Secure Computation with Preprocessing via Function Secret Sharing[*]

Elette Boyle[†] Niv Gilboa[‡] Yuval Ishai[§]

Abstract

We propose a simple and powerful new approach for secure computation with input-independent preprocessing, building on the general tool of *function secret sharing* (FSS) and its efficient instantiations. Using this approach, we can make efficient use of correlated randomness to compute any type of gate, as long as a function class naturally corresponding to this gate admits an efficient FSS scheme. Our approach can be viewed as a generalization of the "TinyTable" protocol of Damgård et al. (Crypto 2017), where our generalized variant uses FSS to achieve exponential efficiency improvement for useful types of gates.

By instantiating this general approach with efficient PRG-based FSS schemes of Boyle et al. (Eurocrypt 2015, CCS 2016), we can implement useful nonlinear gates for equality tests, integer comparison, bit-decomposition and more with optimal online communication and with a relatively small amount of correlated randomness. We also provide a unified and simplified view of several existing protocols in the preprocessing model via the FSS framework.

Our positive results provide a useful tool for secure computation tasks that involve secure integer comparisons or conversions between arithmetic and binary representations. These arise in the contexts of approximating real-valued functions, machine-learning classification, and more.

Finally, we study the necessity of the FSS machinery that we employ, in the simple context of secure string equality testing. First, we show that any "online-optimal" secure equality protocol implies an FSS scheme for point functions, which in turn implies one-way functions. Then, we show that *information-theoretic* secure equality protocols with relaxed optimality requirements would follow from the existence of big families of "matching vectors." This suggests that proving strong lower bounds on the efficiency of such protocols would be difficult.

本文主要内容：
1）基于FSS构造了针对相等、整数比较、位分解等等的非线性门计算方案，为包含整数比较、A2B或B2A的应用提供有用工具
2）提供了一个统一且简单的视角取看待已存在的预处理模型下的安全计算协议
3）最后，系统研究了FSS在预处理模型下MPC的必要性

[*] This is a full version of [7].

[†] IDC, Israel, eboyle@alum.mit.edu.
[‡] Ben Gurion University, Israel, gilboan@bgu.ac.il.
[§] Technion, Israel, yuvali@cs.technion.ac.il.

1 Introduction

The power of correlated randomness in secure computation has recently been an active area of research. In the setting of *secure computation with preprocessing*, two or more parties receive correlated random inputs from a trusted dealer in an offline phase, before the inputs are known. In a subsequent online phase, once the inputs are known, the parties use this correlated randomness to obtain significant speedup over similar protocols in the plain model, either unconditionally or under weaker cryptographic assumptions. Alternatively, in the absence of a trusted dealer, the correlated randomness can be generated via an interactive secure protocol that is executed offline, before the inputs are known, and only the outputs of this protocol need to be stored for later use. For simplicity, we focus in this work on the case of secure *two-party* computation with security against *semi-honest* adversaries.[1]

半诚实模型下的安全两方计算

Originating from the work of Beaver [2], who showed how to use "multiplication triples" for secure arithmetic computation with no honest majority, many current protocols for secure computation make extensive use of correlated randomness. Commonly used types of two-party correlations include garbled circuit correlations, OT and OLE correlations, multiplication ("Beaver") triples and their authenticated version, and one-time truth-tables [2, 4, 15, 23, 14, 12].

Motivated by secure computation applications that involve integer comparisons or conversions between arithmetic and boolean values, we introduce a simple and powerful new approach for secure computation in the preprocessing model. Our approach is based on the general tool of *function secret sharing* (FSS) [8] and its efficient instantiations from any pseudorandom generator. Informally, a (2-party) FSS scheme splits a function $f : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$ from a function class $\mathcal{F}$, where $\mathbb{G}^{\mathsf{in}}$ and $\mathbb{G}^{\mathsf{out}}$ are finite Abelian[2] groups, into two functions, f_0 and f_1, such that (1) each f_σ is represented by a compact key k_σ that allows its efficient evaluation; (2) each key k_σ hides the function f; and (3) for any input $x \in \mathbb{G}^{\mathsf{in}}$ we have $f(x) = f_0(x) + f_1(x)$.

函数秘密共享的概念

The idea in a nutshell. Our FSS-based approach for secure computation with preprocessing is very simple. Denote the two parties by P_0 and P_1. We represent the function being evaluated as a circuit C, in which inputs and internal wires take values from (possibly distinct) groups. The circuit nodes are labeled by gates, where each gate g maps an input from a group $\mathbb{G}^{\mathsf{in}}$ into an output from a group $\mathbb{G}^{\mathsf{out}}$. Note that we can use product groups to capture a gate with multiple input or output wires. To securely evaluate C in the preprocessing model, the dealer generates and distributes the following types of correlated randomness. First, for every wire j in C, the dealer picks a random mask r_j from the corresponding group. Each party P_σ receives the random masks of the input wires it owns. The online phase evaluates the circuit gate-by-gate in a topological order, maintaining the following invariant: for every wire value w_j in C, *both* parties learn the masked value $w_j + r_j$. This is easy to achieve at the inputs level: if input x_i is owned by party P_σ, this party can simply compute and send $x_i + r_i$ to the other party.

The key idea is the following FSS-based gate evaluation procedure. For each gate $g : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$, the dealer uses an FSS scheme for the class of *offset* functions $\mathcal{G}$ that includes all functions of the form $g_{r^{\mathsf{in}},r^{\mathsf{out}}}(x) = g(x - r^{\mathsf{in}}) + r^{\mathsf{out}}$. If the input to gate g is wire i and the output is wire j, the

[1] Our techniques naturally generalize to the multi-party setting, though typically with reduced efficiency benefits over alternative approaches. Moreover, most of our protocols can be extended to the malicious security model by employing simple authentication techniques (as in [4, 15]).

[2] Unlike previous applications of FSS, here it is important that the *input* domain additionally be endowed with group structure. From here on, the term "group" will always refer to a finite Abelian group.

dealer uses the FSS scheme for $\mathcal{G}$ to split the function g_{r_i,r_j} into two functions with keys k_0, k_1, and delivers each key k_σ to party P_σ. Now, evaluating their FSS shares on the common masked input $w_i + r_i$, the parties obtain additive shares of the masked output $w_j + r_j$, which they can exchange and maintain the invariant for wire j. Finally, the outputs are reconstructed by having the dealer reveal to both parties the masks of the output wires.

The above protocol is not only simple, but in a sense is implicit in the literature. It can be viewed as a generalization of the "TinyTable" protocol of Damgård et al. [14], where the novel idea is to use efficient FSS for achieving exponential compression (and speedup[3]) for natural types of gates that are useful in applications. We discuss several useful instances of this approach below.

相关随机性和电路独立：输出线r^out被定义为下一个电路门的输入线

While the correlated randomness in the above protocol depends on the topology of C, we also present a *circuit-independent* variant where the input and output masks of different gates are independent of each other. In this variant, for each gate g the dealer chooses additive r^{in} offsets only for the *input* wires of g, and provides FSS shares for the function $g_{r^{\mathsf{in}},0}(x) = g(x - r^{\mathsf{in}}) + 0$, together with *additive shares* of r^{in}. During the online phase, the parties can "match up" the offsets for adjacent gates, and non-interactively emulate FSS shares of $g_{r^{\mathsf{in}},r^{\mathsf{out}}}(x) = g(x - r^{\mathsf{in}}) + r^{\mathsf{out}}$ using the additive shares, where r^{out} is defined to be $(r^{\mathsf{in}})'$ for the appropriate next gate g'. The resulting online communication is one element per *wire*, as opposed to only one element per computed wire *value* as in the circuit-dependent version (where circuit fan-out introduces extra wires but not new wire values).

Finally, one could alternatively consider a variant of our protocols in which FSS is used to convert *secret-shared* inputs to secret-shared outputs rather than common *masked* inputs to masked outputs. Whereas in the above protocol both parties first apply FSS on the common masked input and then exchange their output shares to obtain a masked output, in the alternative variant they start by reconstructing a common masked input from their input shares, and then apply FSS to directly obtain the output shares.

应用FSS直接获得输出的秘密份额

Application: simple derivation of existing protocols. By using simple information-theoretic FSS schemes for truth-tables and low-degree polynomials, our FSS-based approach can be used to derive in a simple and unified way several previous protocols for secure computation in the preprocessing model. For instance, protocols from [2, 23, 14, 24, 12] can be easily cast in this framework. We also present useful generalizations of such protocols to broader classes of algebraic computations.

Application: online-optimal secure equality, comparison, bit decomposition, and more. Our FSS-based technique yields a simple new approach for securely performing useful nonlinear operations on masked or secret-shared values. We first describe the types of nonlinear operations we can efficiently support, and then the efficiency features of our FSS-based solution.

When performing secure arithmetic computations, it is often useful to switch between an *arithmetic* representation, where the values are secret-shared over a big modulus $\mathbb{Z}_q$, and a Boolean representation, where the values are secret shared bit-by-bit over $\mathbb{Z}_2$. Other useful nonlinear operations include zero-testing of a shared value or equality testing of two shared values, comparing between different integer values (i.e., the "greater than" predicate), or checking if an integer value is in an interval. For all of the above predicates, the input is secret-shared over $\mathbb{Z}_q$ and the 0/1

[3] A general method for compressing truth-table correlations was recently suggested in [6]. However, the *running time* still grows linearly with the truth-table size, or exponentially with the gate input length.

output is secret-shared for further computations over either $\mathbb{Z}_2$, $\mathbb{Z}_q$, or another group. A more general class of nonlinear computations are *spline functions* that output a different polynomial on each interval. A useful special case is the ReLU function $g(x) = \max(0, x)$ that is commonly used as an activation function in neural networks. Finally, one can also consider a garbling-compatible variant of the above operations, where the bits of the output select between pairs of secret keys that can be fed to a garbled circuit.

In all of the above cases, we can use computationally secure FSS schemes based on one-way functions [20, 8, 9] to efficiently realize the corresponding offset classes $\mathcal{G}$ using only symmetric cryptography. Concretely, for all the above types of gates we can use efficient preprocessing to convert shares of an input into shares of an output with optimal online communication that only involves a *single round* of exchanging masked input shares and no further interaction. Each party can then directly compute its share of the output given its part of the correlated randomness and the message received from the other party.

The above types of nonlinear "FSS gates" can provide a valuable toolbox for the large body of work on secure machine learning classification, secure implementation of bounded-precision arithmetic, and secure approximations of real-valued functions. In fact, they can even be useful for evaluating standard Boolean circuits. For instance, evaluating an AND/OR gate with fan-in m reduces to a secure equality of m-bit strings.

Comparison with prior approaches. There is a long line of work on secure implementation of useful nonlinear computations such as bit-decomposition in different models (see [13, 16, 11, 26] and references therein). As discussed above, our FSS-based technique has an *optimal online cost* of converting secret-shared inputs to secret-shared outputs. Compared to the commonly used "ABY framework" [16] for performing such operations using garbled circuits, our approach has better round complexity (1 instead of 2 rounds) and, more importantly, it avoids the big overhead of sending a key for each bit of the input. In concrete terms, this improves the online communication complexity by two orders of magnitude. Even in the relatively simple cases of equality testing and integer comparison, where improved special-purpose protocols are known (see [11] and references therein), our FSS-based approach has significant advantages over the best previous protocols.

[16]ABY, [26]ABY3

The low online cost of our FSS-based protocols is inherited from the efficiency of recent constructions of FSS schemes for point functions, intervals, and decision trees [20, 8, 9]. These constructions make a black-box use of any pseudorandom generator, which can be instantiated by AES in practice. Thus, for the type of "gates" supported by such simple FSS schemes, our protocols significantly improve the online communication complexity and round complexity of prior approaches while still being very computationally efficient in the online phase. See Table 1 for comparison.

生成相关随机性

Realizing the dealer. We turn to discuss the offline cost of securely generating and storing the correlated randomness. The amount of correlated randomness used by our protocols is dominated by the size of the FSS keys. For equality and comparison gates, this includes a linear number of PRG seeds (e.g., AES keys) in the bit-length of the inputs, and for bit-decomposition it involves a quadratic number of PRG seeds. When the input domain is not too big, the distributed generation of this correlated randomness can be done with good concrete efficiency using a distributed FSS key generation protocol of Doerner and shelat [17]. Otherwise one can use concretely efficient general-purpose secure computation protocols (such as [25]) for emulating the dealer. Finally, one can avoid the cost of securely distributing the correlated randomness by using a third party as a dealer and settling for security against a single corrupted party. This is similar to the 3-party ABY^3 framework from [26], except that here the third party is only used to generate correlated randomness and can remain offline during the actual computation.

Is FSS necessary? Our most useful positive results make use of symmetric cryptography. Given that most protocols in the preprocessing model are *information-theoretic*, one may ask if it is possible to obtain similar results in the information-theoretic model with a polynomial amount of correlated randomness. For simplicity, we consider a *shared equality* protocol with optimal online complexity. In such a protocol, the parties hold n-bit strings x_0 and x_1, and in a single round of interaction they send an n-bit message to each other. These messages should hide their inputs. Following this interaction, they each locally output a single output bit such that the exclusive-or of the two bits is 1 if and only if $x_0 = x_1$. We show that our FSS machinery is not only sufficient for obtaining this type of protocols, but is also necessary. In particular, any protocol as above implies the existence of a one-way function. (This implication is more subtle than it may seem since unlike our simple FSS-based protocol, a general shared equality protocol may correlate the randomness used to mask the inputs with the randomness used to compute the output shares.) On the other hand, we show that efficient *information-theoretic* shared equality protocols with constant-size output shares would follow from the existence of big families of "matching vectors" [22, 19, 18], a longstanding open problem in extremal combinatorics. This suggests that strong lower bounds on the efficiency of information-theoretic shared equality protocols would be difficult to obtain.

Organization. In Section 2, we provide necessary preliminaries. In Section 3, we present our general framework for secure computation with preprocessing via FSS. In Section 4, we present applications, instantiating the necessary FSS schemes for specific motivated computation tasks. We conclude in Section 5 by exploring negative results and barriers.

2 Preliminaries

2.1 Representing Functions

In order to seamlessly handle both arithmetic and Boolean operations, we will consider all functions to be defined over Abelian groups. For instance, a Boolean function $f : \{0,1\}^n \to \{0,1\}^m$ will be viewed as a mapping from the group $\mathbb{Z}_2^n$ to the group $\mathbb{Z}_2^m$. Given our heavy use of function secret sharing, we use a similar convention for function representation to the one used in [9] (the only difference being that here we also endow the input domain with a group structure).

Definition 2.1 (Function families)**.** A *function family* is defined by $\mathcal{F} = (P_{\mathcal{F}}, E_{\mathcal{F}})$, where $P_{\mathcal{F}} \subseteq \{0,1\}^*$ is an infinite collection of function descriptions $\hat{f}$ and $E_{\mathcal{F}} : P_{\mathcal{F}} \times \{0,1\}^* \to \{0,1\}^*$ is a polynomial-time algorithm defining the function described by $\hat{f}$. Concretely, $\hat{f} \in P_{\mathcal{F}}$ describes a corresponding function $f : D_f \to R_f$ defined by $f(x) = E_{\mathcal{F}}(\hat{f}, x)$. We require D_f and R_f to be finite Abelian groups, denoted by $\mathbb{G}^{\mathsf{in}}$ and $\mathbb{G}^{\mathsf{out}}$ respectively. We will typically let $\mathbb{G}^{\mathsf{in}}$ and $\mathbb{G}^{\mathsf{out}}$ be product groups, which can capture the case of multiple inputs and outputs. When there is no risk of confusion, we will sometimes write f instead of $\hat{f}$ and $f \in \mathcal{F}$ instead of $\hat{f} \in P_{\mathcal{F}}$. We assume that $\hat{f}$ includes an explicit description of $\mathbb{G}^{\mathsf{in}}$ and $\mathbb{G}^{\mathsf{out}}$.

By convention, we denote by $0 \in \mathbb{G}$ the identity element of $\mathbb{G}$. We will use the notation $1 \in \mathbb{G}$ to denote a fixed canonical nonzero element of $\mathbb{G}$; when $\mathbb{G}$ is additionally endowed with a

Gate	Protocol	Online communication	Online	Offline storage
Type		(bits per party)	rounds	(bits)
Zero test	[11]	$m + o(m)$	≥ 3	$2m + o(m)$
	ABY [16]	$O(\lambda m)$	2	$O(\lambda m)$
	Prop. 4.2	m	1	$\approx \lambda m$
Zero test example	[11]	77	3	152
$m = 64$	**Prop. 4.2**	64	1	8384
Integer comparison	[11] SC1	$O(m)$	$O(\log \log m)$	$3m + o(m)$
	ABY [16]	$O(\lambda m)$	2	$O(\lambda m)$
	Prop. 4.5	m	1	$\approx 4\lambda m$
Comparison example	[11] SC1	1120	12	≈ 300
$m = 64$	**Prop. 4.5**	64	1	≈ 33000
Bit decomposition	ABY [16]	$O(\lambda m)$	2	$O(\lambda m)$
	Prop. 4.9	m	1	$\approx 2\lambda m^2$
Spline over $\mathbb{Z}_{2^m}$	ABY [16]	$O(m(\lambda k + d))$	2	$O(m(\lambda k + d))$
$k + 1$ deg.-d polynomials	**Prop. 4.6**	m	1	$\approx 8km(\lambda + d)$

Table 1: Comparison of the performance of our protocols to the ABY framework by Demmler, Schneider and Zohner [16] and protocols of Couteau [11] (bit-decomposition is not directly supported by [11]). The inputs are taken from a group $\mathbb{G}^{\mathsf{in}}$ with $m = \lceil \log |\mathbb{G}^{\mathsf{in}}| \rceil$ (e.g., $\mathbb{G}^{\mathsf{in}} = \mathbb{Z}_{2^m}$). We let λ denote the seed length of a length-doubling PRG ($\lambda = 128$ for an AES-based implementation) and use big-O notation to hide small constants that are strictly bigger than 1. Online rounds allow one message per party per round. The specified complexity refers to converting secret-shared input to secret-shared output, where the input sharing is over $\mathbb{G}^{\mathsf{in}}$ and the output sharing is over $\mathbb{Z}_2$ (for zero test or comparison) or $\mathbb{Z}_2^m$ (for bit-decomposition). The online computational cost of our protocols is dominated by roughly s/λ invocations of the PRG, where s is the offline storage in bits.

multiplicative structure, e.g., when $\mathbb{G}$ is the additive group of a finite *ring*, 1 will be set to the multiplicative identity.

2.2 Secure Computation with Preprocessing

We follow the standard definitional framework for secure computation (cf. [10, 21]), except that we allow a trusted input-independent setup phase that distributes correlated secret randomness to the parties. This setup phase can be securely emulated by an interactive preprocessing protocol that can be carried out before the inputs are known. We focus here on protocols with security against a *semi-honest* adversary who may non-adaptively corrupt any strict subset of parties. For simplicity, we explicitly spell out the definitions for the two-party case, and later explain the (straightforward) extension to the multi-party case.

Functionalities. We denote the two parties by P_0 and P_1 and a party index by $\sigma \in \{0,1\}$. We consider by default protocols for *deterministic* functionalities that deliver the *same output* to the two parties. The general case (of randomized functionalities with different outputs) can be reduced to this case via a standard reduction [10, 21]. A two-party functionality f is described by a bitstring $\hat{f}$ via a function family $\mathcal{F}$, as in Definition 2.1. We assume that the input domain $\mathbb{G}^{\mathsf{in}}$ is split into $\mathbb{G}^{\mathsf{in}} = \mathbb{G}^{\mathsf{in}}_0 \times \mathbb{G}^{\mathsf{in}}_1$, capturing the inputs of the two parties.

Protocols with preprocessing. A two-party protocol is defined by a pair of PPT algorithms $\Pi = (\mathsf{Setup}, \mathsf{NextMsg})$. The setup algorithm $\mathsf{Setup}(1^\lambda, \hat{f})$, given a security parameter λ and functionality description $\hat{f}$, outputs a pair of correlated random strings (r_0, r_1). We also consider protocols with *function-independent preprocessing*, in which Setup only receives a bound 1^S on the size of $\hat{f}$ instead of $\hat{f}$ itself. The *next-message function* $\mathsf{NextMsg}$ determines the messages sent by the two parties. Concretely, the function $\mathsf{NextMsg}$, on input $(\sigma, j, \hat{f}, x_\sigma, r_\sigma, \mathbf{m})$, specifies the message sent by party P_σ in Round j depending on the functionality description $\hat{f}$, input x_σ, random input r_σ, and vector $\mathbf{m}$ of previous messages received from $P_{1-\sigma}$. We assume both parties can speak to each other in the same round. (In the semi-honest model, one can eliminate this assumption by at most doubling the number of rounds.) If the output of $\mathsf{NextMsg}$ is of the form (Out, y) then party P_σ terminates the protocol with output y. We denote by $\mathsf{Out}_{\Pi,\sigma}(\lambda, \hat{f}, (x_0, x_1))$ and $\mathsf{View}_{\Pi,\sigma}(\lambda, \hat{f}, (x_0, x_1))$ the random variables containing the output and view of party P_σ (respectively) in the execution of Π on inputs (x_0, x_1), where the view includes r_σ and messages received from $P_{1-\sigma}$.

Security definition. We require both *correctness* and *security*, where security is captured by the existence of a PPT algorithm Sim that simulates the view of a party given its input and output alone. We formalize this below.

Definition 2.2 (Secure computation with preprocessing)**.** We say that $\Pi = (\mathsf{Setup}, \mathsf{NextMsg})$ *securely realizes a function family $\mathcal{F}$ in the preprocessing model* if the following holds:

- *Correctness:* For all $\hat{f} \in P_{\mathcal{F}}$ describing $f : \mathbb{G}^{\mathsf{in}}_0 \times \mathbb{G}^{\mathsf{in}}_1 \to \mathbb{G}^{\mathsf{out}}$, $(x_0, x_1) \in \mathbb{G}^{\mathsf{in}}_0 \times \mathbb{G}^{\mathsf{in}}_1$, $\lambda \in \mathbb{N}$, $\sigma \in \{0,1\}$, we have $\Pr[\mathsf{Out}_{\Pi,\sigma}(\lambda, \hat{f}, (x_0, x_1)) = f(x_0, x_1)] = 1$.
- *Security:* For each corrupted party $\sigma \in \{0,1\}$ there exists a PPT algorithm Sim_σ (simulator), such that for every infinite sequence $(\hat{f}_\lambda)_{\lambda \in \mathbb{N}}$ of polynomial-size function descriptions

from $P_\mathcal{F}$ and polynomial-size input sequence $(x_0^\lambda, x_1^\lambda)_{\lambda\in\mathbb{N}}$ for f_λ, the outputs of the following experiments Real and Ideal are computationally indistinguishable:

– Real_λ: Output $\mathsf{View}_{\Pi,\sigma}(\lambda, \hat{f}_\lambda, (x_0^\lambda, x_1^\lambda))$

– Ideal_λ: Output $\mathsf{Sim}_\sigma(1^\lambda, \hat{f}_\lambda, x_\sigma^\lambda, f_\lambda(x_0^\lambda, x_1^\lambda))$

We say that Π realizes $\mathcal{F}$ with *statistical* (resp., *perfect*) security if the above security requirement holds with statistical (resp., perfect) indistinguishability instead of computational indistinguishability.

2.3 Function Secret Sharing

We follow the definition of function secret sharing (FSS) from [9]. Intuitively, a (2-party) FSS scheme is an efficient algorithm that splits a function $f \in \mathcal{F}$ into two *additive* shares f_0, f_1, such that: (1) each f_σ hides f; (2) for every input x, $f_0(x)+f_1(x) = f(x)$. The main challenge is to make the descriptions of f_0 and f_1 compact, while still allowing their efficient evaluation. As in [8, 9], we insist on an additive representation of the output rather than settle for an arbitrary compact output representation. The additive representation is critical for the applications we consider in this work and is achieved by existing constructions.

We now formally define the notion of FSS. While in this work we consider the 2-party case for simplicity, the definitions and the applications can be extended in a natural way to the k-party case.

Definition 2.3 (FSS: Syntax)**.** A (2-party) *function secret sharing (FSS) scheme* is a pair of algorithms $(\mathsf{Gen}, \mathsf{Eval})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda, \hat{f})$ is a PPT *key generation* algorithm, which on input 1^λ (security parameter) and $\hat{f} \in \{0,1\}^*$ (description of a function f) outputs a pair of keys (k_0, k_1). We assume that $\hat{f}$ explicitly contains descriptions of input and output groups $\mathbb{G}^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}}$.

- $\mathsf{Eval}(\sigma, k_\sigma, x)$ is a polynomial-time *evaluation algorithm*, which on input $\sigma \in \{0,1\}$ (party index), k_σ (key defining $f_\sigma : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$) and $x \in \mathbb{G}^{\mathsf{in}}$ (input for f_σ) outputs a group element $y_\sigma \in \mathbb{G}^{\mathsf{out}}$ (the value of $f_\sigma(x)$, the σ-th share of $f(x)$).

Definition 2.4 (FSS: Correctness and Security)**.** Let $\mathcal{F} = (P_\mathcal{F}, E_\mathcal{F})$ be a function family (as defined in Definition 2.1) and Leak be a polynomial-time computable function specifying the allowable leakage about $\hat{f}$. When Leak is omitted, it is understood to output only $\mathbb{G}^{\mathsf{in}}$ and $\mathbb{G}^{\mathsf{out}}$. We say that $(\mathsf{Gen}, \mathsf{Eval})$ as in Definition 2.3 is an *FSS scheme for the function family* $\mathcal{F}$ (with respect to leakage Leak) if it satisfies the following requirements.

- **Correctness:** For all $\hat{f} \in P_\mathcal{F}$ describing $f : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$, and every $x \in \mathbb{G}^{\mathsf{in}}$, if $(k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda, \hat{f})$ then $\Pr[\mathsf{Eval}(0, k_0, x) + \mathsf{Eval}(1, k_1, x) = f(x)] = 1$.

- **Security:** For each $\sigma \in \{0,1\}$ there is a PPT algorithm Sim_σ (simulator), such that for every infinite sequence $(\hat{f}_\lambda)_{\lambda\in\mathbb{N}}$ of polynomial-size function descriptions from $P_\mathcal{F}$ and polynomial-size input sequence x_λ for f_λ, the outputs of the following experiments Real and Ideal are computationally indistinguishable:

  – Real_λ: $(k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda, \hat{f}_\lambda)$; Output k_σ.

– Ideal_λ: Output $\mathsf{Sim}_\sigma(1^\lambda, \mathsf{Leak}(\hat{f}_\lambda))$.

We refer to the FSS scheme as being statistical (resp., perfect) if the above holds with statistical (resp., perfect) indistinguishability instead of computational indistinguishability.

Definition 2.5 (Distributed Point Function (DPF))**.** A *point function* $f_{\alpha,\beta}$, for $\alpha \in \mathbb{G}^{\mathsf{in}}$ and $\beta \in \mathbb{G}^{\mathsf{out}}$, is defined to be the function $f : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$ such that $f(\alpha) = \beta$ and $f(x) = 0$ for $x \neq \alpha$. A *Distributed Point Function* (DPF) is an FSS scheme for the family of all point functions, with the default leakage (i.e., $\mathsf{Leak}(\hat{f}) = (\mathbb{G}^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}})$).

Definition 2.6 (Distributed Interval Function (DIF))**.** An *interval function* $f_{(a,b),\beta}$, for $a, b \in \mathbb{G}^{\mathsf{in}}$, and $\beta \in \mathbb{G}^{\mathsf{out}}$, and given an arbitrary total order $\leq$ on $\mathbb{G}^{\mathsf{in}}$, is defined to be the function $f : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$ such that $f(x) = \beta$ for $x \in \mathbb{G}^{\mathsf{in}}, a \leq x \leq b$, while $f(x) = 0$ for $x < a$ or $x > b$. If $a = 0$ (the minimal element of $\mathbb{G}^{\mathsf{in}}$) or $b = |\mathbb{G}^{\mathsf{in}}| - 1$ (the maximal element) then we say that $f_{(a,b),\beta}$ is a *special* interval function. A *Distributed Interval Function* (DIF) is an FSS scheme for the family of all interval functions, with the default leakage (i.e., $\mathsf{Leak}(\hat{f}) = (\mathbb{G}^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}})$) and a similar definition holds for Distributed Special Interval Functions.

The following theorem captures the complexity of the best known constructions of DPF and DIF from a PRG. The associated DPF scheme is described explicitly in [9], while the DIF scheme is an instance of the construction for distributed decision trees in [9]. A special interval function, e.g. $f_{(0,b),\beta}$, can be regarded as a decision tree with $\lceil \log_2 |\mathbb{G}^{\mathsf{in}}| \rceil - 1$ internal nodes representing the path to b and $\lceil \log_2 |\mathbb{G}^{\mathsf{in}}| \rceil + 1$ leaves, representing b and any departure from the path. The b leaf and any leaf to the left of the path is assigned the value β while any leaf to the right of the path is assigned 0. A general interval function can be similarly represented by a decision tree of less than twice the size of the decision tree for a special interval.

Theorem 2.7 (Concrete complexity of DPF and DIF schemes, Theorems 3.3 and 3.17 of [9])**.** *Given a PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$, there exists a DPF for $f_{\alpha,\beta} : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$ with key size $m \cdot (\lambda + 2) + \lambda + \ell$ bits, where $m = \lceil \log_2 |\mathbb{G}^{\mathsf{in}}| \rceil$ and $\ell = \lceil \log_2 |\mathbb{G}^{\mathsf{out}}| \rceil$. For $\ell' = \lceil \frac{\ell}{\lambda+2} \rceil$, the key generation algorithm* Gen *invokes G at most $2(m + \ell')$ times and the evaluation algorithm* Eval *invokes G at most $m + \ell'$ times. For special DIF, the key size is $4m \cdot (\lambda + 1) + m\ell + \lambda$, the key generation algorithm invokes G at most $m \cdot (4 + \ell')$ times and the evaluation algorithm invokes G at most $m \cdot (2 + \ell')$ times. For general DIF, the above costs for special DIF are multiplied by two.*

3 Secure Computation with Preprocessing from FSS

In this section, we develop our primary general transformation for using FSS to obtain secure 2PC with preprocessing. We then demonstrate how this approach captures and generalizes existing techniques within this regime.

3.1 Circuit and Offset-Family Notation

We begin by introducing some notation for modeling circuits of computation gates.

Definition 3.1 (Computation Gate)**.** A *computation gate* is a function family $\mathcal{G}$ (Definition 2.1), where each function describes a pair of Abelian groups $(\mathbb{G}^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}})$, and a mapping $g : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$. In some cases it will be convenient to interpret $\mathbb{G}^{\mathsf{in}}$ and $\mathbb{G}^{\mathsf{out}}$ explicitly as product groups, of the form $\mathbb{G}^{\mathsf{in}} = \prod_{i \in [\ell]} \mathbb{G}_i^{\mathsf{in}}$ and $\mathbb{G}^{\mathsf{out}} = \prod_{i \in [m]} \mathbb{G}_i^{\mathsf{out}}$.

For example, one may consider a *zero-test gate*, corresponding to the family of zero-test functions parameterized by different input and output groups.

For syntactic purposes, it will be useful to define notation for the following type of (trivial) *input and output* gates.

Definition 3.2 (Input & Output Gates)**.** An *input* gate is a gate $\mathcal{G}_{\mathsf{Inp}}$ which syntactically receives no input from other gates ($\mathbb{G}^{\mathsf{in}} = \emptyset$), and outputs a single value. An *output* gate is a gate $\mathcal{G}_{\mathsf{Out}}$ which syntactically sends no output to further gates ($\mathbb{G}^{\mathsf{out}} = \emptyset$), and receives as input a single value.

We now define a *circuit* of input, output, and computation gates, via two parts: (1) the circuit syntax, dictating its topological connectivity amongst gates, and (2) the circuit instantiation, selecting a specific function for each gate, such that the choices of input/output groups are consistent across edges. For example, given multiplication gates followed by a zero-test gate (each corresponding to a family of functions), these gates could be instantiated over any arithmetic ring R followed by zero-test from $\mathbb{G}^{\mathsf{in}} = R$ to any other space $\mathbb{G}^{\mathsf{out}}$ with canonical 0 and 1 values.

The syntax of the circuit will be modeled by the structure of a directed acyclic graph, with nodes serving as gates and edges serving as wires. In order to model fan-out, each gate will be associated with both an *out-degree* (dictated by the graph) and an *out-arity* ℓ^{out}, which may not be the same. The out-arity corresponds to the number of values output by the gate computation. Each outgoing edge from the gate corresponds to a wire carrying the value of one of these outputs to another gate, and is labeled with the corresponding index $j \in [\ell^{\mathsf{out}}]$.

Definition 3.3 (Circuit syntax)**.** Let $\mathcal{B}$ be a finite set ("basis") of gates. A circuit C over basis $\mathcal{B}$ specifies a directed acyclic graph (V, E), where each node $v \in V$ is labeled with an input and output arity $(\ell^{\mathsf{in}}_v, \ell^{\mathsf{out}}_v)$, and a gate type $\mathcal{G}_v \in \mathcal{B}$, such that:

每个源节点由一个输入门标记，每个最终接收节点由一个输出门标记

- Each source node is labeled by an *input* gate and every sink an *output* gate (as per Definition 3.2). We sometimes denote the set of input and output gates of C by Inp and Out.

- The in-arity ℓ^{in}_v of each node $v \in V$ is equal to its in-degree; each incoming edge into v is associated with a distinct index $i \in [\ell^{\mathsf{in}}_v]$. Each outgoing edge from v is labeled with an index $j \in [\ell^{\mathsf{out}}_v]$, possibly with repetition (representing fan-out).

- The *depth* of C, denoted $\mathsf{depth}(C)$, is defined as the length of the longest directed path in C.

Definition 3.4 (Circuit instantiation)**.** Let C be a circuit over basis $\mathcal{B}$ with graph (V, E). An *instantiation* $C_{\vec{g}}$ of C is a selection for each $v \in V$ of a function $g_v : \mathbb{G}^{\mathsf{in}}_v \to \mathbb{G}^{\mathsf{out}}_v$ from the gate function family $\mathcal{G}_v$, subject to the following constraints:

1. $\mathbb{G}^{\mathsf{in}}_v = \prod_{i \in [\ell^{\mathsf{in}}_v]} \mathbb{G}^{\mathsf{in}}_{(v,i)}$ and $\mathbb{G}^{\mathsf{out}}_v = \prod_{j \in [\ell^{\mathsf{out}}_v]} \mathbb{G}^{\mathsf{out}}_{(v,j)}$ for some abelian groups $\mathbb{G}^{\mathsf{in}}_{(v,i)}, \mathbb{G}^{\mathsf{out}}_{(v,j)}$, where $\ell^{\mathsf{in}}_v, \ell^{\mathsf{out}}_v$ are the arity of v.

2. For every edge $(u, v) \in E$ labeled by $i \in [\ell^{\mathsf{out}}_u]$ and $j \in [\ell^{\mathsf{in}}_v]$, it holds that $\mathbb{G}^{\mathsf{out}}_{(u,i)} = \mathbb{G}^{\mathsf{in}}_{(v,j)}$.

   We will sometimes refer to edges $(u, v) \in E$ as *wires* $w \in C$, denoting $\mathbb{G}_w := \mathbb{G}^{\mathsf{out}}_{(u,i)} = \mathbb{G}^{\mathsf{in}}_{(v,j)}$.

Remark 3.5 (Instantiation-dependent topology)**.** In some cases, the circuit topology cannot be completely decoupled from the instantiation. For example, instantiating a bit-decomposition gate with $\mathbb{G}^{\mathsf{in}} = \mathbb{Z}_{2^k}$ would yield output $\mathbb{G}^{\mathsf{out}} = \mathbb{Z}_2^k$ of arity k. However, we will attempt to keep syntax and instantiation separate whenever possible for sake of modularity.

Our approach for preprocessing a gate computation relies on FSS sharing of a corresponding family of functions, formed by allowing different additive offset values to both the input and output value. In our application to 2PC, these will serve as the values of random wire masks.

Definition 3.6 (Offset function family)**.** Let $\mathcal{G}$ be a computation gate. The *family of offset functions* $\hat{\mathcal{G}}$ of $\mathcal{G}$ is given by

$$\hat{\mathcal{G}} := \left\{ g^{[r^{\mathsf{in}}, r^{\mathsf{out}}]} : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}} \;\middle|\; \begin{array}{c} g : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}} \in \mathcal{G}, \\ r^{\mathsf{in}} \in \mathbb{G}^{\mathsf{in}}, r^{\mathsf{out}} \in \mathbb{G}^{\mathsf{out}} \end{array} \right\}, \text{ where}$$

$$g^{[r^{\mathsf{in}}, r^{\mathsf{out}}]}(x) := g(x - r^{\mathsf{in}}) + r^{\mathsf{out}},$$

and where each $g^{[r^{\mathsf{in}}, r^{\mathsf{out}}]}$ contains an explicit description of $r^{\mathsf{in}}, r^{\mathsf{out}}$.

3.2 Secure 2-Party Computation with Preprocessing from FSS

We now demonstrate how to apply the ideas from the introduction to obtain a secure 2-party computation protocol in the preprocessing model with cheap online complexity. We restrict our protocol descriptions to the 2-party setting, both for purposes of simplicity, and since this is currently the setting of most efficient FSS constructions. However, the statements generalize to the multiparty case (given corresponding multi-party FSS) with any number of corrupted parties.

The following statement constitutes our core protocol, which leverages the structure of the circuit to provide tailored preprocessing information. Later, in Theorem 3.9, we extend the approach to support circuit-independent preprocessing, at small extra offline and communication cost. Roughly, the extra communication corresponds to an element communicated for every *wire* as opposed to every *gate output* value; note that multiple wires may correspond to the same gate output, in the case of circuit fanout.

Theorem 3.7 (Circuit-Dependent Preprocessing)**.** *Let C be a circuit over basis $\mathcal{B}$. For each $\mathcal{G} \in \mathcal{B}$, let $(\mathsf{Gen}_{\hat{\mathcal{G}}}, \mathsf{Eval}_{\hat{\mathcal{G}}})$ be an FSS for the offset-function family $\hat{\mathcal{G}}$ with key size $\mathsf{size}_{\hat{\mathcal{G}}}(\lambda, |\mathbb{G}^{\mathsf{in}}|, |\mathbb{G}^{\mathsf{out}}|)$. Then for any instantiation $C_{\vec{g}}$ of C, there exists a 2-party protocol for securely computing $C_{\vec{g}}$ with the following properties:*

- Preprocessing. *Given circuit C with gate ("vertex") indices $v \in C$, denote the set of gates by $\mathcal{G}_v$ and their instantiations by g_v, which in particular specify input/output groups $\mathbb{G}^{\mathsf{in}}_v, \mathbb{G}^{\mathsf{out}}_v$. The preprocessing phase executes $\mathsf{Gen}_{\hat{\mathcal{G}}_v}$ for each g_v and produces output of size*

$$\sum_{v \in C} \mathsf{size}_{\hat{\mathcal{G}}}(\lambda, |\mathbb{G}^{\mathsf{in}}|, |\mathbb{G}^{\mathsf{out}}|).$$

- Online. *The online protocol requires local execution of $\mathsf{Eval}_{\hat{\mathcal{G}}}$ for each gate, yielding the following properties:*
  - *Rounds:* $\mathsf{depth}_{\mathcal{B}}(C)$*.*
  - *Communication:* $\sum_{v \in C} \log |\mathbb{G}^{\mathsf{out}}_v|$ *bits per party.*

If the FSS schemes are perfectly (resp., statistically) secure, then the resulting protocol is perfectly (resp., statistically) secure in the preprocessing model.

The proof of Theorem 3.7 follows the high-level description from the Introduction. We give the details below.

Proof. Let $C_{\vec{g}}$ be an instantiation of the circuit C. This corresponds to a consistent choice of function for every gate $v \in C$ (labeled by gate function family $\mathcal{G}_v \in \mathcal{B}$), $g_v : \mathbb{G}_v^{\mathsf{in}} \to \mathbb{G}_v^{\mathsf{out}}$, where $g_v \in \mathcal{G}_v$. For every gate $v \in C$, and every outgoing position $i \in [\ell_v^{\mathsf{out}}]$ of gate v, let $\mathbb{G}_{v,i}^{\mathsf{out}}$ denote the corresponding assigned group as per the instantiation $C_{\vec{g}}$. (In particular, $\mathbb{G}_v^{\mathsf{out}} = \prod_{i \in [\ell_v^{\mathsf{out}}]} \mathbb{G}_{v,i}^{\mathsf{out}}$.)

Preprocessing Phase

1. For every (non-output) gate $v \in C$ and $i \in [\ell_v^{\mathsf{out}}]$ as above, sample a random offset mask $r_{v,i} \leftarrow \mathbb{G}_{v,i}^{\mathsf{out}}$. (Recall output gates syntactically have empty $\mathbb{G}_v^{\mathsf{out}} = \emptyset$.) For each $v \in C$, this induces a corresponding pair of elements:

   $$r_v^{\mathsf{in}} := (r_{v'_1,1}, \ldots, r_{v'_{\ell_v^{\mathsf{in}}},\ell_v^{\mathsf{in}}}) \in \mathbb{G}_v^{\mathsf{in}} \quad \text{and} \quad r_v^{\mathsf{out}} := (r_{v,1}, \ldots, r_{v,\ell^{\mathsf{in}}{}_v}) \in \mathbb{G}_v^{\mathsf{out}}.$$

   (Here the components of r_v^{out} are directly determined by the $r_{v,i}$, whereas components of r_v^{in} are inherited from the gates v' for whom these wires were output.)
2. For every *input* gate $v \in C$ owned by party P_0, set $(k_v^0, k_v^1) = (r_v^{\mathsf{out}}, \emptyset)$; if $v \in C$ is an input gate is owned by P_1, set $(k_v^0, k_v^1) = (\emptyset, r_v^{\mathsf{out}})$.
3. For every *computation* gate $v \in C$, labeled by $g_v \in \mathcal{G}_v$, consider the function $g_v^{[r_v^{\mathsf{in}}, r_v^{\mathsf{out}}]} : \mathbb{G}_v^{\mathsf{in}} \to \mathbb{G}_v^{\mathsf{out}}$ contained in $\hat{\mathcal{G}}_v$, where $r_v^{\mathsf{in}}, r_v^{\mathsf{out}}$ are as above. Sample a pair of FSS keys:

   $$(k_v^0, k_v^1) \leftarrow \mathsf{Gen}_{\hat{\mathcal{G}}}(1^\lambda, g_v^{[r_v^{\mathsf{in}}, r_v^{\mathsf{out}}]}).$$
4. Output the collection of keys for all gates: $(k_v^0)_{v \in C}$ to party P_0 and $(k_v^1)_{v \in C}$ to party P_1.

Online Phase

Execution proceeds topologically through circuit C. For each level $\ell = 1$ to $\mathsf{depth}_{\mathcal{B}}(C)$, each party P_σ ($\sigma \in \{0, 1\}$) performs the following. Denote by $v \in C_\ell$ a gate in level ℓ of C.

1. *Parse messages from previous round.* If $\ell > 1$: For each $v' \in C_{\ell-1}$ and $i' \in [\ell_{v'}^{\mathsf{out}}]$, let $\mathsf{msg}_{v',i'}^{\mathsf{out}} = \mathsf{msg}_{v',i'}^0 + \mathsf{msg}_{v',i'}^1 \in \mathbb{G}_{v',i'}^{\mathsf{out}}$, where $\mathsf{msg}_{v',i'}^\sigma$, $\sigma \in \{0, 1\}$, were exchanged in the previous round.

   The combined set of values msg_w from all previous levels $\ell' < \ell$, induces a corresponding element for each gate $v \in C_\ell$,

   $$\mathsf{msg}_v^{\mathsf{in}} := (\mathsf{msg}_{v'_1,i'_1}, \ldots, \mathsf{msg}_{v'_{\ell_v^{\mathsf{in}}},i'_{\ell_v^{\mathsf{in}}}}) \in \mathbb{G}_v^{\mathsf{in}},$$

   where each component of the input into v is dictated by the output v', i' from the previous level which feeds into this input.
2. *Compute & send messages for this round.* For each gate $v \in C_\ell$:
   - If $v \in C_\ell$ is an *input* gate owned by party $P_b \in \{0, 1\}$, then parse $k_v^\sigma = r_v$ and set $\mathsf{msg}_v^\sigma = x_v + r_v$; if v is an input gate owned by $P_{1-\sigma}$, then set $\mathsf{msg}_v^\sigma = \emptyset$.

- If $v \in C_\ell$ is a *computation* gate of type $\mathcal{G}_v$, then parse k_v^σ as an FSS key, and compute
$$\mathsf{msg}_v^\sigma = \mathsf{Eval}_{\mathcal{G}_v}(\sigma, k_v^\sigma, \mathsf{msg}_v^{\mathsf{in}}).$$
Note $\mathsf{msg}_v^{\mathsf{in}} \in \mathbb{G}_v^{\mathsf{in}}$ and $\mathsf{msg}_v^\sigma \in \mathbb{G}_v^{\mathsf{out}}$.

Send the vector $(\mathsf{msg}_v^\sigma)_{v \in C_\ell} \in \prod_{v \in C_\ell} \mathbb{G}_v^{\mathsf{out}}$ to party $P_{1-\sigma}$.

Final output. For each *output* gate $v \in C$, output $\mathsf{msg}_v^{\mathsf{out}} = \mathsf{msg}_v^0 + \mathsf{msg}_v^1 \in \mathbb{G}_v^{\mathsf{out}}$.

We now proceed to analyze the complexity and security of the above protocol Π. The complexity of the protocol is:

- Preprocessing: Computation is dominated by generation of FSS keys: one execution of $\mathsf{Gen}_{\hat{\mathcal{G}}}$ on security parameter λ for each gate $\mathcal{G}$. The preprocessing output size is $\sum_{v \in C} \mathsf{size}_{\hat{\mathcal{G}}_v}(\lambda, |\mathbb{G}_v^{\mathsf{in}}|, |\mathbb{G}_v^{\mathsf{out}}|)$ where $\mathsf{size}_{\hat{\mathcal{G}}_v}(\lambda, |\mathbb{G}_v^{\mathsf{in}}|, |\mathbb{G}_v^{\mathsf{out}}|)$ denotes the FSS share size for $g_v \in \mathcal{G}_v$ (where we define notationally $\mathsf{size}_{\hat{\mathcal{G}}_v}(\lambda, |\mathbb{G}_v^{\mathsf{in}}|, |\mathbb{G}_v^{\mathsf{out}}|) := |\mathbb{G}_v^{\mathsf{in}}|$ for input gates v).
- Online Round complexity: Messages are exchanged for every level of the circuit $\ell \in [\mathsf{depth}_{\mathcal{B}}(C)]$.
- Online Communication: The total per-party communication is equal to $\sum_{v \in C} \log |\mathbb{G}_v^{\mathsf{out}}|$.
- Online Computation: Dominated by evaluation of FSS instances: $\mathsf{Eval}_{\hat{\mathcal{G}}_v}$ for each gate $g_v \in \mathcal{G}_v$.

To prove security of the protocol, we demonstrate and analyze a simulator $\mathcal{S}$, below.

Simulator $\mathcal{S}$: Given the circuit instantiation $C_{\vec{g}}$, identity of corrupt party $\sigma \in \{0,1\}$, corrupt-party inputs $\{x_v\}_{v \in \mathsf{Inp}_\sigma}$, and output $\{y_v\}_{v \in \mathsf{Out}}$.

- *Simulate preprocessing.*
  1. For each input gate $v \in \mathsf{Inp}_\sigma$ owned by P_σ, sample random $k_v^\sigma \leftarrow \mathbb{G}_v^{\mathsf{out}}$.
  2. For each input gate $v \in \mathsf{Inp}_{1-\sigma}$ owned by honest party $P_{1-\sigma}$, set $k_v^\sigma = \emptyset$.
  3. For each $v \in C$ labeled by computation gate $g_v \in \mathcal{G}_v$ with input/output groups $\mathbb{G}_v^{\mathsf{in}}, \mathbb{G}_v^{\mathsf{out}}$, *simulate* the corresponding FSS key. Namely, denote by $\mathsf{Sim}_{\mathcal{G}}$ the FSS simulator for $(\mathsf{Gen}_{\hat{\mathcal{G}}}, \mathsf{Eval}_{\hat{\mathcal{G}}})$, and let $\mathsf{Leak}_v = (\mathbb{G}_v^{\mathsf{in}}, \mathbb{G}_v^{\mathsf{out}})$. Simulate $k_v^\sigma \leftarrow \mathsf{Sim}_{\mathcal{G}}(1^\lambda, \mathsf{Leak}_v)$.

  Output $(k_v^\sigma)_{v \in C}$ as the preprocessing output for the corrupted party P_σ.

- *Simulate online protocol.* For each level $\ell = 1$ to $\mathsf{depth}_{\mathcal{B}}(C)$ of the circuit C:
  1. *Parse messages from previous round.* If $\ell > 1$, then parse messages sent/received in the previous round as specified in the honest protocol (see protocol above), yielding $\mathsf{msg}_v^{\mathsf{in}} \in \mathbb{G}_v^{\mathsf{in}}$ for each $v \in C_\ell$ in the present level.
  2. *Compute & send messages for this round.* For each gate $v \in C_\ell$, select the message $\mathsf{msg}_v^{1-\sigma}$ on behalf of the honest party:
     - If $v \notin \mathsf{Out}$ is *not* an output gate: Sample a random element $\mathsf{msg}_v^{1-\sigma} \leftarrow \mathbb{G}_v^{\mathsf{out}}$.

- If $v \in \mathsf{Out}$ *is* an output gate: Choose $\mathsf{msg}_v^{1-\sigma}$ so as to yield the correct output y_v, as follows. Emulate the actions of the corrupt party in computing $\mathsf{msg}_v^\sigma = \mathsf{Eval}_{\hat{\mathcal{G}}_v}(\sigma, k_v^\sigma, \mathsf{msg}_v^{\mathsf{in}})$, where k_v^σ was the simulated FSS key for $v \in C$ from the preprocessing phase. Set $\mathsf{msg}_v^{1-\sigma} = y_v - \mathsf{msg}_v^\sigma$.

Simulate sending $(\mathsf{msg}_v^{1-\sigma})_{v \in C_\ell}$ on behalf of the honest party.

To prove indistinguishability of the simulated experiment, we consider a sequence of intermediate hybrid experiments.

- **Hybrid 0:** Real-world execution of Π on honest inputs.

- **Hybrid 1:** (Correctness of FSS.)

  Real-world execution, except that the honest party's messages $\mathsf{msg}_v^{1-\sigma}$ (previously computed as $\mathsf{msg}_v^{1-\sigma} = \mathsf{Eval}_{\hat{\mathcal{G}}_v}(k_v^{1-\sigma}, \mathsf{msg}_v^{\mathsf{in}})$) are now computed to be the induced share value to yield *correct* combined output $g_v^{[r_v^{\mathsf{in}}, r_v^{\mathsf{out}}]}(\mathsf{msg}_v^{\mathsf{in}})$ given the corrupt party's contribution $\mathsf{msg}_v^\sigma = \mathsf{Eval}_{\hat{\mathcal{G}}_v}(k_v^\sigma, \mathsf{msg}_v^{\mathsf{in}})$: i.e.,

  $$\mathsf{msg}_v^{1-\sigma} = g_v^{[r_v^{\mathsf{in}}, r_v^{\mathsf{out}}]}(\mathsf{msg}_v^{\mathsf{in}}) - \mathsf{Eval}_{\hat{\mathcal{G}}_v}(k_v^\sigma, \mathsf{msg}_v^{\mathsf{in}}).$$

  In particular, this means for every gate $v \in C$, the corresponding reconstructed value msg_v is equal to the correct intermediate computation value x'_v masked by r_v.

  By correctness of the FSS scheme (and a standard hybrid argument across the polynomially many gates $v \in C$), with all but negligible probability over the choice of the keys in the preprocessing phase, Hybrids 0 and 1 induce an identical experiment output distribution.

- **Hybrid 2:** (Security of FSS.)

  Similar to Hybrid 1, except that the corrupt party's FSS keys k_v^σ in the preprocessing phase are now *simulated* as in $\mathsf{Sim}_{\mathcal{G}}$. Honest party messages are still computed in the same way, using the simulated keys.

  Indistinguishability of the two experiments follows directly by the security of the FSS (together with a hybrid argument across gates $v \in C$), as all honest-party messages are presently computed independently of the honest FSS keys $k_v^{1-\sigma}$.

- **Hybrid 3:** (Randomness of r_v.)

  Simulated experiment. The difference between Hybrids 2 and 3 is the following.

  - In Hybrid 2, the honest party sends values $\mathsf{msg}_v^{1-\sigma}$ so that combining with the corrupt party's messages msg_v^σ yields values msg_v per gate v equal to the masked true intermediate computation values, $x'_v + r_v$.
  - In Hybrid 3, the honest party computes and sends a value $\mathsf{msg}_v^{1-\sigma}$ in this fashion for all *output* gates, but sends *random* values for all other gates.

  However, the values r_v were selected at random for each gate v. Since masked output values are communicated exactly once per each gate; in particular each mask is used only in one message, meaning the corresponding values $x'_v + r_v$ are uniform conditioned on the adversarial view thus far. Further, each corrupt message msg_v^σ is computed as $\mathsf{Eval}_{\hat{\mathcal{G}}_v}(k_v^\sigma, \mathsf{msg}_v^{\mathsf{in}})$, where $\mathsf{msg}_v^{\mathsf{in}}$ depends only on *previous-round* message values (independent of r_v), and the *simulated* key k_v^σ (which is also independent of the true r_v). Thus, the next-round messages of honest parties (i.e., reverse-computed shares, formed by subtracting the $(x'_v + r_v)$ values from the appropriate pieces of msg_v^σ), are also uniform.

That is, the two experiments yield an identical distribution.

This concludes the proof of Theorem 3.7. □

Remark 3.8 (Compressing preprocessing output)**.** In some cases, the size of the offline preprocessing information can be compressed, when e.g. FSS keys of neighboring gates contain redundant information. This will be the case, for example, when generating FSS keys for neighboring gates which are each instantiated by degree-2 functions. (Here, the output mask r_w of the first gate will be identical to the input mask of the second, as they correspond to the same wire; thus including secret shares of r_w as part of both FSS keys is unnecessary.) See discussion in the following section for further cases.

Circuit-Independent Preprocessing. The protocol construction in Theorem 3.7 used preprocessing information that was *tailored* to the topology of the given circuit C. More concretely, we were able to "match up" the input/output offset masks r_v of every pair of gates sharing a wire, hardcoding the same offset into the FSS keys for the respective functions. In particular, this enabled "for free" a direct translation from masked output of one gate to appropriate masked input to *all* gates in the next level which accepted this value as input (via fan-out).

In some cases, it may be advantageous to produce generic preprocessing information that depends on the individual gate structure, but which can be used for any circuit built from such gates (independent of the topology linking the gates together). Our approach generalizes to this circuit-independent setting with a small amount of additional overhead, via a few small changes, which we now describe.

The only difference between the two constructions is that the circuit-dependent correlation could directly "match up" the outgoing mask r^{out} for a gate to be equal to the incoming mask r^{in} of any gate to which it enters. In contrast, when the structure of the circuit C is not a priori known, this can be effectively emulated as follows.

- For each gate g, we will sample a random *input* offset mask r^{in} (but not r^{out}), and provide FSS shares for the offset function $g^{[r^{\mathsf{in}},0]} = g(x - r^{\mathsf{in}}) + 0$, together with *additive secret shares* of the mask r^{in} (which was not needed previously). Note that a mask per input corresponds directly to a mask per *wire* in the circuit.
- Then, once the structure of the circuit C is known (during the protocol), a party P_σ can locally convert his overall collection of preprocessing information over all gates $(k_v^\sigma, (r_v^{\mathsf{in}})^\sigma)_{v \in C}$ into FSS shares for the desired "matched up" $g^{[r^{\mathsf{in}},r^{\mathsf{out}}]}$ (where r^{out} is equal to the input mask r^{in} for the next gate that the gate v output value will enter into), by leveraging the additive secret shares of all wire masks r^{in} together with linearity of FSS reconstruction: i.e., outputting $\mathsf{Eval}(\sigma, k_v^\sigma, x) + (r_v^{\mathsf{out}})^\sigma$.

This effectively reduces us back to the circuit-dependent version, in terms of correctness and security. Observe, however, that whereas in the circuit-dependent version, r^{in} values of all target gates for fan-out wires of the same value could a priori be coordinated, in this setting (when this structure is not a priori known), the parties must send a separate element per fan-out wire. We also must provide the additive shares of the input masks r^{in} as part of the correlated randomness.

Theorem 3.9 (Circuit-Independent Preprocessing)**.** *Let $\mathcal{B}$ be a finite gate basis; for each $\mathcal{G} \in \mathcal{B}$, let $(\mathsf{Gen}_{\hat{\mathcal{G}}}, \mathsf{Eval}_{\hat{\mathcal{G}}})$ be an FSS for the offset-function family $\hat{\mathcal{G}}$ with key size $\mathsf{size}_{\hat{\mathcal{G}}}(\lambda, |\mathbb{G}^{\mathsf{in}}|, |\mathbb{G}^{\mathsf{out}}|)$. Then there exists a 2-party protocol for securely computing any $\mathcal{B}$-circuit instantiation $C_{\vec{\mathcal{G}}}$ consisting of $s_{\mathcal{G}} \in \mathbb{N}$ gates g of type $\mathcal{G}$ for each $\mathcal{G} \in \mathcal{B}$, with the following complexity:*

- Preprocessing. *(Independent of $C_{\vec{\mathcal{G}}}$) The preprocessing phase executes $s_{\mathcal{G}}$ executions of $\mathsf{Gen}_{\hat{\mathcal{G}}}$ for each gate $\mathcal{G} \in \mathcal{B}$ and produces output size*

$$\sum_{\mathcal{G} \in \mathcal{B}} s_{\mathcal{G}} \cdot \Big(\mathsf{size}_{\hat{\mathcal{G}}}(\lambda, |\mathbb{G}^{\mathsf{in}}|, |\mathbb{G}^{\mathsf{out}}|) + \log |\mathbb{G}^{\mathsf{in}}| \Big) .$$

- Online. *The online execution takes $\mathsf{depth}(C)$ rounds (as before), but requires communication $\sum_{v \in C} \log |\mathbb{G}_v^{\mathsf{in}}|$ bits per party (vs. $\sum_{v \in C} \log |\mathbb{G}_v^{\mathsf{out}}|$). Equivalently, one element is communicated per* wire*, as opposed to only one element per* value *(where fan-out introduces extra wires but not values).*

Proof. (sketch) Ultimately, the discussion above leads to the following changes in the protocol construction from Theorem 3.7:

- Preprocessing. For each gate v, we *independently* sample an input mask $r_v^{\mathsf{in}} \leftarrow \mathbb{G}_v^{\mathsf{in}}$. In order to translate to the target input mask $r_{v'}^{\mathsf{in}}$ for some future gate, in the preprocessing we also give *additive shares* $[r_v^{\mathsf{in}}]$ of r_v^{in}. This induces the extra term of $\log |\mathbb{G}_{\mathcal{G}}^{\mathsf{in}}|$ bits for each gate within the preprocessing output information. Note this is independent of the topology of the circuit.

- Online. For each computation gate, compute FSS output shares evaluated as $\mathsf{share}_v^\sigma := \mathsf{Eval}_{\hat{\mathcal{G}}_v}(k_v^\sigma, \mathsf{msg}_v^{\mathsf{in}})$. Recall these yield (additive) shares of the *unmasked* intermediate computation value (y_v). However, instead of directly exchanging these shares (which will never be done!), for each fan-out copy that becomes input of a new gate v', the parties will now exchange v-to-v' “additively translated” share values: that is, exchanging the value $\mathsf{share}_v + [r_{v'}^{\mathsf{in}}]$.

  Correctness: By the additive reconstruction of the FSS, adding in this translation offset directly yields the $r_{v'}^{\mathsf{in}}$-masked value $(y_v + r_{v'}^{\mathsf{in}})$, exactly as needed to input into the FSS'ed gate-offset function for the gate v'.

  Security: The additive shares of the masks r_v^{in} are perfectly hiding. During the online execution, observe that each reconstructed pair of messages is masked by a *fresh* mask r_v^{in}, since independent of fan-out, each input mask will be used only once.

  Complexity: This requires additional elements to be added to the correlated randomness. For gates with fan-out 1, there is no affect to online communication, as this translation can be absorbed into the exchange of shares directly. However, the parties must now send separate “translations” for each copy of a fan-out wire in this setting (as the gates they are entering will be different), whereas this was not needed in the C-dependent solution.

This concludes the proof of Theorem 3.9. □

3.3 Recasting and Generalizing Existing Protocols

We begin by briefly demonstrating that common existing approaches to 2PC with preprocessing (and even useful extensions) can be cast as instances of the FSS-based framework, for special simple cases of FSS.

3.3.1 Low-Degree Gates

The first category is FSS of low-degree polynomials, which can be attained simply by providing additive secret shares of each coefficient. More broadly:

Observation 3.10 (FSS via Coefficient-Sharing)**.** For any module M over coefficient ring R, and family of functions of the form $\mathcal{F} = \{\sum_{i=1}^{m} \alpha_i F_i(x) \mid \alpha_i \in R\}$ for *public* functions $(F_i)_{i\in[m]} : \mathbb{G}^{\mathsf{in}} \to M$, there exists an FSS scheme for $\mathcal{F}$ with *perfect* security and correctness, as follows:

- $\mathsf{Gen}(1^\lambda, f)$: Parse the description of $f \in \mathcal{F}$ as secret coefficients $(\alpha_i)_{i\in[m]} \in R^m$. The output FSS keys are additive secret shares of each α_i over R, yielding key size $m \log |R|$.
- $\mathsf{Eval}(\sigma, k_\sigma, x)$: Parse $k_\sigma = (\alpha_i^\sigma)_{i\in[m]}$. Output $\sum_{i\in[m]} \alpha_i^\sigma F_i(x)$.

Note that FSS keys perfectly hide the coefficients α_i, and thus f. Correctness holds by the distributive law within the module M.

As an example of "public functions" F_i, one can consider, e.g., input monomials of a certain degree. Indeed, we can use this approach to instantiate FSS schemes for offset-function classes $\hat{\mathcal{G}}$ for the following types of low-degree gates.

Definition 3.11 (Low-Degree Gates)**.**

1. The *degree-d gate* $\mathcal{G}_{\mathsf{deg\text{-}d}}$ is the class of functions $g_{\mathsf{deg\text{-}d}} : R^n \to R^m$ parameterized by a ring R and $n, m \in \mathbb{N}$, such that for each $i \in [m]$, the ith output function $(g_{\mathsf{deg\text{-}d}})_i(x_1, \ldots, x_n)$ is a polynomial over R of degree no greater than d.
2. The *bilinear map gate* $\mathcal{G}_{\mathsf{blin}}$ is the class of functions $g_{\mathsf{blin}} : \mathbb{G}_1^{\mathsf{in}} \times \mathbb{G}_2^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$ such that $\mathbb{G}^{\mathsf{in}} = \mathbb{G}_1^{\mathsf{in}} \times \mathbb{G}_2^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}}$ are Abelian groups, and g_{blin} is a bilinear map.

Note that these two classes are incomparable: $\mathcal{G}_{\mathsf{deg\text{-}d}}$ addresses higher-order polynomials, beyond degree 2. On the other hand, $\mathcal{G}_{\mathsf{blin}}$ captures bilinear operations across different structures beyond a single ring R: e.g., multiplication of non-square matrices, $\mathbb{G}_1^{\mathsf{in}} = R^{m_1 \times m_2}$, $\mathbb{G}_2^{\mathsf{in}} = R^{m_2 \times m_3}$, and $\mathbb{G}^{\mathsf{out}} = R^{m_1 \times m_3}$.

Proposition 3.12 (Information-Theoretic FSS for Low-Degree Gates)**.** *Let $d \in N$. Then there exists* perfectly *secure FSS for the following offset-function families, with the given complexities:*

- $\hat{\mathcal{G}}_{\mathsf{deg\text{-}d}}$*: For $\mathbb{G}^{\mathsf{in}} = R^n$, $\mathbb{G}^{\mathsf{out}} = R^m$, key size is $m\binom{n+d}{d}(\log |R|)$ bits.*
- $\hat{\mathcal{G}}_{\mathsf{blin}}$*: For $\mathbb{G}^{\mathsf{in}} = \mathbb{G}_1^{\mathsf{in}} \times \mathbb{G}_2^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}}$, key size is $(\log |\mathbb{G}^{\mathsf{in}}| + \log |\mathbb{G}^{\mathsf{out}}|)$ bits.*

Proof. Consider the following FSS constructions.

- For $\hat{\mathcal{G}}_{\mathsf{deg\text{-}d}}$: Recall we are sharing offset functions of the form $g_{\mathsf{deg\text{-}d}}^{[r^{\mathsf{in}},r^{\mathsf{out}}]}$, where $g = (g_1, \ldots, g_m)$ is a degree-d polynomial $g : R^n \to R^m$, and with offsets $r^{\mathsf{in}} = (r_1^{\mathsf{in}}, \ldots, r_n^{\mathsf{in}}) \in R^n$ and $r^{\mathsf{out}} = (r_1^{\mathsf{out}}, \ldots r_m^{\mathsf{out}}) \in R^m$. By definition, for each $i \in [m]$,

$$(g_{\mathsf{deg\text{-}d}}^{[r^{\mathsf{in}},r^{\mathsf{out}}]})_i(x_1, \ldots, x_n) = g_i(x_1 - r_1^{\mathsf{in}}, \ldots, x_m - r_m^{\mathsf{in}}) + r_i^{\mathsf{out}}.$$

  In particular, each $(g_{\mathsf{deg\text{-}d}}^{[r^{\mathsf{in}},r^{\mathsf{out}}]})_i$ itself is a degree-d polynomial in the inputs, where the coefficients of each degree $\leq d$ monomial in the variables x_i depends on the secret values $r^{\mathsf{in}}, r^{\mathsf{out}}$. By Observation 3.10, we can thus obtain secure FSS by giving additive secret shares of each of these coefficients. There are $\binom{n+d}{d}$ distinct monomials of degree $\leq d$ in the n input variables; for each output $i \in [m]$, the FSS key will contain an additive share of size $\log |R|$ for each monomial.

- For $\hat{\mathcal{G}}_{\mathsf{blin}}$: Given an offset function of the form $g_{\mathsf{blin}}^{[r^{\mathsf{in}},r^{\mathsf{out}}]}$, parse as a *bilinear* function $g : \mathbb{G}_1^{\mathsf{in}} \times \mathbb{G}_2^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$, and $r^{\mathsf{in}} = (r_1^{\mathsf{in}}, r_2^{\mathsf{in}}) \in \mathbb{G}_1^{\mathsf{in}} \times \mathbb{G}_2^{\mathsf{in}}$, and $r^{\mathsf{out}} \in \mathbb{G}^{\mathsf{out}}$. By definition,

$$\begin{aligned} g_{\mathsf{blin}}^{[r^{\mathsf{in}},r^{\mathsf{out}}]}(x_1, x_2) &= g(x_1 - r_1^{\mathsf{in}}, x_2 - r_2^{\mathsf{in}}) + r^{\mathsf{out}} \\ &= g(x_1, x_2) - g(r_1^{\mathsf{in}}, x_2) - g(x_1, r_2^{\mathsf{in}}) + g(r_1^{\mathsf{in}}, r_2^{\mathsf{in}}) + r^{\mathsf{out}}. \end{aligned}$$

  Consider the following observations: (1) $g(x_1, x_2)$ is publicly computable. (2) $r_3 := g(r_1^{\mathsf{in}}, r_2^{\mathsf{in}}) + r^{\mathsf{out}}$ is a fixed additive term, independent of the input x. (3) Bilinearity of g implies the functions $g(\cdot, x_2)$ and $g(x_1, \cdot)$ are *linear* in the corresponding second position.

  We can thus achieve FSS for this function class by giving out *additive secret shares* of the values $r_1^{\mathsf{in}}, r_2^{\mathsf{in}}$, and $r_3 := g(r_1^{\mathsf{in}}, r_2^{\mathsf{in}}) + r^{\mathsf{out}}$. The corresponding FSS key size is $\log |\mathbb{G}_1^{\mathsf{in}}| + \log |\mathbb{G}_2^{\mathsf{in}}| + \log |\mathbb{G}^{\mathsf{out}}| = (\log |\mathbb{G}^{\mathsf{in}}| + \log |\mathbb{G}^{\mathsf{out}}|)$ bits.

□

Plugging these FSS constructions into our protocols from the previous section (Theorems 3.7 and 3.9), we obtain secure computation protocols isomorphic to existing protocols from the literature. In addition, the FSS abstraction extends directly to broader classes: e.g., directly supporting general bilinear gates over different rings R_i (such as matrix multiplications), as well as arbitrary low-degree gates over a ring R.

Note that a degree-d mapping can have circuit complexity $\sim n^d$. In the corresponding approach, this increases only the size of the FSS preprocessing information (corresponding to more coefficients) whereas the online communication scales just with the input and output size of the gate. Similarly, bilinear operations such as matrix multiplication when expressed as circuits over the base ring R require significantly more small gates as compared to a single matrix input and output when viewed as a single large bilinear gate.

Corollary 3.13 (2PC with Preprocessing: Low-Degree & Bilinear Gates)**.** *Applying our FSS framework (Theorems 3.7, 3.9) for circuits of degree-d and bilinear gates $\mathcal{G}_{\mathsf{deg\text{-}d}}, \mathcal{G}_{\mathsf{blin}}$ as above yields perfectly secure protocols in the preprocessing model isomorphic to (and generalizing) the following:*

- Beaver Triples [2]: *Applying Theorem 3.9, yielding circuit-independent preprocessing 2PC for low-degree and bilinear gates.*

- Circuit-Dependent Beaver (e.g., [14, 24, 12, 3]): *Applying Theorem 3.7, yielding circuit-dependent preprocessing 2PC for low-degree and bilinear gates.*

Proof. Consider the two approaches.

(Circuit-independent). Applying the protocol framework of Theorem 3.9, we obtain the following structure. We describe for the case of multiplication gates over a ring R to illustrate the Beaver triple structure (but observe that the construction extends directly to more general degree-d and bilinear gates).

- For each multiplication gate v with input wires (w_1, w_2) and output wire w_3, sampling random $r_1, r_2 \leftarrow R$, and generating FSS shares for the gate-offset function will correspond to sharing the function

$$\begin{aligned} g_v^{[(r_1,r_2),0]}(x_1, x_2) &= (x_1 - r_1)(x_2 - r_2) \\ &= x_1x_2 - r_1x_2 - x_2r_1 + r_1r_2. \end{aligned}$$

  Note that x_1x_2 is always with coefficient 1 and publicly computable. Applying Observation 3.10 for the remaining (secret) coefficients yields FSS keys that are additive secret shares of 3 values: r_1, r_2, and r_1r_2.

- The circuit-independent 2PC preprocessing included FSS keys of each such gate offset function, as well as additive shares of the input masks themselves. In this case, additive shares of the input masks are *already* included as part of the FSS keys. Thus, the final resulting correlation corresponds directly to Beaver triples: for each gate, additive shares of random r_1, r_2, and r_1r_2.

(Circuit-dependent). Applying Theorem 3.7 results in an optimization of this approach, as in [14, 24, 12], where the offset masks are correlated across gates. □

3.3.2 Truth-Table Gates

The second category is a straightforward FSS of arbitrary functions of polynomial-size domain, formed by simply providing additive secret shares of each element of the truth table. Perfect secrecy and evaluation with additive reconstruction follow in a trivial manner.

Observation 3.14 (FSS via Shared Truth Table)**.** Let $\mathcal{F}$ be any family of functions where for a given $(\mathbb{G}^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}})$, the truth table of a function $f \in \mathcal{F}$ can be described by $s = s(\mathbb{G}^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}})$ elements of the output space $\mathbb{G}^{\mathsf{out}}$. Then there exists an FSS scheme for $\mathcal{F}$ with *perfect* security and correctness, with key size $s \cdot \log |\mathbb{G}^{\mathsf{out}}|$ bits.

Note that one can *always* express the truth table of a function $g : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$ using $|\mathbb{G}^{\mathsf{in}}|$ many elements of $\mathbb{G}^{\mathsf{out}}$. However, for some interesting function classes, this can be made even smaller. For example, functions with bounded locality: where $\mathbb{G}^{\mathsf{in}} = \prod_{i=1}^{n} \widetilde{\mathbb{G}^{\mathsf{in}}}$, and each output of the function depends only on a bounded number $\ell(\mathbb{G}^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}})$ of fixed coordinates of the input; in such case, the full truth table of the function can be expressed given just $|\widetilde{\mathbb{G}^{\mathsf{in}}}|^{\ell} \cdot \log |\mathbb{G}^{\mathsf{out}}|$ bits, as opposed to $|\mathbb{G}^{\mathsf{in}}| \cdot \log |\mathbb{G}^{\mathsf{out}}| = |\widetilde{\mathbb{G}^{\mathsf{in}}}|^{n} \cdot \log |\mathbb{G}^{\mathsf{out}}|$ bits.

In a straightforward way, this translates to the *offset-function* family $\hat{\mathcal{F}}$ of any such function family $\mathcal{F}$.

Analogous to the case of low-degree functions, plugging these general truth table FSS constructions into our 2PC protocols from Theorems 3.7 and 3.9 yields secure computation protocols that reproduce existing protocols from the literature.

Corollary 3.15 (2PC with Preprocessing: Truth Table Gates)**.** *Applying our FSS framework (Theorems 3.7, 3.9) for circuits of arbitrary gates $\mathcal{G}$ as above rederives perfectly secure protocols in the preprocessing model isomorphic to the following:*

- One-Time Truth Tables [23, 14]: *Applying Theorem 3.9 together with Observation 3.14 for arbitrary truth tables.*
- Leveled circuits, with sublinear online communication [12]: *Applying Theorem 3.7, together with Observation 3.14 for circuits with bounded locality.*

Proof. (One-Time Truth Tables). For a given gate function g, the truth table of the offset-function $g^{[r^{\mathsf{in}},0]}$ (recall in the circuit-independent setting, we take $r^{\mathsf{out}} = 0$) is simply a randomly shifted version of the original truth table, and FSS shares of this function will be precisely additive shares of the shifted truth table.

(Leveled Circuits). The core technical insight in [12] is that a leveled circuit of size s can be partitioned into large "gates" of depth $\log\log s$, whose input locality are each bounded by $\log s$, and thus whose truth tables can each be described in polynomial size. Applying Theorem 3.7 to such circuit decomposition yields a comparable protocol, with polynomial-size preprocessing information, and where parties need only communicate $O(s/\log\log s)$ elements, corresponding to just inputs and outputs of these gates, in $\mathsf{depth}(C)/\log\log s$ rounds. □

4 Applications

In this section we explore applications of our technique to useful types of gates for which we can obtain significant improvements over the current state of the art.

4.1 Zero Test / Equality

We start with gates that either compare a single group element to 0 or check that two group elements are equal.

Definition 4.1 (Equality-Type Gates)**.**

1. The *zero-test gate* $\mathcal{G}_{\mathsf{zt}}$ is the class of functions $g_{\mathsf{zt}} : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$ parameterized by Abelian groups $\mathbb{G}^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}}$ and given by
$$g_{\mathsf{zt}}(x) = \begin{cases} 0 \in \mathbb{G}^{\mathsf{out}} & \text{if } x = 0 \in \mathbb{G}^{\mathsf{in}} \\ 1 \in \mathbb{G}^{\mathsf{out}} & \text{else} \end{cases} .$$

2. The *equality-test gate* $\mathcal{G}_{\mathsf{eq}}$ is the class of functions $g_{\mathsf{eq}} : \mathbb{G}^{\mathsf{in}} \times \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$ parameterized by $\mathbb{G}^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}}$ and given by
$$g_{\mathsf{eq}}(x, x') = \begin{cases} 0 \in \mathbb{G}^{\mathsf{out}} & \text{if } x = x' \in \mathbb{G}^{\mathsf{in}} \\ 1 \in \mathbb{G}^{\mathsf{out}} & \text{else} \end{cases} .$$

Note that the offset function class of the zero-test gate is precisely the class of point functions, where the special input α corresponds to the input offset and the output value β to the output offset. Hence, realizing a zero-test gate (on a masked input) reduces to a single DPF evaluation.

Proposition 4.2 (Zero Test from DPF)**.** *There is an FSS scheme* $(\mathsf{Gen}_{\mathsf{zt}}, \mathsf{Eval}_{\mathsf{zt}})$ *for the offset function family* $\hat{\mathcal{G}}_{\mathsf{zt}}$ *making black-box use of a PRG. The scheme has the same key size and number of PRG invocations as a DPF with input domain* $\mathbb{G}^{\mathsf{in}}$ *and output domain* $\mathbb{G}^{\mathsf{out}}$*.*

Proof. Consider the following construction, where $(\mathsf{Gen}_{\mathsf{DPF}}, \mathsf{Eval}_{\mathsf{DPF}})$ is a distributed point function.

- $\mathsf{Gen}_{\mathsf{zt}}(1^\lambda, g_{\mathsf{zt}}^{[r^{\mathsf{in}}, r^{\mathsf{out}}]})$: Parse $g_{\mathsf{zt}}^{[r^{\mathsf{in}}, r^{\mathsf{out}}]}$ to recover $\mathbb{G}^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}}, r^{\mathsf{in}}, r^{\mathsf{out}}$. Sample and output keys $(k_0', k_1') \leftarrow \mathsf{Gen}_{\mathsf{DPF}}(1^\lambda, f_{\alpha,\beta})$, for $\alpha = r^{\mathsf{in}} \in \mathbb{G}^{\mathsf{in}}$ and $\beta = 1 \in \mathbb{G}^{\mathsf{out}}$. Sample random additive secret shares $\langle r_0, r_1 \rangle$ of $r^{\mathsf{out}} \in \mathbb{G}^{\mathsf{out}}$. Output keys $k_0 = (k_0', r_0)$ and $k_1 = (k_1', r_1)$.
- $\mathsf{Eval}_{\mathsf{zt}}(\sigma, k_\sigma, x)$: Output $\mathsf{Eval}_{\mathsf{DPF}}(\sigma, k_\sigma', x) + r_\sigma$.

Correctness and security can be easily seen to follow from those of the DPF. Moreover, the construction does not involve additional cryptographic operations beyond making a single call to the DPF. □

The case of comparing two group elements can be easily reduced to the above case of a zero-test. Indeed, by taking the difference between the two masked inputs, the problem reduces to a zero-test of a masked input whose mask is the difference between the two masks, where the latter are known to the key generation algorithm. We provide an explicit description of the corresponding FSS scheme below.

Theorem 4.3 (Equality Test from DPF)**.** *There is an FSS scheme* $(\mathsf{Gen}_{\mathsf{eq}}, \mathsf{Eval}_{\mathsf{eq}})$ *for the offset function family* $\hat{\mathcal{G}}_{\mathsf{eq}}$ *making black-box use of a PRG. The scheme has the same key size and number of PRG invocations as a DPF with input domain* $\mathbb{G}^{\mathsf{in}}$ *and output domain* $\mathbb{G}^{\mathsf{out}}$*.*

Proof. Consider the following construction, where $(\mathsf{Gen}_{\mathsf{DPF}}, \mathsf{Eval}_{\mathsf{DPF}})$ is a distributed point function.

- $\mathsf{Gen}_{\mathsf{eq}}(1^\lambda, g_{\mathsf{eq}}^{[r^{\mathsf{in}}, r^{\mathsf{out}}]})$: Parse $g_{\mathsf{eq}}^{[r^{\mathsf{in}}, r^{\mathsf{out}}]}$ to recover $\mathbb{G}^{\mathsf{in}} = (\mathbb{G}_1^{\mathsf{in}} \times \mathbb{G}_2^{\mathsf{in}}), \mathbb{G}^{\mathsf{out}}, r^{\mathsf{in}}, r^{\mathsf{out}}$, where $r^{\mathsf{in}} = (r_1^{\mathsf{in}}, r_2^{\mathsf{in}}) \in \mathbb{G}^{\mathsf{in}}$. Sample and output keys $(k_0', k_1') \leftarrow \mathsf{Gen}_{\mathsf{DPF}}(1^\lambda, f_{\alpha,\beta})$, for $\alpha = (r_1^{\mathsf{in}} - r_2^{\mathsf{in}}) \in \mathbb{G}^{\mathsf{in}}$ and $\beta = 1 \in \mathbb{G}^{\mathsf{out}}$. Sample random additive secret shares $\langle r_0, r_1 \rangle$ of $r^{\mathsf{out}} \in \mathbb{G}^{\mathsf{out}}$. Output keys $k_0 = (k_0', r_0)$ and $k_1 = (k_1', r_1)$.
- $\mathsf{Eval}_{\mathsf{eq}}(\sigma, k_\sigma, (x_1, x_2))$: Output $\mathsf{Eval}_{\mathsf{DPF}}(\sigma, k_\sigma, (x_1 - x_2)) + r_\sigma$.

Security follows directly. Correctness holds since the point function $f_{\alpha,\beta}(x)$ evaluates to $\beta = 1$ exactly when $(x_1 - x_2) = \alpha = (r_1^{\mathsf{in}} - r_2^{\mathsf{in}})$, or equivalently, when $(x_1 - r_1^{\mathsf{in}}) = (x_2 - r_2^{\mathsf{in}})$. As before, the only cryptographic operations involve a single call to the DPF. □

4.2 Integer Comparison, Interval Membership, and Splines

We turn from equality-type gates to the slightly more involved case of gates related to integer comparisons. The offset functions of such gates can be easily expressed in terms of distributed *interval* functions (DIFs) as constructed in [9]. See Definition 2.6 and Theorem 2.7.

Definition 4.4 (Comparison-Type Gates)**.**

1. The *interval-containment gate* $\mathcal{G}_{(a,b)}$ is the class of functions $g_{(a,b)} : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$ parameterized by Abelian groups $\mathbb{G}^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}}$ endowed with a total ordering $a \leq b \in \mathbb{G}^{\mathsf{in}}$, and given by

$$g_{(a,b)}(x) = \begin{cases} 0 \in \mathbb{G}^{\mathsf{out}} & \text{if } a \leq x \leq b \in \mathbb{G}^{\mathsf{in}} \\ 1 \in \mathbb{G}^{\mathsf{out}} & \text{else} \end{cases} .$$

   We also sometimes consider the sub-family of "special" (one-sided) intervals, in which $a = 0$ is set to the minimum element of $\mathbb{G}^{\mathsf{in}}$ (or, alternatively, the family wherein b is set to the maximum element of $\mathbb{G}^{\mathsf{in}}$). For these sub-families, Leak is amended to include this information.

2. Let $\mathbb{G}^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}}$ be Abelian groups endowed with a total ordering, and define $R \subseteq \mathbb{G}^{\mathsf{in}} \times \mathbb{G}^{\mathsf{in}}$ to be the set of all pairs $(x, x') \in \mathbb{G}^{\mathsf{in}} \times \mathbb{G}^{\mathsf{in}}$ such that the absolute difference between x and x' is less than $|\mathbb{G}^{\mathsf{in}}|/2$. The *comparison gate* $\mathcal{G}_{\leq}$ is the class of functions $g_{\leq} : R \to \mathbb{G}^{\mathsf{out}}$ parameterized by $\mathbb{G}^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}}$, and given by

$$g_{\leq}(x, x') = \begin{cases} 0 \in \mathbb{G}^{\mathsf{out}} & \text{if } x \leq x' \in \mathbb{G}^{\mathsf{in}} \\ 1 \in \mathbb{G}^{\mathsf{out}} & \text{else} \end{cases} .$$

   The promise that $|x - x'| < |\mathbb{G}^{\mathsf{in}}|/2$ is needed for the correctness of our implementation. Since $g_{\leq}$ is typically invoked on integers from a bounded range, one can ensure that the promise is met by using $\mathbb{G}^{\mathsf{in}}$ whose size is at least twice the range size.

3. The *spline gate* $\mathcal{G}_{\mathsf{spline}}$ is the class of functions $g_{(\vec{a},\vec{f})} : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$ parameterized by Abelian groups $\mathbb{G}^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}}$ endowed with a total ordering, a list $\vec{a} = a_1 < a_2 < \cdots < a_k \in \mathbb{G}^{\mathsf{in}}$, and a list of functions $\vec{f} = f_0, \ldots, f_k : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$, given by

$$g_{(\vec{a},\vec{f})}(x) = \begin{cases} f_0(x) \in \mathbb{G}^{\mathsf{out}} & \text{if } x \leq a_1 \in \mathbb{G}^{\mathsf{in}} \\ f_1(x) \in \mathbb{G}^{\mathsf{out}} & \text{if } a_1 < x \leq a_2 \in \mathbb{G}^{\mathsf{in}} \\ \quad\vdots & \\ f_k(x) \in \mathbb{G}^{\mathsf{out}} & \text{if } a_k < x \end{cases} .$$

   By default, we consider the case where $\mathbb{G}^{\mathsf{in}}$ and $\mathbb{G}^{\mathsf{out}}$ are the additive groups of the same finite ring (e.g., $R = \mathbb{Z}_{2^m}$), and each f_i is a degree-d univariate polynomial over R. This is useful in the context of approximating real-valued functions.

We start with the case of interval containment. The key observation is that the offset function of an interval (a, b) is can be expressed as the sum of two *special* intervals.

Proposition 4.5 (Interval-Containment from FSS for Intervals)**.** *There exists an FSS scheme* $(\mathsf{Gen}_{(a,b)}, \mathsf{Eval}_{(a,b)})$ *for the offset function family* $\hat{\mathcal{G}}_{(a,b)}$ *making black-box use of a PRG. The scheme has the same cost (in key size and number of PRG invocations) as* two *instances of a* special *DIF with input domain* $\mathbb{G}^{\mathsf{in}}$ *and output domain* $\mathbb{G}^{\mathsf{out}}$*, except that each key includes an additional element of* $\mathbb{G}^{\mathsf{out}}$*. Moreover, there is an FSS scheme with the same parameters for the offset function family* $\hat{\mathcal{G}}_{\leq}$ *of comparison gates.*

Proof. We argue that each function in the offset family $\hat{\mathcal{G}}_{(a,b)}$ can be expressed as the sum of two special intervals plus the constant offset r^{out}. Indeed, the effect of the input offset r^{in} is cyclically shifting the interval function $f_{(a,b),1}$ to the right. There are two possible cases:

1. There is no wrap-around, namely we get another standard interval of the form $f_{(a',b'),1}$. If $a' = 0$, this is a special interval. Otherwise it can be expressed as the sum of two special intervals: $f_{(a',b'),1} = f_{(0,b'),1} + f_{(0,a'-1),-1}$.
2. There is a wrap-around, in which case we get a sum of two disjoint special intervals: one starting with $a + r^{\mathsf{in}}$ and one ending with $(b + r^{\mathsf{in}}) \mod |\mathbb{G}^{\mathsf{in}}|$.

We can now realize FSS for the offset function by letting Gen generate independent keys for the two instances of a DIF, and Eval output the sum of the two output shares. Finally, given additive shares of the output offset r^{out} as part of the key, Eval can add r^{out} to the output. We can obtain an analogous statement for comparison gates $\hat{\mathcal{G}}_{\leq}$ similarly to the reduction of $\hat{\mathcal{G}}_{\mathsf{eq}}$ to $\hat{\mathcal{G}}_{\mathsf{zt}}$. Specifically, identifying $\mathbb{G}^{\mathsf{in}}$ with $\mathbb{Z}_{|\mathbb{G}^{\mathsf{in}}|}$ (with the natural order $0, 1, 2, \ldots$), generate keys for the offset function family $\hat{\mathcal{G}}_{(0,\lfloor|\mathbb{G}^{\mathsf{in}}|/2\rfloor)}$. To compare $x, x' \in \mathbb{G}^{\mathsf{in}}$ run the evaluation procedure on $x' - x$. The promise that $|x - x'| < |\mathbb{G}^{\mathsf{in}}|/2$ ensures that $x \leq x'$ if and only if $0 \leq x' - x \leq \lfloor|\mathbb{G}^{\mathsf{in}}|/2\rfloor$, where subtraction is in $\mathbb{Z}_{|\mathbb{G}^{\mathsf{in}}|}$. □

We turn to the case of spline functions, starting with the default case where $\mathbb{G}^{\mathsf{in}}$ and $\mathbb{G}^{\mathsf{out}}$ are the same finite ring R and each function $f_i(x)$ is a degree-d univariate polynomial over R. Here the high level idea is to use $2(k+1)$ instances of special DIF to additively share, for each interval, the $d+1$ *coefficients* of either the degree-d polynomial $f'_i(x') = f_i(x' - r^{\mathsf{in}}) + r^{\mathsf{out}}$ in case the input x' is in the shifted i-th interval or the 0 polynomial if x' is not in this interval. The $2(k+1)(d+1)$ coefficients can then be linearly combined with public coefficients to yield additive output shares.

Proposition 4.6 (Splines from FSS for Intervals)**.** *There exists an FSS scheme* $(\mathsf{Gen}_{(\vec{a},\vec{f})}, \mathsf{Eval}_{(\vec{a},\vec{f})})$ *for the offset function family* $\hat{\mathcal{G}}_{\mathsf{spline}}$*, where* $\mathbb{G}^{\mathsf{in}} = \mathbb{G}^{\mathsf{out}} = R$ *and each* f_i*,* $0 \leq i \leq k$*, is a polynomial of degree at most* d *over* R*, making black-box use of a PRG. The scheme has the same cost (in key size and number of PRG invocations) as* $2(k+1)$ *instances of a* special *DIF with input domain* R *and output domain* R^{d+1}*.*

Proof. We express the shifted spline function as the sum of $k+1$ cyclically shifted interval functions. As before, each shifted interval can be expressed as the sum of two special intervals. For the shifted interval i, $0 \leq i \leq k$, the payload $\beta_i \in R^{d+1}$ is the coefficient vector of the univariate polynomial f'_i, where $f'_i(x') = f_i(x' - r^{\mathsf{in}}) + r^{\mathsf{out}}$. Note that if x is *not* in the shifted interval, the output of the shifted interval function will be $\beta_i = (0, 0, \ldots, 0) \in R^{d+1}$. Finally, given the $k+1$ additively-shared coefficient vectors β_i, the parties can homomorphically evaluate $\langle \sum_{i=0}^{k} \beta_i \, , \, (1, x', (x')^2, \ldots, (x')^d) \rangle = g^{[r^{\mathsf{in}}, r^{\mathsf{out}}]}_{(\vec{a},\vec{f})}(x')$, where $\langle \cdot, \cdot \rangle$ denotes inner product over R. Since x' is public, this can be done via a local linear combination of the $(k+1)(d+1)$ ring elements included in the payloads β_i. □

We note that, with some loss of concrete efficiency, the spline construction can be generalized to accommodate any functions f_i from a class that supports efficient FSS. Such a construction can be obtained by using the general tensoring operator for FSS from [9] (Theorem 3.2 of full version) to obtain an FSS scheme for functions that output the same output as f_i on a shifted interval and 0 outside the interval.

4.3 Bit Decomposition

As a concluding item, we turn our attention to the more involved task of bit decomposition.

Definition 4.7 (Bit-Decomposition Gate)**.** The *bit-decomposition gate* $\mathcal{G}_{\mathsf{bit}}$ is the class of functions $g_{\mathsf{bit}} : \mathbb{Z}_M \to \mathbb{Z}_2^m$ parameterized by $M \in \mathbb{N}$ (and $m := \lceil \log M \rceil$), given by

$$g_{\mathsf{bit}}(a) = (a_{m-1}, \ldots, a_0) \in \mathbb{Z}_2^m \quad \text{such that } \sum_{i=0}^{m-1} 2^i a_i = a \in \mathbb{Z}_M.$$

We now describe how to obtain the required FSS for these gates.

Remark 4.8 (Bit Decomposition for Special Modulus)**.** For the sake of simplicity, we present in Proposition 4.9 a construction of bit decomposition for the special case of $\mathbb{Z}_M$ for $M = 2^m$. In this setting, modular arithmetic over M does not incur wraparound carries. The same construction and analysis covers also a promise setting where M is arbitrary, but both the input $x \in \mathbb{Z}_M$ and the secret offset $r^{\mathsf{in}} \in \mathbb{Z}_M$ are guaranteed to be of low magnitude (bounded by e.g. $\sqrt{M}$), as stated in Corollary 4.10. The general case of $\mathbb{Z}_M$ with arbitrary inputs and offsets requires a slightly more sophisticated treatment. We discuss the extension of our construction in Remark 4.11 below.

Proposition 4.9 (Bit-Decomposition for $M = 2^m$)**.** *There exists an FSS* $(\mathsf{Gen}_{\mathsf{bit}}, \mathsf{Eval}_{\mathsf{bit}})$ *for the offset function family* $\hat{\mathcal{G}}_{\mathsf{bit}}$ *(restricted to* $\mathbb{Z}_M$ *with* $M = 2^m$*) making black-box use of a pseudorandom generator* $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2(\lambda+1)}$ *with the following complexities.*

- $\mathsf{Gen}_{\mathsf{bit}}$ *for function* $g_{\mathsf{bit}} : \mathbb{Z}_M \to \mathbb{Z}_2^m$ *(with* $M = 2^m$*) makes* $4m(m-1)$ *calls to* PRG*. It outputs keys* k_0, k_1 *each of size* $2(\lambda + 4)m(m-1) + m$ *bits.*

- $\mathsf{Eval}_{\mathsf{bit}}$ *makes* $2m(m-1)$ *calls to* PRG.

Proof. Consider the following construction, making use of an FSS for special intervals $(\mathsf{Gen}_{\mathsf{SI}}, \mathsf{Eval}_{\mathsf{SI}})$. Recall the goal is to recover shares of the (r^{out}-shifted) bit representation of $x+(-r^{\mathsf{in}}) \in \mathbb{Z}_M$, where $r^{\mathsf{in}}, r^{\mathsf{out}}$ are known at time of FSS generation. Let $\vec{r} = (r_{m-1}, \ldots, r_0) \in \mathbb{Z}_2^m$ denote the bit representation of $(-r^{\mathsf{in}}) \in \mathbb{Z}_M$ (note the additive inverse for notational convenience). For (public) input $x \in \mathbb{Z}_M$, we similarly denote its bit representation as $(x_{m-1}, \ldots, x_0)$.

Given public input x, we will compute (shares of) each bit of $(x+(-r^{\mathsf{in}}))$ over $\mathbb{Z}_M$ by computing "grade-school" addition on the bits. Each desired output bit y_i, $i \in \{0, \ldots, m-1\}$ can be expressed as a sum *over* $\mathbb{Z}_2$: $y_i = x_i \oplus r_i \oplus \mathsf{carry}_{i,\vec{r}}(x)$, where $\mathsf{carry}_{i,\vec{r}}(x) \in \{0,1\}$ is equal to 1 precisely when there is a carry entering into bit i from the lower-order bits, indexes $j < i$. Note that for $M = 2^m$, there are no wraparound carries.

The function $x_i \oplus r_i$ is linear over the output space $\mathbb{Z}_2$ and can thus be directly evaluated given the public input x and additive secret shares of r_i (over $\mathbb{Z}_2$). The challenge is in implementing the nonlinear function $\mathsf{carry}_{i,\vec{r}}(x)$, while hiding the value of $\vec{r}$. To do so, we make a simple observation: $\mathsf{carry}_{i,\vec{r}}(x) = 1$ if and only if $(\sum_{j=0}^{i-1} 2^j x_j) \geq 2^i - (\sum_{j=0}^{i-1} 2^j r_j) \in \mathbb{Z}_{2^i}$. That is, there is a carry exactly if the numbers formed by the two truncated bit strings $(x_{i-1}, \ldots, x_0)$ and $(r_{i-1}, \ldots, r_0)$ sum to greater than 2^i (note they will never reach 2^{i+1}). For each index $i \in \{0, \ldots, m-1\}$, we can thus implement FSS for $\mathsf{carry}_{i,\vec{r}}(x)$ directly by one FSS for a *special (one-sided) interval* $f_{(a,2^{i-1})} : \mathbb{Z}_{2^i} \to \{0,1\}$, which evaluates to 1 on input $x' \in \mathbb{Z}_{2^i}$ precisely if $x' > a$.

We thus achieve the desired FSS with the following construction.

- $\mathsf{Gen}_{\mathsf{bit}}(1^\lambda, g_{\mathsf{bit}}^{[r^{\mathsf{in}}, r^{\mathsf{out}}]})$:

1. Parse $g_{\mathsf{bit}}^{[r^{\mathsf{in}},r^{\mathsf{out}}]}$ to recover $M, r^{\mathsf{in}} \in \mathbb{Z}_M, r^{\mathsf{out}} \in \mathbb{Z}_2^m$. Parse r^{in} as its bit representation $\vec{r} = (r_{m-1}, \dots, r_0)$.
2. For each $i \in \{0, \dots, m-1\}$, do the following.
   (a) Sample special interval FSS keys $(k_0^i, k_1^i) \leftarrow \mathsf{Gen}_{\mathsf{SI}}(1^\lambda, f_{(a,2^i-1)})$, for $f_{(a,2^i-1)} : \mathbb{Z}_{2^i} \to \{0,1\}$, with $a = 2^i - (\sum_{j=0}^{i-1} 2^j r_j) \in \mathbb{Z}_{2^i}$.
   (b) Sample random additive secret shares $\langle z_0^i, z_1^i \rangle$ of $(r_i \oplus r_i^{\mathsf{out}})$ over $\mathbb{Z}_2$.
3. Output keys $k_0 = (k_0^i, z_0^i)_{i=0}^{m-1}$ and $k_1 = (k_1^i, z_1^i)_{i=0}^{m-1}$.

- $\mathsf{Eval}_{\mathsf{bit}}(\sigma, k_\sigma, x)$: Parse $k_\sigma = (k_\sigma^i, z_\sigma^i)_{i=0}^{m-1}$ and $x = (x_{m-1}, \dots, x_0)$ For each $i \in \{0, \dots, m-1\}$, do the following.
  1. Execute $\mathsf{carry}_\sigma^i = \mathsf{Eval}_{\mathsf{SI}}\left(\sigma, k_\sigma^i, \sum_{j=0}^{i-1} 2^j x_j\right)$.
  2. Let $y_\sigma^i = \sigma \cdot x_i \oplus \mathsf{carry}_\sigma^i \oplus z_\sigma^i \in \mathbb{Z}_2$. Note that a single party will contribute x_i. (Recall z_σ^i incorporates party σ's shares of both the r_i bit itself, as well as output offset bit r_i^{out}.)
- Output $(y_\sigma^{m-1}, \dots, y_\sigma^0) \in \mathbb{Z}_m^2$ as party σ's output share.

Correctness of the construction holds as argued above; FSS security holds directly by the security of the underlying FSS scheme for special intervals (and additive secret sharing). □

As mentioned, this case extends beyond just $\mathbb{Z}_M$ for $M = 2^m$, if we are in a promise setting of small inputs as compared to the modulus size. This can be useful within applications, e.g., in order to emulate computations over a non-2^m modulus by emulating over $\mathbb{Z}_M$ for an artificially large M.

Corollary 4.10 (Bit Decomposition for Small Inputs)**.** *There is an FSS scheme for the family of bit-decomposition functions with small inputs*

$$\hat{\mathcal{G}}_{\mathsf{bit}}^{\mathsf{small\text{-}input}} := \left\{ g^{[r^{\mathsf{in}},r^{\mathsf{out}}]} : \mathbb{Z}_M \to \mathbb{Z}_2^m \;\middle|\; \begin{array}{c} g_{\mathsf{bit}} : \mathbb{Z}_M \to \mathbb{Z}_2^m \in \mathcal{G}_{\mathsf{bit}}, \\ r^{\mathsf{in}} \in \mathbb{Z}_M, |r^{\mathsf{in}}| \le \sqrt{M}, r^{\mathsf{out}} \in \mathbb{Z}_2^m \end{array} \right\},$$

where the FSS guarantees correctness for inputs $x \in \mathbb{Z}_M$ of small magnitude $|x| \le \sqrt{M}$. *The complexities of the FSS are as in Proposition 4.9.*

Remark 4.11 (Bit Decomposition for General Modulus)**.** Our bit-arithmetic approach can be extended to the setting of general modulus M, by combining with an additional branch that either computes the same function as in the $M = 2^m$ case (if no wraparound occurs), or the function with an additional additive offset of $2^m - M$ (if a wraparound does occur). Ultimately, the computation of each $\mathsf{carry}_{i,\vec{r}}$ can be expressed by the linear combination of two different functions, each an AND of two special intervals (namely, [> value to wraparound] ∧ [> value to induce carry given wraparound] as well as [< value to wraparound] ∧ [> value to induce carry without wraparound]).

This can be instantiated via FSS for 2-dimensional intervals, as described in [9]. We leave optimization of such scheme to future work.

4.4 Garbling-Compatible Variants

For the purpose of minimizing round complexity, it can be beneficial to combine FSS-based gate evaluation with garbled circuits, where the outputs of FSS gates are fed into a garbled circuit. This motivates garbling-compatible variants of the above types of gates, where the bits of the output select between pairs of secret keys that correspond to inputs of the garbled circuit.

We can realize this modified functionality with a low additional cost for almost all of the above types of gates (the only exception is spline gates, whose output is not binary). This is done in the following way. The secret keys are incorporated into the function families as part of the function description. The key selection is done by incorporating the keys in the DPF or DIF payload β. For instance, in the case of interval membership, the input domain is partitioned into intervals, where for each interval a DIF whose payload is the corresponding key is used to produce an additive secret-sharing of the key corresponding to membership in the (shifted) interval.

5 Negative Results and Barriers

In this section we rule out information-theoretic protocols that achieve the efficiency features of our FSS-based protocols, showing that the machinery we use is in a sense necessary. This should be contrasted with the fact that most positive results on secure computation given a trusted source of correlated randomness are information-theoretic.

We also give evidence that ruling out information-theoretic protocols with slightly relaxed efficiency features is difficult, by establishing a link with the existence of big matching vector families, a well known open problem in extremal combinatorics.

5.1 Online-Optimal Shared Equality Implies DPF

One of the simplest nontrivial instances of our positive results is a secure protocol for *string equality* with preprocessing and with secret-shared output. Concretely, we consider a protocol that given a pair of n-bit strings (x_0, x_1) and correlated randomness (r_0, r_1) outputs a *secret-sharing* of a single bit that indicates whether $x_0 = x_1$.

We define an *online-optimal protocol* to be one that has a single online round in which the message sent by each party is of the same length as its input. (By the perfect correctness requirement, the message length cannot be shorter than the input.) Note that this optimality feature is indeed satisfied by our DPF-based protocol, whose existence can be based on any OWF. We show that any online-optimal protocol can be used to build a DPF, though possibly with an exponential computation overhead in the input length. The latter suffices to prove that an online-optimal shared equality protocol implies a OWF.

Given the very restricted nature of an online-optimal protocol, this converse direction of showing that it implies a DPF may appear to be a mere syntactic translation. However, there are two main challenges that complicate this proof. First, the mapping of the inputs to messages does not necessarily rely on just additive masking. We get around this by requiring this masking to be efficiently invertible. (This requirement is not needed in case the input domain size is polynomial in the security parameter.) Second, the class of online-optimal protocols can deviate from the template of first masking the inputs using a pair of independent random strings and then *independently* applying an FSS scheme to the offset class defined by mapping the masked inputs to the secret-shared output. Indeed, a general protocol allows an arbitrary dependence between the two parts.

As a result of these subtleties, it is not clear how to extend our argument from equality to general functions. Even in the case of equality, we need to assume the protocol to have the extra "efficient inversion" property mentioned above, unless the input domain is small.

We now formalize the notion of an online-optimal shared equality protocol and prove that it implies a DPF.

Definition 5.1 (Online-optimal shared equality protocol)**.** A protocol $\Pi = (\mathsf{Setup}, \mathsf{Msg}, \mathsf{Out})$ is an *online-optimal shared equality protocol* with inversion algorithm Inv if it satisfies the following requirements:

- **Syntax**: The protocol has the following structure.
  1. $\mathsf{Setup}(1^\lambda, 1^n)$ outputs correlated randomness (r_0, r_1) of size $\mathsf{poly}(\lambda, n)$.
  2. In the online phase party P_σ, on input $x_\sigma \in \{0,1\}^n$, sends a single message $m_\sigma = \mathsf{Msg}(\sigma, r_\sigma, x_\sigma)$ to $P_{1-\sigma}$, where $m_\sigma \in \{0,1\}^n$.
  3. Party P_σ outputs $y_\sigma \in \{0,1\}$ where $y_\sigma = \mathsf{Out}(\sigma, r_\sigma, x_\sigma, m_{1-\sigma})$.
- **Correctness**: For any $x_0, x_1 \in \{0,1\}^n$, the resulting outputs y_0, y_1 always satisfy $y_0 \oplus y_1 = EQ(x_0, x_1)$, where EQ outputs 1 if the two inputs are equal and outputs 0 otherwise.
- **Security**: The protocol Π computationally hides from P_σ the input $x_{1-\sigma}$. Formally, it satisfies the security requirement of Definition 2.2 with respect to the constant function $f(x_0, x_1) = 0$.
- **Inversion**: The algorithm Inv extracts the input from the corresponding randomness and message. That is, for any $\sigma \in \{0,1\}$ and $r_\sigma, x_\sigma, m_\sigma$ consistent with an execution of Π, we have $\mathsf{Inv}(\sigma, 1^\lambda, r_\sigma, m_\sigma) = x_\sigma$.

Note that the correctness requirement implies that x_σ is uniquely determined by r_σ and m_σ. Moreover, whenever $n = O(\log \lambda)$, one can implement Inv in polynomial-time via brute-force search. This will suffice for constructing a DPF on a polynomial-size input domain, which implies a OWF. In our DPF-based shared equality protocol, however, the message is obtained by simply adding (or XORing) the input and a part of the randomness, and thus Inv can be implemented in linear time.

We now construct a DPF given oracle access to Π and Inv as in Definition 5.1. The intuition for the construction is the following. Given Inv, the output of each party σ can be computed from r_σ, m_0, m_1 alone (without relying on the input x_σ). The correlated randomness (r_0, r_1) then defines a pair of tables $T_\sigma^{r_\sigma}$, where $T_\sigma^{r_\sigma}[m_0, m_1]$ contains the output of P_σ on randomness r_σ and messages (m_0, m_1). The two tables can be viewed as shares of a shifted identity matrix $T = T_0^{r_0} \oplus T_1^{r_1}$, where the location Δ of the 1-entry in the first row of T is masked by randomness of both parties. To convert this into a DPF for a point function $f_{\alpha,1}$, we include $\alpha \oplus \Delta$ in both keys. This does not reveal α to either party and yet effectively allows the parties to convert the first row of T into one that contains 1 only in position α and 0 elsewhere, as required for sharing $f_{\alpha,1}$. Finally, to guarantee security even in the case of super-polynomial input domains, we need to replace the first row with a random row ρ, where ρ is included in both DPF keys.

The construction is formally described in Figure 1.

Theorem 5.2. *If* $(\mathsf{Setup}, \mathsf{Msg}, \mathsf{Out}, \mathsf{Inv})$ *form an online-optimal shared equality protocol as in Definition 5.1, then* $(\mathsf{Gen}, \mathsf{Eval})$ *defined in Figure 1 is a DPF.*

```
$\mathsf{Gen}(1^\lambda, \alpha)$:        ▷ We assume here that $\mathbb{G}^{\mathsf{in}} = \mathbb{Z}_2^{|\alpha|}$, $\mathbb{G}^{\mathsf{out}} = \mathbb{Z}_2$, and $\beta = 1$
  1: $n \leftarrow |\alpha|$; Sample random $\rho \leftarrow \{0,1\}^n$;
  2: $(r_0, r_1) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$;
  3: Let $\pi_0, \pi_1 : \{0,1\}^n \to \{0,1\}^n$ defined by $\pi_\sigma(x_\sigma) = \mathsf{Msg}(\sigma, r_\sigma, x_\sigma)$;
  4: $\Delta \leftarrow \pi_1(\pi_0^{-1}(\rho))$;        ▷ Use $\mathsf{Msg}$ to compute $\pi_1$ and $\mathsf{Inv}$ to compute $\pi_0^{-1}$
  5: Output $(k_0, k_1)$ where $k_\sigma = (r_\sigma, \rho, \alpha \oplus \Delta)$;
$\mathsf{Eval}(\sigma, k_\sigma, x)$:
  1: $n \leftarrow |x|$;
  2: Parse $k_\sigma$ as $(r_\sigma, \rho, \alpha')$;
  3: Let $x_\sigma \leftarrow \mathsf{Inv}(\sigma, 1^\lambda, r_\sigma, \rho)$ if $\sigma = 0$ or $\mathsf{Inv}(\sigma, 1^\lambda, r_\sigma, \alpha' \oplus x)$ if $\sigma = 1$;
  4: Let $y_\sigma \leftarrow \mathsf{Out}(\sigma, r_\sigma, x_\sigma, \alpha' \oplus x)$ if $\sigma = 0$ or $\mathsf{Out}(\sigma, r_\sigma, x_\sigma, \rho)$ if $\sigma = 1$;
  5: Output $y_\sigma$        ▷ $y_\sigma = T_\sigma^{r_\sigma}[\rho, \alpha' \oplus x]$
```

Figure 1: DPF from online-optimal shared equality protocol ($\mathsf{Setup}, \mathsf{Msg}, \mathsf{Out}, \mathsf{Inv}$).

Proof. We separately argue correctness and security.

For correctness, letting $\pi_\sigma, T_\sigma^{r_\sigma}$ be as in Figure 1 and $T = T_0^{r_0} \oplus T_1^{r_1}$, we can write:

$$
\begin{aligned}
y_0 \oplus y_1 &= T[\rho, \alpha' \oplus x] && (1)\\
&= EQ(\pi_0^{-1}(\rho), \pi_1^{-1}(\alpha' \oplus x)) && (2)\\
&= EQ(\pi_1(\pi_0^{-1}(\rho)), \alpha' \oplus x) && (3)\\
&= EQ(\Delta, \alpha' \oplus x) && (4)\\
&= EQ(\Delta, (\alpha \oplus \Delta) \oplus x) && (5)\\
&= EQ(\alpha, x) && (6)
\end{aligned}
$$

as required, where (1) follows from Lines 3-4 of Eval, (2) from the correctness of the equality protocol, (3) from applying π_1 to both sides from the left, (4) from Line 4 of Gen, (5) from Line 5 of Gen and Line 2 of Eval, and (6) by masking both sides with $\alpha \oplus \Delta$.

We turn to argue security. A key k_σ produced by Gen is of the form $k_\sigma = (r_\sigma, \rho, \alpha \oplus \Delta)$, where $\Delta = \pi_1(\pi_0^{-1}(\rho))$. From the (computational) security of the equality protocol against P_0, it follows that $(r_0, \rho, \pi_1(\rho)) \approx (r_0, \rho, \pi_1(\rho'))$, where ρ' is distributed uniformly over $\{0,1\}^n$ independently of ρ, and since π_1 is a permutation we have

$$(r_0, \rho, \pi_1(\rho)) \approx (r_0, \rho, \rho'). \tag{7}$$

Similarly, from the security against P_1 it follows that

$$(r_1, \rho, \pi_0(\rho)) \approx (r_1, \rho, \rho'). \tag{8}$$

It follows from (7) that

$$(r_0, \rho, \pi_1(\pi_0^{-1}(\rho))) \equiv (r_0, \pi_0(\rho), \pi_1(\rho)) \approx (r_0, \pi_0(\rho), \pi_1(\rho')) \equiv (r_0, \rho, \rho')$$

and hence $k_0 = (r_0, \rho, \alpha \oplus \Delta) \approx (r_0, \rho, \rho')$ as required. Similarly, it follows from (8) and from $(\rho, \pi_0^{-1}(\rho)) \equiv (\pi_0(\rho), \rho)$ that $k_1 = (r_1, \rho, \alpha \oplus \Delta) \approx (r_1, \rho, \rho')$ as required. □

Corollary 5.3. *If there exists an online-optimal shared equality protocol as in Definition 5.1 with efficient* $(\mathsf{Setup}, \mathsf{Msg}, \mathsf{Out})$ *(but possibly without an efficient inversion algorithm* Inv*) then a one-way function exists.*

Proof. The algorithm Inv can be implemented in time $\mathsf{poly}(\lambda, 2^n)$ via a brute-force search that enumerates over all possible choices of x_σ. Suppose that the DPF described in Figure 1 (with oracle to Inv) has key size $|k_\sigma| = O((\lambda + n)^c)$ for a positive integer c. Then, letting $n(\lambda) = (c+1)\log\lambda$, we get a DPF with domain size $N(\lambda) = \lambda^{c+1}$ and asymptotically smaller key size $|k_\sigma| = O(\lambda^c)$. Using Theorem 5 from [20], this implies a one-way function. □

5.2 Matching Vectors Imply Shared Equality

In the previous section we have shown that any shared equality protocol that has *optimal* online complexity implies a DPF, which in turn implies a one-way function. This raises the following question: suppose we relax the optimality requirement by, say, allowing each online message to be of length $10n$ or even $\mathsf{poly}(n)$ rather than n. Does such a protocol still imply a DPF? Alternatively, can we get an information-theoretic protocol with this complexity?

We do not know the answer to the above question. However, we show that if we slightly relax the output sharing requirement by allowing a constant-size (rather than single-bit) shares, then the problem of shared equality reduces to finding big families of *matching vectors* modulo a composite [22, 19, 18, 5], a well studied problem in combinatorics. Given known constructions of matching vectors, this connection implies some unexpected (but rather weak) upper bounds. Perhaps more interestingly, the lack of progress on ruling out much bigger matching vector families suggests that proving strong lower bounds on information-theoretic shared equality protocols would be difficult.

Definition 5.4 (Matching vectors)**.** [18] Let m be a positive integer and $S \subseteq \mathbb{Z}_m \setminus \{0\}$. We say that subsets $U = \{u_1, \ldots, u_N\}$ and $V = \{v_1, \ldots, v_N\}$ of vectors in $\mathbb{Z}_m^h$ form an *S-matching family* if the following two conditions are satisfied:

- For all $i \in [N]$, $\langle u_i, v_i \rangle = 0$, where $\langle \cdot, \cdot \rangle$ denotes inner product over $\mathbb{Z}_m$;
- For all $i, j \in [N]$ such that $i \neq j$, $\langle u_i, v_j \rangle \in S$.

The best known constructions of matching vectors over a constant composite modulus m are of quasi-polynomial size. For instance, for $m = 6$ the best known construction is of size $N = h^{O(\log h / \log\log h)}$ [22]. Whether bigger sets of matching vectors exist is a well known open problem, and only weak upper bounds are known; see [5] for the current state of the art.

We now show how to use families of matching vectors over a constant-size modulus m to obtain an information-theoretic shared equality protocol that has a single online round and constant-size output shares.

Theorem 5.5. *Let n be a positive integer and $N = 2^n$. Suppose there is a family of matching vectors with parameters m, h, N as in Definition 5.4. Then there is a* perfectly secure *shared equality protocol* $(\mathsf{Setup}, \mathsf{Msg}, \mathsf{Out})$ *for n-bit inputs with the following efficiency features:*

- Setup *outputs correlated randomness* (r_0, r_1) *consisting of* $O(h)$ *elements of* $\mathbb{Z}_m$*;*
- $\mathsf{Msg}(\sigma, r_\sigma, x_\sigma)$ *outputs a message in* $\mathbb{Z}_m^h$*;*

- $\mathsf{Out}(\sigma, r_\sigma, x_\sigma, m_{1-\sigma})$ *outputs an output share in* $\mathbb{Z}_m$.

Moreover, the two output shares produced by Out *are equal if and only if* $x_0 = x_1$.

Proof. The protocol first encodes each input into a corresponding matching vector, and then computes shares of the inner product using the correlated randomness. In more detail, Setup generates a pair of random masks $R_0, R_1 \in \mathbb{Z}_m^h$ and additive shares of their inner product (this can be viewed as a generalized Beaver triple, or an instance of our FSS-based construction for a bilinear gate). Msg first encodes x_σ into a corresponding matching vector X_σ (where x_0 is encoded using U and x_1 using V) and outputs $X_\sigma + R_\sigma$. Finally, Out uses the correlated randomness and the two messages to compute *subtractive* shares of the inner product of X_0 and X_1 (namely, the difference between the outputs is the inner product). Security follows from the masking, and correctness from the definition of a matching vector family. □

Generalizing Theorem 5.5 to other useful predicates beyond equality seems challenging. Indeed, there are strong limitations on the existence of big sets of matching vectors with respect predicates other than equality, even for simple ones such as the "greater than" predicate [1]. This should be contrasted with our (computational) FSS-based protocols, which are not only more efficient for the simple case of equality but also apply almost as efficiently to the "greater than" predicate and other types of simple predicates.

Acknowledgements. The authors would like to thank Nishanth Chandran, Divya Gupta, Nishant Kumar and Mayank Rathee for helpful discussions and comments.

Research supported by ERC Project NTSC (742754). E. Boyle additionally supported by ISF grant 1861/16 and AFOSR Award FA9550-17-1-0069. N. Gilboa additionally supported by ISF grant 1638/15, ERC grant 876110, and a grant by the BGU Cyber Center. Y. Ishai additionally supported by ISF grant 1709/14, NSF-BSF grant 2015782, and a grant from the Ministry of Science and Technology, Israel and Department of Science and Technology, Government of India.

References

[1] B. Bauer, J. Vihrovs, and H. Wee. On the inner product predicate and a generalization of matching vector families. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, pages 41:1–41:13, 2018.

[2] D. Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Proceedings*, pages 420–432, 1991.

[3] A. Ben-Efraim, M. Nielsen, and E. Omri. Turbospeedz: Double your online spdz! improving SPDZ using function dependent preprocessing. In *ACNS 2019*, pages 530–549, 2019.

[4] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-homomorphic encryption and multiparty computation. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Proceedings*, pages 169–188, 2011.

[5] A. Bhowmick, Z. Dvir, and S. Lovett. New bounds for matching vector families. *SIAM J. Comput.*, 43(5):1654–1683, 2014.

[6] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO 2019, Part III*, pages 489–518, 2019.

[7] E. Boyle, N. Gilboa, and Y. Ishai. Secure computation with preprocessing via function secret sharing. In *Proceedings of TCC 2019, to appear.*

[8] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing. In *Advances in Cryptology - EUROCRYPT*, pages 337–367, 2015.

[9] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the ACM Conference on Computer and Communications Security, 2016*, pages 1292–1303, 2016. Full version: ePrint report 2018/707.

[10] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, pages 143–202, 2000.

[11] G. Couteau. New protocols for secure equality test and comparison. In *International Conference on Applied Cryptography and Network Security*, pages 303–320. Springer, 2018.

[12] G. Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, pages 473–503, 2019.

[13] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, pages 285–304, 2006.

[14] I. Damgård, J. B. Nielsen, M. Nielsen, and S. Ranellucci. The TinyTable protocol for 2-party secure computation, or: Gate-scrambling revisited. In *CRYPTO 2017, Part I*, pages 167–187, 2017.

[15] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 643–662, 2012.

[16] D. Demmler, T. Schneider, and M. Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *NDSS 2015*, 2015.

[17] J. Doerner and a. shelat. Scaling ORAM for secure computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 523–535, 2017.

[18] Z. Dvir, P. Gopalan, and S. Yekhanin. Matching vector codes. *SIAM J. Comput.*, 40(4):1154–1178, 2011.

[19] K. Efremenko. 3-query locally decodable codes of subexponential length. *SIAM J. Comput.*, 41(6):1694–1703, 2012.

[20] N. Gilboa and Y. Ishai. Distributed point functions and their applications. In *Advances in Cryptology - EUROCRYPT*, pages 640–658, 2014.

[21] O. Goldreich. *Foundations of Cryptography — Basic Applications.* Cambridge University Press, 2004.

[22] V. Grolmusz. On set systems with restricted intersections modulo a composite number. In *COCOON '97*, pages 82–90, 1997.

[23] Y. Ishai, E. Kushilevitz, S. Meldgaard, C. Orlandi, and A. Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In *TCC*, pages 600–620, 2013.

[24] J. Katz, V. Kolesnikov, and X. Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 525–537, 2018.

[25] J. Katz, S. Ranellucci, M. Rosulek, and X. Wang. Optimizing authenticated garbling for faster secure two-party computation. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, pages 365–391, 2018.

[26] P. Mohassel and P. Rindal. ABY^3: A mixed protocol framework for machine learning. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 35–52, 2018.