ sends $\mathbf{m}$ to B.
- If any of the vectors $\mathbf{b}^i$ do not satisfy the relations in $\mathcal{R}$, $\mathcal{F}_{\mathsf{CDS}}$ sends a random vector to B.

6.2 Non-interactive authenticated circuit wires and authenticated garbling

The remainder of the construction is similar to the construction given in Section 5. We give a brief overview here, and a more detailed description in the appendices.

As discussed in 1.1, the randomness computation time and the seed size grow with the number of piece-wise products required, i.e. with $|Q|$ using the notation in the functionality description. In order to minimize the numbers p, q required, we construct for A three vectors of inputs: $\mathbf{a}, \mathbf{a}^L, \mathbf{a}^R$, where $\mathbf{a}$, chosen randomly, represents authenticated bits for all wires, $\mathbf{a}^L$ represents only bit labels for wires used as labels for left inputs to multiplication gates, so that a_k^L is the left input to the kth multiplication gate, and $\mathbf{a}^R$ likewise represents only bit labels for wires used as labels for right inputs to multiplication gates.

We similarly define $\mathbf{b}, \mathbf{b}^L, \mathbf{b}^R$. The full construction of the preprocessing functionality is similar to the protocol $\mathbf{\Pi}_{\mathsf{cp}}$. We give this protocol and a proof of its correctness in Appendix B.4.

The authenticated garbling functionality is also similar to the protocol $\mathbf{\Pi}_{\mathsf{2pc}}^{\mathsf{VOLE}}$ used in the VOLE-only case, replacing $\mathcal{F}_{\mathsf{cp}}$ with the NISC preprocessing functionality. Besides the first step of generating the preprocessing functionality, which is non-interactive by construction, the only message from B to A is given in Step 3, when B sends bit masks of its input values $y_i \oplus b_i$. This communication can be moved to Step 1 with no loss of security, making the entire protocol non-interactive, which we prove in Appendix B.5.

Acknowledgements. Supported in part by DARPA Contract No. HR001120C0087. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of

Fig. 12. Conditional disclosure of secrets

Protocol Π_{CDS}: Conditional disclosure of secrets over programmable OLE.

Parametrized by integers $n, \kappa, p, q, \in \mathbb{N}$, a set of relations $Q \subseteq \{1,\ldots,p\} \times \{1,\ldots,q\}$ as above, and a set of relations $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ as above. Additionally, let $\mathbf{m}$ be a message that A plans to send to B.

1. A and B choose random values $\alpha, \beta \in \mathbb{F}_{2^\kappa}$ and define the vectors $\boldsymbol{\alpha} := (\alpha,\ldots,\alpha)$ and $\boldsymbol{\beta} := (\beta,\ldots,\beta)$.
2. A and B invoke $\mathcal{F}_{\mathsf{OLE}}^{(n,\kappa,p+1,q+1,Q')}$ with the additional inputs $\mathbf{a}^{p+1} := \boldsymbol{\alpha}$, $\mathbf{b}^{q+1} := \boldsymbol{\beta}$. Let $(\mathbf{m}_A^i)$ be the messages that A sends during the random-to-fixed OLE compiler, and let $\hat{\mathbf{c}}_i^{(j,k)}$ be the vectors held by B before receiving $(\mathbf{m}_A^i)$.
3. A computes $\mathbf{s}_A^1 := (v_1^{(p+1,q+1)} - v_2^{(p+1,q+1)}, \ldots, v_1^{(p+1,q+1)} - v_n^{(p+1,q+1)})$.
4. B computes $\mathbf{s}_B^1 := (c_1^{(p+1,q+1)} - c_2^{(p+1,q+1)}, \ldots, c_1^{(p+1,q+1)} - c_n^{(p+1,q+1)})$.
5. For each relation $b_i^j = b_k^\ell \in \mathcal{R}_1$, A appends $v_i^{(p+1,j)} - v_k^{(p+1,\ell)}$ to $\mathbf{s}_A^2$ and B appends $c_i^{(p+1,j)} - c_k^{(p+1,\ell)}$ to $\mathbf{s}_B^2$
6. B constructs three additional vectors, each of length equal to $|\mathcal{R}_2|$, with $\mathbf{b}^{q+2} = (b_i^1 \hat{c}_j^{(p+1,1)} + (b_j^1 \hat{c}_i^{(p+1,1)})$, $\mathbf{b}^{q+3} = (b_i^1 b_j^1)$, and $\mathbf{b}^{q+4} = \hat{c}_k^{(p+1,1)}$ for triples $(i,j,k) \in \mathcal{R}_2$, and both parties call $\mathsf{Extend}(\mathcal{F}_{\mathsf{OLE}})$,so that A and B now hold $\mathcal{F}_{\mathsf{OLE}}^{(n,\kappa,p+1,q+4,Q'')}$.
7. For each relation $b_i^1 \cdot b_j^1 = b_k^1 \in \mathcal{R}_2$, let r be the index of this relation in $\mathcal{R}_2$. A appends $v_i^{(p+1,1)} \cdot v_j^{(p+1,1)} - \alpha v_k^{(p+1,1)} - v_r^{(p+1,q+2)} - (v_r^{(p+1,q+3)}) \cdot (m_{A,i}^1 + m_{A,j}^1) - (v_r^{(p+1,q+4)}) m_{A,k}^1$ to $\mathbf{s}_A^3$, and B appends $c_i^{(p+1,1)} \cdot c_j^{(p+1,1)} - (c_r^{(p+1,q+3)}) \cdot (m_{A,i}^1 + m_{A,j}^1) - (c_r^{(p+1,q+4)}) m_{A,k}^1$ to $\mathbf{s}_B^3$.
8. Each party P computes $\mathbf{s}_P := \cup_i \mathbf{s}_P^i$.
9. A sends $\mathbf{m}_1 := \mathbf{m} + H(\mathbf{s}_A)$ to B.
10. B computes $\mathbf{m}_2 := \mathbf{m}_1 + H(\mathbf{s}_B)$ and outputs $\mathbf{m}_2$.

DARPA. Y. Ishai supported in part by ERC Project NTSC (742754), BSF grant 2018393, and ISF grant 2774/20.

References

1. Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In *Eurocrypt*, pages 387–404, 2014.
2. Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO '91*, pages 420–432, 1991.
3. Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *CCS 2018*, pages 896–912, 2018.
4. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *CCS 2019*, pages 291–308, 2019.
5. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO 2019, Part III*, pages 489–518, 2019.

6. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-lpn. In *Crypto 2020*, pages 387–416, 2020.
7. Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In *CRYPTO 2021*, pages 502–534. Springer, 2021.
8. SB Damelin, G Michalski, and Gary L Mullen. The cardinality of sets of k-independent vectors over finite fields. *Monatshefte für Mathematik*, 150(4):289–295, 2007.
9. SB Damelin, G Michalski, GL Mullen, and D Stone. The number of linearly independent binary vectors with applications to the construction of hypercubes and orthogonal arrays, pseudo (t, m, s)-nets and linear codes. *Monatshefte für Mathematik*, 141(4):277–288, 2004.
10. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, 2012.
11. Sam Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. In *ITC 2021*, 2021. Full version: https://eprint.iacr.org/2020/1446.
12. Yevgeniy Dodis and Sanjeev Khanna. Space-time tradeoffs for graph properties. In *ICALP 1999*, pages 291–300. Springer, 1999.
13. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In *EUROCRYPT 2011*, pages 406–425, 2011.
14. Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In *Crypto 2018*, pages 365–391. Springer, 2018.
15. Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, pages 52–78, 2007.
16. Payman Mohassel and Matthew K. Franklin. Efficiency tradeoffs for malicious two-party computation. In *PKC*, pages 458–473, 2006.
17. Mike Rosulek and Lawrence Roy. Three halves make a whole? beating the half-gates lower bound for garbled circuits. In *CRYPTO 2021*, pages 94–124, 2021.
18. Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-OLE: Improved constructions and implementation. In *CCS 2019*, pages 1055–1072, 2019.
19. Tamir Tassa and Jorge L Villar. On proper secrets, (t, k)-bases and linear codes. *Designs, Codes and Cryptography*, 52(2):129–154, 2009.
20. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *CCS 2017*, pages 21–37, 2017.
21. Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In *CCS*, 2021. Full version: https://eprint.iacr.org/2021/076.
22. Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In *CCS '20*, pages 1607–1626, 2020.
23. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.
24. Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In *Eurocrypt 2015*, pages 220–250, 2015.

A Supplemental: Additional certification protocols

A.1 Certification between VOLE instances and authenticated triples

We give a lightweight protocol for establishing that the parameter α for an instance of VOLE matches the parameter α from an authenticated triple.

Fig. 13. Certification between authenticated triples and VOLE

Protocol $\Pi_{\mathsf{cert}}^{\mathsf{DAMT}\wedge\mathsf{VOLE}}$: The certification that a value α is consistent across a call to $\mathcal{F}_{\mathsf{DAMT}}$ and a call to $\mathcal{F}_{\mathsf{VOLE}}$.

Inputs are the functionalities $\mathcal{F}_{\mathsf{DAMT}}$, $\mathcal{F}_{\mathsf{VOLE}}$, equipped with the operation Extend, and party A's inputs α, α' to $\mathcal{F}_{\mathsf{DAMT}}, \mathcal{F}_{\mathsf{VOLE}}$, respectively.

1. Both parties call $\mathsf{Extend}(\mathcal{F}_{\mathsf{DAMT}})$ so that the parties learn $(x_{A,1}, x_{A,2})$ and $(x_{B,1}, x_{B,2})$, A and B's respective shares of $(x, \alpha x)$ from a fresh two-sided authenticated triple.
2. Both parties call $\mathsf{Extend}(\mathcal{F}_{\mathsf{subVOLE}})$ so that B learns (a, b) and A learns $v := a\alpha' + b$ generated by the VOLE.
3. B sends $x_{B,1} - a$ to A, so that A now holds $v' := x_{B,1}\alpha' + b$.
4. A computes
$$m := H(v' + x_{A,1}\alpha' - x_{A,2})$$
to B.
5. B verifies that $m = H(x_{B,2} + b)$ and otherwise aborts.

Lemma 8. *$\mathcal{F}_{\mathsf{DAMT}}^{(\rho,n_1)} \wedge_{\alpha,\alpha,\mathbb{F}_{2^\rho}} \mathcal{F}_{\mathsf{VOLE}}^{\rho,n_2}$ is realizable in the $\mathcal{F}_{\mathsf{DAMT}}^{(\rho,n_1+1)} - \mathcal{F}_{\mathsf{VOLE}}^{\rho,n_2+1}$-hybrid model.*

Proof. Let $(x_{A,1}, x_{A,2})$ and $(x_{B,1}, x_{B,2})$ be A and B's respective shares of $(x, \alpha x)$ from a two-sided authenticated triple, and let B hold (a, b) and A hold $\alpha', v := a\alpha' + b$ generated by the VOLE. Using the standard compiler from random VOLE to fixed VOLE, B sends $x_{B,1} - a$ to A, so that A now holds $v' := x_{B,1}\alpha' + b$. Then A computes

$$v' + x_{A,1}\alpha' - x_{A,2} = x\alpha' - x_{A,2} + b = x(\alpha' - \alpha) + x_{B,2} + b,$$

applies a cryptographic hash function H to the result, and sends this to B. Now B compares this value to $H(x_{B,2}+b)$, which they can compute locally, and aborts if the values are not equal. Otherwise they continue with $\mathcal{F}_{\mathsf{VOLE}}$ and $\mathcal{F}_{\mathsf{DAMT}}$.

For security against a malicious B, note that if an adversary $\mathcal{A}$ corrupts B, a simulator $\mathcal{S}$ can simulate the message they receive from A as a random value

from $\mathbb{F}_{2^\kappa}$, by the security of H, and if $\mathcal{A}$ sends the correct information, the simulator simply outputs $H(x_{B,2} + b)$.

Security against a malicious A follows from the security of the random VOLE to fixed VOLE compiler, and because the randomness of b guarantees that a simulator $\mathcal{S}$ can generate v' uniformly at random and match the distribution under the real world protocol execution. Finally, if a malicious $\mathcal{A}$ has $\alpha' \neq \alpha$, guessing $H(x_{B,2} + b)$ is equivalent to guessing x, which $\mathcal{A}$ can only do with negligible probability, and so the ideal world and real world abort probabilities are equal up to a negligible term.

A.2 Certification across VOLEs with reversed sender and receiver

Our second protocol is used to certify that a party playing the role of both VOLE receiver and VOLE sender uses the same value in both protocols.

Fig. 14. Certified across VOLEs with reversed sender and receiver

Protocol $\Pi_{\mathsf{cert}}^{\mathsf{VOLE}\wedge\mathsf{ELOV}}$: The certification that a value α is consistent across two calls to $\mathcal{F}_{\mathsf{VOLE}}$ with roles of sender and receiver reversed.

Inputs are the functionalities $\mathcal{F}_{\mathsf{VOLE}}$, $\mathcal{F}_{\mathsf{ELOV}}$, equipped with the operation Extend, and party B's inputs β, β' to $\mathcal{F}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{ELOV}}$, respectively.

1. Both parties call $\mathsf{Extend}(\mathcal{F}_{\mathsf{VOLE}})$ so that A learns a, c and B learns $w := a\beta + c$.
2. Both parties call $\mathsf{Extend}(\mathcal{F}_{\mathsf{ELOV}})$ so that B holds (β', d) and A learns $v := \beta'\alpha + d$.
3. A chooses a random value e and sends $(m_1, m_2) := (\alpha - a, v + e)$ to B.
4. B computes $m_3 := H(w + m_1\beta + d - m_2)$ to A.
5. A verifies that $m_3 = H(c - e)$ and otherwise aborts.

Lemma 9. *$\mathcal{F}_{\mathsf{VOLE}}^{(\rho,n_1)} \wedge_{\alpha,a_1,\mathbb{F}_{2^\rho}} \mathcal{F}_{\mathsf{ELOV}}^{\rho,n_2}$ is realizable in the $\mathcal{F}_{\mathsf{VOLE}}^{(\rho,n_1+1)} - \mathcal{F}_{\mathsf{ELOV}}^{\rho,n_2+1}$-hybrid model.*

Proof. Suppose A holds the values $a, c, v := \beta'\alpha + d$ and B holds the values $\beta', d, w := a\beta + c$, where the values a, c, α, β are chosen uniformly at random, and B wishes to convince A that $\beta' = \beta$. Then A chooses some random value e and sends $m_1 := (\alpha - a)$ and $m_2 = v + e$ to B. B computes $w + m_1\beta + d - m_2 = c - e$, and sends $m_3 = H(c - e)$ to A. A verifies that $m_3 = H(c - e)$, and otherwise aborts. If A does not abort, both parties continue evaluating the $\mathcal{F}_{\mathsf{VOLE}}$ and $\mathcal{F}_{\mathsf{ELOV}}$ functionalities.

For security against a malicious A, the simulator outputs a random message for m_3 when $\mathcal{A}$ cheats in its message for m_1, and otherwise outputs the correct value of $H(c - e)$, where it computes e as $m_2 - v$, where m_2 is output by $\mathcal{A}$.

When $\mathcal{A}$ cheats in m_1, the value of m_3 is equal to $H((m_1 + a - \alpha)\beta + c - e)$, and so appears random to $\mathcal{A}$, since the value of β is random.

For security against a malicious B, the randomness of a and e ensures that a simulator $\mathcal{S}$ can generate the messages m_1, m_2 uniformly at random and match the distribution under the real world protocol execution. Additionally, if an adversary $\mathcal{A}$ has $\beta' \neq \beta$, then $w + m_1\beta + d - m_2 = \alpha(\beta - \beta') + (c - e)$, and so guessing $H(c-e)$ is equivalent to guessing α, which $\mathcal{A}$ can only do with negligible probability, and so the ideal world and real world abort probabilities are equal up to a negligible term.

A.3 Certification between subfield VOLE and interactively generated subfield VOLE

Fig. 15. Certification of senders' inputs between subVOLE instances with distinct parameters ρ, κ

Protocol $\Pi_{\mathsf{cert}}^{\rho\wedge\kappa}$: The certification that a value b is consistent across two distinct calls to $\mathcal{F}_{\mathsf{subVOLE}}$, which may be generated non-silently.

Inputs are the functionalities $\mathcal{F}_{\mathsf{subVOLE}}^{\rho}$, $\mathcal{F}_{\mathsf{subVOLE}}^{\kappa}$, $\mathcal{F}_{\mathsf{VOLE}}$, equipped with the operation Extend, party A's inputs α, Δ_A, and party B's inputs $\mathbf{b}_1, \mathbf{b}_2$, where we desire to certify that $\mathbf{b}_1 = \mathbf{b}_2$. Assume ρ divides κ for simplicity.

1. The parties A and B call $\mathsf{Extend}(\mathcal{F}_{\mathsf{subVOLE}})$ twice, using the correlation calculus so that B's inputs match across both instances, so that B holds $\mathbf{b}_3 \in \mathbb{F}$, $\mathbf{d}_3 \in \mathbb{F}_{2^\rho}$ and $\mathbf{d}_4 \in \mathbb{F}_{2^\kappa}$, and A holds $\mathbf{v}_3 := \mathbf{b}_3\alpha + \mathbf{d}_3$, $\mathbf{v}_4 := \mathbf{b}_3\Delta_A + \mathbf{d}_4$.
2. B sends $\mathbf{m}_1 := \mathbf{b}_1 - \mathbf{b}_3$ to A.
3. A adds $\mathbf{v}_3 \leftarrow \mathbf{v}_3 + \mathbf{m}_1\alpha$ and $\mathbf{v}_4 \leftarrow \mathbf{v}_4 \leftarrow \mathbf{v}_4 + \mathbf{m}_1\Delta_A$.
4. B computes $m_2 = H(\mathbf{d}_1 - \mathbf{d}_3; \mathbf{d}_2 - \mathbf{d}_4)$ and sends to A.
5. A aborts if $m_2 \neq H(\mathbf{v}_1 - \mathbf{v}_3; \mathbf{v}_2 - \mathbf{v}_4)$.

We note that this protocol is necessary only in the particular setting where A and B generate an instance of fixed $\mathcal{F}_{\mathsf{subVOLE}}$ using some optimized protocol, rather than the generic compiler from random to fixed VOLE, and so the "correlation calculus" cannot be used to ensure that the fixed $\mathcal{F}_{\mathsf{subVOLE}}$ outputs match. We obtain the desired certification simply by replacing these artifically generated instances of $\mathcal{F}_{\mathsf{subVOLE}}$ with fresh instances that are generated with the same vector $\mathbf{b}_3$.

Lemma 10. *$\mathcal{F}_{\mathsf{subVOLE}}^{(\rho,n)} \wedge_{\mathbf{b}_1,\mathbf{b}_2,\mathbb{F}_2} \mathcal{F}_{\mathsf{subVOLE}}^{(\kappa,n)}$ is realizable in the RO-$\mathcal{F}_{\mathsf{subVOLE}}^{(\rho,2n)}$-$\mathcal{F}_{\mathsf{subVOLE}}^{(\kappa,2n)}$-hybrid model under the "correlation calculus" with $O(\kappa)$ bits of communication when A's inputs α, Δ_A are chosen uniformly at random.*

Proof. Correctness follows because, if A and B follow the protocl, each of $\mathbf{v}_i$, for $i = 1, 2, 3, 4$, is of the form $\mathbf{v}_i = \mathbf{b}_1\alpha + \mathbf{d}_i$ or $\mathbf{b}_1\Delta_A + \mathbf{d}_i$, so that $\mathbf{v}_1 - \mathbf{v}_3 = \mathbf{d}_1 - \mathbf{d}_3$ and $\mathbf{v}_2 - \mathbf{v}_4 = \mathbf{d}_2 - \mathbf{d}_4$. A sends no messages in the protocol, so security against a malicious A follows from the security of the underlying correlated randomness functionalities.

Security against a malicious B follows because the vector $\mathbf{m}_1$ is distributed uniformly at random under an honest run of the protocol, by the randomness of $\mathbf{b}_3$, and so a simulator for an adversary $\mathcal{A}$ simply generates $\mathbf{m}_1$ uniformly at random and computes m_2 from $\mathbf{m}_1$.

B Deferred proofs

B.1 proof of Lemma 4

Completeness. When both parties are honest, in Step 13 we have

$$\begin{aligned}\hat{b}_i &:= (\hat{v}_i + \hat{c}_i + v_{i,4} + c_{i,4})\,\beta^{-1} + b_i b_j \\ &= a_i a_j + a_i b_j + a_j b_i + b_i b_j + \hat{a}_i\end{aligned}$$

and

$$\begin{aligned}\hat{d}_i &:= (v_{i,2} + c_{i,2} + v_{i,3} + c_{i,3} + v_{i,5} + c_{i,5})\beta^{-1} + d_{i,j} \\ &= (a_i a_j + a_i b_j + a_j b_i + \hat{a}_i)\alpha + \hat{a}_{i,2} + d_{i,j},\end{aligned}$$

respectively, so that A holds $\hat{w}_i := \hat{a}_{i,2} + w_{i,j} = \hat{b}_i\alpha + \hat{d}_i$, as desired.

Security.

Most of the communication in this protocol involves compilers to fixed VOLE and subfield VOLE from random VOLE and subfield VOLE that only touch the linear terms (that is, the a term in $a\beta + c$, not the c term), and so appear uniformly random by the randomness of each c term. Because of the certification of inputs, B has no space to cheat that will not be detected by A with overwhelming probability, and the only place A can cheat is in the messages $m_{i,1} := \hat{c}_i + c_{i,4}$ and $m_{i,2} := c_{i,2} + c_{i,3} + c_{i,5}$.

For security against a malicious A, in the ideal world if A cheats on any messages $m_{i,1}$ by sending $m_{i,1^*}$, the simulator $\mathcal{S}$ aborts. In the real world, B computes $\hat{b}_i^* = \hat{b}_i + (m_{i,1}^* - m_{i,1})\beta^{-1}$, and aborts unless this lies in $\{0, 1\}$, which happens with negligible probability, since it only happens if A guesses β.

If A cheats on $m_{i,2}$ by sending instead $(m_{i,2}^*)$ the simulator $\mathcal{S}$ records the message $(\mathsf{Cheat}, \mathbf{m}^*)$, where $\mathbf{m}^* = (m_{i,2} - m_{i,2}^*)$, and sends A the vector $\hat{\mathbf{w}} + \mathbf{m}^*\beta^{-1}$. In the real world, let $(\hat{b}_i, \hat{d}_i^*)$ be the values computed by B when A sends $(m_{i,1}, \hat{m}_{i,2})$. Then A can compute

$$\hat{a}_{i,2} + w_{i,j} = \hat{b}_i\alpha + \hat{d}_i = \hat{b}_i\alpha + \hat{d}_i^* + (m_{i,2} - m_{i,2}^*)\beta^{-1} = \hat{w}_i^* + m_i^*\beta^{-1}$$

and so $\mathcal{A}$'s view in the ideal world and real world are indistinguishable. Note that when A holds $w_i^* := \hat{b}_i\alpha + \hat{d}_i^* + m_i^*\beta^{-1}$ and B holds $\hat{b}_i, \hat{d}_i^*$, A has a negligible

probability of sending some false $w_i^* + s$ to B and convincing B that $m_i^* = 0$, since this is equivalent to guessing β. We use this in the proof of Lemma 6.

For security against a malicious B, as noted above the abort probabilities in the ideal world and real world are identical. In the ideal world, a simulator $\mathcal{S}$ chooses all values $\hat{v}_i$ and $v_{i,j}$ uniformly at random except for $\hat{v}_i$ and $v_{i,2}$, and chooses the values $m_{i,1}, m_{i,2}, \hat{b}_i, \hat{d}_i$ uniformly at random.

Then $\mathcal{S}$ computes $\hat{v}_i = (\hat{b}_i - b_i b_j)\beta - m_{i,1} - v_{i,4}$ and $v_{i,2} := (\hat{d}_i - d_{i,j})\beta - m_{i,2} - v_{i,3} - v_{i,5}$.

B.2 Proof of Lemma 6

Proof. **Completeness.** The computation of $L_{k,z_k \oplus \lambda_k}$ is unaltered from [14]. The correctness of the computation of $\lambda_k \oplus z_k$ follows from expanding the expression in Step 5(b) of the protocol for each of the four possible values of $(\lambda_i \oplus z_i, \lambda_j \oplus z_j)$. **Security against a malicious A.** If A cheats during $\mathcal{F}_{\mathsf{cp}}$, then the computation of $L_{k,z_k \oplus \lambda_k}$ will be off by the value $m_k^* \beta^{-1}$, and A has only a negligible probability of successfully offsetting this with suitable adjustments to $G_{k,0}, G_{k,1}$. Thus B will abort with overwhelming probability, and A learns nothing.

The only messages A receives during the protocol are in the compiler to $\mathcal{F}_{\mathsf{cp}}^{(C,\rho,\kappa)}$ and in step 3 and 4, along with a message $\perp$ if B aborts. Let $\mathcal{A}$ be an adversary corrupting A. A simulator $\mathcal{S}$ can match the real world view of $\mathcal{A}$ on steps 3 and 4 by choosing random bits in steps 3 and 4, and the security of step 1 is established by Lemma 4.

Learning whether or not B aborts is equivalent to corrupting some subset of t table entries (by corrupting the messages $G_{i,j}$ or $G'_{i,j}$), and learning whether B opened *any* of those table entries during circuit evaluation. If the t table entries chosen correspond to rows of M_H that are linearly independent, then the labels $M_H \widetilde{b}$ are independent, and A's view can be simulated as the logical conjunction of t random values. Therefore we restrict our attention to the case where the t entries correspond to linearly dependent rows of M_H.

To treat this case, we recall the notion of (t, k)-independent sets (the concept was first introduced in [12], see [19] for a thorough treatment, and [9,8] for additional discussion).

A (t, k)-independent set over $\mathbb{F}_q$ is a subset of $\mathbb{F}_q^k$ such that no $t+1$ element subset is linearly dependent. For our purposes, it is sufficient to construct a $(\rho - 1, L)$-independent set $\mathbb{B} \subseteq \mathbb{F}_2^L$ such that $|\mathbb{B}| = n$ via a randomized algorithm. If we generate n uniformly random vectors from $\mathbb{F}_2^L$, and let $\mathcal{R}$ be a random variable denoting the number of relations on $\mathbb{B}$ with at most ρ elements. We then have

$$\mathbb{E}[\mathcal{R}] \leq \sum_{k=1}^{\rho} \binom{n}{k} (\tfrac{1}{2})^L$$

by linearity of expectation, and so by Markov's inequality we have

$$\Pr[\mathcal{R} \geq 1] \leq \frac{(\rho+1)n^\rho}{(\rho)!2^L},$$

and taking

$$L = \frac{\rho \log n - \rho \log \rho}{\log 2} + 2\rho$$

and by Stirling's approximation, this gives

$$\Pr[\mathcal{R} = 0] \geq 1 - 2^{-\rho}.$$

Thus if the t entries chosen above correspond to linearly dependent rows of M_H, we have $t \geq \rho$. The probability that corrupting ρ independent random table entries causes an abort is equal to $1 - 2^{-\rho}$, and so with $t \geq \rho$, B aborts except with negligible probability, and again A learns nothing. Formally, a simulator $\mathcal{S}$ aborts with probability $1-2^{-t}$ for $t < \rho$, and aborts with probability 1 otherwise, and the view of $\mathcal{A}$ interacting with $\mathcal{S}$ is indistinguishable from the real world execution in both cases.

Security against a malicious B. The proof here is similar to the proof in [14]. When a simulator $\mathcal{S}$ acts as an honest A with input $x = 0$, the view of an adversary $\mathcal{A}$ corrupting B is identical to the view of $\mathcal{A}$ when A uses their actual input, by the security of H and H'.

Similarly, because the wire values a_k are still drawn from a uniformly independent distribution, $\mathcal{A}$'s view of $\lambda_k \oplus z_k$ is uniformly random, whether $x = 0$ or x is A's actual input.

B.3 Proof of Lemma 7

Correctness. When both parties are honest we have

$$s^1_{A,i-1} := v_1^{(p+1,q+1)} - v_i^{(p+1,q+1)} = \alpha\beta + c_1^{(p+1,q+1)} - (\alpha\beta + c_i^{(p+1,q+1)}) = s^1_{B,i-1},$$

and

$$s^2_{A,r} := v_i^{(p+1,j)} - v_k^{(p+1,\ell)} = \alpha b_i^j + c_i^{(p+1,j)} - (\alpha b_k^\ell + c_k^{(p+1,\ell)}) = s^1_{B,r},$$

as desired. For the relation $b_i^1 \cdot b_j^1 = b_k^1$, the calculation is more involved, but similar:

$$\begin{aligned}
s^3_{A,r} &:= v_i^{(p+1,1)} \cdot v_j^{(p+1,1)} - \alpha v_k^{(p+1,1)} - v_r^{(p+1,q+2)} \\
&\quad -(v_r^{(p+1,q+3)}) \cdot (m^1_{A,i} + m^1_{A,j}) - (v_r^{(p+1,q+4)}) m^1_{A,k} \\
&= \alpha^2(b_i b_j - b_k) + (c_i^{(p+1,1)} b_j + c_j^{(p+1,1)} b_i - c_k^{(p+1,1)} \\
&\quad -b_r^{(q+2)} - b_r^{(q+3)}(m^1_{A,i} + m^1_{A,j}) - b_r^{(q+4)} m^1_{A,k})\alpha \\
&\quad +c_i^{(p+1,1)} \cdot c_j^{(p+1,1)} - (c_r^{(p+1,q+3)}) \cdot (m^1_{A,i} + m^1_{A,j}) - (c_r^{(p+1,q+4)}) m^1_{A,k} \\
&= c_i^{(p+1,1)} \cdot c_j^{(p+1,1)} - (c_r^{(p+1,q+3)}) \cdot (m^1_{A,i} + m^1_{A,j}) - (c_r^{(p+1,q+4)}) m^1_{A,k} \\
&= s^3_{B,r}.
\end{aligned}$$

Security against malicious A. Party A receives no messages from B during the protocol besides the compiler from random to fixed $\mathcal{F}_{\mathsf{OLE}}$ in Steps 2 and 6,

and whether or not B aborts. Recall that the message in Step 6 can be moved to Step 2, preserving non-interactivity. The messages in Steps 2 and 6 can be simulated as uniformly random vectors, by the security of the random to fixed $\mathcal{F}_{\mathsf{OLE}}$ compiler.

Assume that B aborts whenever $s_A \neq s_B$, and instruct the simulator $\mathcal{S}$ to construct the correct message $\mathbf{s}'_A$, and abort if the vector $\mathbf{a}^{p+1} \neq \boldsymbol{\alpha}$ or if $\mathbf{s}'_A \neq \mathbf{s}_A$.

If $\mathbf{a}^{p+1} \neq \boldsymbol{\alpha}$, then $a_i^* := a_1^{p+1} - a_i^{p+1} \neq 0$ for some index $i > 1$. But then $c_1^{(p+1,q+1)} - c_i^{(p+1,q+1)} = a_i^*\beta + v_1^{(p+1,q+1)} - v_i^{(p+1,q+1)}$, which A can guess only with probability $1/2^\kappa$, and so B will abort with probability $1 - 1/2^\kappa$ while $\mathcal{S}$ aborts with probability 1.

If $\mathbf{a}^{p+1} = \boldsymbol{\alpha}$, then all terms $c_i^{(p+1,j)}$ will be computed correctly by b, and so we have $\mathbf{s}'_A = \mathbf{s}_B$, by the correctness of the protocol. Therefore the probability that $\mathcal{S}$ aborts is computationally indistinguishable from the probability that B aborts during a real world execution of the protocol, and a cheating A is detected with overwhelming probability.

Security against malicious B. We construct a simulator $\mathcal{S}$ that has access to the intended message $\mathbf{m}$ in the ideal world setting. $\mathcal{S}$ generates $(\mathbf{m}_A^i)$ uniformly at random, computes $H(\mathbf{s}_B)$, and outputs $\mathbf{m}_1 = \mathbf{m} + H(\mathbf{s}_B)$ if B follows the protocol honestly, and a random message otherwise. The distribution of $(\mathbf{m}_A^i)$ matches the distribution under a real world execution of the protocol by the correctness of $\mathcal{F}_{\mathsf{OLE}}$.

As above, if $\mathbf{b}^{q+1} \neq \boldsymbol{\beta}$, then $b_i^* := b_1^{q+1} - b_i^{q+1} \neq 0$ for some index $i > 1$. But then $v_1^{(p+1,q+1)} - v_i^{(p+1,q+1)} = b_i^*\alpha + c_1^{(p+1,q+1)} - c_i^{(p+1,q+1)}$, which B can guess only with probability $1/2^\kappa$, and so $s_{B,i-1} \neq s_{A,i-1}$ with overwhelming probability. Similarly, if $b_i^j \neq b_k^\ell$, the value $v_i^{(p+1,j)} - v_k^{(p+1,\ell)}$ will be off by some multiple of α, and if $b_i^1 b_j^1 \neq b_k^1$, then the expression $s_{B,r}^3$ will be off by some multiple of α^2 (and a possibly additional multiple of α). Finally, if B has inputs that satisfy the relations $\mathcal{R}$, but cheats on the values $\mathbf{b}^{q+i}$, for $i \in \{2,3,4\}$, then in Step 7, B will hold some linear expression in α, whose coefficients B can compute. Then the expression $H(\mathbf{s}_B)$ will not be equal $H(\mathbf{s}_A)$ with overwhelming probability if the coefficient of α in this expression is nonzero. In the ideal world execution, the simulator $\mathcal{S}$ can likewise compute the coefficients of this linear expression from the adversary's messages and the randomly generated $(\mathbf{m}_A^i)$, and outputs a random string for $\mathbf{m}_2$ if and only if the coefficient of α is nonzero.

B.4 Non-interactive circuit wires from programmable OLE

We define a modified form of the functionality $\mathcal{F}_{\mathsf{pre}}^{(C,\rho,\kappa)}$, replacing the last line of $\mathcal{F}_{\mathsf{pre}}$ with the last four lines of $\mathcal{F}_{\mathsf{cp}}$, and call it $\mathcal{F}_{\mathsf{pre-wbc}}^{(C,\rho,\kappa)}$, preprocessing with blind cheating. In other words, as in $\mathcal{F}_{\mathsf{cp}}$, we allow party A to cheat in such a way that is not immediately detected, but leaves A with corrupted shares that cannot be used as shares of $\hat{a}_i\Delta_A$.

We give the protocol realizing this functionality in Figure 16. The proof of correctness is similar to the proof for our VOLE-based protocol. Here we give a careful accounting of the total communication cost.

The CDS protocol requires an additional κn bits of communication for A and an additional $3|\mathcal{R}_2|\kappa$ bits for B. The relations we need to verify on B's inputs are $\mathbf{b}_L \cdot \beta = \mathbf{b}_L\beta$, $\mathbf{b}_R \cdot \beta = \mathbf{b}_R\beta$, and $\mathbf{b}_L \cdot \mathbf{b}_R\beta = b_i b_j \beta$, so the cost of CDS is equal to $9\kappa n$ bits of communication for B.

We can use A's constant value from $\mathbf{\Pi}_{\mathsf{CDS}}$ as Δ_A, and the values $a_{i,2}$ can be chosen uniformly at random over $\mathbb{F}_{2^\kappa}$, but the remainder of A's inputs require communication, giving an additional 10κ bits of communication for A. Similarly B requires another 7κ bits of communication.

This gives 11κ bits of communication for A and 16κ bits of communication for B.

Adding in the $2\kappa + 3\rho$ bits of communication for A in the garbled circuit gives total communication of $13\kappa + 3\rho$ bits for A and 16κ bits for B.

B.5 Non-interactive authenticated garbling

We only make two changes from $\mathbf{\Pi}^{\mathsf{VOLE}}_{\mathsf{2pc}}$ to give the garbling protocol $\mathbf{\Pi}^{\mathsf{NISC}}_{\mathsf{2pc}}$. First, we replace $\mathbf{\Pi}_{\mathsf{cp}}$ with $\mathbf{\Pi}^{\mathsf{nisc}}_{\mathsf{pre}}(C, \rho, \kappa)$ in Step 1. Second, we move the communication in Step 3 to Step 1.

The correctness and non-interactivity of $\mathbf{\Pi}^{\mathsf{nisc}}_{\mathsf{pre}}$ follows from the previous subsection. Moving B's message earlier does nothing to help a malicious B, so security against B still holds. What remains is to verify security against a malicious A.

The proof of security is very close to the proof of security in [14]. Essentially, we notice that the proof does not anywhere require Step 3 to occur after Step 2, and so everything goes through after re-ordering. We give the formal proof below for completeness.

First, we note that, if $\lambda_i \oplus z_i$ and $\lambda_j \oplus z_j$ are correct, the expression in $5(b)$ will either be equal to $\lambda_k \oplus z_k$ or $\lambda_k \oplus z_k \oplus \beta^{-1}x^*$, for some value x^*. Therefore either B aborts or B outputs $f(x, y)$ with overwhelming probability.

Next, let $\mathcal{A}$ be an adversary corrupting A. We construct a simulator $\mathcal{S}$ that runs $\mathcal{A}$ as a subroutine and plays the role of A in the ideal world involving an ideal functionality $\mathcal{F}$ evaluating f. $\mathcal{S}$ is defined as follows.

- Step 1. $\mathcal{S}$ plays the role of $\mathcal{F}^{(C,\rho,\kappa)}_{\mathsf{pre-wbc}}$ and records all values that $\mathcal{F}^{(C,\rho,\kappa)}_{\mathsf{pre-wbc}}$ sends to both parties.
- Step 3. (re-ordered) $\mathcal{S}$ acts as an honest B using $y := 0$.
- Step 2. $\mathcal{S}$ receives A's messages.
- Step 4. $\mathcal{S}$ receives A's message $\hat{x}_i$ and computes $x_i := \hat{x}_i \oplus a_i$, where a_i is the value used by $\mathcal{F}^{(C,\rho,\kappa)}_{\mathsf{pre-wbc}}$ previously.
- Steps 5-6. $\mathcal{S}$ acts as an honest B, and aborts if B would abort, and otherwise sends x to $\mathcal{F}$.

We show that the joint distribution on the outputs of $\mathcal{A}$ and an honest B in the real world execution is indistinguishable from the joint distribution on the outputs of $\mathcal{A}$ and $\mathcal{S}$ in an ideal world execution, through a series of hybrid model protocols.

Fig. 16. Non-interactive authenticated wire labels from programmable OLE

Protocol $\mathbf{\Pi}^{\mathsf{nisc}}_{\mathsf{pre}}(C, \rho, \kappa)$: Non-interactive pre-processing of wire labels for authenticated garbling from programmable OLE.

Parametrized by the value ρ, and a circuit C consisting of $\mathcal{W}$ wires, $\mathcal{I}$ input wires, $\mathcal{O}$ output wires, and gates $\mathcal{G}$ of the form (i, j, k, T), for $T \in \{\wedge, \oplus\}$, $i, j \in \mathcal{I} \cup \mathcal{W}$, and $k \in \mathcal{W} \cup \mathcal{O}$. Let n be the number of AND gates.

1. The parties invoke a $\mathbf{\Pi}^{(n,\kappa,11,7,Q,\mathcal{R})}_{\mathsf{CDS}}$ functionality, where A's inputs are $(\boldsymbol{\Delta}_A, \mathbf{a}, \mathbf{a}_L, \mathbf{a}_R, (a_i a_j), \hat{\mathbf{a}}, \Delta_A \mathbf{a}, \Delta_A \mathbf{a}_L, \Delta_A \mathbf{a}_R, \Delta_A (a_i a_j), \Delta_A \hat{\mathbf{a}}, (a_{i,2}))$ and B's inputs are the set $(\mathbf{b} + \gamma, \mathbf{b}_L + \gamma, \mathbf{b}_R + \gamma, \mathbf{b}_L \beta + \gamma, \mathbf{b}_R \beta + \gamma, b_i b_j \beta + \gamma, \boldsymbol{\beta}, \boldsymbol{\gamma})$. The correspondence between A inputs and B inputs given by Q arise in the protocol description. The relations $\mathcal{R}_1$ are the permutations required to construct $\mathbf{b}_L$ and $\mathbf{b}_R$. The relations $\mathcal{R}_2$ are the product relations on b_i, b_j, β.
2. A and B store the resulting value $\mathbf{s}$ and add $H(\mathbf{s})$ to all subsequent messages.
3. Party B computes $\hat{v}_i, v_{i,2}$ as in $\mathbf{\Pi}_{\mathsf{cp}}$, and computes

$$v_{i,3} := \hat{a}_i \Delta_A \beta + c_{i,3}$$

$$v_{i,4} := (a_i a_j + a_i b_j + a_j b_i + b_i b_j)\beta + c_{i,4}$$

$$v_{i,5} := (a_i a_j + a_i b_j + a_j b_i + b_i b_j)\Delta_A \beta + c_{i,5}$$

4. A and B construct as entries of OLE:

$$u_i = \Delta_A (b_i + \gamma) + e_i$$

and

$$u'_i = \Delta_A (\gamma) + e'_i,$$

The parties then take $w_i = e_i + e'_i$ and $d_i = u_i + u'_i$ in $\mathcal{F}^{(C,\rho,\kappa)}_{\mathsf{pre}}$. The value $w_{i,j} := b_i b_j \Delta_A \beta + d_{i,j}$ is computed similarly.
5. As in $\mathbf{\Pi}_{\mathsf{cp}}$, A sends the messages $\hat{c}_i + c_{i,4}$ and $c_{i,2} + c_{i,3} + c_{i,5}$ to B to open $\hat{b}_i, \hat{d}_i$ to B, and A can locally compute $\hat{w}_i$.
6. The parties produce entries of OLE corresponding to secret shares of the product of each of $a_i a_j, \Delta_A a_i, \Delta_A a_i a_j, \Delta_A \hat{a}_i$ with β.
7. Treating the resulting secret sharing as a realization of VOLE, A and B invoke $\mathbf{\Pi}_{\mathsf{LPZK}}$ to verify that the terms $a_i a_j, \Delta_A a_i, \Delta_A a_i a_j, \Delta_A \hat{a}_i$ have been computed correctly.

Hybrid-1. $\mathcal{S}$ plays the role of an honest B, using B's input y, and also plays the role of $\mathcal{F}_{\mathsf{pre-wbc}}^{(C,\rho,\kappa)}$. **Hybrid-2.** $\mathcal{S}$ plays the role of an honest B, using B's input y, and also plays the role of $\mathcal{F}_{\mathsf{pre-wbc}}^{(C,\rho,\kappa)}$, in steps 1-3. In step 4, $\mathcal{S}$ extracts the value $x_i := \hat{x}_i \oplus a_i$, and then aborts if B would abort and otherwise sends x to $\mathcal{F}$. **Hybrid-3.** $\mathcal{S}$ follows the ideal-world setting laid out above.

There is no difference in $\mathcal{A}$'s view between **Hybrid-1** and **Hybrid-2**, because $\mathcal{S}$ aborts in both cases if B would abort, and as noted above, if B fails to abort, it will necessarily output $f(x, y)$, which is also what is output by **Hybrid-2**.

The only difference between **Hybrid-2** and **Hybrid-3** is that $\mathcal{S}$ sets $y := 0$, but the values $y_i \oplus b_i = b_i$ will still appear totally random to $\mathcal{A}$. All of B's calculations in steps 5 and 6 do not depend on y, only on the values $y_i \oplus b_i$ and $z_i \oplus \lambda_i$, so that whether or not B aborts is not affected by which scenario we are in, and $\mathcal{A}$ will be unable to distinguish between **Hybrid-2** and **Hybrid-3**. This completes the proof.

C Formalization of the correlation calculus