Function Secret Sharing for Mixed-Mode and Fixed-Point Secure Computation

Elette Boyle[1], Nishanth Chandran[2], Niv Gilboa[3], Divya Gupta[2], Yuval Ishai[4], Nishant Kumar*[5], and Mayank Rathee*[6]

[1] IDC Herzliya
[2] Microsoft Research, India
[3] Ben-Gurion University of the Negev
[4] Technion
[5] University of Illinois at Urbana-Champaign
[6] University of California, Berkeley

Abstract. Boyle *et al.* (TCC 2019) proposed a new approach for secure computation in the *preprocessing model* building on *function secret sharing* (FSS), where a gate g is evaluated using an FSS scheme for the related *offset family* $g_r(x) = g(x + r)$. They further presented efficient FSS schemes based on any pseudorandom generator (PRG) for the offset families of several useful gates g that arise in "mixed-mode" secure computation. These include gates for zero test, integer comparison, ReLU, and spline functions. The FSS-based approach offers significant savings in online communication and round complexity compared to alternative techniques based on garbled circuits or secret sharing.

In this work, we improve and extend the previous results of Boyle *et al.* by making the following three kinds of contributions:

- **Improved Key Size.** The preprocessing and storage costs of the FSS-based approach directly depend on the FSS key size. We improve the key size of previous constructions through two steps. First, we obtain roughly $4\times$ reduction in key size for Distributed Comparison Function (DCF), i.e., FSS for the family of functions $f^{<}_{\alpha,\beta}(x)$ that output β if $x < \alpha$ and 0 otherwise. DCF serves as a central building block in the constructions of Boyle *et al.*. Second, we improve the number of DCF instances required for realizing useful gates g. For example, whereas previous FSS schemes for ReLU and m-piece spline required 2 and $2m$ DCF instances, respectively, ours require only a *single instance of DCF* in both cases. This improves the FSS key size by $6-22\times$ for commonly used gates such as ReLU and sigmoid.
- **New Gates.** We present the first PRG-based FSS schemes for arithmetic and logical shift gates, as well as for bit-decomposition where both the input and outputs are shared over $\mathbb{Z}_{2^n}$. These gates are crucial for many applications related to fixed-point arithmetic and machine learning.
- **A Barrier.** The above results enable a 2-round PRG-based secure evaluation of "multiply-then-truncate," a central operation in fixed-point arithmetic, by sequentially invoking FSS schemes for multiplication

* Work done while at Microsoft Research, India

and shift. We identify a barrier to obtaining a 1-round implementation via a single FSS scheme, showing that this would require settling a major open problem in the area of FSS: namely, a PRG-based FSS for the class of bit-conjunction functions.

1 Introduction

Secure multi-party computation (or MPC) [8, 20, 29, 53] allows two or more parties to compute any function on their private inputs without revealing anything other than the output. A useful intermediate construction goal is that of MPC *in the preprocessing model*, wherein the parties receive *correlated randomness* from a trusted dealer in an offline input-independent phase, and then use this correlated randomness in the online phase once the inputs are known. Such protocols can be directly converted to ones in the standard model (without a dealer) via an assortment of general transformations, e.g. emulating the role of the dealer jointly using a targeted MPC protocol between the parties (see discussion in Appendix A in full version [9]). This modular design approach facilitates significant performance benefits, and indeed is followed by essentially all concretely efficient MPC protocols to date. Common types of correlated randomness include Beaver triples for multiplication [6], garbled circuit correlations [25, 53], OT [16, 31, 35] and OLE [32, 42] correlations, and one-time truth tables [23, 30].

When used to evaluate "pure" Boolean or arithmetic circuits, MPC protocols in the preprocessing model have the benefit of a very fast online phase in which the local computation performed by the parties is comparable to computing the circuit in the clear. Furthermore, the online *communication* is roughly the same as communicating the values of all wires in the circuit, and the number of online *rounds* is equal to the circuit depth.

Unfortunately, typical applications of MPC in areas such as machine learning and scientific computing apply computations that cannot be succinctly represented by pure Boolean or arithmetic circuits. Instead, they involve a mixture of arithmetic operations (additions and multiplications over a large field or ring) and "non-arithmetic" operations such as truncation, rounding, integer comparison, ReLU, bit-decomposition, or piecewise-polynomial functions known as splines. The cost of naively emulating such mixed computations by pure Boolean or arithmetic circuits is prohibitively high.

This motivated a long line of work on "mixed-mode" MPC, which supports efficient inter-conversions between arithmetic and Boolean domains and supports the above kinds of non-arithmetic operations. General frameworks such as [15, 19, 25, 36, 40] allow mixing of arithmetic gates (additions and multiplications) and Boolean gates (such as integer comparison), performing a suitable conversion whenever the type of gate changes. Together with MPC protocols for Boolean circuits based on garbled circuits or secret sharing, they can support the above kinds of non-arithmetic operations. However, the efficiency of these techniques leaves much to be desired, as they typically incur a significant overhead in communication and rounds even when ignoring the cost of input-independent preprocessing.

Recently, Boyle *et al.* [13] proposed a powerful approach for mixed-mode MPC in the preprocessing model, using *function secret sharing* (FSS) [10, 12] (their approach can be seen as a generalization of an earlier truth-table based protocol of Damgård *et al.* [23]). The FSS-based approach to MPC with preprocessing can support arithmetic operations that are mixed with the above kinds of non-arithmetic operations with the same online communication and round complexity as pure arithmetic computations, and while only making use of *symmetric* cryptography. In the present work, we significantly improve the efficiency of this FSS-based approach and extend it by supporting useful new types of non-arithmetic operations. Before giving a more detailed account of our results, we give an overview of the FSS-based approach to MPC with preprocessing.

1.1 MPC with Preprocessing Through FSS

At a high level, a (2-party) FSS scheme [10, 12] for a function family $\mathcal{F}$ splits a function $f \in \mathcal{F}$ into two *additive* shares f_0, f_1, such that each f_σ hides f and $f_0(x) + f_1(x) = f(x)$ for every input x. Here we assume that the output domain of f is a finite Abelian group $\mathbb{G}$, where addition is taken over $\mathbb{G}$. While this can be trivially solved by secret-sharing the truth-table of f, the goal of FSS is to obtain *succinct* descriptions of f_0 and f_1 using short keys k_0 and k_1, while still allowing their efficient evaluation.

For simplicity, consider *semi-honest* 2-party secure computation (2PC) with a *trusted dealer* – in the full version [9] we discuss how to emulate the trusted dealer with 2PC (building upon [27]) as well as extensions to malicious security, in Appendix A and B, respectively. The main idea, from [13], to obtain 2PC with trusted dealer is as follows. Consider a mixed circuit whose wires take values from (possibly different) Abelian groups and where each gate g maps a single input wire to a single output wire. We can additionally make free use of fan-out gates that duplicate wires, "splitters" that break a wire from a product group $\mathbb{G}_1 \times \mathbb{G}_2$ into two wires, and "joiners" that concatenate two wires into a single wire from the product group. This allows us to view a two-input gate (such as addition or multiplication) as a single-input gate applied on top of a joiner gate.

The FSS-based evaluation of such a circuit proceeds by maintaining the following invariant: for every wire w_i in the circuit, both parties learn the *masked* wire value $w_i + r_i$, where r_i is a random secret mask (from the group associated with w_i) which is picked by the dealer and is not revealed to any of the parties. The only exceptions are input wires, where the mask r_i is revealed to the party owning the input, and the circuit output wires, where the masks are revealed to both parties.

This above is easy to achieve for input wires by simply letting the dealer send to each party the masks of the inputs owned by this party, and having the parties reveal the masked inputs to each other. The challenge is to maintain the invariant when evaluating a gate g with input wire w_i and output wire $w_j = g(w_i)$ without revealing any information about the wire values. The idea is to consider the function mapping the masked input $w'_i = w_i + r_i$ to the masked

output $w'_j = g(w_i) + r_j$ as a *secret* function f determined by r_i and r_j, applied to the *public* input w'_i. Concretely, $f(w'_i) = g(w'_i - r_i) + r_j$.

Since the secret function f is known to the dealer (who picks all random masks), the dealer can securely delegate the evaluation of f to the two parties by splitting it into f_0 and f_1 via FSS and sending to each party σ its corresponding FSS key k_σ. Letting party σ evaluate $f_\sigma(w'_i)$, the parties obtain additive shares of w'_j, which they can safely exchange and recover the masked output w'_j. Finally, the circuit output wires are unmasked by having the dealer provide their masks to both parties.

The key observation is that given a gate g, the secret function f comes from the family of *offset* functions $\mathcal{F}_g$ that includes all functions of the form $g^{[\mathsf{r}^{\mathsf{in}},\mathsf{r}^{\mathsf{out}}]}(x) = g(x - \mathsf{r}^{\mathsf{in}}) + \mathsf{r}^{\mathsf{out}}$. (Alternatively, up to a slight loss of efficiency, it is enough to use FSS for the simpler class of functions of the form $g_r(x) = g(x+r)$, together with separate shares of the masks.) We refer to an FSS scheme for the offset function family $\mathcal{F}_g$ as an *FSS gate* for g. The key technical challenge in implementing the approach of [13] is in efficiently realizing FSS gates for useful types of gates g.

For addition and multiplication gates over a finite ring, the FSS gates are information-theoretic and essentially coincide with Beaver's protocol [6] (more accurately, its circuit-dependent variant from [7,21,23]). A key observation of [13] is that for a variety of useful non-arithmetic gates, including zero test, integer comparison, ReLU, splines, and bit-decomposition (mapping an input in $\mathbb{Z}_{2^n}$ to the corresponding output in $\mathbb{Z}_2^n$), FSS gates can be efficiently constructed using a small number of invocations of FSS schemes from [12]. The latter FSS schemes have the appealing feature of making a black-box use of any pseudorandom generator (PRG). This gives rise to relatively short keys and fast implementations using hardware support for AES.

Alternative variants. The above protocol uses *circuit-dependent* correlated randomness, since a wire mask is used in two or more gates incident to this wire, and this incidence relation depends on the circuit topology. At a small additional cost, one can break the correlations between FSS gates and obtain a circuit independent variant; see [13] for details. Another variant, which corresponds to how standard MPC protocols are typically described, is to use an FSS gate for mapping a *secret-shared* input to a *secret-shared* output (rather than a masked input to a masked output). This variant proceeds as described above, except that the parties start by reconstructing the masked input using a single round of interaction, and then use the FSS gate to locally compute shares of the output (without any interaction). With this variant, one can seamlessly use FSS gates in combination with other kinds of MPC protocols are based on garbled circuits, secret sharing, or homomorphic encryption.

Efficiency. When mapping a masked input to a masked output, processing a gate g requires only a *single* round of interaction, where each party sends a message to the other party. This message consists of a single element in the output group of g. Similarly, the variant mapping a secret-shared input to a

secret-shared output still requires only a single round of interaction, where the message here consists of a single element in the input group of g. Assuming a single round of interaction, this online communication complexity is optimal [13]. Overall, when evaluating a full circuit the communication by each party (using either the masked-input to masked-output or the shared-input to shared-output variant) is equal to that of communicating all wire values. The round complexity is equal to the circuit depth, no matter how complex the gates g are. The only complexity measures which are sensitive to the FSS gate implementation are the evaluation time and, typically more significantly, the size of the correlated randomness communicated by the dealer and stored by the parties. Optimizing the latter is a central focus of our work.

When is the FSS-based approach attractive? It is instructive to compare the efficiency features of the above FSS-based approach with that of the two main approaches for MPC with preprocessing: a Yao-style protocol based on garbled circuits (GC) [53] and a GMW-style protocol based on secret sharing [29].[7] Consider the goal of securely converting input shares for g into output shares when g is a nontrivial gate, say ReLU, over elements of $\mathbb{Z}_N$ for $N = 2^n$.

The FSS-based online protocol requires only *one* round of interaction in which each party sends only n bits (as argued above, this is optimal). In contrast, in a GC-based protocol the online phase (as used in several related works [15, 19, 25, 33, 39–41]) requires one of the parties to communicate $256n$ bits (a pair of AES keys for each input), which is 128× bigger. Furthermore, the parties need to interact in *two* sequential rounds. In the full version of this paper [9], we discuss a way to reduce the online communication of a GC-based protocol by 2×, which still leaves a 64× overhead in communication and 2× overhead in rounds over the FSS-based protocol. A GMW-style protocol typically requires a large number of rounds (depending on the multiplicative depth of a Boolean circuit implementing g), and has online per-party communication which is bigger than n by a multiplicative factor which depends on the number of multiplication gates in the circuit. See full version for a more concrete comparison with previous works taking the GC-based or GMW-based approach.

Even when considering MPC *without* preprocessing, namely, when the offline and online phases are combined, the FSS-based approach can still maintain some of its advantages. For instance, since keys for all FSS gates in a deep circuit can be generated in parallel, the advantage in round complexity is maintained. In the 3PC setting where one party emulates the role of the dealer, or in the 2PC setting with a relatively small input length n (see Appendix A of full version [9]), one can potentially beat the communication complexity of a GC-based protocol, depending on the FSS key size. This will be further discussed below.

To conclude, FSS-based protocols will typically outperform competing approaches in two common scenarios: (1) when offline communication is cheaper

[7] Here we only consider protocols whose online phase is based on *symmetric* cryptography. This excludes protocols based on homomorphic encryption, whose concrete costs are typically much higher.

than online communication, or alternatively (2) when latency is the bottleneck and minimizing rounds is a primary goal. In the setting of MPC with preprocessing, the FSS-based approach beats *all* previous practical approaches to mixed-mode secure computation with respect to *both* online communication and round complexity.

Finally, we stress that while the above discussion mainly focuses on semi-honest 2PC with a trusted dealer, most of the above benefits also apply to malicious security (see full version), and when emulating the trusted dealer using the different options we discuss: third party, 2PC protocol (full version), or semi-trusted hardware [43].

Bottlenecks for the FSS-based approach. Given the optimality of rounds and communication in the online evaluation of a gate g, the main bottleneck in the FSS-based approach lies in the size of the correlated randomness provided by the trusted dealer, namely the size of the FSS keys k_σ. This affects both *offline communication* and *online storage*. In the 3PC setting, where the trusted dealer is emulated by a third party, the FSS key size directly translates to offline communication from the third party to the other two parties. In the 2PC setting, where the dealer is emulated by an offline protocol for securely generating correlated randomness (see full version [9] for more details), the communication and computation costs of the offline protocol grow significantly with the key size. Thus, minimizing key size of useful FSS gates is strongly motivated by all application scenarios of FSS-based MPC.

Many compelling use-cases of MPC, such as privacy-preserving machine learning, finance, and scientific computing, involve numerical computation with finite precision, also known as "fixed-point arithmetic." Arithmetic over fixed-point numbers not only requires arithmetic operations such as additions and multiplications, for which efficient protocols can be based on traditional techniques, but also other kinds of operations that cannot be efficiently reduced to arithmetic operations over large rings. These include Boolean shift operators needed for adjusting the "scale" of fixed-point numbers. Concretely, for $N = 2^n$, a logical (resp., arithmetic) right shift by s converts an element $x \in \mathbb{Z}_N$ representing an n-bit unsigned (resp., signed) number to $y \in \mathbb{Z}_N$ representing $\lfloor x/2^s \rfloor$. To date, there are no PRG-based realizations of FSS gates for these Boolean operations,[8] and hence, fixed-point arithmetic operations cannot be realized securely using existing lightweight FSS machinery.

We now discuss our contributions that address these bottlenecks.

[8] An FSS-based protocol for right-shift can be obtained using the FSS gate for bit-decomposition from [13]. However, their construction only allows output shares of bits over $\mathbb{Z}_2$, whereas such a reduction (as well as other applications) requires output shares over $\mathbb{Z}_N$. Conversion of shares from $\mathbb{Z}_2$ to $\mathbb{Z}_N$ would thus require an additional round of interaction. Furthermore, this approach would require key size quadratic in input length: $O(n^2\lambda)$ for $N = 2^n$ (i.e., n-bit numbers) and PRG seed length λ.

1.2 Our Contributions

In this work, we make the following contributions:

- **Improved Key Size.** We obtain both concrete and asymptotic improvements in key size for widely applicable FSS gates such as integer comparisons, interval containment, bit-decomposition, and splines.
- **New Gates.** We extend the scope of FSS-based MPC by providing the first efficient FSS gates for several useful function families that include (logical and arithmetic) right shift, as well as bit-decomposition with outputs shared in $\mathbb{Z}_N$ (rather than $\mathbb{Z}_2$ in the construction from [13]).
- **A Barrier.** We provide a barrier result explaining the difficulty of obtaining PRG-based FSS gates for functions such as fixed-point multiplication.

We now give more details about these three kinds of contributions.

Improved Key Size. In Table 1 we summarize our improvements in key size over [13] and compare our improved FSS key size with garbled circuit size for the same gates. We provide the key size both as a function of input bitlength n and for the special case $n = 16$. Compared to [13], we observe a reduction in key size ranging from 6× for ReLU to 22× for splines and 77× for multiple interval containment (MIC) with 12 intervals. (Please refer to Appendix D in full version [9] for precise definitions of all gate types.) As can be observed, for all of the FSS gates considered in [13], their key size was significantly larger than the garbled circuit size. With our constructions, the key size is significantly *lower* than garbled circuits, for all gates except bit-decomposition (with output in $\mathbb{Z}_2^n$). For instance, our key size is at least 2× better than garbled circuits for ReLU and 15× and 27× better for splines and MIC, respectively. Recall that when compared to MPC protocols that use garbled circuits for preprocessing, protocols that follow the FSS-based approach have 64× lower online communication and 2× less rounds. So with our new schemes, FSS-based MPC with preprocessing will typically become more efficient in storage as well. The offline cost can also be smaller in some MPC settings (such as the 3PC case).

Our improvements in key size are obtained in two steps. The first step is a roughly 4× improvement for a central building block of useful FSS gates that we call Distributed Comparison Function (DCF). A DCF is an FSS scheme for the family of functions $f^{<}_{\alpha,\beta}(x)$ that output β if $x < \alpha$ and 0 otherwise, where $\alpha, \beta \in \mathbb{Z}_N$. This improvement is independently motivated by several other applications, including Yao's millionaires' problem and 2-server PIR with range queries. However, our primary motivation is the fact that previous FSS gate constructions from [13] are cast as *reductions* that invoke multiple instances of DCF. As a second step, we significantly improve the previous reductions from [13] of useful non-arithmetic FSS gates to DCF. We describe these two types of improvements in more detail below.

Optimized DCF. The best previous DCF construction is an instance of an FSS scheme for decision trees from [12]. Instead, we provide a tighter direct construction that reduces the key size by roughly $4\times$. Concretely, the total key size is improved from $\approx 2n(4\lambda + n)$ to $\approx 2n(\lambda + n)$ for input and output domains of size $N = 2^n$ and PRG seed length λ, with similar savings for general input and output domains.[9]

Better reductions to DCF. We significantly reduce the number of DCF instances required by most of the non-arithmetic FSS gates from [13]. The main new building block is a new FSS scheme for the offset families of interval containment (IC for short) and splines (piecewise polynomial functions) when the comparison points are public. Our construction uses only one DCF instance compared to the analogous constructions from [13] that require 2 and $2m$ DCF instances for IC and splines with m pieces, respectively, but can hide the comparison points. We note that comparison points are public for almost all important applications - e.g. the popular activation function in machine learning, ReLU,[10] absolute value, as well as approximations of transcendental functions [38, 41].

Concretely, for $n = 16$ (where inputs and outputs are in $\mathbb{Z}_N$ for $N = 2^n$), including our improvement in DCF key size, we improve the key size from [13] by roughly $6\times$, $12\times$, and $22\times$ for the spline functions ReLU, absolute value and sigmoid, respectively, where the sigmoid function is approximated using 12 pieces [38]. Moreover, this improvement in key size makes the FSS-based construction beat garbled circuits not only in terms of online communication but also in terms of per-gate storage requirements. See Table 1 for a more detailed comparison.

The main technical idea that enables the above improvement is that an FSS scheme for the offset family of a *public* IC function $f_{[p,q]}$ (that outputs is 1 if $p \leqslant x \leqslant q$ and 0 otherwise) can be reduced to a single DCF instance with $\alpha = N-1+\mathsf{r}^{\mathsf{in}}$. We build on this construction to reduce FSS keys for multiple intervals (and hence splines with constant payload) to this single DCF instance. See Section 4 for details. Constructions for splines with general polynomial outputs employ additional techniques to embed secret payloads (see Section 5.1).

Another kind of FSS gate for which we get an asymptotic improvement in key size over [13] is *bit-decomposition* with outputs shared over $\mathbb{Z}_2$. Here an input $x \in \mathbb{Z}_N$ is split to its bit-representation $(x_{n-1}, \ldots, x_0) \in \{0,1\}^n$, where each x_i is individually shared over $\mathbb{Z}_2$. (This type of "arithmetic to Boolean" conversion can be useful for applying a garbled circuit to compute a complex function of x that is not efficiently handled by FSS gates.) Non-trivial protocols for bit-decomposition have been proposed in different MPC models [22, 44, 49, 50]. An FSS gate for the above flavor of bit-decomposition was given in [13] with $O(n^2\lambda)$

[9] A concurrent work by Ryffel *et al.* [48] on privacy-preserving machine learning using FSS also proposes an optimized DCF scheme. Our construction is around $1.7\times$ better in key size than theirs.

[10] A ReLU operator, or Rectified Linear Unit, is a function on signed numbers defined by $g(x) = \max(x, 0)$.

key size. Here we substantially improve the hidden constant by reducing the bit-decomposition problem to a series of public interval containments. Moreover, we show how to further reduce the key size by an extra factor of w at the cost of computational overhead that grows exponentially with w. Setting $w = \log n$, we get an asymptotic improvement in key size over [13], while maintaining poly(n) computation time.

Table 1. Comparison of our FSS gate key sizes, with those of [13], and Garbled Circuits (GC) [52]. For FSS (i.e., our work and [13]), we list total key size for *both* P_0, P_1. For GC, we under-approximate and consider only the size of garbled circuit. The table only captures the size of correlated randomness (offline communication in the 3PC case); the online communication corresponding to both FSS columns is at least $\frac{\lambda}{2}\times$ better than GC (and rounds $2\times$ better). $\mathbb{U}_N$, $\mathbb{S}_N$ denote unsigned and signed n-bit integers, respectively. We consider gates for: Interval containment (IC), multiple interval containment (MIC) with m intervals, splines with m intervals and d-degree polynomial outputs, ReLU, Absolute value (ABS), Bit Decomposition (BD), Logical/Arithmetic Right Shifts (LRS/ARS) by s. Syntax and definitions of all gates are described in appendix D in our full version [9]. We provide cost in terms of number of $\mathsf{DCF}_{n,\mathbb{G}}$ keys for DCF with input bitlength n and output group $\mathbb{G}$. To disambiguate between our optimized DCF and DCF used in [13], we use $\mathsf{DCF}^{\mathsf{BGI}}_{n,\mathbb{G}}$ for the latter. Let $\ell = \lceil \log |\mathbb{G}| \rceil$. Size of our optimized $\mathsf{DCF}_{n,\mathbb{G}}$ key is total $2\left(n(\lambda+\ell+2)+\lambda+\ell\right)$ bits. Size of $\mathsf{DCF}^{\mathsf{BGI}}_{n,\mathbb{G}}$ key (using [12]) is $2\left(4n(\lambda+1)+n\ell+\lambda\right)$ bits. For our BD scheme (with output over $\mathbb{U}_2^n$), w is a parameter (here we assume $w \mid n$) and compute grows exponentially with w. We provide approximate key size expressions here by ignoring lower order terms; refer to Table 2 in Appendix C.2 of full version [9] for exact expressions. The values in parenthesis give exact key size in bits for $\lambda = 128$, $n = 16$, $m = 12$, $d = 1$, $w = 4$, $s = 7$.

Gate	This work	BGI'19 [13]	GC
IC (n)	$\mathsf{DCF}_{n,\mathbb{U}_N}$ (4992)	$2 \times \mathsf{DCF}^{\mathsf{BGI}}_{n,\mathbb{U}_N}$ (34592)	$8\lambda n$ (15616)
MIC (n, m)	$\mathsf{DCF}_{n,\mathbb{U}_N} + 2mn$ (5344)	$2m \times \mathsf{DCF}^{\mathsf{BGI}}_{n,\mathbb{U}_N}$ (415104)	$6\lambda mn$ (145152)
Splines (n, m, d)	$\mathsf{DCF}_{n,\mathbb{U}_N^{m(d+1)}} + 4mn(d+1)$ (19040)	$2m \times \mathsf{DCF}^{\mathsf{BGI}}_{n,\mathbb{U}_N^{(d+1)}}$ (427008)	$4\lambda mn(d+2)$ (289536)
ReLU (n)	$\mathsf{DCF}_{n,\mathbb{U}_N^2}$ (5664)	$2 \times \mathsf{DCF}^{\mathsf{BGI}}_{n,\mathbb{U}_N^2}$ (35616)	$6\lambda n$ (11776)
ABS (n)	$\mathsf{DCF}_{n,\mathbb{U}_N^2}$ (5728)	$4 \times \mathsf{DCF}^{\mathsf{BGI}}_{n,\mathbb{U}_N^2}$ (71168)	$8\lambda n$ (15616)
BD (n, w)	$\frac{n}{w} \times \mathsf{DCF}_{\frac{n+w}{2},\mathbb{U}_2}$ (11544)	$(n-1) \times \mathsf{DCF}^{\mathsf{BGI}}_{\frac{n}{2},\mathbb{U}_2}$ (127952)	$2\lambda n$ (3840)
LRS (n, s)	$\mathsf{DCF}_{s,\mathbb{U}_N} + \mathsf{DCF}_{n,\mathbb{U}_N}$ (7324)	- (-)	$4\lambda n$ (7680)
ARS (n, s)	$\mathsf{DCF}_{s,\mathbb{S}_N} + \mathsf{DCF}_{n-1,\mathbb{S}_N^2}$ (7608)	- (-)	$4\lambda n$ (7680)

New FSS Gates. A central operation that underlies fixed-point arithmetic with bounded precision is a Boolean *right shift* operation that maps a number $x \in \mathbb{Z}_N$ to $y \in \mathbb{Z}_N$ representing $\lfloor x/2^s \rfloor$ for shift amount s. This operation comes in two flavors: *logical* that applies to unsigned numbers and *arithmetic* that applies to signed numbers in 2's complement representation. These operations are typically applied following a multiplication operation to enable further computations while keeping the significant bits. Previous results from the literature do not give rise to efficient PRG-based FSS gates for these shift operators. We present a new design approach to FSS for right shift that uses only two invocations of DCF, obtaining asymptotic key size of $O(n\lambda + n^2)$. See Section 6 for definitions and construction details and Table 1 for comparison of key size with garbled circuits.

Another new feasibility result is related to the bit-decomposition problem discussed above. The FSS gate for bit-decomposition from [13] crucially relies on the output bits x_i being shared over $\mathbb{Z}_2$, whereas in some applications one needs the bits x_i to be individually shared over $\mathbb{Z}_N$ (or a different $\mathbb{Z}_{N'}$). While a conversion from $\mathbb{Z}_2$ to $\mathbb{Z}_N$ can be done directly using another FSS gate or oblivious transfer, this costs at least one more round of interaction. We realize this generalized form of bit-decomposition directly by a single FSS gate, via a similar approach of reducing the problem to a series of public interval containments.

A Barrier. Most applications of MPC in the areas of machine learning (see [40, 41, 46] and references therein) and scientific computing (see [4, 5, 17, 18] and references therein) use fixed-point arithmetic for efficiently obtaining an approximate output. Fixed-point addition is defined to be the same as integer addition; however, fixed-point multiplication requires an integer multiplication followed by an appropriate right shift operation for preventing integer overflows (see Section 6). Many prior works, for efficiency reasons, implement this right shift (or truncation) through a non-interactive "local truncation" procedure [26, 37, 40, 41, 51]. This has two issues. First, the truncated output can be *totally* incorrect, in the sense of being random, with some (small) probability. Since this probability accumulates with the number of such multiplications (and hence truncations), it necessitates an increase of the modulus N that can take a toll on efficiency. While this overhead is reasonable in some cases [2, 47], local truncation may be too costly for large scale applications [46]. Second, even when a big error does not occur, the least significant bit resulting from local truncation is erroneous with high probability. Such small errors are aggregated over the course of the computation. This makes the correctness of the implementation more difficult to verify, and can potentially lead to fraud through salami slicing (or penny shaving) in financial applications [1], where the adversary ensures that the small errors are biased in its favorable direction.

Our new FSS gate constructions for right shifts provide an effective solution for performing fixed-point multiplication operations in two rounds by sequentially invoking two FSS gates: one FSS gate for performing multiplication over $\mathbb{Z}_N$ (implemented via [13] or a standard multiplication triple), followed by a second FSS gate to perform an arithmetic right shift for signed integers (or logical

shift for unsigned integers). This approach gives a faithful error-free implementation of secure fixed-point multiplication for inputs of all bitlengths. A natural question is whether it is possible to replace the two FSS gates by a single FSS gate, avoiding the additional round of communication, using only cheap symmetric cryptographic primitives such as a PRG.

We demonstrate a barrier toward this goal, showing that this requires settling a major open problem in the area of FSS: namely, whether the family of conjunctions of a subset of n bits has an FSS scheme based on symmetric cryptography. Currently, FSS schemes for this family are known only under structured, public-key computational hardness assumptions such as Decisional Diffie-Hellman [11], Paillier [28] or Learning With Errors [14, 26], that imply homomorphic public key encryption. Such FSS schemes are less efficient than the PRG-based schemes considered in this work by several orders of magnitude, with respect to both communication and computation.

2 Preliminaries

We provide an abbreviated version of preliminaries and notation. A more detailed formal treatment can be found in Appendix E in our full version [9].

Notation. We use arithmetic operations in the ring $\mathbb{Z}_N$ for $N = 2^n$. We naturally identify elements of $\mathbb{Z}_N$ with their n-bit binary representation, where 0 is represented by 0^n and $N-1$ by 1^n. Unless otherwise specified, we parse $x \in \{0,1\}^n$ as $x_{[n-1]}, \ldots, x_{[0]}$, where $x_{[n-1]}$ is the most significant bit (MSB) and $x_{[0]}$ is the least significant bit (LSB). For $0 \leqslant j < k \leqslant n$, $z = x_{[j,k)} \in \mathbb{Z}_{2^{k-j}}$ denotes the ring element corresponding to the bit-string $x_{[k-1]}, \ldots, x_{[j]}$. $||$ denotes string concatenation. *Function family* denotes an infinite collection of functions specified by the same representation. λ denotes computational security parameter.

2.1 Data Types and Operators

Unsigned and signed integers. We consider computations over finite bit unsigned and signed integers, denoted by $\mathbb{U}_N$ and $\mathbb{S}_N$, respectively, over n-bits. We note that $\mathbb{U}_N = \{0, \ldots, N-1\}$ is isomorphic to $\mathbb{Z}_N$. Moreover, $\mathbb{S}_N = \{-N/2, \ldots, 0, \ldots, N/2-1\}$ can be encoded into $\mathbb{Z}_N$ or $\mathbb{U}_N$ using 2's complement notation or mod N operation. The positive values $\{0, \ldots, N/2-1\}$ are mapped identically to $\{0, \ldots, N/2-1\}$ and negative values $\{-N/2, \ldots, -1\}$ are mapped to $\{N/2, \ldots, N-1\}$. In this notation, the MSB of (the binary representation of) x is 0 if $x \geqslant 0$ and 1 if $x < 0$. Note that addition, subtraction and multiplication of signed integers modulo N respect this representation as long as the result is in the range $[-N/2, N/2)$. Our work also considers fixed-point representation of numbers and its associated arithmetic. Section 6 provides a more detailed description of the mapping of rationals into the fixed-point space as well as fixed-point arithmetic.

Operators. We consider several standard operators, which can be thought of as applying to (signed or unsigned) integers. Each operator is defined by a *gate*: a function family parameterized by input and output domains and possibly other parameters. Some of the operators we consider are single and multiple interval containments (Section 4), splines and applications to ReLU and absolute value (Section 5.1), bit decomposition (Section 5.2 in full version [9]), as well as operators required for the realization of fixed-point arithmetic - such as fixed-point addition and multiplication (Section 6.1), logical right shifts (Section 6.2), arithmetic right shifts, and comparison (Section 6.3 & 6.4 in full version [9]).

2.2 Function Secret Sharing

We follow the definition of function secret sharing (FSS) from [12]. 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 [10, 12, 13], we insist on an additive representation of the output that is critical for applications.

Definition 1 (FSS: Syntax). *A (2-party)* function secret sharing (FSS) scheme *is a pair of algorithms* $(\mathsf{Gen}, \mathsf{Eval})$ *such that:*

- $\mathsf{Gen}(1^\lambda, \hat{f})$ *is a PPT* key generation algorithm *that given* 1^λ *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 *that given* $\sigma \in \{0,1\}$ *(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* $f_\sigma(x)$*).*

Definition 2 (FSS: Correctness and Security). *Let* $\mathcal{F} = \{f\}$ *be a function family and* Leak *be a 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 1 is an* FSS scheme for $\mathcal{F}$ *(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 sequence* $(\hat{f}_\lambda)_{\lambda \in \mathbb{N}}$ *of polynomial-size function descriptions from* $\mathcal{F}$ *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* $\mathsf{Sim}_\sigma(1^\lambda, \mathsf{Leak}(\hat{f}_\lambda))$*.*

A central building block for many of our constructions is an FSS scheme for a *special interval function* referred to as a *distributed comparison function* (DCF) as defined below. We formalize it below.

Definition 3 (DCF). *A* special interval *function $f^{<}_{\alpha,\beta}$, also referred to as a* comparison *function, outputs β if $x < \alpha$ and 0 otherwise. We refer to an FSS schemes for comparison functions as* distributed comparison function *(DCF). Analogously, function $f^{\leqslant}_{\alpha,\beta}$ outputs β if $x \leqslant \alpha$ and 0 otherwise. In all of these cases, we allow the default leakage $\mathsf{Leak}(\hat{f}) = (\mathbb{G}^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}})$.*

The following theorem captures the concrete costs of the best known construction of DCF from a PRG (Theorem 3.17 in the full version of [12]):

Theorem 1 (Concrete cost of DCF [12]). *Given a PRG $G : \{0,1\}^{\lambda} \to \{0,1\}^{2\lambda+2}$, there exists a DCF for $f^{<}_{\alpha,\beta} : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$ with key size $4n \cdot (\lambda + 1) + n\ell + \lambda$,where $n = \lceil \log |\mathbb{G}^{\mathsf{in}}| \rceil$ and $\ell = \lceil \log |\mathbb{G}^{\mathsf{out}}| \rceil$. For $\ell' = \lceil \frac{\ell}{\lambda+2} \rceil$, the key generation algorithm Gen invokes G at most $n \cdot (4 + \ell')$ times and the algorithm Eval invokes G at most $n \cdot (2 + \ell')$ times.*

We use $\mathbf{DCF}_{n,\mathbb{G}}$ to denote the total key size, i.e. $|k_0| + |k_1|$, of the DCF key with input length n and output group $\mathbb{G}$ (see Table 1). This captures the output length of Gen algorithm. On the other hand, we use $\mathsf{DCF}_{n,\mathbb{G}}$ (non-bold) to denote the key size per party, i.e., $|k_b|, b \in \{0,1\}$. This captures the key size used in Eval algorithm. In the rest of the paper, we use $\mathsf{DCF}_{n,\mathbb{G}}$ to count number of invocations/evaluations as well as key size per evaluator P_b, $b \in \{0,1\}$.

2.3 FSS Gates

The recent work of Boyle *et al.* [13] provided general-purpose transformations for obtaining efficient secure computation protocols in the preprocessing model via FSS schemes for corresponding function families.

The key idea is the following FSS-based gate evaluation procedure. For each gate $g : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$ in the circuit to be securely evaluated, the dealer uses an FSS scheme for the class of *offset* functions $\hat{\mathcal{G}}$ that includes all functions of the form $g^{[\mathsf{r}^{\mathsf{in}},\mathsf{r}^{\mathsf{out}}]}(x) = g(x - \mathsf{r}^{\mathsf{in}}) + \mathsf{r}^{\mathsf{out}}$. If the input to gate g is wire i and the output is wire j, the dealer uses the FSS scheme for $\hat{\mathcal{G}}$ to split the function $g^{[\mathsf{r}^{\mathsf{in}},\mathsf{r}^{\mathsf{out}}]}$ 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. We defer a formal statement of the corresponding transformation to Appendix E in our full version [9]. In what follows we introduce necessary terminology.

Definition 4 (Offset function family and FSS gates). *Let $\mathcal{G} = \{g : \mathbb{G}^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}\}$ be a computation gate (parameterized by input and output groups $\mathbb{G}^{\mathsf{in}}, \mathbb{G}^{\mathsf{out}}$). The* family of offset functions *$\hat{\mathcal{G}}$ of $\mathcal{G}$ is given by*

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

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

and $g^{[\mathsf{r}^{in},\mathsf{r}^{out}]}$ *contains an explicit description of* $\mathsf{r}^{in}, \mathsf{r}^{out}$*. Finally, we use the term* FSS gate for $\mathcal{G}$ *to denote an FSS scheme for the corresponding offset family* $\hat{\mathcal{G}}$*.*

As explained above, an FSS gate for $\mathcal{G}$ implies an "online-optimal" protocol for converting a masked input x to a masked output $g(x)$ for $g \in \mathcal{G}$. Concretely, the online phase consists of only one round in which each party sends a message of length $|g(x)|$. Alternatively, we can have a similar one-round protocol converting additively shared input to additively shared output, where here the message length is $|x|$. The offline communication and storage correspond to the FSS key size produced by Gen, and the online compute time corresponds to the computational cost of Eval.

Boyle *et al.* [13] constructed FSS gates for most of the operators from Section 2.1 by reducing them to multiple invocations of DCF. In this work we will improve the efficiency of previous DCF constructions, and provide better reductions (both asymptotically and concretely) from gates in Section 2.1 to DCF.

3 Optimized Distributed Comparison Function

A Distributed Comparison Function (DCF), as formalized in Definition 3, is an FSS scheme for the family of comparison functions. We reduce the key size of prior best known construction of [12] from roughly $n(4\lambda+n)$ to roughly $n(\lambda+n)$, i.e. roughly $4\times$, for input and output domains of size $N = 2^n$ and security parameter λ, with similar savings for general input and output domains.

Our construction draws inspiration from the DPF of [12]. The Gen algorithm uses a PRG G and generates two keys (k_0, k_1) such that $\forall b \in \{0,1\}$, k_b includes a random PRG seed s_b and $n+1$ shared *correction words*. A key implicitly defines a binary tree with $N = 2^n$ leaves where a node u is associated with a tuple (s_b, V_b, t_b), for a PRG seed s_b, an output group element $V_b \in \mathbb{G}$ and a bit t_b. The construction ensures that the sum $V_0 + V_1$ over all nodes leading to an input x is exactly equal to $f^{<}_{\alpha,\beta}(x)$. Therefore, evaluating a key k_b on an input x requires traversing the tree generated by k_b from the root to the leaf representing x, computing (s_b, V_b, t_b) at each node and summing up the values V_b.

The tuple (s_b, V_b, t_b) associated with u is a function of the seed associated with the parent of u and the correction words. Therefore, if $s_0 = s_1$ then for any descendent of u, k_0 and k_1 generate identical tuples. The correction words are chosen such that when a path to x departs from the path to α, the two seeds s_0 and s_1 on the first node off the path are identical, and the sum of $V_0 + V_1$ along the whole path to u is exactly zero if the departure is to the right of the path to α, i.e. $x > \alpha$, and is β if the departure is to the left of the path to α. Finally, along the path to α any seed s_b is computationally indistinguishable from a random string given the key k_{1-b}, which ensures the security of the construction.

The DCF scheme is presented in Fig. 1, and a formal statement of the scheme's complexity appears in Theorem 2 (see Appendix F.1 in full version [9] for detailed security proof). The scheme uses the function $\mathsf{Convert}_{\mathbb{G}} : \{0,1\}^{\lambda} \to \mathbb{G}$

Distributed Comparison Function $(\mathsf{Gen}_n^<, \mathsf{Eval}_n^<)$
Let $G : \{0,1\}^\lambda \to \{0,1\}^{2(2\lambda+1)}$ be a pseudorandom generator.
Let $\mathsf{Convert}_\mathbb{G} : \{0,1\}^\lambda \to \mathbb{G}$ be a map converting a random λ-bit string to a pseudorandom group element of $\mathbb{G}$.

$\mathsf{Gen}_n^<(1^\lambda, \alpha, \beta, \mathbb{G})$:

Let $\alpha = \alpha_1, \ldots, \alpha_n \in \{0,1\}^n$ be the bit decomposition of α
Sample random $s_0^{(0)} \leftarrow \{0,1\}^\lambda$ and $s_1^{(0)} \leftarrow \{0,1\}^\lambda$
Let $V_\alpha = 0 \in \mathbb{G}$, let $t_0^{(0)} = 0$ and $t_1^{(0)} = 1$
for $i = 1$ to n **do**
    $s_0^L || v_0^L || t_0^L \;||\; s_0^R || v_0^R || t_0^R \leftarrow G(s_0^{(i-1)})$
    $s_1^L || v_1^L || t_1^L \;||\; s_1^R || v_1^R || t_1^R \leftarrow G(s_1^{(i-1)})$
    if $\alpha_i = 0$ **then** $\mathsf{Keep} \leftarrow L$, $\mathsf{Lose} \leftarrow R$
    else $\mathsf{Keep} \leftarrow R$, $\mathsf{Lose} \leftarrow L$
    end if
    $s_{CW} \leftarrow s_0^{\mathsf{Lose}} \oplus s_1^{\mathsf{Lose}}$
    $V_{CW} \leftarrow (-1)^{t_1^{(i-1)}} \cdot [\mathsf{Convert}_\mathbb{G}(v_1^{\mathsf{Lose}}) - \mathsf{Convert}_\mathbb{G}(v_0^{\mathsf{Lose}}) - V_\alpha]$
    if $\mathsf{Lose} = L$ **then** $V_{CW} \leftarrow V_{CW} + (-1)^{t_1^{(i-1)}} \cdot \beta$
    end if
    $V_\alpha \leftarrow V_\alpha - \mathsf{Convert}_\mathbb{G}(v_1^{\mathsf{Keep}}) + \mathsf{Convert}_\mathbb{G}(v_0^{\mathsf{Keep}}) + (-1)^{t_1^{(i-1)}} \cdot V_{CW}$
    $t_{CW}^L \leftarrow t_0^L \oplus t_1^L \oplus \alpha_i \oplus 1$ and $t_{CW}^R \leftarrow t_0^R \oplus t_1^R \oplus \alpha_i$
    $CW^{(i)} \leftarrow s_{CW} || V_{CW} || t_{CW}^L || t_{CW}^R$
    $s_b^{(i)} \leftarrow s_b^{\mathsf{Keep}} \oplus t_b^{(i-1)} \cdot s_{CW}$ for $b = 0, 1$
    $t_b^{(i)} \leftarrow t_b^{\mathsf{Keep}} \oplus t_b^{(i-1)} \cdot t_{CW}^{\mathsf{Keep}}$ for $b = 0, 1$
end for
$CW^{(n+1)} \leftarrow (-1)^{t_1^n} \cdot [\mathsf{Convert}_\mathbb{G}(s_1^{(n)}) - \mathsf{Convert}_\mathbb{G}(s_0^{(n)}) - V_\alpha]$
Let $k_b = s_b^{(0)} || CW^{(1)} || \cdots || CW^{(n+1)}$
return (k_0, k_1)

$\mathsf{Eval}_n^<(b, k_b, x)$:

Parse $k_b = s^{(0)} || CW^{(1)} || \cdots || CW^{(n+1)}$, $x = x_1, \ldots, x_n$, let $V = 0 \in \mathbb{G}$, $t^{(0)} = b$.
for $i = 1$ to n **do**
    Parse $CW^{(i)} = s_{CW} || V_{CW} || t_{CW}^L || t_{CW}^R$
    Parse $G(s^{(i-1)}) = \hat{s}^L || \hat{v}^L || \hat{t}^L \;||\; \hat{s}^R || \hat{v}^R || \hat{t}^R$
    $\tau^{(i)} \leftarrow (\hat{s}^L || \hat{t}^L \;||\; \hat{s}^R || \hat{t}^R) \oplus (t^{(i-1)} \cdot [s_{CW} || t_{CW}^L || s_{CW} || t_{CW}^R])$
    Parse $\tau^{(i)} = s^L || t^L \;||\; s^R || t^R \in \{0,1\}^{2(\lambda+1)}$
    if $x_i = 0$ **then** $V \leftarrow V + (-1)^b \cdot [\mathsf{Convert}_\mathbb{G}(\hat{v}^L) + t^{(i-1)} \cdot V_{CW}]$
        $s^{(i)} \leftarrow s^L$, $t^{(i)} \leftarrow t^L$
    else $V \leftarrow V + (-1)^b \cdot [\mathsf{Convert}_\mathbb{G}(\hat{v}^R) + t^{(i-1)} \cdot V_{CW}]$
        $s^{(i)} \leftarrow s^R$, $t^{(i)} \leftarrow t^R$
    end if
end for
$V \leftarrow V + (-1)^b \cdot [\mathsf{Convert}_\mathbb{G}(s^{(n)}) + t^{(n)} \cdot CW^{(n+1)}]$
Return V

Fig. 1: Optimized FSS scheme for the class $\mathcal{F}_{n,\mathbb{G}}^<$ of comparison functions $f_{\alpha,\beta}^< : \{0,1\}^n \to \mathbb{G}$, outputting β for $0 \leqslant x < \alpha$ and 0 for $x \geqslant \alpha$. $||$ denotes string concatenation. b refers to party id. All s and v values are λ-bit strings, V values are elements in $\mathbb{G}$, which are represented in $\lceil \log |\mathbb{G}| \rceil$ bits and t values are single bits. α_1 and x_1 refer to MSBs of α and x, respectively. Similarly, α_n and x_n are the corresponding LSBs.

[12] that converts a pseudo-random string to a pseudo-random group element. When $|\mathbb{G}| = 2^k$ and $k \leqslant \lambda$, the function simply outputs the first k bits of the input. In any other case, the function expands the input s to a string $G(s)$ of length at least $\log|\mathbb{G}|$ using a PRG G, regards $G(s)$ as an integer and returns $G(s) \bmod |\mathbb{G}|$.

Theorem 2. *Let λ be a security parameter, let $\mathbb{G}$ be an Abelian group, $\ell = \lceil \log|\mathbb{G}| \rceil$, and let $G : \{0,1\}^\lambda \to \{0,1\}^{4\lambda+2}$ be a PRG. The scheme in Fig. 1 is a DCF for $f^{<}_{\alpha,\beta} : \{0,1\}^n \to \mathbb{G}$ with key size $n(\lambda+\ell+2)+\lambda+\ell$ bits. For $\ell' = \lceil \frac{\ell}{4\lambda+2} \rceil$, the key generation algorithm* Gen *invokes G at most $2n(1+2\ell')+2\ell'$ times and the evaluation algorithm* Eval *invokes G at most $n(1+\ell')+\ell'$ times. In the special case that $|\mathbb{G}| = 2^c$ for $c \leqslant \lambda$ the number of PRG invocations in* Gen *is $2n$ and the number of PRG invocations in* Eval *is n.*

Dual Distributed Comparison Function (DDCF). Consider a variant of DCF, called *Dual Distributed Comparison Function*, denoted by $\mathcal{F}^{\mathsf{DDCF}}_{n,\mathbb{G}}$. It is a class of comparison functions $f_{\alpha,\beta_1,\beta_2} : \{0,1\}^n \to \mathbb{G}$, that outputs β_1 for $0 \leqslant x < \alpha$ and β_2 for $x \geqslant \alpha$. The FSS scheme for DDCF, denoted by $\mathsf{DDCF}_{n,\mathbb{G}}$, follows easily from DCF using $f_{\alpha,\beta_1,\beta_2}(x) = \beta_2 + f^{<}_{\alpha,\beta_1-\beta_2}(x)$. We provide a formal construction in Fig. 12 of Appendix F.2 in our full version [9].

4 Public Intervals and Multiple Interval Containments

Computing interval containment for a secret value w.r.t. a publicly known interval, that is, whether $x \in [p,q]$, is an important building block for many tasks occurring in scientific computations [4] as well as machine learning [37, 41, 51]. Moreover, many popular functions such as splines (Section 5.1) and most significant non-zero bit (MSNZB) (Appendix H.1 of full version [9]) reduce to computing multiple interval containments on the same secret value x. The work of [13] provided the first constructions of an FSS gate for interval containment as well as splines. In their work, the key size of an FSS gate for interval containment was ≈ 2 DCF keys. They build on this to construct an FSS gate for splines and multiple interval containment with m different intervals using key size proportional to $2m$ DCF keys, which is quite expensive. We provide the following constructions:

- In Section 4.1, we show how to reduce the key size required for a single interval containment to a *single* DCF key, compared to two DCF keys needed in [13]. Including the gains from our optimized DCF, we get around $7\times$ reduction in key size over [13] for $n = 32$.
- In Section 4.2 of our full version [9], we show how to *compress* the FSS keys for multiple interval containments to that of an FSS key for a *single* interval containment (and ring elements proportional to m). More concretely, over inputs of length n, and for computing the output of m interval containment functions on the same input, we reduce the FSS key size from $\approx 2m(4n\lambda + n^2 + 4n) + mn$ to $\approx n\lambda + n^2 + mn$ (including gains from our optimized DCF

construction). As an example, taking $n = 32$, we reduce the key size by up to 1100× and for instance, for $m = 10$, the reduction is about 62×.

While the construction from [13] also works when the interval boundaries are secret, i.e., known only to the dealer, our techniques crucially rely on the interval boundaries being public. However, we show that our techniques enable the reduction of key size for several important applications, such as splines (Section 5.1), bit decomposition and MSNZB (Section 5.2 and Appendix H.1 of full version [9]).

We start by setting notation for single and multiple interval containments. For ease of exposition, in this section, we only consider the ring $\mathbb{U}_N$; however our ideas easily extend to $\mathbb{S}_N$ as well. In particular, for signed intervals checking whether $x \in [p, q]$, where $p, q \in \mathbb{S}_N$, can be reduced to the following unsigned interval containment: $(x+N/2 \bmod N) \in [(p+N/2 \bmod N), (q+N/2 \mod N)]$. We define $\mathbf{1}\{b\}$ as 1 when b is true and 0 otherwise.

Interval Containment gate. The (single) interval containment gate $\mathcal{G}_{\mathsf{IC}}$ is the family of functions $g_{\mathsf{IC},n,p,q} : \mathbb{U}_N \to \mathbb{U}_N$ parameterized by input and output groups $\mathbb{G}^{\mathsf{in}} = \mathbb{G}^{\mathsf{out}} = \mathbb{U}_N$, and given by

$$\mathcal{G}_{\mathsf{IC}} = \left\{ g_{\mathsf{IC},n,p,q} : \mathbb{U}_N \to \mathbb{U}_N \right\}_{0 \leqslant p \leqslant q \leqslant N-1}, g_{\mathsf{IC},n,p,q}(x) = \mathbf{1}\{p \leqslant x \leqslant q\}.$$

Multiple Interval Containment Gate. The multiple interval containment gate $\mathcal{G}_{\mathsf{MIC}}$ is the family of functions $g_{\mathsf{MIC},n,m,P,Q} : \mathbb{U}_N \to \mathbb{U}_N^m$ for m interval containments parameterized by input and output groups $\mathbb{G}^{\mathsf{in}} = \mathbb{U}_N$ and $\mathbb{G}^{\mathsf{out}} = \mathbb{U}_N^m$, respectively, and for $P = \{p_1, p_2, \ldots, p_m\}$ and $Q = \{q_1, q_2, \ldots, q_m\}$, given by

$$\mathcal{G}_{\mathsf{MIC}} = \left\{ g_{\mathsf{MIC},n,m,P,Q} : \mathbb{U}_N \to \mathbb{U}_N^m \right\}_{0 \leqslant p_i \leqslant q_i \leqslant N-1}, g_{\mathsf{MIC},n,m,P,Q}(x) = \left\{ \mathbf{1}\{p_i \leqslant x \leqslant q_i\} \right\}_{1 \leqslant i \leqslant m},$$

Next, we describe our construction for single interval containment that reduces to universal comparison function $f^{<}_{(N-1)+\mathsf{r}^{\mathsf{in}},1}$ and this is the key idea that allows us to compress keys for multiple interval containments.

4.1 Realizing FSS gate for $[p, q]$ using FSS scheme for $f^{<}_{(N-1)+\mathsf{r}^{\mathsf{in}},1}$

First, in Fig. 2, we describe a construction of an FSS gate for $\mathcal{G}_{\mathsf{IC}}$ that is a slight modification of the construction in [13]. This will enable us to build upon it to obtain an FSS gate for $\mathcal{G}_{\mathsf{IC}}$ with a reduced key size (when the intervals are public). The modification that we make is as follows: in [13], the FSS keys for $\mathcal{G}_{\mathsf{IC}}$ were generated differently in the case when only $q + \mathsf{r}^{\mathsf{in}}$ wraps around in $\mathbb{U}_N$ as opposed to when either both or none of $p + \mathsf{r}^{\mathsf{in}}$ and $q + \mathsf{r}^{\mathsf{in}}$ wrap around. In our construction (Fig. 2), we unify these cases, except that the dealer additionally includes an additive correction term $\mathbf{1}\{(p + \mathsf{r}^{\mathsf{in}} \bmod N) > (q + \mathsf{r}^{\mathsf{in}} \bmod N)\}$ in the key, which makes up for the difference between the cases. For completeness,

we provide a correctness proof in Appendix G.1 of full version [9]. We note that the key size of our construction in Fig. 2 is identical to the scheme presented in [13], that is, 2 DCF keys and a ring element in $\mathbb{U}_N$.

Next, we present an alternate construction of FSS gate for $\mathcal{G}_{\mathsf{IC}}$ again using two DCF keys that are *independent of interval* $[p, q]$. Later, we will optimize this construction to use only a single DCF key.

Interval Containment Gate $(\mathsf{Gen}^{\mathsf{IC}}_{n,p,q}, \mathsf{Eval}^{\mathsf{IC}}_{n,p,q})$

$\mathsf{Gen}^{\mathsf{IC}}_{n,p,q}(1^\lambda, \mathsf{r}^{\mathsf{in}}, \mathsf{r}^{\mathsf{out}})$:

1: $(k_0^{(p)}, k_1^{(p)}) \leftarrow \mathsf{Gen}^{<}_n(1^\lambda, \alpha^{(p)}, N-1, \mathbb{U}_N)$, $\alpha^{(p)} = p + \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$.
2: $(k_0^{(q)}, k_1^{(q)}) \leftarrow \mathsf{Gen}^{\leqslant}_n(1^\lambda, \alpha^{(q)}, 1, \mathbb{U}_N)$, $\alpha^{(q)} = q + \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$.
3: Sample random $w_0, w_1 \leftarrow \mathbb{U}_N$ s.t. $w_0 + w_1 = \mathsf{r}^{\mathsf{out}} + \mathbf{1}\{\alpha^{(p)} > \alpha^{(q)}\}$.
4: For $b \in \{0,1\}$, let $k_b = k_b^{(p)} || k_b^{(q)} || w_b$.
5: **return** (k_0, k_1).

$\mathsf{Eval}^{\mathsf{IC}}_{n,p,q}(b, k_b, x)$:

1: Parse $k_b = k_b^{(p)} || k_b^{(q)} || w_b$.
2: Set $t_b^{(p)} \leftarrow \mathsf{Eval}^{<}_n(b, k_b^{(p)}, x)$.
3: Set $t_b^{(q)} \leftarrow \mathsf{Eval}^{\leqslant}_n(b, k_b^{(q)}, x)$.
4: **return** $t_b^{(p)} + t_b^{(q)} + w_b$.

Fig. 2: FSS Gate for $\mathcal{G}_{\mathsf{IC}}$ using 2 DCFs [13], b refers to party id.

Using 2 DCF keys independent of p and q. Below, we state our main technical lemma that allows us to give an alternate construction of FSS gate for $g_{\mathsf{IC},n,p,q}$ using 2 keys for comparison that are *independent of the interval* $[p, q]$ and only depend on r^{in}. More concretely, we will use FSS keys for $f^{<}_{(N-1)+\mathsf{r}^{\mathsf{in}},N-1}$ and $f^{\leqslant}_{(N-1)+\mathsf{r}^{\mathsf{in}},1}$. In the lemma statement and its proof (Appendix G.2 of full version [9]), unless explicitly stated using mod N, all expressions and equations are over $\mathbb{Z}$ and we consider the natural embedding of $\mathbb{U}_N$ into $\mathbb{Z}$.

Lemma 1. *Let $a, \tilde{a}, b, \tilde{b}, r \in \mathbb{U}_N$, where $a \leqslant b$, $\tilde{a} = a + r \bmod N$ and $\tilde{b} = b + r \bmod N$. Define 4 boolean predicates over $\mathbb{U}_N \to \{0,1\}$ as follows: $P(x)$ denotes $x < \tilde{a}$, $P'(x)$ denotes $x \leqslant \tilde{a}$, $Q(x)$ denotes $(x + (b-a) \bmod N) < \tilde{b}$, $Q'(x)$ denotes $(x + (b-a) \bmod N) \leqslant \tilde{b}$. Then, the following holds:*

$$P(x) = Q(x) + (e_a - e_x) \text{ and } P'(x) = Q'(x) + (e_a - e_x)$$

$$\text{where } e_a = \mathbf{1}\{\tilde{a} + (b-a) > N-1\} \text{ and } e_x = \mathbf{1}\{x + (b-a) > N-1\}$$

Intuitively, Lemma 1 allows us to reduce comparison of x with $\tilde{a}$ (both $<$ and $\leqslant$) to similar comparison with $\tilde{b}$ modulo some additive correction terms, i.e. e_a and e_x. Our next observation is that in the FSS setting, e_a can be computed by the dealer (with the knowledge of r) and e_x can be locally computed by

Interval Containment Gate $(\mathsf{Gen}^{\mathsf{IC}}_{n,p,q}, \mathsf{Eval}^{\mathsf{IC}}_{n,p,q})$

$\mathsf{Gen}^{\mathsf{IC}}_{n,p,q}(1^\lambda, \mathsf{r}^{\mathsf{in}}, \mathsf{r}^{\mathsf{out}})$:

1: Set $\gamma = (N-1) + \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$.
2: $(k_0^{(N-1)}, k_1^{(N-1)}) \leftarrow \mathsf{Gen}^{<}_n(1^\lambda, \gamma, 1, \mathbb{U}_N)$.
3: Set $q' = q+1 \in \mathbb{U}_N$, $\alpha^{(p)} = p + \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$, $\alpha^{(q)} = q + \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$ and $\alpha^{(q')} = q + 1 + \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$.
4: Sample random $z_0, z_1 \leftarrow \mathbb{U}_N$ s.t. $z_0 + z_1 = \mathsf{r}^{\mathsf{out}} + \mathbf{1}\{\alpha^{(p)} > \alpha^{(q)}\} - \mathbf{1}\{\alpha^{(p)} > p\} + \mathbf{1}\{\alpha^{(q')} > q'\} + \mathbf{1}\{\alpha^{(q)} = N-1\}$.
5: For $b \in \{0,1\}$, let $k_b = k_b^{(N-1)} || z_b$.
6: **return** (k_0, k_1).

$\mathsf{Eval}^{\mathsf{IC}}_{n,p,q}(b, k_b, x)$:

1: Parse $k_b = k_b^{(N-1)} || z_b$.
2: Set $q' = q+1 \in \mathbb{U}_N$, $x^{(p)} = x + (N-1-p) \in \mathbb{U}_N$ and $x^{(q')} = x + (N-1-q') \in \mathbb{U}_N$.
3: Set $s_b^{(p)} \leftarrow \mathsf{Eval}^{<}_n(b, k_b^{(N-1)}, x^{(p)})$.
4: Set $s_b^{(q')} \leftarrow \mathsf{Eval}^{<}_n(b, k_b^{(N-1)}, x^{(q')})$.
5: **return** $y_b = b \cdot (\mathbf{1}\{x > p\} - \mathbf{1}\{x > q'\}) - s_b^{(p)} + s_b^{(q')} + z_b$.

Fig. 3: FSS Gate for $\mathcal{G}_{\mathsf{IC}}$ using DCF key for $f^{<}_{(N-1)+\mathsf{r}^{\mathsf{in}},1}$, b refers to party id.

P_0, P_1 (with the knowledge of x at runtime). Using Lemma 1 and this observation, we can construct an FSS gate for $g_{\mathsf{IC},n,p,q}$ using 2 DCF keys, for functions $f^{<}_{(N-1)+\mathsf{r}^{\mathsf{in}},N-1}$ and $f^{\leqslant}_{(N-1)+\mathsf{r}^{\mathsf{in}},1}$ (see Appendix G.4 of full version [9] for this).

Reducing to 1 DCF key. We now optimize the key size of our construction to a single DCF key using Lemma 2 (proof in Appendix G.3 of full version).

Lemma 2. *Let $c, c' \in \mathbb{U}_N$, where $c' = c+1 \bmod N$. Define 2 boolean predicates over $\mathbb{U}_N \to \{0,1\}$ as follows: $R(x)$ denotes $x \leqslant c$ and $S(x)$ denotes $x < c'$. Then the following holds: $R(x) = S(x) + \mathbf{1}\{c = N-1\}$*

This lemma lets us get rid of the DCF key for $f^{\leqslant}_{(N-1)+\mathsf{r}^{\mathsf{in}},1}$ and work with the key for $f^{<}_{(N-1)+\mathsf{r}^{\mathsf{in}},1}$ using an additional correction term which can be computed by the dealer. Formally, we have the following theorem.

Theorem 3. *There is an FSS Gate $(\mathsf{Gen}^{\mathsf{IC}}_{n,p,q}, \mathsf{Eval}^{\mathsf{IC}}_{n,p,q})$ for $\mathcal{G}_{\mathsf{IC}}$ that requires 2 invocations of $\mathsf{DCF}_{n,\mathbb{U}_N}$, and has a total key size of n bits plus key size of $\mathsf{DCF}_{n,\mathbb{U}_N}$.*

Proof. We present our construction formally in Fig. 3. For arguing correctness we need to prove that $y = y_0 + y_1 \bmod N = \mathbf{1}\{p \leqslant (x - \mathsf{r}^{\mathsf{in}} \bmod N) \leqslant q\} + \mathsf{r}^{\mathsf{out}}$. We use correctness of FSS gate in Fig. 2 and prove that output of Fig. 3 is identical to output of Fig. 2. In Fig. 2, using correctness of FSS schemes for $f^{<}_{\alpha,\beta}$ and $f^{\leqslant}_{\alpha,\beta}$,

$$t^{(p)} = t_0^{(p)} + t_1^{(p)} \bmod N = -1 \cdot \mathbf{1}\{x < \alpha^{(p)}\} \text{ and}$$
$$t^{(q)} = t_0^{(q)} + t_1^{(q)} \bmod N = \mathbf{1}\{x \leqslant \alpha^{(q)}\}$$

Also, from correctness of FSS gate in Fig. 2, $t^{(p)} + t^{(q)} + \mathbf{1}\{\alpha^{(p)} > \alpha^{(q)}\} + \mathsf{r}^{\mathsf{out}} = \mathbf{1}\{p \leqslant (x - \mathsf{r}^{\mathsf{in}} \bmod N) \leqslant q\} + \mathsf{r}^{\mathsf{out}}$.

First, we look at $t^{(q)} = \mathbf{1}\{x \leqslant \alpha^{(q)}\}$. From Lemma 2, we can write $t^{(q)} = \mathbf{1}\{x < \alpha^{(q')}\} + \mathbf{1}\{\alpha^{(q)} = N-1\}$, where $\alpha^{(q')} = \alpha^{(q)} + 1 \bmod N$. Now, using Lemma 1 with $a = q'$, $b = N-1$, $r = \mathsf{r}^{\mathsf{in}}$, $\tilde{a} = \alpha^{(q')}$, and $\tilde{b} = \gamma$:

$$\begin{aligned}
t^{(q)} &= \mathbf{1}\{x < \alpha^{(q')}\} + \mathbf{1}\{\alpha^{(q)} = N-1\} \\
&= \mathbf{1}\{x + (N-1-q') \bmod N < \gamma\} + \mathbf{1}\{\alpha^{(q')} + (N-1-q') > (N-1)\} \\
&\quad - \mathbf{1}\{x + (N-1-q') > (N-1)\} + \mathbf{1}\{\alpha^{(q)} = N-1\} \\
&= \mathbf{1}\{x^{(q')} < \gamma\} + \mathbf{1}\{\alpha^{(q')} > q'\} - \mathbf{1}\{x > q'\} + \mathbf{1}\{\alpha^{(q)} = N-1\} \\
&= s_0^{(q')} + s_1^{(q')} + \mathbf{1}\{\alpha^{(q')} > q'\} - \mathbf{1}\{x > q'\} + \mathbf{1}\{\alpha^{(q)} = N-1\}
\end{aligned}$$

Similarly, using Lemma 1, it can be proven that: $t^{(p)} = -1 \cdot (s_0^{(p)} + s_1^{(p)}) - \mathbf{1}\{\alpha^{(p)} > p\} + \mathbf{1}\{x > p\}$. Therefore, in Fig. 3, $y = y_0 + y_1 = t^{(p)} + t^{(q)} + \mathbf{1}\{\alpha^{(p)} > \alpha^{(q)}\} + \mathsf{r}^{\mathsf{out}}$ matches the output of Fig. 2.

5 Applications of Public Intervals

5.1 Splines with Public Intervals

A spline is a special function defined piecewise by polynomials. Formally, consider $P = \{p_i\}_i \in \mathbb{U}_N^m$ such that $0 \leqslant p_1 < p_2 < \ldots < p_{m-1} < p_m$ ($p_m = N-1$) and $d-degree$ univariate polynomials $F = \{f_i\}_i$. Then, a spline function $h_{n,m,d,P,F} : \mathbb{U}_N \to \mathbb{U}_N$ parameterized by input and output rings $\mathbb{U}_N$, list of m interval boundaries P and degree d polynomials F is defined as

$$h_{n,m,d,P,F}(x) = \begin{cases} f_1(x) & \text{if } x \in [0, p_1] \\ f_2(x) & \text{if } x \in [p_1 + 1, p_2] \\ & \vdots \\ f_m(x) & \text{if } x \in [p_{m-1} + 1, p_m] \end{cases}$$

Commonly used functions such as Rectified Linear Unit (ReLU) and Absolute value are special cases of splines. Moreover, splines have been used to approximate transcendental functions such as sigmoid [38, 41], sometimes with up to $m = 12$ intervals. Boyle *et al.* [13], gave a construction of an FSS gate for splines by reducing it to m instances of interval containment, resulting in both key size and online evaluation cost being proportional to the cost of $2m$ DCF keys. In this work, building upon our techniques for multiple interval containment[11], we reduce both the key size as well as online evaluation. More concretely, [13] requires $2m$ $\mathsf{DCF}_{n,\mathbb{Z}_N^{d+1}}$ keys and each key is evaluated once during online phase. We

[11] As we explain later, our FSS gate for splines requires secret payload (function of r^{in}) in DCF known only to the dealer and hence, it does not black-box reduce to $\mathcal{G}_{\mathsf{MIC}}$.

provide a construction using a *single* $\mathsf{DCF}_{n,\mathbb{Z}_N^{(d+1)m}}$ key that is evaluated m times and additional $2m(d+1)+1$ ring elements. Hence, including our improved DCF construction, we reduce the overall key size from $\approx 2m\left(4n(\lambda+1)+n^2(d+1)\right)$ to $\approx \left(\lambda(n+1)+mn^2(d+1)\right)+2mn(d+1)$ bits. As an example, for $n=32$, $m \geqslant 2$ and degree 1 polynomials, this represents a reduction in key size of about $8-17\times$, and for instance, for $m=10$, the reduction is $14\times$.

The spline gate $\mathcal{G}_{\mathsf{spline}}$ is the family of functions $g_{\mathsf{spline},n,m,d,P,F} : \mathbb{U}_N \to \mathbb{U}_N$ with m intervals parameterized by input and output rings $\mathbb{U}_N$, and for $P = \{p_1, p_2, \ldots, p_m\}$ and $F = \{f_1, f_2, \ldots, f_m\}$, given by

$$\mathcal{G}_{\mathsf{spline}} = \left\{ g_{\mathsf{spline},n,m,d,P,F} : \mathbb{U}_N \to \mathbb{U}_N \right\}_{\substack{0 \leqslant p_i < p_{i+1} \leqslant N-1, \\ p_0 = p_m = N-1}} g_{\mathsf{spline},n,m,d,P,F}(x) = h_{n,m,d,P,F}(x).$$

Construction Overview. Our FSS gate for splines builds upon our techniques from multiple interval containment to incorporate secret payloads as required. At a high level, the basic idea, also used in [13], is to check for interval containment $[p_{i-1}+1, p_i]$ and output the coefficients of the polynomial $f_i' = f_i(x - \mathsf{r}^{\mathsf{in}})$ as payload. Once the evaluators P_0 and P_1 learn the shares of the correct coefficients, they compute an inner product with $(x^d, \ldots, x^0)$ to learn shares of final output. We note that coefficients of f_i' depend on the randomness r^{in} that is secret and known only to the dealer. Due to this, we cannot invoke our FSS gate for multiple interval containment $\mathcal{G}_{\mathsf{MIC}}$ directly. Next, [13] used a different interval containment key for each interval with payload as the corresponding coefficients of the polynomials. In our construction, we only use a single DCF key for all intervals, and hence, the payload of this key has to encode the coefficients of all the polynomials. Moreover, naively building on $\mathcal{G}_{\mathsf{MIC}}$, the online computation would require $2m$ evaluations of the DCF key. However, for the case of splines, we use the property that the intervals are consecutive, that is, of the form $[p_{i-1}+1, p_i]$, to reduce this to m evaluations.

We present our final construction in 2 steps. First, we present the construction for a simpler spline gate, $\mathcal{G}_{\mathsf{spline\text{-}one}}$ that is a family of functions $h_{n,d,p,f}$ with only 1 interesting interval i.e., it outputs $f(x)$ on $[0,p]$ and 0 otherwise. With this construction, we describe our techniques for embedding secret payloads in our optimized FSS gate for $\mathcal{G}_{\mathsf{IC}}$ that uses a single DCF key. Note that ReLU function, the most commonly used activation in machine learning, is a function in $\mathcal{G}_{\mathsf{spline\text{-}one}}$. We discuss about ReLU and absolute value function in the full version of this paper [9]. Then, we will give our construction for general splines using our ideas of common payload for all intervals and reducing number of DCF evaluations.

Spline with one interesting interval. The simple spline gate $\mathcal{G}_{\mathsf{spline\text{-}one}}$ is a family of functions $h_{n,d,p,f} : \mathbb{U}_N \to \mathbb{U}_N$ such that $p \in \mathbb{U}_N$, f is a d-degree univariate polynomial and $h_{n,d,p,f}(x) = f(x)$ for $x \in [0,p]$ and 0 otherwise. We give a formal construction for FSS gate for $\mathcal{G}_{\mathsf{spline\text{-}one}}$ in Fig. 4. At a high level, we build on our construction for $\mathcal{G}_{\mathsf{IC}}$ and modify it to allow for secret payloads as follows: Recall that in FSS gate for $\mathcal{G}_{\mathsf{IC}}$, we give out a DCF key with payload 1 and shares of a correction term that depends on r^{in}, say c_r. Also,

during evaluation, P_0, P_1 compute a correction term, say c_x, that depends on x. Overall, at the time of evaluation, P_0, P_1 evaluate the DCF key and add c_r and c_x. Now we desire the payload to be coefficients of $f' = f(x - \mathsf{r}^{\mathsf{in}})$, say β. To enable this, the dealer sets the payload of the DCF key as β. But now, this β also needs to be multiplied with c_r and c_x. For this the dealer gives out shares of $c_r \cdot \beta$ and shares of β. Shares of β allow P_0 and P_1 to compute shares of $c_x \cdot \beta$, as c_x can be computed locally.

Spline Gate $(\mathsf{Gen}^{\mathsf{spline\text{-}one}}_{n,d,p}, \mathsf{Eval}^{\mathsf{spline\text{-}one}}_{n,d,p})$

$\mathsf{Gen}^{\mathsf{spline\text{-}one}}_{n,d,p}(1^\lambda, f, \mathsf{r}^{\mathsf{in}}, \mathsf{r}^{\mathsf{out}})$:

1: Let $(f'_d, \ldots, f'_0) \in \mathbb{U}_N^{(d+1)}$ be coefficients of f' such that $f'(x) = f(x - \mathsf{r}^{\mathsf{in}})$.
2: Set $\beta = (f'_d, \ldots, f'_0) \in \mathbb{U}_N^{(d+1)}$ and $\gamma = (N-1) + \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$.
3: $(k_0^{(N-1)}, k_1^{(N-1)}) \leftarrow \mathsf{Gen}_n^<(1^\lambda, \gamma, \beta, \mathbb{U}_N^{(d+1)})$.
4: Set $\alpha^{(L)} = \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$, $\alpha^{(R)} = p + \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$ and $\alpha^{(R')} = p + 1 + \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$.
5: Set
$c_r = \mathbf{1}\{\alpha^{(L)} > \alpha^{(R)}\} - \mathbf{1}\{\alpha^{(L)} > 0\} + \mathbf{1}\{\alpha^{(R')} > (p+1 \bmod N)\} + \mathbf{1}\{\alpha^{(R)} = N-1\}$.
6: Sample random $e_0, e_1 \leftarrow \mathbb{U}_N^{(d+1)}$ s.t. $e_0 + e_1 = c_r \cdot \beta$.
7: Sample random $\beta_0, \beta_1 \leftarrow \mathbb{U}_N^{(d+1)}$ s.t. $\beta_0 + \beta_1 = \beta$.
8: Sample random $r_0, r_1 \leftarrow \mathbb{U}_N$ s.t. $r_0 + r_1 = \mathsf{r}^{\mathsf{out}}$.
9: For $b \in \{0,1\}$, let $k_b = k_b^{(N-1)} || e_b || \beta_b || r_b$.
10: **return** (k_0, k_1).

$\mathsf{Eval}^{\mathsf{spline\text{-}one}}_{n,d,p}(b, k_b, x)$:

1: Parse $k_b = k_b^{(N-1)} || e_b || \beta_b || r_b$.
2: Set $x^{(L)} = x + (N-1) \in \mathbb{U}_N$ and $x^{(R')} = x + (N - 1 - (p+1)) \in \mathbb{U}_N$.
3: Set $s_b^{(L)} \leftarrow \mathsf{Eval}_n^<(b, k_b^{(N-1)}, x^{(L)})$.
4: Set $s_b^{(R')} \leftarrow \mathsf{Eval}_n^<(b, k_b^{(N-1)}, x^{(R')})$.
5: Set $c_x = (\mathbf{1}\{x > 0\} - \mathbf{1}\{x > (p+1 \bmod N)\})$.
6: $w_b = (w_{d,b}, \ldots, w_{0,b}) = c_x \cdot \beta_b - s_b^{(L)} + s_b^{(R')} + e_b$.
7: **return** $u_b = r_b + \sum_{i=0}^{d} (w_{i,b} \cdot x^i) \bmod N$.

Fig. 4: FSS Gate for single interval splines $\mathcal{G}_{\mathsf{spline\text{-}one}}$, b refers to party id.

Theorem 4. *There is an FSS Gate* $(\mathsf{Gen}^{\mathsf{spline\text{-}one}}_{n,d,p}, \mathsf{Eval}^{\mathsf{spline\text{-}one}}_{n,d,p})$ *for* $\mathcal{G}_{\mathsf{spline\text{-}one}}$ *that requires* 2 *invocations of* $\mathsf{DCF}_{n,\mathbb{U}_N^{(d+1)}}$*, and has a total key size of* $n(2d+3)$ *bits plus the key size of* $\mathsf{DCF}_{n,\mathbb{U}_N^{(d+1)}}$.

Proof. We present our construction of FSS Gate for single interval spline formally in Fig. 4. To prove correctness of our scheme it suffices to show that $w = w_0 + w_1$ is β when $(x - \mathsf{r}^{\mathsf{in}}) \in [0, p]$ and 0^{d+1} otherwise. In our scheme, $w = \sum_b (c_x \cdot \beta_b - s_b^{(L)} + s_b^{(R')} + e_b) = c_x \cdot \beta - s^{(L)} + s^{(R')} + c_r \cdot \beta$. Now, by correctness of DCF keys, $s^{(L)} = \beta \cdot \mathbf{1}\{x^{(L)} < \gamma\}$ and $s^{(R')} = \beta \cdot \mathbf{1}\{x^{(R')} < \gamma\}$. Using these, we get

that $w = \Big(c_x - \mathbf{1}\{x^{(L)} < \gamma\} + \mathbf{1}\{x^{(R')} < \gamma\} + c_r\Big) \cdot \beta = \mathbf{1}\{0 \leqslant (x - \mathsf{r}^{\mathsf{in}}) \leqslant p\} \cdot \beta$ as required, by using similar arguments as in correctness of $\mathcal{G}_{\mathsf{IC}}$ in Fig. 3.

General Splines. To construct an FSS gate for general splines, we make two modifications to the previous construction. First, we change the payload of our DCF key to be the long vector containing coefficients of all polynomials $\{f'_i\}_i$, where $f'_i = f(x - \mathsf{r}^{\mathsf{in}})$. Now, during evaluation, we do DCF evaluations similar to $\mathcal{G}_{\mathsf{MIC}}$ separately for each interval. For each interval, output would be over $\mathbb{U}_N^{m(d+1)}$. While considering the i^{th} interval, i.e., $[p_{i-1}+1, p_i]$, we will only use the i^{th} segment of $(d+1)$ ring elements. These would either be shares of coefficients of f'_i (if $(x - \mathsf{r}^{\mathsf{in}}) \in [p_{i-1}+1, p_i]$) or 0^{d+1}. Next, to reduce number of evaluations from $2m$ to m, we rely on intervals in splines being consecutive, i.e., an interval ends at p_i and next interval starts at $p_i + 1$. Recall from our construction of $\mathcal{G}_{\mathsf{MIC}}$, that we need to do two DCF evaluations for each interval of interest, one for the left point and one for the right point. This is also true for Fig. 4, where we do one DCF evaluation each for $x^{(L)}$ and $x^{(R')}$. In general splines, for the i^{th} interval $[p_{i-1}+1, p_i]$, let these points be $x_i^{(L)}$ and $x_i^{(R')}$. Now, observe that since $x_i^{(R')} = x_{i+1}^{(L)}$, we need to evaluate the DCF only once for them. For consistency of notation, we set $p_0 = p_m = N - 1$, so that the first interval, i.e., $[0, p_1]$ can also be written as $[p_0+1, p_1]$ and similarly the last interval, i.e., $[p_{m-1}+1, N-1]$ can be written as $[p_{m-1}+1, p_m]$. In our construction, we do DCF evaluations for all points $x_i = x_i^{(L)} = x + (N - 1 - (p_{i-1} + 1))$ for $i \in \{1, \ldots, m\}$.

Theorem 5. *There is an FSS Gate* $(\mathsf{Gen}^{\mathsf{spline}}_{n,m,d,\{p_i\}_i}, \mathsf{Eval}^{\mathsf{spline}}_{n,m,d,\{p_i\}_i})$ *for* $\mathcal{G}_{\mathsf{spline}}$ *that requires* m *invocations of* $\mathsf{DCF}_{n,\mathbb{U}_N^{m(d+1)}}$*, and has a total key size of* $2mn(d+1) + n$ *bits plus the key size of* $\mathsf{DCF}_{n,\mathbb{U}_N^{m(d+1)}}$.

We provide our scheme and its proof formally in the full version [9].

6 FSS Gates for Fixed-point Arithmetic

Fixed-point representation allows us to embed rational numbers into fixed bit-width integers. Let $\mathbb{Q}^{\mathsf{u}}$ denote non-negative rational numbers. Assuming no overflows, the unsigned (resp. signed) forward mapping $f^{\mathsf{ufix}}_{n,s} : \mathbb{Q}^{\mathsf{u}} \to \mathbb{U}_N$ (resp. $f^{\mathsf{sfix}}_{n,s} : \mathbb{Q} \to \mathbb{S}_N$) is defined by $\lfloor x \cdot 2^s \rfloor$ and the reverse mapping $h^{\mathsf{ufix}}_{n,s} : \mathbb{U}_N \to \mathbb{Q}^{\mathsf{u}}$ (resp. $h^{\mathsf{sfix}}_{n,s} : \mathbb{S}_N \to \mathbb{Q}$) is defined by $x/2^s$, where x is lifted to $\mathbb{Q}$ and "/" denotes the regular division over $\mathbb{Q}$. The value s associated with a fixed-point representation is called the "scale" which defines the precision, i.e., the number of bits after the decimal point, that the fixed-point number preserves. When 2 fixed-point numbers are added or multiplied in n-bit integer ring, the bits at the top (significant bits) can overflow leading to incorrect results. To prevent this from happening, these operations are accompanied by a "scale adjustment" step where the scale of operands are appropriately reduced to create enough room in

the top bits for the computation to fit. Scale adjustment is also used in multiplication to maintain the scale of the output at s instead of getting doubled for every multiplication performed. Many applications of secure computation require computing over the rational numbers. One such application is privacy-preserving machine learning where most prior works use fixed-point representation to deal with rational numbers [33, 37, 39–41, 46, 51][12].

In this section we build efficient FSS gates for realizing secure fixed-point arithmetic. In particular, we consider the following operations: addition, multiplication, and comparison. We begin (in Section 6.1) by first describing how fixed-point addition and multiplication work given access to a FSS gates for secure right shift operations. We then describe the FSS gate constructions for right shift operator - logical right shift (LRS) in Section 6.2, which enables scale adjustment, and hence fixed-point multiplication, over unsigned integers. We defer the details on arithmetic right shift (ARS) and fixed-point comparison to Sections 6.3 and 6.4 respectively of our full version [9].

6.1 Fixed-point Addition and Multiplication

We describe the case when the scales of both operands is the same, i.e. s - the case of different scales is similar[14]. Fixed-point addition is a local operation where the corresponding shares of the operands are added together by each party and no scale adjustment is typically performed. This is same as the construction of FSS gate for addition from [13] as described in Fig. 17, Appendix I.1 (of full version [9]). Fixed-point multiplication involves 2 steps: first, using the FSS gate for multiplication from [13] (presented in Fig. 18, Appendix I.2 of full version for completeness) the operands are multiplied resulting in an output of scale $2s$, and second, using our FSS gate for right shift, values are shifted (ARS/LRS for signed/unsigned operands respectively) by s to reduce the scale back to s.

6.2 Logical Right Shift

Logical right shift of unsigned integers is done by shifting the integer by a prescribed number of bits to the right while removing the low-order bits and inserting zeros as the high order bits. Implementing the shift operation on secret shared values is a nontrivial task even when the shift s is public, and is typically achieved via an expensive secure bit-decomposition operation. Prior FSS gate for bit-decomposition [13] output shares of bits in $\mathbb{U}_2$ (which must then be converted into shares over $\mathbb{U}_N$, if it is to be used in computing logical right shift).

[12] Although there are a handful of works outside the secure ML context that give secure protocols directly for floating-point numbers [13] [3, 24, 34, 45], they are usually orders of magnitude slower than the ones based on fixed-point.

[14] When scales of the operands differ, they need to be aligned before addition can happen. For this, a common practice is to left shift (locally) the operand with smaller scale by the difference of the scales. Fixed-point multiplication remains the same and shift parameter for the right shift at the end can be chosen depending on the scale required for the output.

Hence, this leads to construction for right shift that has 2 online rounds. Here we provide a much more efficient construction, which a) requires only 1 online round of communication of a single group element; and b) further, improves upon the key size of the approach based on bit-decomposition, by roughly a factor of n (when $n \leqslant \lambda$), i.e. $O(n\lambda + n^2)$ vs $O(n^2\lambda)$.

If an integer $x \in \mathbb{U}_N$ ($N = 2^n$) is additively shared into $x \equiv x_0 + x_1 \bmod N$ with one party holding x_0 and the other holding x_1 then locally shifting x_0 and x_1 by s bits is not sufficient to additively share a logically shifted x. Lemma 3 (proof appears in Appendix I.3 of full version [9]) gives an identity showing that the LRS of a secret shared x can be computed as the sum of the LRS of the shares and the output of two comparison functions. This identity is the basis for an FSS gate realizing the offset family associated with LRS.

Notation. *Given integers $0 < n, 0 \leqslant s \leqslant n$, let $(\gg_L s) : \mathbb{U}_N \to \mathbb{U}_N, 0 \leqslant s \leqslant n$ be the* logical right shift *function with action on input x denoted by $(\gg_L s)(x) = (x \gg_L s)$ and defined by $(x \gg_L s) = \frac{x - (x \bmod 2^s)}{2^s}$ over $\mathbb{Z}$.*

Lemma 3. *For any integers $0 < n, 0 \leqslant s \leqslant n$, any $x \in \mathbb{U}_N$ and any $x_0, x_1 \in \mathbb{U}_N$ such that $x_0 + x_1 \equiv x \bmod N$, the following holds over $\mathbb{Z}$ (and in particular over $\mathbb{U}_N$) $(x \gg_L s) = (x_0 \gg_L s) + (x_1 \gg_L s) + t^{(s)} - 2^{n-s} \cdot t^{(n)}$, where for any $0 \leqslant i \leqslant n$, $t^{(i)}$ is defined by:*

$$t^{(i)} = \begin{cases} 1 & (x_0 \bmod 2^i) + (x_1 \bmod 2^i) > 2^i - 1 \\ 0 & \textit{otherwise} \end{cases},$$

The logical right-shift gate $\mathcal{G}_{\gg_L}$ is the family of functions $g_{\gg_L,s,n} : \mathbb{U}_N \to \mathbb{U}_N$ parameterized by input/output groups $\mathbb{G}^{\mathsf{in}} = \mathbb{G}^{\mathsf{out}} = \mathbb{U}_N$, shift s and given by

$$\mathcal{G}_{\gg_L} = \left\{ g_{\gg_L,s,n} : \mathbb{U}_N \to \mathbb{U}_N \right\}_{0 \leqslant s \leqslant n}, g_{\gg_L,s,n}(x) = (x \gg_L s).$$

We denote the corresponding offset gate class by $\hat{\mathcal{G}}_{\gg_L}$ and the offset functions by $\hat{g}^{[\mathsf{r}^{\mathsf{in}}, \mathsf{r}^{\mathsf{out}}]}_{\gg_L,s,n}(x) = g_{\gg_L,s,n}(x - \mathsf{r}^{\mathsf{in}}) + \mathsf{r}^{\mathsf{out}} = ((x - \mathsf{r}^{\mathsf{in}}) \gg_L s) + \mathsf{r}^{\mathsf{out}}$. We use Lemma 3 to construct our FSS gate for LRS (as described in Fig. 6 of full version [9]) and which satisfies the following theorem.

Theorem 6 (LRS from DCF). *There is an FSS Gate $(\mathsf{Gen}^{\gg_L}_{n,s}, \mathsf{Eval}^{\gg_L}_{n,s})$ for $\mathcal{G}_{\gg_L}$ that requires a single invocation each of $\mathsf{DCF}_{n,\mathbb{U}_N}$ and $\mathsf{DCF}_{s,\mathbb{U}_N}$, and has a total key size of n bits plus the key sizes of $\mathsf{DCF}_{n,\mathbb{U}_N}$ and $\mathsf{DCF}_{s,\mathbb{U}_N}$.*

7 FSS Barrier for Fixed-Point Multiplication

In the previous sections, we presented FSS gates for several fixed-point operations, enabling secure computation of fixed-point multiplication $\mathcal{F}_{\mathsf{FPM}}$ with "FSS depth 2": namely, one FSS gate for performing multiplication of the two integer inputs over $\mathbb{U}_N$ (resp. $\mathbb{S}_N$), followed by a second FSS gate to perform a logical

right shift (resp. arithmetic right shift). While this provides an effective solution, a downside of two sequential FSS gates is that the resulting secure computation protocol requires information communicated between parties via two sequential rounds, and a natural goal would be to construct a *single* FSS gate to perform both steps of the fixed-point multiplication together. Such a single FSS gate would not only lead to optimal round complexity (one instead of two rounds), but also to optimal online communication complexity (a factor-2 improvement over the current implementation). In this section, we demonstrate a barrier toward achieving this goal using only symmetric-key cryptography.

More specifically, we show that the existence of any FSS gate construction for fixed-point multiplication, denoted by $\mathcal{G}_{\mathsf{uFPM}}$ (resp. $\mathcal{G}_{\mathsf{sFPM}}$) for operation over unsigned (resp. signed) integers, (with polynomial key size) directly implies the existence of FSS scheme for the class of all bitwise conjunction formulas (with polynomial key size), from the same underlying assumptions. As discussed below, FSS schemes for conjunctions from symmetric-key primitives have remained elusive despite significant research effort. As such, this constitutes a barrier toward symmetric-key constructions for fixed-point multiplication.

FSS for conjunctions. We will denote by $\mathcal{F}^{\wedge}_{n,\mathbb{U}_N}$ the collection of bit-conjunction functions on n-bit inputs, each parameterized by a subset $S \subseteq [n]$, where $[n] = \{i \mid (0 \leqslant i \leqslant n-1) \wedge (i \in \mathbb{Z})\}$), of input bits, evaluating to a given nonzero value if the corresponding input bits are all 1.

Definition 5. *The family $\mathcal{F}^{\wedge}_{n,\mathbb{U}_N}$ of conjunction functions is*

$$\mathcal{F}^{\wedge}_{n,\mathbb{U}_N} = \left\{ f_S : \{0,1\}^n \to \mathbb{U}_N \right\}_{S \subseteq [n]}, \text{ where } f_S(x) = \begin{cases} \beta & \bigwedge_{i \in S} x_{[i]} = 1 \\ 0 & \text{otherwise} \end{cases}.$$

Presently the only existing construction of FSS scheme for $\mathcal{F}^{\wedge}_{n,\mathbb{U}_N}$ with negligible correctness error relies on the Learning With Errors (LWE) assumption [14, 26]. A construction with inverse-polynomial correctness error can be obtained from the Decisional Diffie-Hellman (DDH) assumption [11] or from the Paillier assumption [28]. All assumptions are specific structured assumptions, and corresponding constructions require heavy public-key cryptographic machinery. It remains a highly motivated open question to attain such an FSS construction using only symmetric-key cryptography, even in the case when payload β is public.

Open Question (FSS for conjunctions)**.** *Construct FSS scheme for the class $\mathcal{F}^{\wedge}_{n,\mathbb{U}_N}$ of bit-conjunction functions (with key size polynomial in the security parameter and input length n) based on symmetric-key cryptographic primitives.*

The barrier result. We prove the desired barrier result via an intermediate function family: $\mathcal{F}^{\times\mathsf{MSB}}_{\eta,\mathbb{U}_N}$, a simplified version of fixed-point multiplication.

Definition 6. *The family $\mathcal{F}^{\times\mathsf{MSB}}_{\eta,\mathbb{U}_N}$ of multiply-then-MSB functions is given by*

$$\mathcal{F}^{\times\mathsf{MSB}}_{\eta,\mathbb{U}_N} = \left\{ f_c : \mathbb{U}_{2^\eta} \to \mathbb{U}_N \right\}_{c \in \mathbb{U}_{2^\eta}}, \text{ where } f_c(x) = \mathsf{MSB}(c \cdot x),$$

and where $n \leqslant \eta$ and $c \cdot x$ is multiplication over $\mathbb{U}_{2^\eta}$.

The description of a function f_c above is assumed to explicitly contain a description of the respective parameter $c \in \mathbb{U}_{2^\eta}$ (similarly for $f_S \in \mathcal{F}^{\wedge}_{n,\mathbb{U}_N}$ and $S \subseteq [n]$).

Our overall barrier result will proceed in two steps. First, we build an FSS scheme for conjunctions $\mathcal{F}^{\wedge}_{n,\mathbb{U}_N}$ from an FSS scheme for multiply-then-MSB $\mathcal{F}^{\times\mathsf{MSB}}_{n(\lceil \log n \rceil+1),\mathbb{U}_N}$. Next, we give a reduction from the FSS scheme for $\mathcal{F}^{\times\mathsf{MSB}}_{\eta,\mathbb{U}_N}$ to the FSS gate for unsigned fixed-point multiplication, $\mathcal{G}_{\mathsf{uFPM}}$ over $\mathbb{U}_{2^\eta}$, and set $\eta = n(\lceil \log n \rceil + 1)$. We now focus only on the case of unsigned fixed point multiplication - the case of signed fixed point multiplication follows in an analogous manner (details of the changes needed can be found in the full version [9]).

Step one of the barrier result. Intuitively, for a function $f_S \in \mathcal{F}^{\wedge}_{n,\mathbb{U}_N}$, the input/output behavior will be emulated by a corresponding function $f_{c_S} \in \mathcal{F}^{\times\mathsf{MSB}}_{n(\lceil \log n \rceil+1),\mathbb{U}_N}$, i.e., $f_S(x) = f_{c_S}(x) = \mathsf{MSB}(x' \cdot c_S)$, where x' is a public encoding of the input x, and c_S is a (secret) constant determined as a function of S. The Gen algorithm of FSS scheme for $f_S \in \mathcal{F}^{\wedge}_{n,\mathbb{U}_N}$ will output FSS keys for $f_{c_S} \in \mathcal{F}^{\times\mathsf{MSB}}_{n(\lceil \log n \rceil+1),\mathbb{U}_N}$, where c_S is determined from S. The Eval algorithm will encode the public $x \in \mathbb{U}_{2^n}$ to $x' \in \mathbb{U}_{2^{n(\lceil \log n \rceil+1)}}$ and evaluate the given FSS key for f_{c_S}.

More concretely, the new FSS evaluation will encode the input x to x' by "spacing out" the bits of x with $m = \lceil \log n \rceil$ zeros with $x_{[0]}$ as the least significant bit (as depicted below). Now, c_S is carefully crafted to "extract" and add the bits in x at indices in S such that: the value $x' \cdot c_S$ will have most significant bit (MSB) as 1 if and only if bits of x in all indices of S are equal to 1. For ease of exposition, first consider the case when size $h = |S|$ is a power of 2 and let $\ell = \log h$. Moreover, consider an alternate representation of $S \subseteq [n]$ as $(s_{n-1}, \ldots, s_0) \in \{0,1\}^n$ such that $s_i = 1$ iff $i \in S$, else 0. Then, $c_S \in \mathbb{U}_{2^{n(m+1)}}$ (depicted below) will be constructed by spacing out the bits s_i by m zeros and put in reverse order, and has ℓ leading zeros and $m - \ell$ trailing zeros.

Mathematically, we can write, $x' = \sum_{i=0}^{n-1} x_{[i]} \cdot 2^{i(m+1)} \in \mathbb{U}_{2^{n(m+1)}}$ and $c_S = 2^{n(m+1)-\ell-1} \cdot \sum_{i=0}^{n-1} s_i \cdot 2^{-i(m+1)} \in \mathbb{U}_{2^{n(m+1)}}$. We will make use of these equations in formal construction and correctness of reduction.

$$x' = \overbrace{0\cdots0}^{m}\,|\,x_{[n-1]}\,|\,\overbrace{0\cdots0}^{m}\,|\,x_{[n-2]}\,|\,\overbrace{0\cdots0}^{m}\,|\cdots|\,\overbrace{0\cdots0}^{m}\,|\,x_{[0]}$$

$$c_S = \overbrace{0\cdots0}^{\ell}\,|\,s_0\,|\,\overbrace{0\cdots0}^{m}\,|\,s_1\,|\,\overbrace{0\cdots0}^{m}\,|\cdots|\,\overbrace{0\cdots0}^{m}\,|\,s_{n-1}\,|\,\overbrace{0\cdots0}^{m-\ell}$$

The interesting part in the product $x' \cdot c_S$ is the upper $\ell + 1$ bits which will capture the sum $\sum_{i=0}^{n-1} x_{[i]} \cdot s_i$. Things have been structured so that none of the other terms in $x' \cdot c_S$ affect these upper bits due to the large spacing of 0s

(preventing additive carries), as shown in the proof of Theorem 7. Therefore, $\mathsf{MSB}(x' \cdot c_S) = \mathsf{MSB}(\sum_{i=0}^{n-1} x_{[i]} \cdot s_i) = \mathsf{MSB}(\sum_{i \in S} x_{[i]})$ (because $s_i = 1$ for $i \in S$, else 0), which is equal to 1 precisely if all bits $\{x_{[i]}\}_{i \in S}$ are equal to 1. Namely, precisely if $f_S(x) = 1$, as desired.

The more general case where $h = |S|$ is not necessarily a power of 2 can be addressed by replacing $s_i \in \{0, 1\}$ with arbitrary positive integer values such that the sum of *all* terms $\sum_{i \in S} s_i$ is precisely equal to 2^ℓ, where $\ell = \lceil \log h \rceil$, and $\{s_i\}_{i \notin S} = 0$. The analysis remains the same.

Theorem 7. *Assume the existence of an FSS scheme for the function class* $\mathcal{F}^{\times\mathsf{MSB}}_{n(m+1),\mathbb{U}_N}$*, where* $m = \lceil \log n \rceil$*. Then there exists an FSS scheme for* $\mathcal{F}^{\wedge}_{n,\mathbb{U}_N}$*.*

Proof. Details of this proof can be found in our full version [9]. □

Step two of the barrier result. In the full version, we give a formal reduction from the FSS scheme for $\mathcal{F}^{\times\mathsf{MSB}}_{\eta,\mathbb{U}_N}$ to $\mathcal{G}_{\mathsf{uFPM}}$ over $\mathbb{U}_{2^n}$. Setting $\eta = n(\lceil \log n \rceil + 1)$ completes the barrier result for unsigned fixed-point multiplication. The high level idea is as follows: we set the shift parameter of $\mathcal{G}_{\mathsf{uFPM}}$ as $s = \eta - 1$ and include $c + r$ as a part of the FSS key (along with the key for $\mathcal{G}_{\mathsf{uFPM}}$) which still hides the secret constant c of member functions in $\mathcal{F}^{\times\mathsf{MSB}}_{\eta,\mathbb{U}_N}$, where r is randomly sampled from $\mathbb{U}_{2^n}$ and known only to the Gen algorithm. Then using these FSS keys, the evaluation algorithm computes $((x \cdot c) \gg_L \eta - 1) = \mathsf{MSB}(x \cdot c)$, as desired.

Acknowledgments

E. Boyle supported by ISF grant 1861/16, AFOSR Award FA9550-17-1-0069, and ERC Project HSS (852952). N. Gilboa supported by ISF grant 2951/20, ERC grant 876110, and a grant by the BGU Cyber Center. Y. Ishai supported by ERC Project NTSC (742754), ISF grant 2774/20, NSF-BSF grant 2015782, and BSF grant 2018393.

References

1. Salami slicing — Wikipedia. `https://en.wikipedia.org/w/index.php?title=Salami_slicing&oldid=943583075` (2020), [Online; accessed 1-November-2020]
2. Agrawal, N., Shamsabadi, A.S., Kusner, M.J., Gascón, A.: QUOTIENT: Two-Party Secure Neural Network Training and Prediction. In: CCS (2019)
3. Aliasgari, M., Blanton, M., Zhang, Y., Steele, A.: Secure computation on floating point numbers. In: NDSS (2013)
4. Aly, A., Smart, N.P.: Benchmarking privacy preserving scientific operations. In: ACNS 2019 (2019)
5. Atallah, M.J., Pantazopoulos, K.N., Rice, J.R., Spafford, E.H.: Secure outsourcing of scientific computations. Adv. Comput. (2001)
6. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: CRYPTO (1991)

7. Ben-Efraim, A., Nielsen, M., Omri, E.: Turbospeedz: Double your online SPDZ! improving SPDZ using function dependent preprocessing. In: ACNS (2019)
8. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: STOC (1988)
9. Boyle, E., Chandran, N., Gilboa, N., Gupta, D., Ishai, Y., Kumar, N., Rathee, M.: Function secret sharing for mixed-mode and fixed-point secure computation. IACR Cryptol. ePrint Arch. (2020)
10. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: EUROCRYPT (2015)
11. Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: CRYPTO (2016)
12. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: CCS (2016)
13. Boyle, E., Gilboa, N., Ishai, Y.: Secure computation with preprocessing via function secret sharing. In: TCC (2019)
14. Boyle, E., Kohl, L., Scholl, P.: Homomorphic secret sharing from lattices without FHE. In: EUROCRYPT (2019)
15. Büscher, N., Demmler, D., Katzenbeisser, S., Kretzmer, D., Schneider, T.: HyCC: Compilation of hybrid protocols for practical secure computation. In: CCS (2018)
16. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: STOC (2002)
17. Catrina, O., de Hoogh, S.: Secure multiparty linear programming using fixed-point arithmetic. In: ESORICS (2010)
18. Catrina, O., Saxena, A.: Secure computation with fixed-point numbers. In: FC (2010)
19. Chandran, N., Gupta, D., Rastogi, A., Sharma, R., Tripathi, S.: EzPC: Programmable and Efficient Secure Two-Party Computation for Machine Learning. In: IEEE EuroS&P (2019)
20. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: STOC (1988)
21. Couteau, G.: A note on the communication complexity of multiparty computation in the correlated randomness model. In: EUROCRYPT, Part II (2019)
22. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: TCC (2006)
23. Damgård, I., Nielsen, J.B., Nielsen, M., Ranellucci, S.: The tinytable protocol for 2-party secure computation, or: Gate-scrambling revisited. In: CRYPTO, Part I (2017)
24. Demmler, D., Dessouky, G., Koushanfar, F., Sadeghi, A., Schneider, T., Zeitouni, S.: Automated synthesis of optimized circuits for secure computation. In: CCS (2015)
25. Demmler, D., Schneider, T., Zohner, M.: ABY-a framework for efficient mixed-protocol secure two-party computation. In: NDSS (2015)
26. Dodis, Y., Halevi, S., Rothblum, R.D., Wichs, D.: Spooky encryption and its applications. In: CRYPTO (2016)
27. Doerner, J., Shelat, A.: Scaling ORAM for secure computation. In: CCS (2017)
28. Fazio, N., Gennaro, R., Jafarikhah, T., III, W.E.S.: Homomorphic secret sharing from paillier encryption. In: Provable Security (2017)
29. Goldreich, O., Micali, S., Wigderson, A.: How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In: STOC (1987)
30. Ishai, Y., Kushilevitz, E., Meldgaard, S., Orlandi, C., Paskin-Cherniavsky, A.: On the power of correlated randomness in secure computation. In: TCC (2013)

31. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer - efficiently. In: CRYPTO (2008)
32. Ishai, Y., Prabhakaran, M., Sahai, A.: Secure arithmetic computation with no honest majority. In: TCC (2009)
33. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In: USENIX Security (2018)
34. Kerik, L., Laud, P., Randmets, J.: Optimizing MPC for robust and scalable integer and floating-point arithmetic. In: FC (2016)
35. Kilian, J.: More general completeness theorems for secure two-party computation. In: STOC (2000)
36. Kiltz, E., Damgaard, I., Fitzi, M., Nielsen, J.B., Toft, T.: Unconditionally secure constant round multi-party computation for equality, comparison, bits and exponentiation. IACR Cryptology ePrint Archive **2005** (2005)
37. Kumar, N., Rathee, M., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: CrypTFlow: Secure TensorFlow Inference. In: IEEE S&P (2020)
38. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious neural network predictions via minionn transformations. In: CCS (2017)
39. Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., Popa, R.A.: Delphi: A cryptographic inference service for neural networks. In: USENIX Security (2020)
40. Mohassel, P., Rindal, P.: ABY3: A mixed protocol framework for machine learning. In: CCS (2018)
41. Mohassel, P., Zhang, Y.: SecureML: A system for scalable privacy-preserving machine learning. In: IEEE S&P (2017)
42. Naor, M., Pinkas, B.: Oblivious polynomial evaluation. SIAM J. Comput. **35**(5) (2006)
43. Nawaz, M., Gulati, A., Liu, K., Agrawal, V., Ananth, P., Gupta, T.: Accelerating 2PC-based ML with limited trusted hardware. arXiv preprint:2009.05566 (2020)
44. Nishide, T., Ohta, K.: Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In: PKC (2007)
45. Pullonen, P., Siim, S.: Combining secret sharing and garbled circuits for efficient private IEEE 754 floating-point computations. In: FC (2015)
46. Rathee, D., Rathee, M., Kumar, N., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: CrypTFlow2: Practical 2-party secure inference. In: CCS (2020)
47. Riazi, M.S., Samragh, M., Chen, H., Laine, K., Lauter, K.E., Koushanfar, F.: XONN: xnor-based oblivious deep neural network inference. In: USENIX Security (2019)
48. Ryffel, T., Pointcheval, D., Bach, F.: ARIANN: Low-interaction privacy-preserving deep learning via function secret sharing. arXiv preprint:2006.04593 (2020)
49. Schoenmakers, B., Tuyls, P.: Efficient binary conversion for paillier encrypted values. In: EUROCRYPT (2006)
50. Toft, T.: Constant-rounds, almost-linear bit-decomposition of secret shared values. In: CT-RSA (2009)
51. Wagh, S., Gupta, D., Chandran, N.: SecureNN: 3-Party Secure Computation for Neural Network Training. PoPETs (2019)
52. Wang, X., Malozemoff, A.J., Katz, J.: EMP-toolkit: Efficient MultiParty computation toolkit. `https://github.com/emp-toolkit` (2016)
53. Yao, A.C.: How to generate and exchange secrets. In: FOCS (1986)