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[2]

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

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× 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 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.

[⋆] Work done while at Microsoft Research, India

Table of Contents

1 Introduction

Secure multi-party computation (or MPC) [12,29,43,78] 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). 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 [8], garbled circuit correlations [36,78], OT [25,45,50] and OLE [46,60] correlations, and one-time truth tables [33,44].

现有相关随机性的类型

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 [23,28,36,51,57] 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.

[21]Secure computation with preprocessing via function secret sharing

Recently, Boyle *et al.* [21] proposed a powerful approach for mixed-mode MPC in the preprocessing model, using *function secret sharing* (FSS) [18,20] (their approach can be seen as a generalization of an earlier truth-table based protocol of Damgård *et al.* [33]). 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 [18,20] 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* – we discuss how to emulate the trusted dealer with 2PC (building upon [39]) as well as extensions to malicious security, in Appendix A and B, respectively. The main idea, from [21], to obtain 2PC with trusted dealer is as follows.

存在可信第三方的情况下实现2PC

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 [21] is in efficiently realizing FSS gates for useful types of gates g.

[21]Secure computation with preprocessing via function secret sharing

For addition and multiplication gates over a finite ring, the FSS gates are information-theoretic and essentially coincide with Beaver's protocol [8] (more accurately, its circuit-dependent variant from [11,30,33]). A key observation of [21] 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 [20]. 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 [21] 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.

1. 输入的秘密份额 --> 输出的秘密份额
2. 掩蔽的输入 --> 掩蔽的输出

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 [21]. 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) [78] and a GMW-style protocol based on secret sharing [43].[6] 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 [23, 28, 36, 47, 56, 57, 59]) requires one of the parties to communicate $256n$ bits (a pair of AES keys for each input), which is $128\times$ bigger. Furthermore, the parties need to interact in *two* sequential rounds. In Appendix C we discuss a way to reduce the online communication of a GC-based protocol by $2\times$, which still leaves a $64\times$ overhead in communication and $2\times$ 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 Section 1.3 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), 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 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 Appendix B), and when emulating the trusted dealer using the different options we discuss: third party, 2PC protocol (Appendix A), or semi-trusted hardware [61].

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 Appendix A), 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

[6] 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.

Boolean operations,[7] and hence, fixed-point arithmetic operations cannot be realized securely using existing lightweight FSS machinery.

We now discuss our contributions that address these bottlenecks.

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 [21]).
- **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 [21] 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 [21], we observe a reduction in key size ranging from $6\times$ for ReLU to $22\times$ for splines and $77\times$ for multiple interval containment (MIC) with 12 intervals. (See Appendix D for precise definitions of all gate types.) As can be observed, for all of the FSS gates considered in [21], 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 $2\times$ better than garbled circuits for ReLU and $15\times$ and $27\times$ 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 $64\times$ lower online communication and $2\times$ 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 $4\times$ 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 [21] are cast as *reductions* that invoke multiple instances of DCF. As a second step, we significantly improve the previous reductions from [21] 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 [20]. 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.[8]

[7] An FSS-based protocol for right-shift can be obtained using the FSS gate for bit-decomposition from [21]. 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.

[8] A concurrent work by Ryffel *et al.* [70] 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.

Better reductions to DCF. We significantly reduce the number of DCF instances required by most of the non-arithmetic FSS gates from [21]. 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 [21] 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,[9] absolute value, as well as approximations of transcendental functions [55, 59].

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 [21] 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 [55]. 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 [21] 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 [32, 63, 71, 72]. An FSS gate for the above flavor of bit-decomposition was given in [21] with $O(n^2\lambda)$ 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 [21], while maintaining poly(n) computation time.

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 [21] 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 [57,59,67] and references therein) and scientific computing (see [5, 7, 26, 27] 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

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

Table 1: Comparison of our FSS gate key sizes, with those of [21], and Garbled Circuits (GC) [75]. For FSS (i.e., our work and [21]), 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. 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 [21], 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 $2\,(n(\lambda+\ell+2)+\lambda+\ell)$ bits. Size of $\mathsf{DCF}^{\mathsf{BGI}}_{n,\mathbb{G}}$ key (using [20]) is $2\,(4n(\lambda+1)+n\ell+\lambda)$ 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 (Appendix C.2) 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 [21]	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)

right shift (or truncation) through a non-interactive "local truncation" procedure [38, 53, 57, 59, 74]. 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 [2,68], local truncation may be too costly for large scale applications [67]. 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.

2. 即使没有发生大错误，由局部截断产生的最低有效位也很有可能是错误的。

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 [21] 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 [19], Paillier [41] or Learning With Errors [22, 38], 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.

1.3 Other Related Works

FSS-based MPC has some key advantages over works on 2PC and 3PC in the preprocessing model. Prior works on 2PC in the preprocessing model [36, 47, 56, 59, 69] use garbled circuits (GC) to evaluate non-arithmetic gates such as comparison and general splines. This holds for many 3PC works with honest majority [10, 57, 58] that try to minimize online round complexity. In Appendix C.1, we describe an optimized GC-based approach in the preprocessing model that is more communication efficient than prior works. Our FSS-based approach beats this GC-based approach in both correlated randomness size (see Table 1) and online communication (by a factor of $\lambda/2$, namely $64\times$ for AES-based implementation).

Another line of work in the preprocessing model [37, 64, 65] focuses on optimizing the rounds and communication of the online phase without the use of GC (due to its high communication cost). A recent work ABY2.0 [65] reduces online rounds and communication complexity of GMW-like approach [36] for many non-arithmetic functions by constructing efficient protocols for multi input AND gates. Even then, FSS-based approach is more efficient in the online phase in terms of both rounds as well as communication. For instance, for popular functions like ReLU and spline-based approximation of sigmoid, ABY2.0 [65] requires $4 - 5\times$ more rounds and $3 - 6\times$ more communication in the online phase compared to the FSS-based approach. However, [65] requires smaller correlated randomness and it can be generated more efficiently using 2PC based preprocessing.

In the honest-majority setting, several works solve the mixed-mode secure computation problem using secret sharing rather than GC in order to reduce the communication cost [6, 53, 57, 67, 74]. However, the round complexity of these protocols is typically much higher than ours, and even when optimized to the offline-online setting, their online communication is significantly higher than ours.

An alternate line of work considers building secure protocols for floating-point arithmetic [4, 14]; however, these are significantly more expensive than their fixed-point counterparts [27, 53, 59], which suffice for most applications.

The work on pseudorandom correlation generators [16] shows a simple way to use a *single* DPF for compressing a randomly shifted and secret-shared truth-table, which can be used for implementing an arbitrary FSS gate. However, their solution either requires (1) superlinear offline computation and linear storage in $N = 2^n$, or (2) linear online computation in N. In contrast, the storage and online computation costs of our solutions scale polynomially with n.

1.4 Organization

Following some background on data types, gates, and FSS in Section 2, Section 3 gives our FSS construction for DCF. We present our core technical contribution of efficient FSS gates for single and multiple interval containments with public boundaries (Section 4), extend these ideas to splines with public intervals (Section 5.1) and bit-decomposition (Section 5.2), provide constructions for FSS gates for fixed-point arithmetic (Section 6), and demonstrate the barrier towards one round fixed-point multiplication via FSS (Section 7). The appendices include discussion of two-party protocols for emulating the dealer in the two-party setting (Appendix A), achieving malicious security with a trusted dealer (Appendix B), detailed comparison with garbled circuits Appendix C, and other proofs and details that are deferred from the main text.

2 Preliminaries

We provide an abbreviated version of preliminaries and notation. We defer a more detailed formal treatment to Appendix E.

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 $x_{[n-1]}, \dots, x_{[0]}$, 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 $x_{[k-1]}, \dots, x_{[j]}$. $||$ 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 $\mathbb{U}_N = \{0, \dots, N-1\}$ is isomorphic to $\mathbb{Z}_N$. Moreover, $\mathbb{S}_N = \{-N/2, \dots, 0, \dots, N/2-1\}$ can be encoded into $\mathbb{Z}_N$ or $\mathbb{U}_N$ using 2's complement notation or mod N operation. The positive values $\{0, \dots, N/2-1\}$ are mapped identically to $\{0, \dots, N/2-1\}$ and negative values $\{-N/2, \dots, -1\}$ are mapped to $\{N/2, \dots, N-1\}$. 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.

只要结果在 [-N/2, N/2) 范围内，有符号整数模 N 的加法、减法和乘法就会遵守此表示。

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), as well as operators required for the realization of fixed-point arithmetic - such as fixed-point addition and multiplication (Section 6.1), logical and arithmetic right shifts (Section 6.2 & 6.3), and comparison (Section 6.4).

2.2 Function Secret Sharing

We follow the definition of function secret sharing (FSS) from [20]. 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 [18, 20, 21], 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 [20]):

Theorem 1 (Concrete cost of DCF [20]). *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 $\boldsymbol{\mathsf{DCF}}_{n,\mathbb{G}}$ 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.* [21] 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 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.* [21] 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 [20] 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.

```
**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 $\lambda$-bit string to a pseudorandom group element of $\mathbb{G}$.

$\mathsf{Gen}_n^<(1^\lambda, \alpha, \beta, \mathbb{G})$:
 1: Let $\alpha = \alpha_1, \ldots, \alpha_n \in \{0,1\}^n$ be the bit decomposition of $\alpha$
 2: Sample random $s_0^{(0)} \leftarrow \{0,1\}^\lambda$ and $s_1^{(0)} \leftarrow \{0,1\}^\lambda$
 3: Let $V_\alpha = 0 \in \mathbb{G}$, let $t_0^{(0)} = 0$ and $t_1^{(0)} = 1$
 4: for $i = 1$ to $n$ do
 5:     $s_0^L || v_0^L || t_0^L \;||\; s_0^R || v_0^R || t_0^R \leftarrow G(s_0^{(i-1)})$
 6:     $s_1^L || v_1^L || t_1^L \;||\; s_1^R || v_1^R || t_1^R \leftarrow G(s_1^{(i-1)})$
 7:     if $\alpha_i = 0$ then $\mathsf{Keep} \leftarrow L$, $\mathsf{Lose} \leftarrow R$
 8:     else $\mathsf{Keep} \leftarrow R$, $\mathsf{Lose} \leftarrow L$
 9:     end if
10:     $s_{CW} \leftarrow s_0^{\mathsf{Lose}} \oplus s_1^{\mathsf{Lose}}$
11:     $V_{CW} \leftarrow (-1)^{t_1^{(i-1)}} \cdot [\mathsf{Convert}_\mathbb{G}(v_1^{\mathsf{Lose}}) - \mathsf{Convert}_\mathbb{G}(v_0^{\mathsf{Lose}}) - V_\alpha]$
12:     if $\mathsf{Lose} = L$ then $V_{CW} \leftarrow V_{CW} + (-1)^{t_1^{(i-1)}} \cdot \beta$
13:     end if
14:     $V_\alpha \leftarrow V_\alpha - \mathsf{Convert}_\mathbb{G}(v_1^{\mathsf{Keep}}) + \mathsf{Convert}_\mathbb{G}(v_0^{\mathsf{Keep}}) + (-1)^{t_1^{(i-1)}} \cdot V_{CW}$
15:     $t_{CW}^L \leftarrow t_0^L \oplus t_1^L \oplus \alpha_i \oplus 1$ and $t_{CW}^R \leftarrow t_0^R \oplus t_1^R \oplus \alpha_i$
16:     $CW^{(i)} \leftarrow s_{CW} || V_{CW} || t_{CW}^L || t_{CW}^R$
17:     $s_b^{(i)} \leftarrow s_b^{\mathsf{Keep}} \oplus t_b^{(i-1)} \cdot s_{CW}$ for $b = 0, 1$
18:     $t_b^{(i)} \leftarrow t_b^{\mathsf{Keep}} \oplus t_b^{(i-1)} \cdot t_{CW}^{\mathsf{Keep}}$ for $b = 0, 1$
19: end for
20: $CW^{(n+1)} \leftarrow (-1)^{t_1^n} \cdot [\mathsf{Convert}_\mathbb{G}(s_1^{(n)}) - \mathsf{Convert}_\mathbb{G}(s_0^{(n)}) - V_\alpha]$
21: Let $k_b = s_b^{(0)} || CW^{(1)} || \cdots || CW^{(n+1)}$
22: return $(k_0, k_1)$

$\mathsf{Eval}_n^<(b, k_b, x)$:
 1: Parse $k_b = s^{(0)} || CW^{(1)} || \cdots || CW^{(n+1)}$, $x = x_1, \ldots, x_n$, let $V = 0 \in \mathbb{G}$, $t^{(0)} = b$.
 2: for $i = 1$ to $n$ do
 3:     Parse $CW^{(i)} = s_{CW} || V_{CW} || t_{CW}^L || t_{CW}^R$
 4:     Parse $G(s^{(i-1)}) = \hat{s}^L || \hat{v}^L || \hat{t}^L \;||\; \hat{s}^R || \hat{v}^R || \hat{t}^R$
 5:     $\tau^{(i)} \leftarrow (\hat{s}^L || \hat{t}^L \;||\; \hat{s}^R || \hat{t}^R) \oplus (t^{(i-1)} \cdot [s_{CW} || t_{CW}^L || s_{CW} || t_{CW}^R])$
 6:     Parse $\tau^{(i)} = s^L || t^L \;||\; s^R || t^R \in \{0,1\}^{2(\lambda+1)}$
 7:     if $x_i = 0$ then $V \leftarrow V + (-1)^b \cdot [\mathsf{Convert}_\mathbb{G}(\hat{v}^L) + t^{(i-1)} \cdot V_{CW}]$
 8:         $s^{(i)} \leftarrow s^L$, $t^{(i)} \leftarrow t^L$
 9:     else $V \leftarrow V + (-1)^b \cdot [\mathsf{Convert}_\mathbb{G}(\hat{v}^R) + t^{(i-1)} \cdot V_{CW}]$
10:         $s^{(i)} \leftarrow s^R$, $t^{(i)} \leftarrow t^R$
11:     end if
12: end for
13: $V \leftarrow V + (-1)^b \cdot [\mathsf{Convert}_\mathbb{G}(s^{(n)}) + t^{(n)} \cdot CW^{(n+1)}]$
14: 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.

Our construction draws inspiration from the DPF of [20]. 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 for detailed security proof). The scheme uses the function $\mathsf{Convert}_{\mathbb{G}} : \{0,1\}^{\lambda} \to \mathbb{G}$ [20] 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 in Appendix F.2.

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 [5] as well as machine learning [53, 59, 74]. Moreover, many popular functions such as splines (see Section 5.1) and most significant non-zero bit (MSNZB) (see Appendix H.1) reduce to computing multiple interval containments on the same secret value x. The work of [21] provided the first constructions of a PRG-based 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:

- First, 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 [21]. Including the gains from our optimized DCF, we get around $7\times$ reduction in key size over [21] for $n = 32$.
- Next, in Section 4.2, we show how to *compress* the FSS keys for multiple interval containments to essentially 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 $1100\times$ and for instance, for $m = 10$, the reduction is about $62\times$.

While the construction from [21] 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 (Section 5.2) and MSNZB (Appendix H.1).

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: $(x + N/2 \bmod N) \in [(p + N/2 \bmod N), (q + N/2 \bmod N)]$. 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 [21]. 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 [21], 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. We note that the key size of our construction in Fig. 2 is identical to the scheme presented in [21], 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.

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 (see Appendix G.2), 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:*

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

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

```
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 [21], b refers to party id.

```
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.

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 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 for this construction).

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

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.

4.2 FSS Gate for Multiple Interval Containments

We now construct an FSS gate for multiple interval containments $\mathcal{G}_{\mathsf{MIC}}$ using the above idea of reducing FSS gate for arbitrary public interval $[p, q]$ to the universal comparison function $f^{<}_{(N-1)+\mathsf{r}^{\mathsf{in}},1}$. Formally, we reduce key for the FSS gate for m public intervals $[p_1, q_1], \ldots, [p_m, q_m]$ to FSS key for $f^{<}_{(N-1)+\mathsf{r}^{\mathsf{in}},1}$ plus m elements from $\mathbb{U}_N$. We describe our construction of FSS Gate for $g_{\mathsf{MIC},n,m,P,Q}$ formally in Fig. 14, Appendix G.5 that satisfies the following theorem.

Theorem 4. *There is an FSS Gate* $(\mathsf{Gen}^{\mathsf{MIC}}_{n,m}, \mathsf{Eval}^{\mathsf{MIC}}_{n,m})$ *for* $\mathcal{G}_{\mathsf{MIC}}$ *that requires* $2m$ *invocations of* $\mathsf{DCF}_{n,\mathbb{U}_N}$*, and has a total key size of* mn *bits plus the key size of* $\mathsf{DCF}_{n,\mathbb{U}_N}$*.*

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 [55, 59], sometimes with up to $m = 12$ intervals. Boyle *et al.* [21], 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[10], we reduce both the key size as well as online evaluation. More concretely, [21] requires $2m$ $\mathsf{DCF}_{n,\mathbb{Z}_N^{d+1}}$ keys and each key is evaluated once during online phase. We 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,

[10] 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 $\mathcal{G}_{\mathsf{MIC}}$.

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 [21], 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, [21] 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 similar to Section 4.2. 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 $\mathcal{G}_{\mathsf{spline\text{-}one}}$. 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.

Theorem 5. *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 $w=\left(c_x-\mathbf{1}\{x^{(L)}<\gamma\}+\mathbf{1}\{x^{(R')}<\gamma\}+c_r\right)\cdot\beta = \mathbf{1}\{0\leqslant(x-\mathsf{r}^{\mathsf{in}})\leqslant p\}\cdot\beta$ as required, by using similar arguments as in correctness of $\mathcal{G}_{\mathsf{IC}}$ in Fig. 3.

□

ReLU. ReLU is the most commonly used activation function in machine learning [54] and $\mathsf{ReLU}(x)=x$ if x is positive and 0 otherwise. Efficient protocols for this function has been a main focus of many works on secure machine learning [47, 53, 56, 59, 67, 74]. When signed integers are encoded using 2's complement, elements of $\mathbb{S}_N$ have a natural embedding in $\mathbb{U}_N$ using mod$N$ operation. Then, for $x\in\mathbb{U}_N$, $\mathsf{ReLU}(x)=x$

```
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 $(f'_d, \dots, f'_0) \in \mathbb{U}_N^{(d+1)}$ be coefficients of $f'$ such that $f'(x) = f(x - \mathsf{r}^{\mathsf{in}})$.
 2: Set $\beta = (f'_d, \dots, f'_0) \in \mathbb{U}_N^{(d+1)}$ 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: $w_b = (w_{d,b}, \dots, w_{0,b}) = c_x \cdot \beta_b - s_b^{(L)} + s_b^{(R')} + e_b$.
 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.

for $x \in [0, 2^{n-1} - 1]$ and 0 otherwise, and is a special case of $\mathcal{G}_{\mathsf{spline\text{-}one}}$. Hence, as a corollary of Theorem 5, we have an FSS gate for $\mathcal{G}_{\mathsf{ReLU}}$ that requires 2 invocations of $\mathsf{DCF}_{n,\mathbb{U}_N^2}$, and has a total key size of $5n$ bits plus the key size of $\mathsf{DCF}_{n,\mathbb{U}_N^2}$. FSS gate for ReLU can be constructed using splines in [21]. Our construction gives $2\times$ reduction in key size over [21] even when [21] is instantiated using our optimized DCF and overall $\approx 6\times$ reduction.

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 $i \in \{1, \dots, m\}$.

Theorem 6. *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* $\mathsf{DCF}_{n,\mathbb{U}_N^{m(d+1)}}$.

Proof. We provide our construction of FSS gate for general splines formally in Fig. 5. For correctness, it suffices to prove that $t = t_0 + t_1 = \beta_j$ when $(x - \mathsf{r}^{\mathsf{in}} \bmod N)$ lies in the j^{th} interval $[p_{j-1}+1, p_j]$. In Fig. 5, $w^{(i)} = w_0^{(i)} + w_1^{(i)} = \sum_b (c_{x,i} \cdot \beta_{i,b} - s_{i,b}^{(i)} + s_{i,b}^{(i+1)} + e_{i,b}) = c_{x,i} \cdot \beta_i - s_i^{(i)} + s_i^{(i+1)} + c_{r,i} \cdot \beta_i$. Observe that $s_i^{(i)}$ (resp. $s_i^{(i+1)}$) corresponds to the i^{th} segment of $(d+1)$ ring elements in the output of DCF evaluated on the input $x_i = x_i^{(L)} = x + (N - 1 - (p_{i-1}+1)) \in \mathbb{U}_N$ (resp. $x_{i+1} = x_i^{(R')} = x + (N - 1 - (p_i + 1)) \in \mathbb{U}_N$). By the

```
Spline Gate ($\mathsf{Gen}^{\mathsf{spline}}_{n,m,d,\{p_i\}_i}, \mathsf{Eval}^{\mathsf{spline}}_{n,m,d,\{p_i\}_i}$)
$\mathsf{Gen}^{\mathsf{spline}}_{n,m,d,\{p_i\}_i}(1^\lambda, \{f_i\}_i, \mathsf{r}^{\mathsf{in}}, \mathsf{r}^{\mathsf{out}})$:
 1: For $i \in \{1,\ldots,m\}$, let $\beta_i = (f'_{i,d},\ldots,f'_{i,0}) \in \mathbb{U}_N^{(d+1)}$, be the coefficient vector of $f'_i$ s.t. $f'_i(x) = f_i(x - \mathsf{r}^{\mathsf{in}})$.
 2: Set $\beta = (\beta_1,\ldots,\beta_m) \in \mathbb{U}_N^{m(d+1)}$ and $\gamma = (N-1) + \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$.
 3: $(k_0^{(N-1)}, k_1^{(N-1)}) \leftarrow \mathsf{Gen}^<_n(1^\lambda, \gamma, \beta, \mathbb{U}_N^{m(d+1)})$.
 4: for $i = \{1,\ldots,m\}$ do
 5:     Set $\alpha_i^{(L)} = p_{i-1} + 1 + \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$, $\alpha_i^{(R)} = p_i + \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$ and $\alpha_i^{(R')} = p_i + 1 + \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$.
 6:     Set $c_{r,i} = \mathbf{1}\{\alpha_i^{(L)} > \alpha_i^{(R)}\} - \mathbf{1}\{\alpha_i^{(L)} > (p_{i-1}+1 \bmod N)\} + \mathbf{1}\{\alpha_i^{(R')} > (p_i + 1 \bmod N)\} + \mathbf{1}\{\alpha_i^{(R)} = N-1\}$.
 7:     Sample random $e_{i,0}, e_{i,1} \leftarrow \mathbb{U}_N^{(d+1)}$ s.t. $e_{i,0} + e_{i,1} = c_{r,i} \cdot \beta_i$.
 8:     Sample random $\beta_{i,0}, \beta_{i,1} \leftarrow \mathbb{U}_N^{(d+1)}$ s.t. $\beta_{i,0} + \beta_{i,1} = \beta_i$.
 9: end for
10: Sample random $r_0, r_1 \leftarrow \mathbb{U}_N$ s.t. $r_0 + r_1 = \mathsf{r}^{\mathsf{out}}$.
11: For $b \in \{0,1\}$, let $k_b = k_b^{(N-1)} || \{e_{i,b}\}_i || \{\beta_{i,b}\}_i || r_b$.
12: return $(k_0, k_1)$.

$\mathsf{Eval}^{\mathsf{spline}}_{n,m,d,\{p_i\}}(b, k_b, x)$:
 1: Parse $k_b = k_b^{(N-1)} || \{e_{i,b}\}_i || \{\beta_{i,b}\}_i || r_b$.
 2: for $i = \{1,\ldots,m\}$ do
 3:     Set $x_i = x + (N - 1 - (p_{i-1} + 1)) \in \mathbb{U}_N$.
 4:     Set $(s_{1,b}^{(i)},\ldots,s_{m,b}^{(i)}) \leftarrow \mathsf{Eval}^<_n(b, k_b^{(N-1)}, x_i)$.
 5: end for
 6: Set $(s_{1,b}^{(m+1)},\ldots,s_{m,b}^{(m+1)}) = (s_{1,b}^{(1)},\ldots,s_{m,b}^{(1)})$
 7: for $i = \{1,\ldots,m\}$ do
 8:     Set $c_{x,i} = (\mathbf{1}\{x > (p_{i-1} + 1 \bmod N)\} - \mathbf{1}\{x > (p_i + 1 \bmod N)\})$.
 9:     $w_b^{(i)} = (w_{d,b}^{(i)},\ldots,w_{0,b}^{(i)}) = c_{x,i} \cdot \beta_{i,b} - s_{i,b}^{(i)} + s_{i,b}^{(i+1)} + e_{i,b}$.
10: end for
11: Set $t_b = (t_{d,b},\ldots,t_{0,b}) = \sum_{i=1}^m w_b^{(i)} \in \mathbb{U}_N^{(d+1)}$.
12: return $y_b = r_b + \sum_{i=0}^d (t_{i,b} \cdot x^i) \bmod N$.
```

Fig. 5: FSS Gate for splines $\mathcal{G}_{\mathsf{spline}}$, b refers to party id.

correctness of DCF keys, we have that $s_i^{(i)} = \beta_i \cdot \mathbf{1}\{x_i^{(L)} < \gamma\}$ and $s_i^{(i+1)} = \beta_i \cdot \mathbf{1}\{x^{(R')} < \gamma\}$. Using arguments similar to the correctness proof of Fig. 4, we get that $w^{(i)} = \mathbf{1}\{(p_{i-1}+1) \leqslant (x - \mathsf{r}^{\mathsf{in}}) \leqslant p_i\} \cdot \beta_i$. Since intervals in a spline are disjoint, $(x - \mathsf{r}^{\mathsf{in}})$ lies in exactly one of them, say j^{th} interval, so $\forall i \neq j$, $w^{(i)} = 0^{d+1}$ and $w^{(j)} = \beta_j$. Therefore, $t = \sum_{i=1}^m w^{(i)} = \beta_j$. □

Next, we discuss an optimization to key size of FSS gate for splines such as absolute value, where all the polynomials f_i's are constant multiples of a polynomial f, for publicly known constants.

Absolute Value. Absolute value of $x \in \mathbb{S}_N$, denoted by $|x|$ is equal to x if x is positive and $-x$ otherwise. Again, using the natural encoding of $\mathbb{S}_N$ into $\mathbb{U}_N$ using mod N, $|x|$ can be defined as x for $f_1(x) = x \in [0, 2^{n-1} - 1]$ and $f_2(x) = -x$ for $[2^{n-1}, N-1]$. Using Theorem 6, we get an FSS gate for $\mathcal{G}_{|\cdot|}$ with key size $\mathsf{DCF}_{n,\mathbb{U}_N^4}$ and $9n$ bits. We observe that $f_2 = -1 \cdot f_1$ and compress keys as follows. Let β_1 and β_2 be coefficients of f'_1 and f'_2, respectively, as defined above. Now, the dealer creates the DCF key with payload $\beta = \beta_1$. And similarly, gives out shares of β_1 alone and not β_2. It still gives out shares of $c_{r,1} \cdot \beta_1$ and $c_{r,2} \cdot \beta_2$ and $\mathsf{r}^{\mathsf{out}}$. This gives key size of $\mathsf{DCF}_{n,\mathbb{U}_N^2}$ and $7n$ bits. Now, during evaluation, when evaluating for second interval, P_0, P_1 locally multiply the output by -1, to get correct shares of either β_2 or 0^2, and similarly, multiply shares of β_1 by -1 to get shares of β_2. Finally, we note that this optimized construction reduces the key size of $\mathcal{G}_{|\cdot|}$ of prior solution using general splines in [21] by $4\times$ even when we use our optimized DCF construction to instantiate [21] and $\approx 12\times$ overall.

5.2 Bit Decomposition

The bit-decomposition gate $\mathcal{G}_{\mathsf{BIT}}$ is the family of functions $g_{\mathsf{BIT},n} : \mathbb{U}_N \to \mathbb{U}_N^n$, parameterized by input, output groups $\mathbb{G}^{\mathsf{in}} = \mathbb{U}_N$, $\mathbb{G}^{\mathsf{out}} = \mathbb{U}_N^n$ and given by $g_{\mathsf{BIT},n}(x) = (x_{n-1},\ldots,x_0)$ s.t. $\forall i, x_i \in \{0,1\}$ and $\sum_{i=0}^{n-1} 2^i x_i = x$.

Our FSS gate for bit decomposition builds upon ideas in our FSS gate for $\mathcal{G}_{\mathsf{MIC}}$. We illustrate this using 4-bit integers. Let $x = x_3||x_2||x_1||x_0$, where $x \in \mathbb{U}_{16}$. We observe that the bit $x_3 \Leftrightarrow \mathbf{1}\{8 \leqslant x \leqslant 15\}$ and hence, can be computed using the IC gate $g_{\mathsf{IC},8,15}(x)$. Next, x_2 can be computed as $g_{\mathsf{IC},4,7}(x) + g_{\mathsf{IC},12,15}(x) = \mathbf{1}\{4 \leqslant x \leqslant 7\} + \mathbf{1}\{12 \leqslant x \leqslant 15\}$. Similarly, for x_1 and x_0. As we can see, the number of instances of g_{IC} needed increases exponentially in distance from MSB, i.e., x_3. However, using the idea of compressing the IC keys from the previous section, we reduce all these to the universal comparison function $f^{<}_{15+\mathsf{r}^{\mathsf{in}},1}$. Moreover, since $x_2 = g_{\mathsf{IC},4,7}(x) + g_{\mathsf{IC},12,15}(x)$, the dealer's corrections for the IC gates $g_{\mathsf{IC},4,7}$ and $g_{\mathsf{IC},12,15}$ can be put together as one element. Hence, overall we can compute the bit-decomposition for x using a single DCF key along with 4 additional ring elements.

However, the above idea requires an exponential (in bitlength) computation for both the dealer and the evaluators. We address this issue by breaking the input into smaller chunks. As an example, consider 8-bit integers and $x = x_7||x_6||\ldots x_0$. We compute $x_7 \ldots x_4$ using the idea discussed above. For computing x_3, note that $x_3 = 1 \Leftrightarrow x_{[0,4)} \in [8, 15]$. Using this observation, we can recurse on 4-bit integer defined by $x_{[0,4)}$. For the FSS gate, this can be achieved by dealer dropping upper 4 bits of r^{in} in key generation and P_0, P_1 dropping upper 4 bits of x.

The above idea of extracting 4 bits at a time naturally extends to extracting w bits. With this, overall FSS keys are $\approx \frac{n}{w}$ DCF keys of appropriately decreasing input length (by w), along with n additional ring elements. Since the compute for each chunk is exponential in w, we set $w = \lceil \log n \rceil$, ensuring overall compute is polynomial in n. Hence, our construction with $\approx \frac{n}{\log n}$ DCF keys asymptotically improves upon the prior work [21], which required n DCF keys for the special case[11] of output group $\mathbb{U}_2^n$. We stress that our construction is the first to provide a PRG-based FSS gate for bit decomposition with outputs in larger groups, an important feature missing in [21]. We present the FSS gate for $\mathcal{G}_{\mathsf{BIT}}$ (Fig. 16) and its correctness formally in Appendix H.2.

Theorem 7. *There is an FSS Gate* $(\mathsf{Gen}^{\mathsf{BIT}}_n, \mathsf{Eval}^{\mathsf{BIT}}_n)$ *for* $\mathcal{G}_{\mathsf{BIT}}$ *that makes* $2(2^w - 1)$ *invocations of each of* $\{\mathsf{DCF}_{n-i\cdot w,\mathbb{U}_N}\}_{0\leqslant i\leqslant \lfloor \frac{n}{w} \rfloor -1}$ *keys plus* $2(2^{w'} - 1)$ *invocations of* $\mathsf{DCF}_{w',\mathbb{U}_N}$ *and has a key size of* n^2 *bits plus the key size of* $\{\mathsf{DCF}_{n-i\cdot w,\mathbb{U}_N}\}_{0\leqslant i\leqslant \lfloor \frac{n}{w} \rfloor -1}$ *and* $\mathsf{DCF}_{w',\mathbb{U}_N}$*, where* w *is a parameter s.t.* $1 \leqslant w \leqslant n$ *and* $w' = n \bmod w$*.*

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 [47,53,56,57,59,67,74][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 operators - logical right shift (LRS) in Section 6.2, and arithmetic right shift (ARS) in Section 6.3, which enable scale adjustment, and

[11] Our construction trivially works for output group $\mathbb{U}_2^n$ by using $\mathbb{U}_2$ as output group in DCFs and corresponding ring elements.

[12] Although there are a handful of works outside the secure ML context that give secure protocols directly for floating-point numbers [13] [4, 35, 49, 66], they are usually orders of magnitude slower than the ones based on fixed-point.

hence fixed-point multiplication, over unsigned and signed integers respectively. Finally, in Section 6.4, we show how to construct an FSS gate for comparison of two fixed-point numbers.

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 [21] as described in Fig. 17, Appendix I.1. Fixed-point multiplication involves 2 steps: first, using the FSS gate for multiplication from [21] (presented in Fig. 18, Appendix I.2 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 PRG-based FSS gate for bit-decomposition [21] outputs 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). 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) 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 which satisfies the following theorem.

Theorem 8 (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 $\mathsf{DCF}_{s,\mathbb{U}_N}$.*

[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.

```
Logical Right Shift Gate ($\mathsf{Gen}^{\gg_L}_{n,s}$, $\mathsf{Eval}^{\gg_L}_{n,s}$)
$\mathsf{Gen}^{\gg_L}_{n,s}(1^\lambda, \mathsf{r}^{\mathsf{in}}, \mathsf{r}^{\mathsf{out}})$:
 1: Let $y = (2^n - \mathsf{r}^{\mathsf{in}}) \bmod N$.
 2: $(k_0^{(s)}, k_1^{(s)}) \leftarrow \mathsf{Gen}^{<}_s(1^\lambda, \alpha^{(s)}, 1, \mathbb{U}_N)$, $\alpha^{(s)} = y_{[0,s)} \in \{0,1\}^s$.
 3: $(k_0^{(n)}, k_1^{(n)}) \leftarrow \mathsf{Gen}^{<}_n(1^\lambda, y, 1, \mathbb{U}_N)$.
 4: Sample random $r_0, r_1 \leftarrow \mathbb{U}_N$ s.t. $r_0 + r_1 = \mathsf{r}^{\mathsf{out}} + (y \gg_L s)$.
 5: For $b \in \{0,1\}$, let $k_b = k_b^{(s)} || k_b^{(n)} || r_b$.
 6: return $(k_0, k_1)$.

$\mathsf{Eval}^{\gg_L}_{n,s}(b, k_b, x)$:
 1: Parse $k_b = k_b^{(s)} || k_b^{(n)} || r_b$.
 2: Set $t_b^{(s)} \leftarrow \mathsf{Eval}^{<}_s(b, k_b^{(s)}, x^{(s)})$, where $x^{(s)} = 2^s - x_{[0,s)} - 1$.
 3: Set $t_b^{(n)} \leftarrow \mathsf{Eval}^{<}_n(b, k_b^{(n)}, x^{(n)})$, where $x^{(n)} = 2^n - x - 1$.
 4: return $b \cdot (x \gg_L s) + r_b + t_b^{(s)} - 2^{n-s} \cdot t_b^{(n)}$.
```

Fig. 6: FSS Gate for Logical Right Shift $\mathcal{G}_{\gg_L}$, b refers to party id.

6.3 Arithmetic Right Shift

Arithmetic right shift of signed integers by s, is done by removing the lower s bits and copying the most-significant bit (MSB) in the upper s positions. Similar to LRS, ARS of secret values cannot be obtained by simply locally shifting the shares of the values. In this subsection, we present our construction of the FSS gate for ARS operator.

Handling Signed Integers. Note that the ARS operator takes as input a signed integer $x \in \mathbb{S}_N$, but our crypto protocols work over integers modulo N, i.e. over $\mathbb{U}_N$. Relying on the bijective mapping between elements of $\mathbb{S}_N$ and $\mathbb{U}_N$, i.e. $x \in \mathbb{S}_N$ is mapped to $(x \bmod N) \in \mathbb{U}_N$ (forward) and $y \in \mathbb{U}_N$ is mapped to $(y - \mathsf{MSB}(y) \cdot N) \in \mathbb{S}_N$ (backward), we can define shares of x over $\mathbb{U}_N$, i.e., $x_0, x_1 \in \mathbb{U}_N$ s.t. $x_0 + x_1 \bmod N = (x \bmod N)$. This allows us to perform the secure computation over $\mathbb{U}_N$, as required. The forward mapping defined above gives a binary representation corresponding to the two's complement encoding of signed integers which has the following property: fundamental arithmetic operations like addition and multiplication over signed integers become identical to the corresponding operations on unsigned integers of the same bitwidth. Therefore, operating over shares in $\mathbb{U}_N$ preserves the relation that the reverse mapping of their sum is equal to the secret value in $\mathbb{S}_N$. For the ease of exposition, we use $a \equiv_{\mathsf{s}} b$ to denote that the forward mapping of a is equal to the unsigned value b, where $a \in \mathbb{S}_N$ and $b \in \mathbb{U}_N$.

Lemma 4 (proof appears in Appendix I.4) gives an identity showing that ARS of a secret shared x can be computed as the sum of the LRS of integer represented by lower $n-1$ bits of the shares x_0 and x_1, output of two comparison functions and MSB of x.

Notation. *Substring function on signed integer $x \in \mathbb{S}_N$ is defined as: $x_{[0,i)} = (x \bmod N)_{[0,i)} \in \mathbb{U}_{2^i}$. Similarly, the bit function on signed $x \in \mathbb{S}_N$ is: $x_{[i]} = (x \bmod N)_{[i]}$. For $0 \leqslant s \leqslant n$, let $(\gg_A s) : \mathbb{S}_N \to \mathbb{S}_N$ be the* arithmetic right shift *function with action on input x denoted by $(\gg_A s)(x) = (x \gg_A s)$ and defined by $(x \gg_A s) = \frac{x - (x \bmod 2^s)}{2^s}$ over $\mathbb{Z}$ [40]. Pictorially, the two's complement representation of this is the following*

$$(x \gg_A s) = \underbrace{x_{[n-1]} \ldots x_{[n-1]}}_{s} \overbrace{x_{[n-1]} x_{[n-2]} \ldots x_{[s]}}^{n-s} \tag{1}$$

Lemma 4. *For any integers $n > 0, 0 \leqslant s < n$, any $x \in \mathbb{S}_N$ and any $x_0, x_1 \in \mathbb{U}_N$ such that $x \equiv_{\mathsf{s}} x_0 + x_1 \bmod N$, it holds that $(x \gg_A s) \equiv_{\mathsf{s}} (x_{0[0,n-1)} \gg_L s) + (x_{1[0,n-1)} \gg_L s) + t^{(s)} - 2^{n-s-1}\left(t^{(n-1)} + x_{[n-1]}\right) \bmod N$, where:*

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

The arithmetic right-shift gate $\mathcal{G}_{\gg_A}$ is the family of functions $g_{\gg_A,s,n} : \mathbb{S}_N \to \mathbb{S}_N$ parameterized by input and output groups $\mathbb{S}_N$, shift amount s and given by

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

We denote the corresponding offset gate class by $\hat{\mathcal{G}}_{\gg_A}$, its component functions by $\hat{g}^{[\mathsf{r}^{\mathsf{in}},\mathsf{r}^{\mathsf{out}}]}_{\gg_A,s,n} : \mathbb{U}_N \to \mathbb{U}_N$, where $\mathsf{r}^{\mathsf{in}}, \mathsf{r}^{\mathsf{out}} \in \mathbb{U}_N$, and the following relation holds: $(x' \gg_A s) = g_{\gg_A,s,n}(x') \equiv_{\mathsf{s}} \hat{g}^{[\mathsf{r}^{\mathsf{in}},\mathsf{r}^{\mathsf{out}}]}_{\gg_A,s,n}(x) - \mathsf{r}^{\mathsf{out}}$, where $x' \equiv_{\mathsf{s}} (x - \mathsf{r}^{\mathsf{in}})$. We use Lemma 4 to construct our FSS gate for ARS as described in Fig. 7 which satisfies the following theorem.

```
Arithmetic Right Shift Gate ($\mathsf{Gen}^{\gg_A}_{n,s}, \mathsf{Eval}^{\gg_A}_{n,s}$)
$\mathsf{Gen}^{\gg_A}_{n,s}(1^\lambda, \mathsf{r}^{\mathsf{in}}, \mathsf{r}^{\mathsf{out}})$:
 1: Let $y = (2^n - \mathsf{r}^{\mathsf{in}}) \in \mathbb{U}_N$ and $\alpha^{(n-1)} = y_{[0,n-1)}$
 2: $(k_0^{(s)}, k_1^{(s)}) \leftarrow \mathsf{Gen}^{<}_s\left(1^\lambda, \alpha^{(s)}, 1, \mathbb{U}_N\right)$, where $\alpha^{(s)} = y_{[0,s)}$.
 3: $(k_0^{(n-1)}, k_1^{(n-1)}) \leftarrow \mathsf{Gen}^{\mathsf{DDCF}}_{n-1}\left(1^\lambda, \alpha^{(n-1)}, \beta_1, \beta_2, \mathbb{U}_N \times \mathbb{U}_N\right)$, where $\beta_1 = (1, 1 \oplus y_{[n-1]}), \beta_2 = (0, y_{[n-1]}) \in \mathbb{U}_N \times \mathbb{U}_N$.
 4: Sample random $r_0, r_1 \leftarrow \mathbb{U}_N$ s.t. $r_0 + r_1 = \mathsf{r}^{\mathsf{out}} + (\alpha^{(n-1)} \gg_L s)$.
 5: return $(k_0, k_1)$, where $k_b = k_b^{(s)} || k_b^{(n-1)} || r_b$ for $b \in \{0,1\}$.

$\mathsf{Eval}^{\gg_A}_{n,s}(b, k_b, x)$:
 1: Parse $k_b = k_b^{(s)} || k_b^{(n-1)} || r_b$.
 2: Set $t_b^{(s)} \leftarrow \mathsf{Eval}^{<}_s(b, k_b^{(s)}, x^{(s)})$, where $x^{(s)} = 2^s - x_{[0,s)} - 1$.
 3: Set $(t_b^{(n-1)}, m_b^{(n-1)}) \leftarrow \mathsf{Eval}^{\mathsf{DDCF}}_{n-1}(b, k_b^{(n-1)}, x^{(n-1)})$, where $x^{(n-1)} = 2^{n-1} - x_{[0,n-1)} - 1$.
 4: $m_b = b \cdot x_{[n-1]} + m_b^{(n-1)} - 2 \cdot x_{[n-1]} \cdot m_b^{(n-1)}$.
 5: return $b \cdot (x_{[0,n-1)} \gg_L s) + r_b + t_b^{(s)} - 2^{n-s-1} \cdot (t_b^{(n-1)} + m_b)$.
```

Fig. 7: FSS Gate for Arithmetic Right Shift $\mathcal{G}_{\gg_A}$, b refers to party id.

Theorem 9 (ARS from DDCF and DCF). *There is an FSS Gate* $(\mathsf{Gen}^{\gg_A}_{n,s}, \mathsf{Eval}^{\gg_A}_{n,s})$ *for* $\mathcal{G}_{\gg_A}$ *that requires a single invocation each of* $\mathsf{DDCF}_{n-1,\mathbb{S}_N \times \mathbb{S}_N}$ *and* $\mathsf{DCF}_{s,\mathbb{S}_N}$*, and has a total key size of n bits plus the key sizes of* $\mathsf{DDCF}_{n-1,\mathbb{S}_N \times \mathbb{S}_N}$ *and* $\mathsf{DCF}_{s,\mathbb{S}_N}$*.*

6.4 Comparison

Comparison of 2 integers, x, y (unsigned/signed), is outputs 1 if $x > y$ and 0 otherwise. When working with fixed-point numbers, the comparison function is the same as the one that works over integers except for when the operands x and y have differing scales which is handled similar to how fixed-point addition accommodates operands of different scales.

An FSS gate construction for integer comparison was presented in [21] which reduces to an interval containment on $(x - y)$. Using our optimized interval containment construction in Fig. 3, we already have a FSS gate for integer comparison with half the key size compared to [21], however the number of DCF invocations is the same as in [21]. Here, we present a construction which has half the key size as well half the DCF invocations as in [21]. Note that our construction as well as the prior one from [21] require the following precondition for correctness to hold: $|x - y| < N/2$ for unsigned and $|x| + |y| < N/2$ for the signed case.

The signed integer comparison gate $\mathcal{G}_{\mathsf{sCMP}}$ is the family of functions $g_{\mathsf{sCMP},n} : \mathbb{S}_N \times \mathbb{S}_N \to \mathbb{S}_N$ parameterized by input group $\mathbb{S}_N \times \mathbb{S}_N$ and output group $\mathbb{S}_N$, and given by $g_{\mathsf{sCMP},n}(x,y) := \mathbf{1}\{x > y\}$. We denote the corresponding offset gate class by $\hat{\mathcal{G}}_{\mathsf{sCMP}}$, its component offset functions by $\hat{g}^{[\mathsf{r}_1^{\mathsf{in}},\mathsf{r}_2^{\mathsf{in}},\mathsf{r}^{\mathsf{out}}]}_{\mathsf{sCMP},n} : \mathbb{U}_N \times \mathbb{U}_N \to \mathbb{U}_N$, where $\mathsf{r}_1^{\mathsf{in}}, \mathsf{r}_2^{\mathsf{in}}, \mathsf{r}^{\mathsf{out}} \in \mathbb{U}_N$, and the following relation holds: $\mathbf{1}\{x' > y'\} = g_{\mathsf{sCMP},n}(x',y') \equiv_{\mathsf{s}} \hat{g}^{[\mathsf{r}_1^{\mathsf{in}},\mathsf{r}_2^{\mathsf{in}},\mathsf{r}^{\mathsf{out}}]}_{\mathsf{sCMP},n}(x,y) - \mathsf{r}^{\mathsf{out}}$, where $x' \equiv_{\mathsf{s}} (x - \mathsf{r}_1^{\mathsf{in}})$ and $y' \equiv_{\mathsf{s}} (y - \mathsf{r}_2^{\mathsf{in}})$. Unsigned integer comparison gate $\mathcal{G}_{\mathsf{uCMP}}$ is defined in a similar manner.

Theorem 10 (Integer Comparison). *There is an FSS Gate* $(\mathsf{Gen}_n^{\mathsf{sCMP}}, \mathsf{Eval}_n^{\mathsf{sCMP}})$ *for* $\mathcal{G}_{\mathsf{sCMP}}$ *that requires a single invocation of* $\mathsf{DDCF}_{n-1,\mathbb{U}_N}$*, and has a total key size of* n *bits plus key size of* $\mathsf{DDCF}_{n-1,\mathbb{U}_N}$*. Moreover, there is an FSS gate* $(\mathsf{Gen}_n^{\mathsf{uCMP}}, \mathsf{Eval}_n^{\mathsf{uCMP}})$ *for* $\mathcal{G}_{\mathsf{uCMP}}$ *with same parameters.*

Proof. We present our construction of FSS Gate for signed integer comparison formally in Figure Fig. 8 and the unsigned case is similar. Our construction relies on the following observation: for any 2 signed integers $c, d \in \mathbb{S}_N$, $\mathbf{1}\{c > d\} = 1 - \mathsf{MSB}(e)$, where $(c - d) \equiv_{\mathsf{s}} e$, as long as $|c| + |d| < N/2$ (similar relation holds for $c, d \in \mathbb{U}_N$ with $(c - d) = e$ as long as $|c - d| < N/2$). □

Signed Integer Comparison Gate $(\mathsf{Gen}_n^{\mathsf{sCMP}}, \mathsf{Eval}_n^{\mathsf{sCMP}})$

```
$\mathsf{Gen}_n^{\mathsf{sCMP}}(1^\lambda, \mathsf{r}_1^{\mathsf{in}}, \mathsf{r}_2^{\mathsf{in}}, \mathsf{r}^{\mathsf{out}})$:
 1: Let $y = (2^n - (\mathsf{r}_1^{\mathsf{in}} - \mathsf{r}_2^{\mathsf{in}})) \in \mathbb{U}_N$ and $\alpha^{(n-1)} = y_{[0,n-1)}$
 2: $(k_0^{(n-1)}, k_1^{(n-1)}) \leftarrow \mathsf{Gen}_{n-1}^{\mathsf{DDCF}}\left(1^\lambda, \alpha^{(n-1)}, \beta_1, \beta_2, \mathbb{U}_N\right)$, where $\beta_1 = 1 \oplus y_{[n-1]}, \beta_2 = y_{[n-1]} \in \mathbb{U}_N$.
 3: Sample random $r_0, r_1 \leftarrow \mathbb{U}_N$ s.t. $r_0 + r_1 = \mathsf{r}^{\mathsf{out}}$.
 4: For $b \in \{0,1\}$, let $k_b = k_b^{(n-1)} || r_b$.
 5: return $(k_0, k_1)$.

$\mathsf{Eval}_n^{\mathsf{sCMP}}(b, k_b, x, y)$:
 1: Parse $k_b = k_b^{(n-1)} || r_b$.
 2: Set $z = (x - y) \in \mathbb{U}_N$.
 3: Set $m_b^{(n-1)} \leftarrow \mathsf{Eval}_{n-1}^{\mathsf{DDCF}}(b, k_b^{(n-1)}, z^{(n-1)})$, where $z^{(n-1)} = 2^{n-1} - z_{[0,n-1)} - 1$.
 4: return $b - (b \cdot z_{[n-1]} + m_b^{(n-1)} - 2 \cdot z_{[n-1]} \cdot m_b^{(n-1)}) + r_b$.
```

Fig. 8: FSS Gate for Signed Integer Comparison $\mathcal{G}_{\mathsf{sCMP}}$, b refers to party id.

7 FSS Barrier for Fixed-Point Multiplication

In the previous section, 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 $\mathcal{F}_{n,\mathbb{U}_N}^{\wedge}$ 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* $\mathcal{F}_{n,\mathbb{U}_N}^{\wedge}$ *of conjunction functions is*

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

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 [22,38]. A construction with inverse-polynomial correctness error can be obtained from the Decisional Diffie-Hellman (DDH) assumption [19] or from the Paillier assumption [41]. 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 and we describe the changes needed for this reduction at the end of this section.

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.

$$x' = \begin{array}{|c|c|c|c|c|c|c|c|}\hline \overbrace{0\cdots0}^{m} & x_{[n-1]} & \overbrace{0\cdots0}^{m} & x_{[n-2]} & \overbrace{0\cdots0}^{m} & \cdots & \overbrace{0\cdots0}^{m} & x_{[0]} \\ \hline \end{array}$$

$$c_S = \begin{array}{|c|c|c|c|c|c|c|c|c|}\hline \overbrace{0\cdots0}^{\ell} & s_0 & \overbrace{0\cdots0}^{m} & s_1 & \overbrace{0\cdots0}^{m} & \cdots & \overbrace{0\cdots0}^{m} & s_{n-1} & \overbrace{0\cdots0}^{m-\ell} \\ \hline \end{array}$$

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 11. 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 11. *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. Let $(\mathsf{Gen}^{\times\mathsf{MSB}}_{n(m+1)}, \mathsf{Eval}^{\times\mathsf{MSB}}_{n(m+1)})$ be an FSS scheme for $\mathcal{F}^{\times\mathsf{MSB}}_{n(m+1),\mathbb{U}_N}$.
For $\mathsf{Gen}^{\times\mathsf{MSB}}_{n(m+1)}(1^\lambda, f_{c_S}, \mathbb{U}_N) = (k_0, k_1)$, we have that $\sum_{b=0}^{1} \mathsf{Eval}^{\times\mathsf{MSB}}_{n(m+1)}(b, k_b, x') = \mathsf{MSB}(x' \cdot c_S) \in \mathbb{U}_N$ (multiplication over $\mathbb{U}_{2^{n(m+1)}}$), where the description of $f_{c_S} : \mathbb{U}_{2^{n(m+1)}} \to \mathbb{U}_N$ defines c_S and $x' \in \mathbb{U}_{2^{n(m+1)}}$. We construct an FSS scheme $(\mathsf{Gen}^{\wedge}_n, \mathsf{Eval}^{\wedge}_n)$ for $\mathcal{F}^{\wedge}_{n,\mathbb{U}_N}$ as follows.

```
Gen^∧_n(1^λ, f_S, U_N) :
 1: Parse $f_S : \{0,1\}^n \to \mathbb{U}_N$ and $S \subseteq [n]$. Let $h = |S|$ and $\ell = \lceil \log h \rceil$.
 2: For $i \in S$, set $s_i > 0$ as an arbitrary positive integer such that $\sum_{i \in S} s_i = 2^\ell$. In the other case when
    $i \notin S$, set $s_i = 0$.
 3: Let $c_S = 2^{n(m+1)-\ell-1} \cdot \sum_{i=0}^{n-1} s_i \cdot 2^{-i(m+1)} \in \mathbb{U}_{2^{n(m+1)}}$
 4: Sample FSS keys $(k_0, k_1) \leftarrow \mathsf{Gen}^{\times\mathsf{MSB}}_{n(m+1)}(1^\lambda, f_{c_S}, \mathbb{U}_N)$
 5: Output $(k_0, k_1)$.
Eval^∧_n(b, k_b, x) :
 1: Parse the input $x$ as $x_{[n-1]} || x_{[n-2]} || \cdots || x_{[1]} || x_{[0]}$.
 2: Encode $x$ as $x' = \sum_{j=0}^{n-1} x_{[j]} \cdot 2^{j(m+1)}$ (i.e., separating the bits of $x$ each by $m$ 0s).
 3: Output $\mathsf{Eval}^{\times\mathsf{MSB}}_{n(m+1)}(b, k_b, x')$.
```

We prove that $(\mathsf{Gen}^{\wedge}_n, \mathsf{Eval}^{\wedge}_n)$ is a secure FSS scheme for $\mathcal{F}^{\wedge}_{n,\mathbb{U}_N}$. By the security of the underlying FSS scheme for $\mathcal{F}^{\times\mathsf{MSB}}_{n(m+1),\mathbb{U}_N}$, it holds that the key k_b is indistinguishable from a simulated output generated from only the leakage $(\mathbb{U}_{2^{n(m+1)}}, \mathbb{U}_N)$, corresponding to the input and output spaces of f_{c_S}, which in turn is simulatable given $(\mathbb{U}_{2^n}, \mathbb{U}_N)$, corresponding to the input and output spaces of the secret function f_S.

It remains thus to prove correctness. Observe that

$$\begin{aligned} x' \cdot c_S &= 2^{n(m+1)-\ell-1} \left(\sum_{i=0}^{n-1} s_i \cdot 2^{-i(m+1)} \cdot \sum_{j=0}^{n-1} x_{[j]} \cdot 2^{j(m+1)} \right) \\ &= 2^{n(m+1)-\ell-1} \left(\begin{matrix} \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} s_i \cdot x_{[j]} \cdot 2^{(j-i)\cdot(m+1)} + \sum_{i=0}^{n-1} s_i \cdot x_{[i]} \\ + \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} s_i \cdot x_{[j]} \cdot 2^{(j-i)\cdot(m+1)} \end{matrix} \right) \end{aligned}$$

Now, we argue that only the second term in the summation matters for $\mathsf{MSB}(x' \cdot c_S)$. Note that the third term can also be written as: $2^{n(m+1)-\ell-1}\big(2^{m+1} \cdot \sum_{i=0}^{n-1} \sum_{k=0}^{n-i-2} s_i \cdot x_{[i+k+1]} \cdot 2^{k(m+1)}\big) \equiv 0 \in \mathbb{U}_{2^{n(m+1)}}$ because $m \geqslant \ell$. In a similar way, the first term can be rewritten as: $2^{n(m+1)-\ell-1}\big(2^{-m-1} \cdot \sum_{i=0}^{n-1} \sum_{k=0}^{i-1} s_i \cdot x_{[i-k-1]} \cdot 2^{-k(m+1)}\big)$. Here, observe that value of k decides the bit position where components of this term get added and also how many times they get added. For $k = 0$, the number of components added is maximum, which is precisely $n-1$, and their sum $\sum_{i=1}^{n-1} s_i \cdot x_{[i-1]} \leqslant 2^\ell < 2^{m+1}$. Pictorially, for $k = 0$, these components are getting added at the position where s_1 sits in c_S and since, their sum is less than 2^{m+1}, it will not carry into the upper $\ell + 1$ bits due to the m intermediate zeros. Similarly for $k = 1$, the terms are pictorially added where s_2 sits and the summation will not carry into the bit positions where the terms of $k = 0$ get added and so on.

Therefore, neither the first nor the third term affects the value in the upper $\ell + 1$ bits of $(x' \cdot c_S)$, and hence, the following holds:

$$\mathsf{MSB}(x' \cdot c_S) = \mathsf{MSB}\left(2^{n(m+1)-\ell-1}\left(\sum_{i=0}^{n-1} s_i \cdot x_{[i]}\right)\right) = \mathsf{MSB}\left(2^{n(m+1)-\ell-1}\left(\sum_{i \in S} s_i \cdot x_{[i]}\right)\right)$$

This value is 1 exactly if all bits in x corresponding to the set S are 1, i.e. $x_{[i]} = 1$ for $i \in S$. Therefore, $\mathsf{MSB}(x' \cdot c_S) = f_S(x)$. $\square$

Step two of the barrier result. We now present the second part, i.e., a reduction from the FSS scheme for $\mathcal{F}^{\times\mathsf{MSB}}_{\eta,\mathbb{U}_N}$ to $\mathcal{G}_{\mathsf{uFPM}}$ over $\mathbb{U}_{2^\eta}$. 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 $\mathbb{U}_{2^\eta}$ 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.

Theorem 12. *Assume the existence of an FSS gate $\mathcal{G}_{\mathsf{uFPM}}$ over $\mathbb{U}_{2^\eta}$ for unsigned fixed-point multiplication. Then there exists an FSS scheme for $\mathcal{F}^{\times\mathsf{MSB}}_{\eta,\mathbb{U}_N}$, where $n \leqslant \eta$.*

Proof. Let $(\mathsf{Gen}^{\mathsf{uFPM}}_{\eta,s}, \mathsf{Eval}^{\mathsf{uFPM}}_{\eta,s})$ be an FSS gate for unsigned fixed-point multiplication, where s is the parameter that specifies the right shift amount, and both the input and output groups are $\mathbb{U}_{2^\eta}$. For $\mathsf{Gen}^{\mathsf{uFPM}}_{\eta,s}(1^\lambda, \mathsf{r}^{\mathsf{in}}_1, \mathsf{r}^{\mathsf{in}}_2, \mathsf{r}^{\mathsf{out}}) = (k_0, k_1)$, we have that $\sum_{b=0}^{1} \mathsf{Eval}^{\mathsf{uFPM}}_{\eta,s}(b, k_b, x, y) = (((x - \mathsf{r}^{\mathsf{in}}_1) \cdot (y - \mathsf{r}^{\mathsf{in}}_2)) \gg_L s) + \mathsf{r}^{\mathsf{out}} \in \mathbb{U}_{2^\eta}$ (multiplication over $\mathbb{U}_{2^\eta}$), where $x, \mathsf{r}^{\mathsf{in}}_1, \mathsf{r}^{\mathsf{in}}_2, \mathsf{r}^{\mathsf{out}} \in \mathbb{U}_{2^\eta}$. We construct an FSS scheme $(\mathsf{Gen}^{\times\mathsf{MSB}}_{\eta}, \mathsf{Eval}^{\times\mathsf{MSB}}_{\eta})$ for $\mathcal{F}^{\times\mathsf{MSB}}_{\eta,\mathbb{U}_N}$ as follows.

```
$\mathsf{Gen}^{\times\mathsf{MSB}}_{\eta}(1^\lambda, f_c, \mathbb{U}_N)$ :
 1: Parse $f_c : \mathbb{U}_{2^\eta} \to \mathbb{U}_N$ and $c \in \mathbb{U}_{2^\eta}$.
 2: Set $\tilde{c} = c + r$, where $r \leftarrow \mathbb{U}_{2^\eta}$ is randomly sampled.
 3: Sample FSS keys $(k_0, k_1) \leftarrow \mathsf{Gen}^{\mathsf{uFPM}}_{\eta,\eta-1}(1^\lambda, 0, r, 0)$.
 4: For $b \in \{0,1\}$, $K_b = (k_b, \tilde{c})$.
 5: Output $(K_0, K_1)$.
$\mathsf{Eval}^{\times\mathsf{MSB}}_{\eta}(b, K_b, x)$ :
 1: Parse $K_b = (k_b, \tilde{c})$.
 2: Set $z_b = \mathsf{Eval}^{\mathsf{uFPM}}_{\eta,\eta-1}(b, k_b, x, \tilde{c})$.
 3: Output $z'_b = z_b \bmod N$.
```

To argue that $(\mathsf{Gen}^{\times\mathsf{MSB}}_{\eta}, \mathsf{Eval}^{\times\mathsf{MSB}}_{\eta})$ is a secure FSS scheme for $\mathcal{F}^{\times\mathsf{MSB}}_{\eta,\mathbb{U}_N}$, we observe the following: from the security of the FSS gate $\mathcal{G}_{\mathsf{uFPM}}$ over $\mathbb{U}_{2^\eta}$, k_b looks indistinguishable from the output generated by the simulator given access to just the leakage function, i.e., input and output group $(\mathbb{U}_{2^\eta}, \mathbb{U}_{2^\eta})$, of $\mathcal{G}_{\mathsf{uFPM}}$ (which are same as the input group of $\mathcal{F}^{\times\mathsf{MSB}}_{\eta,\mathbb{U}_N}$ and therefore simulatable). The other part of the key, $\tilde{c}$, is indistinguishable from a randomly sampled element from $\mathbb{U}_{2^\eta}$, which is also simulatable with just the knowledge of the input group $\mathbb{U}_{2^\eta}$ of $\mathcal{F}^{\times\mathsf{MSB}}_{\eta,\mathbb{U}_N}$.

We now show the correctness of our reduction. Let $z = z_0 + z_1 \in \mathbb{U}_{2^\eta}$ and $z' = z'_0 + z'_1 \in \mathbb{U}_N$. Observe that $z = \sum_{b=0}^{1} \mathsf{Eval}^{\mathsf{uFPM}}_{\eta,\eta-1}(b, k_b, x, \tilde{c}) = (((x-0) \cdot (\tilde{c} - r)) \gg_L (\eta - 1)) = ((x \cdot c) \gg_L (\eta - 1)) = \mathsf{MSB}(x \cdot c)$. Since $n \leqslant \eta$, $z' = z \bmod N = \mathsf{MSB}(x \cdot c)$.

$\square$

Signed Fixed-Point Multiplication. To prove the case of signed fixed-point multiplication, the first part of the barrier result remains the same as the unsigned case captured by Theorem 11. In the second part, we show how to reduce the FSS scheme for $\mathcal{F}^{\times\mathsf{MSB}}_{\eta,\mathbb{U}_N}$ to $\mathcal{G}_{\mathsf{sFPM}}$ over $\mathbb{S}_{2^\eta}$ in an analogous manner to the unsigned case. Let $(\mathsf{Gen}^{\mathsf{sFPM}}_{\eta,s}, \mathsf{Eval}^{\mathsf{sFPM}}_{\eta,s})$ be an FSS gate for signed fixed-point multiplication, where s is the parameter that specifies the right shift amount, and both the input and output groups are $\mathbb{S}_{2^\eta}$. For $\mathsf{Gen}^{\mathsf{sFPM}}_{\eta,s}(1^\lambda, \mathsf{r}^{\mathsf{in}}_1, \mathsf{r}^{\mathsf{in}}_2, \mathsf{r}^{\mathsf{out}}) = (k_0, k_1)$, we have that $\sum_{b=0}^{1} \mathsf{Eval}^{\mathsf{sFPM}}_{\eta,s}(b, k_b, x, y) = (((x - \mathsf{r}^{\mathsf{in}}_1) \cdot (y - \mathsf{r}^{\mathsf{in}}_2)) \gg_A s) + \mathsf{r}^{\mathsf{out}} \in \mathbb{S}_{2^\eta}$ (multiplication over $\mathbb{S}_{2^\eta}$), where $x, \mathsf{r}^{\mathsf{in}}_1, \mathsf{r}^{\mathsf{in}}_2, \mathsf{r}^{\mathsf{out}} \in \mathbb{S}_{2^\eta}$. All the steps in $\mathsf{Gen}^{\times\mathsf{MSB}}_{\eta}(1^\lambda, f_c, \mathbb{U}_N)$ are same as the unsigned case (Theorem 12) except how FSS keys are generated, i.e., $(k_0, k_1) \leftarrow \mathsf{Gen}^{\mathsf{sFPM}}_{\eta,\eta-1}(1^\lambda, 0, r, 0)$. In $\mathsf{Eval}^{\times\mathsf{MSB}}_{\eta}(b, K_b, x)$, we need to be careful about the use of arithmetic right shift in signed fixed-point multiplication instead of logical right shift used in the unsigned case. Following the same steps as above, we get the following: $z = z_0 + z_1 = \sum_{b=0}^{1} \mathsf{Eval}^{\mathsf{sFPM}}_{\eta,\eta-1}(b, k_b, x, \tilde{c}) = (((x-0) \cdot (\tilde{c} - r)) \gg_A (\eta - 1)) = ((x \cdot c) \gg_A (\eta - 1)) = -1 \cdot \mathsf{MSB}(x \cdot c)$. Multiplying z_b with -1 followed by mod N gives correct shares of the result.

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. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: EUROCRYPT (2015)
4. Aliasgari, M., Blanton, M., Zhang, Y., Steele, A.: Secure computation on floating point numbers. In: NDSS (2013)
5. Aly, A., Smart, N.P.: Benchmarking privacy preserving scientific operations. In: ACNS 2019 (2019)
6. Araki, T., Furukawa, J., Lindell, Y., Nof, A., Ohara, K.: High-throughput semi-honest secure three-party computation with an honest majority. In: CCS (2016)
7. Atallah, M.J., Pantazopoulos, K.N., Rice, J.R., Spafford, E.H.: Secure outsourcing of scientific computations. Adv. Comput. (2001)
8. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: CRYPTO (1991)
9. Beaver, D.: Precomputing oblivious transfer. In: CRYPTO (1995)
10. Ben-Efraim, A., Lindell, Y., Omri, E.: Optimizing semi-honest secure multiparty computation for the internet. In: CCS (2016)
11. Ben-Efraim, A., Nielsen, M., Omri, E.: Turbospeedz: Double your online SPDZ! improving SPDZ using function dependent preprocessing. In: ACNS (2019)
12. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: STOC (1988)
13. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: EUROCRYPT (2011)
14. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: A framework for fast privacy-preserving computations. In: ESORICS (2008)
15. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: Efficient two-round OT extension and silent non-interactive secure computation. In: ACM CCS (2019)
16. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: CRYPTO (2019)
17. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators from ring-lpn. In: CRYPTO (2020)
18. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: EUROCRYPT (2015)
19. Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: CRYPTO (2016)
20. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: CCS (2016)
21. Boyle, E., Gilboa, N., Ishai, Y.: Secure computation with preprocessing via function secret sharing. In: TCC (2019)
22. Boyle, E., Kohl, L., Scholl, P.: Homomorphic secret sharing from lattices without FHE. In: EUROCRYPT (2019)
23. Büscher, N., Demmler, D., Katzenbeisser, S., Kretzmer, D., Schneider, T.: HyCC: Compilation of hybrid protocols for practical secure computation. In: CCS (2018)
24. Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of Cryptology (2000)
25. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: STOC (2002)
26. Catrina, O., de Hoogh, S.: Secure multiparty linear programming using fixed-point arithmetic. In: ESORICS (2010)
27. Catrina, O., Saxena, A.: Secure computation with fixed-point numbers. In: FC (2010)
28. 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)
29. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: STOC (1988)

30. Couteau, G.: A note on the communication complexity of multiparty computation in the correlated randomness model. In: EUROCRYPT, Part II (2019)
31. Cramer, R., Damgård, I., Escudero, D., Scholl, P., Xing, C.: SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In: Crypto. pp. 769–798 (2018)
32. 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)
33. 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)
34. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: CRYPTO (2012)
35. Demmler, D., Dessouky, G., Koushanfar, F., Sadeghi, A., Schneider, T., Zeitouni, S.: Automated synthesis of optimized circuits for secure computation. In: CCS (2015)
36. Demmler, D., Schneider, T., Zohner, M.: ABY-a framework for efficient mixed-protocol secure two-party computation. In: NDSS (2015)
37. Dessouky, G., Koushanfar, F., Sadeghi, A., Schneider, T., Zeitouni, S., Zohner, M.: Pushing the communication barrier in secure computation using lookup tables. In: NDSS (2017)
38. Dodis, Y., Halevi, S., Rothblum, R.D., Wichs, D.: Spooky encryption and its applications. In: CRYPTO (2016)
39. Doerner, J., Shelat, A.: Scaling ORAM for secure computation. In: CCS (2017)
40. Escudero, D., Ghosh, S., Keller, M., Rachuri, R., Scholl, P.: Improved primitives for MPC over mixed arithmetic-binary circuits. In: CRYPTO (2020)
41. Fazio, N., Gennaro, R., Jafarikhah, T., III, W.E.S.: Homomorphic secret sharing from paillier encryption. In: Provable Security (2017)
42. Goldreich, O.: Foundations of Cryptography — Basic Applications. Cambridge University Press (2004)
43. Goldreich, O., Micali, S., Wigderson, A.: How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In: STOC (1987)
44. Ishai, Y., Kushilevitz, E., Meldgaard, S., Orlandi, C., Paskin-Cherniavsky, A.: On the power of correlated randomness in secure computation. In: TCC (2013)
45. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer - efficiently. In: CRYPTO (2008)
46. Ishai, Y., Prabhakaran, M., Sahai, A.: Secure arithmetic computation with no honest majority. In: TCC (2009)
47. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In: USENIX Security (2018)
48. Katz, J., Ranellucci, S., Rosulek, M., Wang, X.: Optimizing authenticated garbling for faster secure two-party computation. In: CRYPTO (2018)
49. Kerik, L., Laud, P., Randmets, J.: Optimizing MPC for robust and scalable integer and floating-point arithmetic. In: FC (2016)
50. Kilian, J.: More general completeness theorems for secure two-party computation. In: STOC (2000)
51. 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)
52. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: ICALP (2008)
53. Kumar, N., Rathee, M., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: CrypTFlow: Secure TensorFlow Inference. In: IEEE S&P (2020)
54. LeCun, Y., Bengio, Y., Hinton, G.E.: Deep learning. Nat. (2015)
55. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious neural network predictions via minionn transformations. In: CCS (2017)
56. Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., Popa, R.A.: Delphi: A cryptographic inference service for neural networks. In: USENIX Security (2020)
57. Mohassel, P., Rindal, P.: ABY3: A mixed protocol framework for machine learning. In: CCS (2018)
58. Mohassel, P., Rosulek, M., Zhang, Y.: Fast and secure three-party computation: The garbled circuit approach. In: CCS (2015)
59. Mohassel, P., Zhang, Y.: SecureML: A system for scalable privacy-preserving machine learning. In: IEEE S&P (2017)
60. Naor, M., Pinkas, B.: Oblivious polynomial evaluation. SIAM J. Comput. **35**(5) (2006)
61. 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)
62. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: CRYPTO (2012)
63. Nishide, T., Ohta, K.: Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In: PKC (2007)

64. Ohata, S., Nuida, K.: Communication-efficient (client-aided) secure two-party protocols and its application. In: FC (2020)
65. Patra, A., Schneider, T., Suresh, A., Yalame, H.: ABY2.0: Improved mixed-protocol secure two-party computation. Cryptology ePrint Archive, Report 2020/1225 (2020)
66. Pullonen, P., Siim, S.: Combining secret sharing and garbled circuits for efficient private IEEE 754 floating-point computations. In: FC (2015)
67. Rathee, D., Rathee, M., Kumar, N., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: CrypTFlow2: Practical 2-party secure inference. In: CCS (2020)
68. 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)
69. Riazi, M.S., Weinert, C., Tkachenko, O., Songhori, E.M., Schneider, T., Koushanfar, F.: Chameleon: A hybrid secure computation framework for machine learning applications. In: AsiaCCS (2018)
70. Ryffel, T., Pointcheval, D., Bach, F.: ARIANN: Low-interaction privacy-preserving deep learning via function secret sharing. arXiv preprint:2006.04593 (2020)
71. Schoenmakers, B., Tuyls, P.: Efficient binary conversion for paillier encrypted values. In: EUROCRYPT (2006)
72. Toft, T.: Constant-rounds, almost-linear bit-decomposition of secret shared values. In: CT-RSA (2009)
73. Trieu, N., Shehata, K., Saxena, P., Shokri, R., Song, D.: Epione: Lightweight contact tracing with strong privacy. IEEE Data Eng. Bull. (2020)
74. Wagh, S., Gupta, D., Chandran, N.: SecureNN: 3-Party Secure Computation for Neural Network Training. PoPETs (2019)
75. Wang, X., Malozemoff, A.J., Katz, J.: EMP-toolkit: Efficient MultiParty computation toolkit. `https://github.com/emp-toolkit` (2016)
76. Wang, X., Ranellucci, S., Katz, J.: Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation. In: CCS (2017)
77. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast Extension for coRRElated oT with small communication. IACR Cryptol. ePrint Arch. (2020)
78. Yao, A.C.: How to generate and exchange secrets. In: FOCS (1986)
79. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole - reducing data transfer in garbled circuits using half gates. In: EUROCRYPT (2015)

A Realizing the Dealer

Protocols in this work are presented in an idealized preprocessing model, where the two parties receive correlated randomness from a trusted dealer before protocol execution. Protocols within this idealized model can be converted to protocols in the standard model, *without* a dealer, via different generic transformations. Examples of such transformations include:

1. *3-party setting:* Given 3 parties and 1 corruption, the role of the dealer can be emulated directly by the third party.
2. *2-party setting with trusted hardware:* As explored recently e.g. by Nawaz *et al.* [61], the role of the dealer can be emulated within a trusted execution environment, such as that provided by Intel SGX.
3. *2-party setting:* Additionally, the role of the dealer can be jointly emulated by the parties via a small-scale two-party secure protocol.

In this section, we discuss concrete approaches for the third path above, where two parties jointly emulate the role of dealer. Recall that this amounts to 2-party protocols for securely executing the Gen procedure for appropriate FSS gates, in turn reducing to secure generation of keys for DCF.

We focus on two specific paths of execution toward this goal: (1) a variant of the secure DPF generation protocol of Doerner and shelat [39] for small and moderate domain sizes, which is *black-box* in the underlying cryptographic pseudorandom generator (PRG); and (2) generic 2-party computation approaches, implementing the PRG via AES or targeted constructions of "MPC-friendly" ciphers. The two approaches have various tradeoffs and respective advantages:

- **Black-box protocol** (à la Doerner-shelat): Restricted to moderate domain size (e.g., 2^{16} or smaller) and semi-honest security, but outperforms alternatives within these regimes.
- **Generic 2PC**: Can be used with large domain sizes, extends efficiently to malicious security, and can obtain small constant round complexity.

A.1 Extending Doerner-shelat to DCF

The most expensive part of securely distributing DCF key generation is the corresponding required secure evaluations of the cryptographic PRG. For the special related case of generating Distributed Point Function (DPF) keys, an alternative method was given by Doerner and shelat [39], within the context of secure 2-party computation on RAM programs. While their protocol is restricted to key generation for moderate-sized domains (computation costs grow linearly with the domain size), and the approach is presently limited to the semi-honest setting, within these settings the protocol offers the significant efficiency advantage that it executes *black-box* in the PRG evaluation. That is, PRG evaluations only take place locally, and do *not* need to be securely emulated. As a result, the protocol performs significantly better within the relevant parameter regimes.

We demonstrate that the DPF key generation protocol of Doerner-shelat can be extended to the case of DCF at small additional cost. In turn, this provides an appealing solution for secure emulation of our FSS key dealer within the setting of semi-honest adversaries and moderate gate domain sizes (e.g., 2^{16} or smaller).

DCF with $\mathbb{Z}_2$ payloads. We first show that, for the special case of DCF keys with $\mathbb{Z}_2$ outputs, the Doerner-shelat protocol *without modification* already is a protocol for secure DCF key generation. This follows from two steps.

Recall that the Doerner-shelat protocol securely evaluates the Gen procedure of the specific DPF construction from [20]. We first appeal to the observation of [17] that a small modification of the corresponding DPF $\mathsf{Eval}^{\mathsf{DPF}}$ procedure yields an FSS $(\mathsf{Gen}, \mathsf{Eval}^{\mathsf{prefix}})$ for the class of *all-prefix point functions* functions with incremental evaluation. That is, for the class of functions $f_{\alpha,\bar{\beta}} : \bigcup_{\ell \in [n]} \{0,1\}^\ell \to \mathbb{Z}_2$ given by $f_{\alpha,\bar{\beta}}(x_1, \dots, x_\ell) = \beta_\ell \in \mathbb{Z}_2$ if $(x_1, \dots, x_\ell) = (\alpha_1, \dots, \alpha_\ell)$, and 0 otherwise, and where the cost of $\mathsf{Eval}^{\mathsf{prefix}}$ on an input $(x_1 \dots, x_\ell)$ is small given the output of $\mathsf{Eval}^{\mathsf{prefix}}$ on $(x_1, \dots, x_{\ell-1})$ (plus short intermediate state information). Intuitively, the desired incremental prefix evaluation output shares are *already* present within the intermediate execution values for $\mathsf{Eval}^{\mathsf{DPF}}$.[15]

As the second step, we demonstrate that an additional small modification of $\mathsf{Eval}^{\mathsf{prefix}}$ (again keeping Gen *unchanged*) yields yet another FSS scheme $(\mathsf{Gen}, \mathsf{Eval}^{\mathsf{DCF}})$, which this time provides support for *DCF with* $\mathbb{Z}_2$ *payloads*. (In fact, it even supports the more general class of decision lists with output in $\mathbb{Z}_2$.) The new $\mathsf{Eval}^{\mathsf{DCF}}$ function has roughly twice the runtime of the original $\mathsf{Eval}^{\mathsf{DPF}}$.

More concretely, consider a secret comparison function $f^{<}_{\alpha,\beta}$, with $\beta \in \mathbb{Z}_2$.

- To generate a DCF key for $f^{<}_{\alpha,\beta}$, run the all-prefix DPF key generation procedure on the function $f_{\alpha,\bar{\beta}'}$ with $\bar{\beta}' \in \mathbb{Z}_2^n$, where: (a) if $\beta = 1$ then $\beta'_i = \alpha_i$ for all $i \in [n]$, and (b) if $\beta = 0$ then $\beta'_i = 0$ for all $i \in [n]$.
- The modified $\mathsf{Eval}^{\mathsf{DCF}}$ procedure takes as input an identity bit $b \in \{0,1\}$, an incremental all-prefix DPF key k_b for $f_{\alpha,\bar{\beta}'}$ as above, and an input $x \in \{0,1\}^n$ to the DCF. Functionality-wise, it outputs the following value:
$$\mathsf{Eval}^{\mathsf{DCF}}(b, k_b, x) = \sum_{i=1}^{n} \mathsf{Eval}^{\mathsf{prefix}}(b, k_b, (x_1, \ldots, \overline{x_i})). \tag{2}$$
Computed naively, the expression in Equation 2 would take *quadratic* time $O(\lambda n^2)$ to evaluate; however, leveraging the incremental evaluation property of the all-prefix DPF scheme, then this value can be computed in time comparable to *two* executions of the original DPF evaluation, $\mathsf{Eval}^{\mathsf{DPF}}$.

Security of the construction follows directly by the underlying all-prefix DPF security. Correctness holds via the following argument. If $\beta = 0$, then correctness follows directly, as all prefix evaluations (and thus their sums) evaluate to 0, as desired. Suppose then $\beta = 1$. Consider an arbitrary input $x = (x_1 \ldots, x_n)$, and let $\ell \in [n]$ denote the unique index for which $(x_1, \ldots, x_{\ell-1}) = (\alpha_1, \ldots, \alpha_{\ell-1})$ but $x_\ell = 0$ and $\alpha_\ell = 1$. Thus, $f_{\alpha,\bar{\beta}'}(x_1, \ldots, \overline{x}_\ell) = \beta'_\ell = \alpha_\ell$ (i.e., this modified prefix of x exactly matches the prefix of α), but $f_{\alpha,\bar{\beta}'}(x_1, \ldots, \overline{x}_i) = 0$ for all other indices $i \neq \ell \in [n]$. Taking the sum, this means $\mathsf{Eval}^{\mathsf{DCF}}$ on input x outputs additive shares of the bit α_ℓ for this index ℓ. If $\alpha_\ell = 0$ (and thus $x_\ell = 1$), this implies $x > \alpha$, and we indeed do want $f^{<}_{\alpha,\beta}(x) = 0$; similarly, if $\alpha_\ell = 1$ (and thus $x_\ell = 0$), this implies $x < \alpha$, and we again do want $f^{<}_{\alpha,\beta}(x) = 1$. In both cases, α_ℓ is the desired output.

Combining the two steps above yields secure DCF generation *immediately* via the Doerner-shelat protocol (as the resulting Gen algorithms to be securely computed are identical). This provides a convenient state of affairs, where existing, optimized implementations of their protocol can be used directly within our new contexts.

In terms of efficiency, the total computation time of the protocol to distribute DCF Gen with $\mathbb{Z}_2$ payloads is dominated by $(n+1)$ string OTs for 129-bit strings and 2^{n+1} locally computed AES operations; the communication is dominated by the $(n+1)$ string OTs. For example, DCF with $\mathbb{Z}_2$ output and domain size 2^{16} (i.e., $n = 16$) corresponds to 17 string OTs (129-bit strings) and 2^{17} local AES computations. The cost of the string OTs can be amortized over many FSS key generation computations (e.g., over the different gates of a given circuit), via OT extension. IKNP-based OT extension, given a one-time cost of base OTs (say roughly 20ms), can yield rates of over 500,000 OTs per second [62], translating here to less than 0.05ms amortized OT cost per DCF key. Recent developments in "silent OT" and extensions [15, 77] can further improve these estimates. For local computation, using an estimate of 360 million AES calls per second on a single-core 3.6 GHz machine (10 machine cycles per AES) [73], local computation will take roughly 0.36ms per DCF key. Given the large parallelism of the respective AES evaluations (2^j parallel calls per level j), executing with multiple cores can further provide significant efficiency benefits.

DCF for general payloads. To extend to more general output spaces, beyond $\mathbb{Z}_2$, we must look more closely at the specific structure of the Doerner-shelat protocol.

The core challenge faced in this setting is that within Gen, the parties must securely evaluate a PRG on specific values $s_b^i \in \{0,1\}^\lambda$ that need to remain secret from both of the parties (as their identity reveals

[15] For those familiar with the DPF construction of [20]: The prefix-evaluation output share at an input $(x_1, \ldots, x_\ell)$ is precisely the t bit generated in the ℓth level evaluation of $\mathsf{Eval}^{\mathsf{DPF}}$ on an input with this prefix. As part of the DPF construction, these t bits are guaranteed to be opposite across the parties for any partial evaluation that remains on the special path, and are guaranteed to agree at any point off the path.

information about the secret input value α). The main insights of Doerner and shelat are that: (1) each s_b^i is directly computable by party $b \in \{0,1\}$, but must be hidden from within a list of 2^i possible values $L_b \in (\{0,1\}^\lambda)^{2^i}$, (2) the lists L_0, L_1 of parties $b = 0,1$ agree aside from this one pair of values s_0^i, s_1^i, and (3) the computation in Gen in fact only relies on the *xor* $G(s_0^i) \oplus G(s_1^i)$, and not on the individual PRG evaluations themselves. Combining these ideas, they designed a protocol which removes the PRG evaluations from the secure computation and replaces them with 2^i *local* PRG evaluations per level: reaching the target value as a simple computation $G(s_0^i) \oplus G(s_1^i) = \left(\sum_{s \in L_0} G(s)\right) \oplus \left(\sum_{s \in L_1} G(s)\right)$.

We observe that the same insights apply also to the case of our Distributed *Comparison* Function (DCF) key generation, and give an extension of the Doerner-shelat procedure to support secure computation of $\mathsf{Gen}^{\mathsf{DCF}}$. Consider a DCF $f_{\alpha,\beta} : \{0,1\}^n \to \mathbb{G}$, where parties enter with secret shares of input $\alpha \in \{0,1\}^n$ and payload $\beta \in \mathbb{G}$ over their respective groups (in particular, *bitwise* shares of α). In the pseudocode that follows, we associate each node w of a binary tree with a seed s_b^w (one for each party $b \in \{0,1\}$), where w indicates the bit string of length $\leqslant n$ that uniquely identifies the given node. By convention, we take $w = \emptyset$ to represent the root node. We provide the pseudocode of the corresponding protocol in Fig. 9.

The execution time for the more general protocol will be comparable to the baseline Doerner-shelat execution, with an appropriate overhead for the larger output group $\mathbb{G}$. This has two contributions: (1) Lengthening the output of the string OTs by $2\lceil \log |\mathbb{G}| \rceil$ additional bits (for computing V values as well; see Fig. 9). (2) Greater expansion required of the PRG to $2(\lambda + 1 + \lceil \log |\mathbb{G}|)$ bits, emulatable, e.g., using $2 + \lceil 2 \log |\mathbb{G}|/128 \rceil$ AES calls in the place of 2. For group size $\mathbb{G}$ up to 128 bits, the overall effect will be relatively minor.

A.2 Distributed Generation via Generic 2PC

Alternatively, one may jointly emulate the dealer's role of generating DCF keys via generic secure 2-party computation. Communication and computation requirements of the corresponding protocol will be dominated by the $(n+1)$ secure evaluations of the length-doubling PRG.

Setting $\lambda = 127$ and instantiating the PRG via two AES evaluations, as suggested previously, results in a necessary $2(n+1)$ secure evaluations of AES (with secret-shared inputs and outputs). Depending on the hardware, network, and on whether one targets semi-honest or malicious security, the throughput for state-of-the-art secure 2PC of AES via garbled circuits is 100-1000 instances per second [48, 76], with communication roughly 200kB per instance. As an example for ReLU on domain size 2^{16}, requiring 34 secure AES evaluations, secure key generation is estimated to take roughly 35-300ms.

The generation numbers can be further improved using an MPC-friendly PRG with few AND gates instead of AES; e.g., using LowMC [3] would give approximately a $\times$10-20 time improvement for communication and computation of the setup. Although usage of non-AES ciphers incurs an efficiency tradeoff between secure key generation and online computation of Eval, since (unlike AES) their operations are not supported in native hardware.

B Malicious Security

All protocols described up to this point are only secure against *semi-honest* parties. In this section we sketch a simple modification of these protocols that achieves security against malicious parties, under the assumption that the correlated randomness setup is performed correctly.[16] To secure the protocols against malicious behavior by one of the two parties we follow the arithmetic MAC approach of [13, 34].

First, we previously viewed each FSS gate as translating a public masked input on each input wire and additive secret shares of the mask to a public masked value and additive shares of the mask on the output wire. It will be more convenient to consider the values on every wire as additive shares of the wire value. These shares are obtained by one party setting its share to minus its share of the mask and the other party

[16] Such a trusted setup can be achieved either by assuming a semi-trusted dealer (e.g., implemented with limited trusted hardware [61]), or a two-party maliciously secure preprocessing protocol. The latter can be done quite efficiently via state-of-the-art generic 2PC protocols (e.g., [48]). Potential efficiency improvements can be obtained via recent techniques for making the Doerner-shelat protocol for distributed DPF key generation maliciously secure [15, 17]. We leave the question of optimizing the concrete cost of such setup protocols to future work.

Protocol: Secure Distributed DCF Generation
Inputs: Each party holds additive shares of $\alpha \in \{0,1\}^n$ (bitwise) and $\beta \in \mathbb{G}$.
Output: DCF keys for $f^{<}_{\alpha,\beta}$

Each party $b \in \{0,1\}$ performs the following:

```
1: Sample $s_b^\emptyset \leftarrow \{0,1\}^\lambda$, set $t_b^\emptyset = b$, and set additive share of $V_\alpha$ to $0 \in \mathbb{G}$.
2: for $j = 1, \dots, n$ do
3:     Initialize $(V_b^L || V_b^R) = 0||0$ and $(s_b^L || s_b^R) = 0||0$
4:     for $w \in \{0,1\}^{j-1}$ do
5:         Compute $(s_b^{w,L} || v_b^{w,L} || t_b^{w,L} \; || \; s_b^{w,R} || v_b^{w,R} || t_b^{w,R}) = G(s_b^w)$
6:         Update $(s_b^L || t_b^L \; || \; s_b^R || t_b^R) \oplus = (s_b^{w,L} || t_b^{w,L} \; || \; s_b^{w,R} || t_b^{w,R})$
7:         Update $V_b^L \mathrel{+}= \mathsf{Convert}_{\mathbb{G}}(v_b^{w,L})$, and $V_b^R \mathrel{+}= \mathsf{Convert}_{\mathbb{G}}(v_b^{w,R})$
8:          // Note: for any $w \neq (\alpha_1, \dots, \alpha_{j-1})$, the parties' updates cancel each other
9:     end for
10:    Secure Computation:
         – Inputs: shares of $\alpha$, shares of $\beta$, $(s_b^L || t_b^L \; || \; s_b^R || t_b^R)$, $(V_b^L || V_b^R)$, $(t_b^w)_{w \in \{0,1\}^{j-1}}$ (from previous level), and
           shares of $V_\alpha$
         – Compute:
```

$$t^* \leftarrow t_1^{(\alpha_1, \dots, \alpha_{j-1})},$$

$$(s^{CW}, t^L_{CW}, t^R_{CW}) \leftarrow \begin{cases} \big((s_0^R \oplus s_1^R), (t_0^L \oplus t_1^L \oplus 1), (t_0^R \oplus t_1^R)\big) & \text{if } \alpha_j = 0 \\ \big((s_0^L \oplus s_1^L), (t_0^L \oplus t_1^L), (t_0^R \oplus t_1^R \oplus 1)\big) & \text{if } \alpha_j = 1 \end{cases},$$

$$V_{CW} \leftarrow \begin{cases} (-1)^{t^*} \cdot \big[(V_1^R - V_0^R) - V_\alpha\big] & \text{if } \alpha_j = 0 \\ (-1)^{t^*} \cdot \big[(V_1^L - V_0^L) - V_\alpha + \beta\big] & \text{if } \alpha_j = 1 \end{cases},$$

$$V_\alpha \leftarrow \begin{cases} V_\alpha - V_1^L + V_0^L + (-1)^{t^*} \cdot V_{CW} & \text{if } \alpha_j = 0 \\ V_\alpha - V_1^R + V_0^R + (-1)^{t^*} \cdot V_{CW} & \text{if } \alpha_j = 1 \end{cases}$$

```
         – Output: $CW^j := (s^{CW} || V_{CW} || t^L_{CW} || t^R_{CW})$ to both, and additive shares of $V_\alpha$
11:    for $w \in \{0,1\}^{j-1}$ do   // Compute $s, t$ values for each node in next level:
12:        Compute $(s_b^{w||0} || s_b^{w||1}) = (s_b^{w,L} || s^{w,R}) \oplus t_b^w \cdot (s^{CW} || s^{CW})$
13:        Compute $(t_b^{w||0} || t_b^{w||1}) = (t_b^{w,L} || t^{w,R}) \oplus t_b^w \cdot (t^L_{CW} || t^R_{CW})$
14:    end for
15: end for
16: Output $k_b = s_b^\emptyset || CW^1 || \cdots || CW^{n+1}$
```

Fig. 9: Extension of Doerner-shelat [39] protocol to secure 2-party generation of DCF keys as per $\mathsf{Gen}_n^<$ in Fig. 1.

setting its share to the masked input minus its share of the mask. The second, and more crucial observation is that all the families of offset functions, for which we construct FSS gates, are closed under multiplication by scalar. For instance, given a comparison function with output x in a group, there is another comparison function in the family with output αx for some scalar α.

We begin by describing a maliciously secure protocol that makes the assumption that all wire values in the protocol are defined over a large field $\mathbb{F}$. Our protocol is similar to the MAC batching protocol in [34]. We then explain how to construct a similar protocol over $\mathbb{Z}_N$ for $N = 2^n$, which is particularly important in light of the output domain for most of the FSS gates in this paper. This second protocol we propose is similar to MAC batching in [31]. Another standard assumption that we make is that the parties can efficiently sample a shared random source with security against a malicious adversary.

Preprocessing phase:

1. Choose a random element $\alpha \in \mathbb{F}$ and create additive secret shares $\alpha = \alpha_1 + \alpha_2$ and $\alpha^2 = \beta_1 + \beta_2$.
2. For every FSS gate in the semi-honest protocol realizing a function $f(x, y)$, create two FSS gates: the original gate for f, and a gate for the function f_α from the same family with output $f_\alpha(x, y) = \alpha f(x, y)$.
3. Distribute the shares of α, α^2, and the FSS keys for all the FSS gates to the two parties.

In the computation phase, the two parties run the semi-honest protocol for every gate, such that on each output wire w with inputs to the gate x_w, y_w the parties in an honest execution hold shares of $f(x_w, y_w)$ and $f_\alpha(x_w, y_w)$. Assume that the shares on all wires of party $b \in \{1, 2\}$ are $s_{b,1}, \ldots, s_{b,m}$ for m gates of functions f and $t_{b,1}, \ldots, t_{b,m}$ for m gates of functions f_α. The two parties complete the execution with a verification step that either outputs the value of the circuit or aborts the computation.

Verification phase:

1. Jointly and securely sample m random non-zero field elements $r_1, \ldots, r_m \in \mathbb{F} \setminus \{0\}$.
2. The first party computes locally $S_1 = \sum_{i=1}^{m} r_i s_{1,i}, T_1 = \sum_{i=1}^{m} r_i t_{1,i}$ and the second party computes locally $S_2 = \sum_{i=1}^{m} r_i s_{2,i}, T_2 = \sum_{i=1}^{m} r_i t_{2,i}$.
3. The two parties run a maliciously secure 2PC which checks two conditions. If either of them fails then the protocol aborts, and otherwise the protocol returns the sum of the shares on the output wire.
   (a) $(\alpha_1 + \alpha_2)^2 = \beta_1 + \beta_2$
   (b) $(\alpha_1 + \alpha_2)(S_1 + S_2) = T_1 + T_2$

The verification step uses a maliciously secure 2PC as a black box, but since the protocol executes on a constant number of field elements, it will be very efficient. Note that if the first check on α and α^2 is not performed then the adversary can execute a type of selective failure attack that tests if all the wire values are zero. It changes α arbitrarily and then all the wire values are zero if and only if the verification does not abort.

It follows from the definition of the verification step that it correctly returns the output after an honest execution of the protocol. We next prove that any malicious behavior by one of the parties results in an abort with high probability.

Lemma 5. *Any dishonest behavior in the computation phase of the protocol by one of the parties results in an abort in the verification step with probability at least* $1 - \frac{3}{|\mathbb{F}|-1} - \mu(1^\lambda)$*, for a negligible function* μ *and the security parameter* λ*.*

Proof. We begin by assuming that every FSS gate in the circuit is information-theoretically secure. That is, the distributions induced by the two experiments Real_λ and Ideal_λ in Definition 2 are identical. We further assume that the malicious party is the first party, which we can do wlog since the security argument is symmetrical for both parties.

If the first party changes the shares of α or α^2 that it enters into the final computation then its inputs are γ_1 instead of α_1 and δ_1 instead of β_1. In order to pass the first verification check it must hold that $(\gamma_1 + \alpha_2)^2 = \delta_1 + \beta_2$ in addition to the fact that $(\alpha_1 + \alpha_2)^2 = (\beta_1 + \beta_2)$. Subtracting the two equations we have that

$$\alpha_2(\alpha_1 - \gamma_1) = \beta_1 - \delta_1 + \gamma_1^2 - \alpha_1^2.$$

If $\alpha_1 = \gamma_1$ then $\beta_1 = \delta_1$ which contradicts the assumption on malicious behavior. Otherwise, the first party can correctly predict α_2. Since α is random in $\mathbb{F}$ the first party can predict the value of α_2 with probability at most $1/|\mathbb{F}|$.

Now, assume that both parties entered the correct shares of α and α^2, but that the first party behaved maliciously during the computation phase. Observe that the only communication between the parties in the computation phase occurs at the output of an FSS gate when the two parties convert their shares of the output and shares of the mask to a public masked output. They achieve that by locally adding the shares of the output and the mask and then exchanging the shares of the masked output. Assume that the first time the first party behaves maliciously was when exchanging values for the i-th wire. Then, the only way it can influence the share of the second party on the i-th wire is to add to its real share some additive value, by changing the reported value of the first party's share of the masked value on this wire.

The result of this attack is that the f value of the wire is $s_{1,i} + s_{2,i} + \gamma$ and the value on the f_α wire is $t_{1,i} + t_{2,i} + \delta$ for some γ, δ which are not both zero. To pass the second check in the verification phase, it holds that $\alpha(S_1 + S_2) - (T_1 + T_2) = 0$ or that $\alpha r_i \gamma - r_i \delta + c = 0$, for some value c that we can assume that the adversary knows. If $\gamma = 0$ and $c = 0$ then $\delta = 0$ which contradicts the assumption on malicious behavior. If $\gamma = 0$ and $c \neq 0$ then $r_i \delta = c$. Since r_i is chosen independently of δ and c from $\mathbb{F} \setminus \{0\}$, the adversary chooses δ correctly with probability at most $\frac{1}{|\mathbb{F}|-1}$. Finally, if $\gamma \neq 0$ then the adversary can compute α, which can only happen with probability at most $1/|\mathbb{F}|$.

Taking the union bound over all possible attacks gives the adversary a success probability of at most $\frac{3}{|\mathbb{F}|-1}$ when the distributions of Real_λ and Ideal_λ are identical for every FSS gate.

In the general case, the distributions of Real_λ and Ideal_λ are computationally indistinguishable in each of the $2m$ FSS gates in the circuit. We use a standard hybrid argument, constructing distributions Hyb_j, $j = 0, 1, \ldots, 2m$, such that in Hyb_j the distributions of the keys of the first j gates are given by Ideal_λ, and the distribution of the rest of the keys is given by Real_λ. Therefore, Hyb_{2m} is the distribution of the ideal experiment on all FSS gates, and Hyb_0 is the distribution of the real experiment on all FSS keys, with independent key generation for each gate. Since every mask is random and independent, and the only communication between the parties is shares of these masks, Hyb_0 is distributed identically to a party's view during the execution of the protocol. By definition, any polynomial time adversary can distinguish between Real_λ and Ideal_λ with negligible probability, and can therefore distinguish between Hyb_j and Hyb_{j+1} with some negligible probability $\mu'(1^\lambda)$. As a consequence, that same adversary can distinguish between Hyb_0 and Hyb_{2m} with probability at most $\mu(1^\lambda) = m\mu'(1^\lambda)$, which is negligible if m is a polynomial in the security parameter. Since any malicious behavior by the adversary in the ideal model succeeds with probability at most $1 - \frac{3}{|\mathbb{F}|-1}$ in the ideal model, it succeeds with probability at most $1 - \frac{3}{|\mathbb{F}|-1} - \mu(1^\lambda)$ in the real model. $\square$

The protocol we described so far relies on the fact that $\mathbb{F}$ is a field, and therefore all elements have an inverse. In the domain $\mathbb{Z}_N, N = 2^n$, that is not the case. For example, using the protocol we described with operations over $\mathbb{Z}_N$ allows an adversary to cheat with good probability by choosing certain inputs, e.g. $x = 2^{n-1}$.

The protocol achieving malicious security in $\mathbb{Z}_N$ uses the method of Cramer et al. in [31]. The idea is to choose a security parameter s, and then to "lift" the computation to $\mathbb{Z}_{N'}$ for $N' = 2^{n+s}$. The input and output domains of all FSS gates must be changed to $\mathbb{Z}_{N'}$, which is possible for all our schemes. For example, inputs $\alpha, \beta \in \mathbb{Z}_N$ for a DCF are padded with extra zeroes and viewed as elements of $\mathbb{Z}_{N'}$.

Then, run the previous verification protocol over $\mathbb{Z}_{N'}$. In further detail, in the preprocessing phase choose a random element $\alpha \in \mathbb{Z}_{N'}$ and create additive secret shares $\alpha \equiv \alpha_1 + \alpha_2 \bmod N'$ and $\alpha^2 \equiv \beta_1 + \beta_2 \bmod N'$. Create FSS gates for the functions $f(x, y)$ and $f_\alpha(x, y) = \alpha f(x, y)$ and distribute the shares to the parties. Then, in the verification phase securely sample m public random ring elements $r_1, \ldots, r_m \in \mathbb{Z}_{N'}$. Party $b \in \{0, 1\}$ computes S_b, T_b as in the protocol over fields, and both together check that the conditions

1. $(\alpha_1 + \alpha_2)^2 = \beta_1 + \beta_2$
2. $(\alpha_1 + \alpha_2)(S_1 + S_2) = T_1 + T_2$

hold.

If the FSS gates are information-theoretically secure then the probability of an adversary acting maliciously without the verification step leading to an abort is at most $2^{-s+\log(s+1)}$ [31]. Therefore, for any computationally secure FSS gates, and a polynomial size circuit, the probability of dishonest computation by one of the parties without the other aborting is at most $2^{-s+\log(s+1)} + \mu(1^\lambda)$ for a negligible function μ.

C Comparison with Garbled Circuits

C.1 Garbled Circuits in the Preprocessing Model

Recall that secure computation with preprocessing involves a trusted dealer giving out correlated randomness to the parties in an input independent setup phase. This is succeeded by a highly efficient online phase where parties use this correlated randomness to perform secure computation on their inputs. We use the term "offline storage" to refer to the correlated randomness that a party needs to store. Similar to our work, we assume that the parties start with additive shares of the input to garbled circuit and learn additive shares of the output from it. Note that this is required for chaining different protocols together as a part of the widely-used mixed-mode computation paradigm [36, 57]. In this section, we describe in detail how we use Garbled Circuits (GC) [78] in the preprocessing model when comparing it with our work in Table 1. In Table 2 (Appendix C.2), we provide a version of Table 1 with exact key size expressions. For completeness, we also provide a comparison with GC for per party offline storage and online AES calls in Appendix C.2.

Let the functionality being computed with GC be $f : \{0,1\}^{n_1} \rightarrow \{0,1\}^{n_2}$, where $y = f(x)$, (x_0, x_1) be additive shares of x and (y_0, y_1) be additive shares of y. We assume that $\{0,1\}^n$ represents the binary representation of elements of $\mathbb{Z}_N$. We denote the garbler by P_0 and the evaluator by P_1.

Setup Phase. The trusted dealer generates a garbled circuit C_g for the functionality $g : \{0,1\}^{n_1} \rightarrow \{0,1\}^{n_2}$, where $g(x) = f(x - r_1) - r_0$ and r_0, r_1 are sampled randomly from $\{0,1\}^{n_2}$, $\{0,1\}^{n_1}$, respectively. Let $\{(w_i^0, w_i^1)\}_i$ be the pair of labels corresponding each input wire i in C_g. She sends $(\{(w_i^0, w_i^1)\}_i, r_0)$ to P_0 and (C_g, r_1) to P_1.

Online Phase. Let x_0, x_1 be the shares of x held by P_0 and P_1, respectively. The online phase proceeds in 2 sequential communication rounds. In the first round, P_1 sends x_1+r_1 to P_0. P_0 computes $\tilde{x} = x_0+x_1+r_1 = x+r_1$ and sends the wire labels corresponding to $\tilde{x}$, i.e. $\{w_i^{\tilde{x}[i]}\}_i$, to P_1 in the second round. P_1 evaluates C_g using the labels $\{w_i^{\tilde{x}[i]}\}_i$ to learn $y_1 = g(\tilde{x}) = g(x + r_1) = f(x) - r_0$. P_0 sets $y_0 = r_0$.

In the above protocol, the offline storage of P_0, which includes $(\{(w_i^0, w_i^1)\}_i, r_0)$, amounts to a total of $(n_1 + 1)\lambda + n_2$ bits (relying on the Free XOR optimization [52]). P_1's offline storage includes (C_g, r_1) which is equivalent to $\#\mathsf{AND} \cdot 2\lambda + n_1$ bits, where $\#\mathsf{AND}$ denotes the number of AND gates in C_g and each such gate requires 2λ bits of storage (using the half-gates optimization from [79]). In the online phase, across both the rounds, a total communication of $(\lambda + 1)n_1$ bits happens.

Computing Splines with GC: Recall that a spline function with m intervals defined by a list of m consecutive intervals $\{[p_{i-1} + 1, p_i]\}_i$ and m degree-d univariate polynomials $\{f_i\}_i$ outputs $f_j(x)$ if $x \in [p_{j-1} + 1, p_j]$. This involves performing multiplications to evaluate $f_j(x)$ which is very inefficient with GC. In mixed-mode paradigm, this step is often performed using arithmetic beaver triples [8]. Additive shares of coefficients of f_j are computed using GC and then beaver triples are consumed to evaluate the polynomial. However, we observe that this step of evaluating the polynomials can be made completely local. For this, instead of giving out r_1 to P_1, the dealer distributes additive shares of r_1 between P_0 and P_1 during the setup phase. Then, in the online phase, P_0 and P_1 reconstruct $x + r_1$ and P_0 sends labels corresponding to it to P_1. The rest of the GC evaluation proceeds normally and the parties get shares of coefficients of f'_j at the end such that $f'_j(x + r_1) = f_j(x)$. Since, P_0 and P_1 both know $x + r_1$ in clear, they can evaluate $f'_j(x + r_1)$ locally to get shares of $f_j(x)$ as output.

Prior Work: Similar to this, a recent work Delphi [56] splits GC into preprocessing and online phases in the 2PC setting, and obtains a total online communication of $\lambda n_1 + n_2$ bits across 2 rounds. However, their offline storage is $(n_1 + 1)\lambda$ for P_0 and $\#\mathsf{AND} \cdot 2\lambda + (n_1 + n_2)(\lambda + 1)$ for P_1, which is more than the total storage required for our GC protocol. Note that one can also use Garbled Circuits in the preprocessing model by having the dealer give out random OT (ROT) correlations in the setup phase. These ROT correlations are then reoriented into general OT (GOT) in the online phase [9] by P_0 and P_1 to transfer the wire labels corresponding to P_1's input. However, it has double the online communication compared to our GC protocol.

C.2 Garbled Circuits vs Our Protocols

We now show how our protocols compare with GC in the preprocessing model. Recall that all our FSS gate constructions are optimal in terms of online communication (equal to n bits per party) and rounds (single round), while GC requires online communication of $(\lambda + 1)n$ bits and 2 rounds, where n denotes the input group bitsize. This translates to our protocols being $\frac{\lambda}{2}\times$ better in online communication and $2\times$ better in rounds. First, we provide a version of Table 1 with exact expressions for key size in Table 2. We then provide a comparison of our protocols for offline storage and online AES calls per party with GC in Table 3a and 3b, respectively.

D FSS Gates Syntax

We summarize the syntax of FSS gates considered in this work below. We use N to denote 2^n for $n > 0$. $\mathbb{U}_N$ and $\mathbb{S}_N$ denote unsigned and signed n-bit integers, respectively. Refer to Section 2.1 for more details on our data types. We define $\mathbf{1}\{b\}$ as 1 when b is true and 0 otherwise.

Table 2: Comparison of key size (i.e., storage and for the 3PC case offline communication) for our FSS gate constructions with [21] and Garbled Circuits (GC) [75]. For FSS, i.e., our work and [21], we use the total key size for both P_0, P_1. For GC, we do an under approximation and consider only the size of garbled circuit. The table only captures the size of correlated randomness (offline communication in the 3PC case) and the online communication corresponding to both the FSS columns is at least $\frac{\lambda}{2}\times$ better than GC (and rounds $2\times$ better). Here, $\mathbb{U}_N$ and $\mathbb{S}_N$ denote unsigned and signed n-bit integers, respectively. We consider following gates: 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. The exact syntax and definitions of all the gates in the table are described in Appendix D. 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 [21], 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 across both parties. Size of $\mathsf{DCF}^{\mathsf{BGI}}_{n,\mathbb{G}}$ key (using [20]) 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. The values in the parenthesis give exact key size in bits for $\lambda = 128$, $n = 16$, $m = 12$, $d = 1$, $w = 4$, $s = 7$.

Gate	This work	[21]	GC
IC (n)	$\mathsf{DCF}_{n,\mathbb{U}_N} + 2n$ (4992)	$2 \times \mathsf{DCF}^{\mathsf{BGI}}_{n,\mathbb{U}_N} + 2n$ (34592)	$2\lambda(4n-3)$ (15616)
MIC (n,m)	$\mathsf{DCF}_{n,\mathbb{U}_N}$ $+2mn$ (5344)	$2m \times \mathsf{DCF}^{\mathsf{BGI}}_{n,\mathbb{U}_N}$ $+2mn$ (415104)	$2\lambda(n-1)(3m+1)$ $+2\lambda m$ (145152)
Splines (n,m,d)	$\mathsf{DCF}_{n,\mathbb{U}_N^{m(d+1)}}$ $+4mn(d+1)+2n$ (19040)	$2m \times \mathsf{DCF}^{\mathsf{BGI}}_{n,\mathbb{U}_N^{(d+1)}}$ (427008)	$2\lambda(n-1)(2m+1)$ $+2\lambda m(2n(d+1)-d)$ (289536)
ReLU (n)	$\mathsf{DCF}_{n,\mathbb{U}_N^2} + 10n$ (5664)	$2 \times \mathsf{DCF}^{\mathsf{BGI}}_{n,\mathbb{U}_N^2} + 2n$ (35616)	$2\lambda(3n-2)$ (11776)
ABS (n)	$\mathsf{DCF}_{n,\mathbb{U}_N^2} + 14n$ (5728)	$4 \times \mathsf{DCF}^{\mathsf{BGI}}_{n,\mathbb{U}_N^2}$ (71168)	$2\lambda(4n-3)$ (15616)
BD (n,w)	$\frac{n}{w} \times \mathsf{DCF}_{\frac{n+w}{2},\mathbb{U}_2} + 2n$ (11544)	$(n-1) \times \mathsf{DCF}^{\mathsf{BGI}}_{\frac{n}{2},\mathbb{U}_2} + 2n$ (127952)	$2\lambda(n-1)$ (3840)
LRS (n,s)	$\mathsf{DCF}_{s,\mathbb{U}_N} + \mathsf{DCF}_{n,\mathbb{U}_N} + 2n$ (7324)	- (-)	$2\lambda(2n-2)$ (7680)
ARS (n,s)	$\mathsf{DCF}_{s,\mathbb{S}_N} + \mathsf{DCF}_{n-1,\mathbb{S}_N^2} + 6n$ (7608)	- (-)	$2\lambda(2n-2)$ (7680)

1. Interval Containment/IC (n): $\mathcal{G}_{\mathsf{IC}}$ gate is a family of functions $g_{\mathsf{IC},n,p,q} : \mathbb{U}_N \to \mathbb{U}_N$ parametrized by the input and output groups $\mathbb{U}_N$ and defined by $g_{\mathsf{IC},n,p,q}(x) = \mathbf{1}\{p \leqslant x \leqslant q\}$.
2. Multiple Interval Containment/MIC (n,m): $\mathcal{G}_{\mathsf{MIC}}$ gate is a family of functions $g_{\mathsf{MIC},n,m,\{p_i\}_i,\{q_i\}_i} : \mathbb{U}_N \to \mathbb{U}_N^m$ for m interval containments parameterized by input and output groups $\mathbb{U}_N$ and $\mathbb{U}_N^m$, respectively, and defined by $g_{\mathsf{MIC},n,m,\{p_i\}_i,\{q_i\}_i}(x) = \Big\{\mathbf{1}\{p_i \leqslant x \leqslant q_i\}\Big\}_{1 \leqslant i \leqslant m}$.
3. Splines (n,m,d): $\mathcal{G}_{\mathsf{spline}}$ gate is a family of functions $g_{\mathsf{spline},n,m,d,\{p_i\}_i,\{f_i\}_i} : \mathbb{U}_N \to \mathbb{U}_N$ with m intervals parameterized by input and output rings $\mathbb{U}_N$, and defined by $g_{\mathsf{spline},n,m,d,\{p_i\}_i,\{f_i\}_i}(x) = h_{n,m,d,\{p_i\}_i,\{f_i\}_i}(x)$, where f_i refers to the d-degree univariate polynomial defining the output of the spline at i-th interval and
$$h_{n,m,d,\{p_i\}_i,\{f_i\}_i}(x) = \begin{cases} f_1(x) & \text{if } x \in [0, p_1] \\ f_2(x) & \text{if } x \in [p_1+1, p_2] \\ \vdots & \\ f_m(x) & \text{if } x \in [p_{m-1}+1, N-1] \end{cases}$$
4. ReLU (n): $\mathcal{G}_{\mathsf{ReLU}}$ gate is a family of functions $g_{\mathsf{ReLU},n} : \mathbb{S}_N \to \mathbb{S}_N$ parametrized by the input and output groups $\mathbb{S}_N$ and defined by $g_{\mathsf{ReLU},n}(x) = x$ if $x \geqslant 0$ and 0 otherwise.
5. Absolute Value/ABS (n): $\mathcal{G}_{|\cdot|}$ gate is a family of functions $g_{|\cdot|,n} : \mathbb{S}_N \to \mathbb{S}_N$ parametrized by the input and output groups $\mathbb{S}_N$ and defined by $g_{|\cdot|,n}(x) = x$ if $x \geqslant 0$ and $-x$ otherwise.

Table 3: Comparison of our FSS gate constructions with GC in the preprocessing model. We consider following gates: 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. n denotes the input bitlength. IC, MIC and Splines have public intervals. For our BD scheme (with output over $\mathbb{Z}_2^n$), w is a parameter and compute grows exponentially with w. In the table, we assume that $w \mid n$. The exact syntax and definitions of all the gates in the table are described in Appendix D.

Gate	This work	GC
IC (n)	$n(\lambda+n)$	$n(8\lambda+1)$
MIC (n,m)	$n(\lambda+n+m)$	$n(6m\lambda+1)$
Splines (n,m,d)	$n(\lambda+mn(d+1))$	$n(4m(d+2)\lambda+1)$
ReLU (n)	$n(\lambda+2n)$	$n(6\lambda+1)$
ABS (n)	$n(\lambda+2n)$	$n(8\lambda+1)$
BD (n,w)	$\frac{n\lambda(n+w)}{2w}$	$n(2\lambda+1)$
LRS (n,s)	$(\lambda+n)(n+s)$	$n(4\lambda+1)$
ARS (n,s)	$(\lambda+n)(n+s)+n^2$	$n(4\lambda+1)$

(a) Offline storage required for our FSS gate constructions vs GC. For our work, offline storage values denote the number of bits of key size that each party needs to store, i.e., $\frac{1}{2}\times$ the total key size. For GC, it refers to the number of bits that GC evaluator (P_1) needs to store. We provide approximate storage costs here by ignoring the lower order terms.

Gate	This work	GC
IC (n)	$8n$	$8n-6$
MIC (n,m)	$8mn$	$(n-1)(6m+2)+2m$
Splines (n,m,d)	$4mn$	$(n-1)(4m+2)+m(4n(d+1)-2d)$
ReLU (n)	$8n$	$6n-4$
ABS (n)	$8n$	$8n-6$
BD (n,w)	$\frac{4n(2^w-1)(n+w)}{w}$	$2n-2$
LRS (n,s)	$4(n+s)$	$4n-4$
ARS (n,s)	$4(n+s-1)$	$4n-4$

(b) Online AES calls required for our FSS gate constructions vs GC. For our work, online AES calls denote the number of AES calls that each party makes during gate evaluation. For GC, it refers to the number of AES calls that GC evaluator (P_1) makes to evaluate the garbled circuit. We assume $nm(d+1) \leqslant \lambda$.

6. Bit Decomposition/BD (n,w): $\mathcal{G}_{\mathsf{BIT}}$ gate is a family of functions $g_{\mathsf{BIT},n} : \mathbb{U}_N \to \mathbb{U}_N^n$, parameterized by the input and output groups $\mathbb{U}_N$ and $\mathbb{U}_N^n$, respectively, and defined by $g_{\mathsf{BIT},n}(x) = \{x_{n-1}, x_{n-2}, \dots x_0\}$ s.t. $\forall i, x_i \in \{0,1\}$ and $\sum_{i=0}^{n-1} 2^i x_i = x$.

   – The key size of our FSS gate construction of BD is $\approx \frac{n}{w}$ DCF keys, where w is tunable parameter. However, our compute grows exponentially with w and to keep it polynomial in n, we set $w = \lceil \log n \rceil$.

   – In Table 1 and 2, we use the output group of $\mathcal{G}_{\mathsf{BIT}}$ as $\mathbb{U}_2^n$ for which the gate definition remains the same except that instead of outputting the bits x_i (in the binary representation of x) over $\mathbb{U}_N$, we output them over $\mathbb{U}_2$. Similarly for Table 3a and 3b.

7. Logical Right Shift/LRS (n,s): $\mathcal{G}_{\gg_L}$ gate is a family of functions $g_{\gg_L,s,n} : \mathbb{U}_N \to \mathbb{U}_N$ parameterized by the input and output groups $\mathbb{U}_N$ and shift amount s, and defined by $g_{\gg_L,s,n}(x) = (x \gg_L s)$, where $(x \gg_L s) = \frac{x-(x \bmod 2^s)}{2^s}$ over $\mathbb{Z}$.
8. Arithmetic Right Shift/ARS (n,s): $\mathcal{G}_{\gg_A}$ gate is a family of functions $g_{\gg_A,s,n} : \mathbb{S}_N \to \mathbb{S}_N$ parameterized by the input and output groups $\mathbb{S}_N$ and shift amount s, and defined by $g_{\gg_A,s,n}(x) = (x \gg_A s)$, where $(x \gg_A s) = \frac{x-(x \bmod 2^s)}{2^s}$ over $\mathbb{Z}$ [40].

E Additional Preliminaries

E.1 Representing Functions

In order to seamlessly handle both arithmetic and Boolean operations, we will consider all functions to be defined over Abelian groups. For instance, a Boolean function $f : \{0,1\}^n \to \{0,1\}^m$ will be viewed as a mapping from the group $\mathbb{Z}_2^n$ to the group $\mathbb{Z}_2^m$. Given our heavy use of function secret sharing, we use a similar convention for function representation to the one used in [20] (the only difference being that here we also endow the input domain with a group structure).

Definition 7 (Function families). *A* function family *is defined by $\mathcal{F} = (P_{\mathcal{F}}, E_{\mathcal{F}})$, where $P_{\mathcal{F}} \subseteq \{0,1\}^*$ is an infinite collection of function descriptions $\hat{f}$ and $E_{\mathcal{F}} : P_{\mathcal{F}} \times \{0,1\}^* \to \{0,1\}^*$ is a polynomial-time algorithm defining the function described by $\hat{f}$. Concretely, $\hat{f} \in P_{\mathcal{F}}$ describes a corresponding function $f : D_f \to R_f$ defined by $f(x) = E_{\mathcal{F}}(\hat{f}, x)$. We require D_f and R_f to be finite Abelian groups, denoted by $\mathbb{G}^{\mathsf{in}}$ and $\mathbb{G}^{\mathsf{out}}$ respectively. We will typically let $\mathbb{G}^{\mathsf{in}}$ and $\mathbb{G}^{\mathsf{out}}$ be product groups, which can capture the case of multiple inputs and outputs. When there is no risk of confusion, we will sometimes write f instead of $\hat{f}$ and $f \in \mathcal{F}$ instead of $\hat{f} \in P_{\mathcal{F}}$. We assume that $\hat{f}$ includes an explicit description of $\mathbb{G}^{\mathsf{in}}$ and $\mathbb{G}^{\mathsf{out}}$.*

By convention, we denote by $0 \in \mathbb{G}$ the identity element of $\mathbb{G}$. We will use the notation $1 \in \mathbb{G}$ to denote a fixed canonical nonzero element of $\mathbb{G}$; when $\mathbb{G}$ is additionally endowed with a multiplicative structure, e.g., when $\mathbb{G}$ is the additive group of a finite *ring*, 1 will be set to the multiplicative identity.

E.2 Secure Computation with Preprocessing

We follow the standard definitional framework for secure computation (cf. [24, 42]), except that we allow a trusted input-independent setup phase that distributes correlated secret randomness to the parties. This setup phase can be securely emulated by an interactive preprocessing protocol that can be carried out before the inputs are known. We focus here on protocols with security against a *semi-honest* adversary who may non-adaptively corrupt any strict subset of parties. For simplicity, we explicitly spell out the definitions for the two-party case, and later explain the (straightforward) extension to the multi-party case.

Functionalities. We denote the two parties by P_0 and P_1 and a party index by $\sigma \in \{0,1\}$. We consider by default protocols for *deterministic* functionalities that deliver the *same output* to the two parties. The general case (of randomized functionalities with different outputs) can be reduced to this case via a standard reduction [24, 42]. A two-party functionality f is described by a bit-string $\hat{f}$ via a function family $\mathcal{F}$, as in Definition 7. We assume that the input domain $\mathbb{G}^{\mathsf{in}}$ is split into $\mathbb{G}^{\mathsf{in}} = \mathbb{G}_0^{\mathsf{in}} \times \mathbb{G}_1^{\mathsf{in}}$, capturing the inputs of the two parties.

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

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

Definition 8 (Secure computation with preprocessing). *We say that $\Pi = (\mathsf{Setup}, \mathsf{NextMsg})$ securely realizes a function family $\mathcal{F}$ in the preprocessing model if the following holds:*

– Correctness: *For all $\hat{f} \in P_{\mathcal{F}}$ describing $f : \mathbb{G}_0^{\mathsf{in}} \times \mathbb{G}_1^{\mathsf{in}} \to \mathbb{G}^{\mathsf{out}}$, $(x_0, x_1) \in \mathbb{G}_0^{\mathsf{in}} \times \mathbb{G}_1^{\mathsf{in}}$, $\lambda \in \mathbb{N}$, $\sigma \in \{0,1\}$, we have $\Pr[\mathsf{Out}_{\Pi,\sigma}(\lambda, \hat{f}, (x_0, x_1)) = f(x_0, x_1)] = 1$.*

– Security: *For each corrupted party $\sigma \in \{0,1\}$ there exists a PPT algorithm Sim_σ (simulator), such that for every infinite sequence $(\hat{f}_\lambda)_{\lambda \in \mathbb{N}}$ of polynomial-size function descriptions from $P_{\mathcal{F}}$ and polynomial-size input sequence $(x_0^\lambda, x_1^\lambda)_{\lambda \in \mathbb{N}}$ for f_λ, the outputs of the following experiments Real and Ideal are computationally indistinguishable:*
  - Real_λ*: Output* $\mathsf{View}_{\Pi,\sigma}(\lambda, \hat{f}_\lambda, (x_0^\lambda, x_1^\lambda))$
  - Ideal_λ*: Output* $\mathsf{Sim}_\sigma(1^\lambda, \hat{f}_\lambda, x_\sigma^\lambda, f_\lambda(x_0^\lambda, x_1^\lambda))$

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

E.3 Secure Computation from FSS

We here present one of the main theorems from [21], attaining secure computation with preprocessing given appropriate FSS schemes.

Theorem 13 (Circuit-Dependent Preprocessing [21]). *Let C be a circuit over basis $\mathcal{B}$. For each $\mathcal{G} \in \mathcal{B}$, let $(\mathsf{Gen}_{\hat{\mathcal{G}}}, \mathsf{Eval}_{\hat{\mathcal{G}}})$ be an FSS for the offset-function family $\hat{\mathcal{G}}$ with key size $\mathsf{size}_{\hat{\mathcal{G}}}(\lambda, |\mathbb{G}^{\mathsf{in}}|, |\mathbb{G}^{\mathsf{out}}|)$. Then for any instantiation $C_{\boldsymbol{g}}$ of C, there exists a 2-party protocol for securely computing $C_{\boldsymbol{g}}$ with the following properties:*

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

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

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

F Optimized DCF: Proof and Dual DCF

F.1 Proof of DCF Theorem

Proof of Theorem 2 . Security: We prove that each party's key k_b is pseudorandom. This will be done via a sequence of hybrids, where in each step we replace another correction word $CW^{(i)}$ within the key from being honestly generated to being random.

The high-level argument for security will go as follows. Each party $b \in \{0,1\}$ begins with a random seed $s_b^{(0)}$ that is completely unknown to the other party. In each level of key generation (for $i = 1$ to n), the parties apply a PRG to their seed $s_b^{(i-1)}$ to generate six items: namely, two seeds s_b^L, s_b^R, two group elements V_b^L, V_b^R and 2 bits t_b^L, t_b^R. This process will *always* be performed on a seed which appears completely random and unknown given the view of the other party; because of this, the security of the PRG guarantees that the six resulting values appear similarly random and unknown given the view of the other party. The ith level correction word $CW^{(i)}$ will "use up" the secret randomness of 4 of these 6 pieces: the two bits t_b^L, t_b^R, the element V_b^{Lose} and the seed s_b^{Lose} for $\mathsf{Lose} \in \{L, R\}$ corresponding to the direction *exiting* the "special path"

α: i.e. $\mathsf{Lose} = L$ if $\alpha = 1$ and $\mathsf{Lose} = R$ if $\alpha = 0$. However, given this $CW^{(i)}$, the remaining seed s_b^{Keep} for $\mathsf{Keep} \neq \mathsf{Lose}$ still appears random to the other party. The argument then continues in similar fashion to the next level, beginning with seeds s_b^{Keep}.

For each $j \in \{0, 1, \ldots, n+1\}$, we will consider a distribution Hyb_j defined roughly as follows:

1. $s_b^{(0)} \leftarrow \{0,1\}^\lambda$ chosen at random (honestly), and $t_b^{(0)} = b$.
2. $CW^{(1)}, \ldots, CW^{(j)} \leftarrow \{0,1\}^{\lambda+1}$ chosen at random.
3. For $i \leqslant j$, $s_b^{(i)} || V_b^{(i)} || t_b^{(i)}$ computed honestly, as a function of $s_b^{(0)} || V_b^{(0)} || t_b^{(0)}$ and $CW^{(1)}, \ldots, CW^{(j)}$.
4. For j, the other party's seed $s_{1-b}^{(j)} \leftarrow \{0,1\}^\lambda$ and the element $V_{1-b}^{(j)} \leftarrow \mathbb{G}$ are chosen at random, and $t_{1-b}^{(j)} = 1 - t_b^{(j)}$.
5. For $i > j$: the remaining values
$$s_b^{(i)} || V_b^{(i)} || t_b^{(i)}, s_{1-b}^{(i)} || V_{1-b}^{(i)} || t_{1-b}^{(i)}, CW^{(i)}$$
are all computed honestly as a function of the previously chosen values.
6. The output of the experiment is
$$k_b := s_b^{(0)} || CW^{(1)} || \cdots || CW^{(n+1)}.$$

Formally, Hyb_j is fully described in Fig. 10. Note that when $j = 0$, this experiment corresponds to the honest key distribution, whereas when $j = n+1$ this yields a completely random key k_b. We claim that each pair of adjacent hybrids $j-1$ and j will be indistinguishable based on the security of the pseudorandom generator.

The proof of Theorem 2 follows from the following four claims:

Claim 1. *For every $b \in \{0,1\}, \alpha \in \{0,1\}^n, \beta \in \mathbb{G}$, it holds that*
$$\{k_b \leftarrow \mathsf{Hyb}_0(1^\lambda, b, \alpha, \beta)\} \equiv \{k_b : (k_0, k_1) \leftarrow \mathsf{Gen}_n^{<}(1^\lambda, \alpha, \beta, \mathbb{G})\}.$$

Claim 2. *For every $b \in \{0,1\}, \alpha \in \{0,1\}^n, \beta \in \mathbb{G}$, it holds that*
$$\{k_b \leftarrow \mathsf{Hyb}_{n+1}(1^\lambda, b, \alpha, \beta)\} \equiv \{k_b \leftarrow U\}.$$

Note that Claim 1 and Claim 2 follow directly by construction of Hyb_j.

Claim 3. *There exists a polynomial p' such that for any $(T, \epsilon_{\mathsf{PRG}})$-secure pseudorandom generator G, then for every $j \in [n]$, every $b \in \{0,1\}, \alpha \in \{0,1\}^n, \beta \in \mathbb{G}$, and every nonuniform adversary $\mathcal{A}$ running in time $T' \leqslant T - p'(\lambda)$, it holds that*
$$\Big| \Pr[k_b \leftarrow \mathsf{Hyb}_{j-1}(1^\lambda, b, \alpha, \beta); c \leftarrow \mathcal{A}(1^\lambda, k_b) : c = 1] \\ - \Pr[k_b \leftarrow \mathsf{Hyb}_j(1^\lambda, b, \alpha, \beta); c \leftarrow \mathcal{A}(1^\lambda, k_b) : c = 1] \Big| < \epsilon_{\mathsf{PRG}}.$$

Proof. Fix an arbitrary $j \in [n], b \in \{0,1\}, \alpha \in \{0,1\}^n$ and $\beta \in \mathbb{G}$. Given a Hyb-distinguishing adversary $\mathcal{A}$ with advantage ϵ for these values, we construct a corresponding PRG adversary $\mathcal{B}$. Recall that in the PRG challenge for G, the adversary $\mathcal{B}$ is given a value r that is either computed by sampling a seed $s \leftarrow \{0,1\}^\lambda$ and computing $r = G(s)$, or sampling a random $r \leftarrow \{0,1\}^{2(2\lambda+1)}$.

Now, consider $\mathcal{B}$'s success in the PRG challenge as a function of $\mathcal{A}$'s success in distinguishing Hyb_{j-1} from Hyb_j. If r is computed *pseudorandomly*, then it is clear the generated k_b is distributed as $\mathsf{Hyb}_{j-1}(1^\lambda, b, \alpha, \beta)$.

It remains to show that if r was sampled at random then the generated k_b is distributed as $\mathsf{Hyb}_j(1^\lambda, b, \alpha, \beta)$. That is, if r is random, then the corresponding computed values of $s_{1-b}^{(j)}$ and $CW^{(j)}$ are distributed *randomly* conditioned on the values of $s_b^{(0)} || t_b^{(0)} || CW^{(1)} || \cdots || CW^{(j-1)}$, and the value of $t_{1-b}^{(j)}$ is given by $1 - t_b^{(j)}$. Note that all remaining values (for "level" $i > j$) are computed as a function of the values up to "level" j.

First, consider $CW^{(j)}$, computed in four parts:

```
$\mathsf{Hyb}_j(1^\lambda, b, \alpha, \beta)$:
 1: Let $\alpha = \alpha_1, \ldots, \alpha_n \in \{0,1\}^n$ be the bit decomposition of $\alpha$
 2: Sample random $s_b^{(0)} \leftarrow \{0,1\}^\lambda$, and let $V_\alpha^{(0)} \leftarrow 0 \in \mathbb{G}$, $t_b^{(0)} = b$, $t_{1-b}^{(0)} = 1-b$.
 3: for $i = 1$ to $n$ do
 4:     if $i < j$ then Sample $CW^{(i)} \leftarrow \{0,1\}^\lambda \times \mathbb{G} \times \{0,1\}^2$.
 5:     else
 6:         if $i = j$ then
 7:             Sample random $s_{1-b}^{(j-1)} \leftarrow \{0,1\}^\lambda$, and random $V_\alpha^{(i)} \leftarrow \mathbb{G}$.
 8:             Let $t_{1-b}^{(j-1)} = 1 - t_b^{(j-1)}$.
 9:         end if
10:         $CW^{(i)} = \mathsf{CompCW}(i, \alpha_i, G(s_b^{(i-1)}), G(s_{1-b}^{(i-1)}), V_\alpha^{(i-1)}, \beta)$.
11:         $(s_{1-b}^{(i)}, t_{1-b}^{(i)}) = \mathsf{NextST}(1-b, i, t_{1-b}^{(i-1)}, s_{1-b}^{\mathsf{Keep}} || t_{1-b}^{\mathsf{Keep}}, CW^{(i)})$.
12:     end if
13:
14:     $(s_b^{(i)}, t_b^{(i)}) = \mathsf{NextST}(b, i, t_b^{(i-1)}, s_b^{\mathsf{Keep}} || t_b^{\mathsf{Keep}}, CW^{(i)})$.
15:     $V_\alpha^{(i)} \leftarrow V_\alpha^{(i-1)} - \mathsf{Convert}_{\mathbb{G}}(v_1^{\mathsf{Keep}}) + \mathsf{Convert}_{\mathbb{G}}(v_0^{\mathsf{Keep}}) + (-1)^{t_1^{(i-1)}} \cdot V_{CW}$.
16: end for
17:
18: if $j = n+1$ then
19:     $CW^{(n+1)} \leftarrow \mathbb{G}$
20: else
21:     $CW^{(n+1)} \leftarrow (-1)^{t_1^n} \cdot \left[\beta - \mathsf{Convert}(s_0^{(n)}) + \mathsf{Convert}(s_1^{(n)})\right] \in \mathbb{G}$
22: end if
23: Let $k_b = s_b^{(0)} || CW^{(1)} || \cdots || CW^{(n+1)}$
24: return $k_b$

$\mathsf{CompCW}(i, \alpha_i, S_b^{(i-1)}, S_{1-b}^{(i-1)}, V_\alpha^{(i-1)}, \beta)$:
 1: Parse $S_{1-b}^{(i-1)} = s_{1-b}^L || v_{1-b}^L || t_{1-b}^L \big|\big| s_{1-b}^R || v_{1-b}^R || t_{1-b}^R$.
 2: Parse $S_b^{(i-1)} = s_b^L || v_b^L || v_b^L || t_b^L \big|\big| s_b^R | v_b^R || v_b^R || t_b^R$.
 3: if $\alpha_i = 0$ then set $\mathsf{Keep} \leftarrow L$, $\mathsf{Lose} \leftarrow R$.
 4: else  Set $\mathsf{Keep} \leftarrow R$, $\mathsf{Lose} \leftarrow L$.
 5: end if
 6: $s_{CW} \leftarrow s_0^{\mathsf{Lose}} \oplus s_1^{\mathsf{Lose}}$.
 7: $V_{CW} \leftarrow (-1)^{t_1^{(i-1)}} \cdot [\mathsf{Convert}_{\mathbb{G}}(v_1^{\mathsf{Lose}}) - \mathsf{Convert}_{\mathbb{G}}(v_0^{\mathsf{Lose}}) - V_\alpha^{(i-1)}]$.
 8: if $\mathsf{Lose} = L$ then $V_{CW} \leftarrow V_{CW} + (-1)^{t_1^{(i-1)}} \cdot \beta$
 9: end if
10: $t_{CW}^L \leftarrow t_0^L \oplus t_1^L \oplus \alpha_i \oplus 1$ and $t_{CW}^R \leftarrow t_0^R \oplus t_1^R \oplus \alpha_i$.
11: return $CW^{(i)} \leftarrow s_{CW} || V_{CW} || t_{CW}^L || t_{CW}^R$

$\mathsf{NextST}(x, i, t_x^{(i-1)}, s_x^{\mathsf{Keep}} || t_x^{\mathsf{Keep}}, CW^{(i)})$:
 1: Parse $CW^{(i)} = s_{CW} || V_{CW} || t_{CW}^L || t_{CW}^R$.
 2: $s_x^{(i)} \leftarrow s_x^{\mathsf{Keep}} \oplus t_x^{(i-1)} \cdot s_{CW}$
 3: $t_x^{(i)} \leftarrow t_x^{\mathsf{Keep}} \oplus t_x^{(i-1)} \cdot t_{CW}^{\mathsf{Keep}}$
 4: return $(s_x^{(i)}, t_x^{(i)})$.
```
Fig. 10: Hybrid distribution j, in which the first j correction words are sampled completely at random, and the remaining correction words are computed honestly.

```
PRG adversary $\mathcal{B}(1^\lambda, (j, b, \alpha, \beta), r)$:
 1: Let $\alpha = \alpha_1, \ldots, \alpha_n \in \{0,1\}^n$ be the bit decomposition of $\alpha$
 2: Sample $s_b^{(0)} \leftarrow \{0,1\}^\lambda$, let $V_\alpha^{(0)} = 0 \in \mathbb{G}$, and let $t_b^{(0)} = b$.
 3:
 4: for $i = 1$ to $(j-1)$ do
 5:     Sample random $CW^{(i)} \leftarrow \{0,1\}^\lambda \times \mathbb{G} \times \{0,1\}^2$.
 6:     Parse $CW^{(i)} = s_{CW}||V_{CW}||t_{CW}^L||t_{CW}^R$.
 7:     Expand $s_b^L||v_b^L||t_b^L\big||s_b^R||v_b^R||t_b^R = G(s_b^{(i-1)})$.
 8:     if $\alpha_i = 0$ then Set $\mathsf{Keep} \leftarrow L$, $\mathsf{Lose} \leftarrow R$. else, Set $\mathsf{Keep} \leftarrow R$, $\mathsf{Lose} \leftarrow L$
 9:     $(s_b^{(i)}, t_b^{(i)}) = \mathsf{NextST}(b, i, t_b^{(i-1)}, s_b^{\mathsf{Keep}}, t_b^{\mathsf{Keep}}, CW^{(i)})$.
10:     $V_\alpha^{(i)} \leftarrow V_\alpha^{(i-1)} - \mathsf{Convert}_{\mathbb{G}}(v_1^{\mathsf{Keep}}) + \mathsf{Convert}_{\mathbb{G}}(v_0^{\mathsf{Keep}}) + (-1)^{t_1^{(i-1)}} \cdot V_{CW}$
11:     Take $t_{1-b}^{(i)} = 1 - t_b^{(i)}$.
12: end for
13:
14: Expand $s_b^L||v_b^L||t_b^L\big||s_b^R||v_b^R||t_b^R = G(s_b^{(j-1)})$.
15: Set $s_b^L||v_b^L||t_b^L\big||s_b^R||v_b^R||t_b^R = r$ (the PRG challenge).
16: $CW^{(j)} = \mathsf{CompCW}(j, \alpha_j, r, G(s_b^{(j-1)}), V_\alpha^{(i-1)}, \beta)$.
17: if $\alpha_j = 0$ then Set $\mathsf{Keep} \leftarrow L$, $\mathsf{Lose} \leftarrow R$. else, Set $\mathsf{Keep} \leftarrow R$, $\mathsf{Lose} \leftarrow L$
18: Compute $(s_x^{(j)}, t_x^{(j)}) = \mathsf{NextST}(x, j, t_x^{(j-1)}, s_x^{\mathsf{Keep}}||t_x^{\mathsf{Keep}}, CW^{(j)})$, for both $x \in \{0,1\}$.
19:
20: Set $P = [s_0^L||v_0^L||t_0^L\big||s_0^R||v_0^R||t_0^R], [s_1^L||v_1^L||t_1^L\big||s_1^R||v_1^R||t_1^R]$.
21: Compute $(CW^{(j+1)}||\cdots||CW^{(n+1)}) =$
        $\mathsf{RemainingKey}(\alpha, j, CW^{(1)}||\cdots||CW^{(j)}, P)$.
22: return $k_b = s_b^{(0)}||CW^{(1)}||\cdots||CW^{(n+1)}$.

$\mathsf{RemainingKey}(\alpha, j, CW^{(1)}||\cdots||CW^{(j)}, t_0^{(j)}, t_1^{(j)}, P)$:
 1: Parse $P = [s_0^L||v_0^L||t_0^L\big||s_0^R||v_0^R||t_0^R], [s_1^L||v_1^L||t_1^L\big||s_1^R||v_1^R||t_1^R]$.
 2: for $i = (j+1)$ to $n$ do
 3:     Expand $s_x^L||v_x^L||t_x^L\big||s_x^R||v_x^R||t_x^R = G(s_x^{(i-1)})$ for both $x \in \{0,1\}$.
 4:     if $\alpha_i = 0$ then Set $\mathsf{Keep} \leftarrow L$, $\mathsf{Lose} \leftarrow R$. else, Set $\mathsf{Keep} \leftarrow R$, $\mathsf{Lose} \leftarrow L$
 5:     $CW^{(i)} = \mathsf{CompCW}(i, \alpha_i, [s_0^L||v_0^L||t_0^L\big||s_0^R||v_0^R||t_0^R], [s_1^L||v_1^L||t_1^L\big||s_1^R||v_1^R||t_1^R], V_\alpha^{(i-1)}, \beta)$.
 6:     Compute $(s_x^{(i)}, t_x^{(i)}) = \mathsf{NextST}(x, i, t_x^{(i-1)}, s_x^{\mathsf{Keep}}||t_x^{\mathsf{Keep}}, CW^{(i)})$, for both $x \in \{0,1\}$.
 7:     $V_\alpha^{(i)} \leftarrow V_\alpha^{(i-1)} - \mathsf{Convert}_{\mathbb{G}}(v_1^{\mathsf{Keep}}) + \mathsf{Convert}_{\mathbb{G}}(v_0^{\mathsf{Keep}}) + (-1)^{t_1^{(i-1)}} \cdot V_{CW}$
 8: end for
 9: $CW^{(n+1)} \leftarrow (-1)^{t_1^n} \cdot \left[\beta - \mathsf{Convert}(s_0^{(n)}) + \mathsf{Convert}(s_1^{(n)})\right] \in \mathbb{G}$
10: return $(CW^{(j)}||CW^{(j+1)}||\cdots||CW^{(n+1)})$
```

Fig. 11: Adversary action.

– $s_{CW} = s_b^{\mathsf{Lose}} \oplus s_{1-b}^{\mathsf{Lose}}$.
– $V_{CW} \leftarrow (-1)^{t_1^{(i-1)}} \cdot [\mathsf{Convert}_{\mathbb{G}}(v_1^{\mathsf{Lose}}) - \mathsf{Convert}_{\mathbb{G}}(v_0^{\mathsf{Lose}}) - V_\alpha^{(i-1)}]$, and if $\mathsf{Lose} = L$ then $V_{CW} \leftarrow V_{CW} + (-1)^{t_1^{(i-1)}} \cdot \beta$.
– $t_{CW}^L = t_b^L \oplus t_{1-b}^L \oplus \alpha_j \oplus 1$.
– $t_{CW}^L = t_b^L \oplus t_{1-b}^L \oplus \alpha_j$.

In the case that r is random, then $s_{1-b}^{\mathsf{Lose}}, t_{1-b}^L$, and t_{1-b}^R (no matter the value of $\mathsf{Lose} \in \{L, R\}$) are each perfect one-time pads, and v_{1-b}^{Lose} is a random element in $\mathbb{G}$. So, $CW^{(j)} = s_{CW}||V_{CW}||t_{CW}^L||t_{CW}^R$ is indeed distributed uniformly.

Now, condition on $CW^{(j)}$ as well, and consider the value of $s_{1-b}^{(j)}$. Depending on the value of $t_{1-b}^{(j-1)}$, $s_{1-b}^{(j)}$ is selected either as s_{1-b}^{Keep} or $s_{1-b}^{\mathsf{Keep}} \oplus s_{CW}$. However, s_{1-b}^{Keep} is distributed uniformly conditioned on the view thus far, and so in either case the resulting value is again distributed uniformly.

Finally, consider the value of $t_{1-b}^{(j)}$. Note that both $t_b^{(j)}$ and $t_{1-b}^{(j)}$ are computed as per NextST, as a function of $t_1^{(j-1)}$ and $t_{1-b}^{(j-1)}$, respectively (and $t_{1-b}^{(j-1)}$ was set to $1 - t_b^{(j-1)}$). In particular,

$$\begin{aligned} t_b^{(j)} \oplus t_{1-b}^{(j)} &= (t_b^{\mathsf{Keep}} \oplus t_b^{(i-1)} \cdot t_{CW}^{\mathsf{Keep}}) \oplus (t_{1-b}^{\mathsf{Keep}} \oplus t_{1-b}^{(i-1)} \cdot t_{CW}^{\mathsf{Keep}}) \\ &= t_b^{\mathsf{Keep}} \oplus t_{1-b}^{\mathsf{Keep}} \oplus (t_b^{(i-1)} \oplus t_{1-b}^{(i-1)}) \cdot t_{CW}^{\mathsf{Keep}} \\ &= t_b^{\mathsf{Keep}} \oplus t_{1-b}^{\mathsf{Keep}} \oplus 1 \cdot (t_0^{\mathsf{Keep}} \oplus t_1^{\mathsf{Keep}} \oplus 1) \\ &= 1 \end{aligned}$$

Combining these pieces, we have that in the case of a random PRG challenge r, the resulting distribution of k_b as generated by $\mathcal{B}$ is precisely distributed as is $\mathsf{Hyb}_j(1^\lambda, b, \alpha, \beta)$. Thus, the advantage of $\mathcal{B}$ in the PRG challenge experiment is equivalent to the advantage ϵ of $\mathcal{A}$ in distinguishing $\mathsf{Hyb}_{j-1}(1^\lambda, b, \alpha, \beta)$ from $\mathsf{Hyb}_j(1^\lambda, b, \alpha, \beta)$. The runtime of $\mathcal{B}$ is equal to the runtime of $\mathcal{A}$ plus a fixed polynomial $p'(\lambda)$. Thus for any $T' \leqslant T - p'(\lambda)$, it must be that the distinguishing advantage ϵ of $\mathcal{A}$ is bounded by ϵ_{PRG}. □

Claim 4. *There exists a polynomial p' such that for any $(T, \epsilon_{\mathsf{Convert}})$-secure pseudorandom* $\mathsf{Convert} : \{0,1\}^\lambda \to \mathbb{G}$*, then for every $b \in \{0,1\}, \alpha \in \{0,1\}^n, \beta \in \mathbb{G}$, and every nonuniform adversary $\mathcal{A}$ running in time $T' \leqslant T - p'(\lambda)$, it holds that*

$$\Big| \Pr[k_b \leftarrow \mathsf{Hyb}_n(1^\lambda, b, \alpha, \beta); c \leftarrow \mathcal{A}(1^\lambda, k_b) : c = 1] \\ - \Pr[k_b \leftarrow \mathsf{Hyb}_{n+1}(1^\lambda, b, \alpha, \beta); c \leftarrow \mathcal{A}(1^\lambda, k_b) : c = 1] \Big| < \epsilon_{\mathsf{Convert}}.$$

Proof. Fix an arbitrary $b \in \{0,1\}, \alpha \in \{0,1\}^n, \beta \in \mathbb{G}$. In a similar fashion to the previous claim, an adversary $\mathcal{A}$ who distinguishes between the corresponding distributions Hyb_n and Hyb_{n+1} with advantage ϵ directly yields a corresponding adversary $\mathcal{B}$ for the pseudo-randomness of $\mathsf{Convert}$ with the same advantage, and only polynomial additional runtime $p'(\lambda)$. Namely, $\mathcal{B}$ samples $s_b^{(n)} \leftarrow \{0,1\}^\lambda$ and all values $CW^{(1)}, \ldots, CW^{(n)} \leftarrow \{0,1\}^{\lambda+2}$ at random, and then embeds the $\mathsf{Convert}$ challenge by setting $CW^{(n+1)} = (-1)^{t_1^n} \cdot [\beta + (-1)^{1-b} \cdot \mathsf{Convert}(s_b^{(n)}) + (-1)^b \cdot r]$. In the case that r is generated pseudo-randomly as the output of $\mathsf{Convert}(s_{1-b}^{(n)})$ for random $s_{1-b}^{(n)}$, this is precisely the distribution generated by Hyb_n. In the case that r is truly random, then it directly acts as a one-time pad on the remaining terms and thus $CW^{(n+1)}$ is distributed uniformly, precisely as per Hyb_{n+1}. The claim follows. □

This concludes the proof of Theorem 2. □

F.2 Dual Distributed Comparison Function (DDCF)

In Fig. 12, we give an FSS scheme for $\mathcal{F}_{n,\mathbb{G}}^{\mathsf{DDCF}}$. The key size of $\mathsf{DDCF}_{n,\mathbb{G}}$ is equal to $\lceil \log |\mathbb{G}| \rceil$ bits plus the key size of $\mathsf{DCF}_{n,\mathbb{G}}$.

Dual Distributed Comparison Function $(\mathsf{Gen}_n^{\mathsf{DDCF}}, \mathsf{Eval}_n^{\mathsf{DDCF}})$

```
$\mathsf{Gen}_n^{\mathsf{DDCF}}(1^\lambda, \alpha, \beta_1, \beta_2, \mathbb{G})$:
 1: Let $\beta = \beta_1 - \beta_2$.
 2: $(k_0^{(n)}, k_1^{(n)}) \leftarrow \mathsf{Gen}_n^<(1^\lambda, \alpha, \beta, \mathbb{G})$.
 3: Sample random $S_0, S_1 \leftarrow \mathbb{G}$ s.t. $S_0 + S_1 = \beta_2$.
 4: For $b \in \{0,1\}$, let $k_b = k_b^{(n)} || S_b$.
 5: return $(k_0, k_1)$.

$\mathsf{Eval}_n^{\mathsf{DDCF}}(b, k_b, x)$:
 1: Parse $k_b = k_b^{(n)} || S_b$.
 2: $y_b^{(n-1)} \leftarrow \mathsf{Eval}_n^<(b, k^{(n)}, x)$.
 3: return $y_b^{(n-1)} + S_b$.
```

Fig. 12: Optimized FSS scheme for the class $\mathcal{F}_{n,\mathbb{G}}^{\mathsf{DDCF}}$ of comparison functions $f_{\alpha,\beta_1,\beta_2} : \{0,1\}^n \to \mathbb{G}$, outputting β_1 for $0 \leqslant x < \alpha$ and β_2 for $x \geqslant \alpha$. b refers to party id.

G Proofs and Supplemental Material for Section 4

In this section, we provide the missing proofs from Section 4 along with some supplemental material.

Following simple lemma is used in the proof of main technical Lemma 1 and also in correctness proof of basic FSS gate for interval containment described in Fig. 2.

Lemma 6. *Let $x, y, z \in \mathbb{U}_N$. Unless stated by an explicit* mod N, $(+,-,<,>,=)$ *are operations over $\mathbb{Z}$ and any operands $\in \mathbb{U}_N$ are seen as elements of $\mathbb{Z}$ for that purpose. Then, the following holds true:*

$$x \geqslant y \text{ implies } e_x \geqslant e_y$$
$$\text{where } e_x = \mathbf{1}\{x + z > N - 1\} \text{ and } e_y = \mathbf{1}\{y + z > N - 1\}.$$

Proof. This Lemma is very simple to prove. We have, $x \geqslant y$, then over $\mathbb{Z}$, we can say that $x + z \geqslant y + z$. If $e_y = 1$, then $y + z > N - 1$ from the definition of e_y. Since $x + z \geqslant y + z > N - 1$, $e_x = 1$. On the other hand, if $e_y = 0$, then $x + z \geqslant y + z \leqslant N - 1$, and e_x can be both 0 or 1 depending on the values of x and z. □

G.1 Proof of correctness of Fig. 2

The proof follows from the following lemma using correctness of FSS schemes for $f_{\alpha,\beta}^<$ and $f_{\alpha,\beta}^\leqslant$.

Lemma 7. *Let $p, \tilde{p}, q, \tilde{q}, r \in \mathbb{U}_N$, where $p \leqslant q$, $\tilde{p} = p + r$* mod N, *$\tilde{q} = q + r$* mod N. *Define boolean predicate over $\mathbb{U}_N \to \{0,1\}$ as follows: $P(x)$ denotes $p \leqslant x \leqslant q$.*
Then the following holds:

$$P(x) = \mathbf{1}\{\tilde{x} \leqslant \tilde{q}\} - \mathbf{1}\{\tilde{x} < \tilde{p}\} + \mathbf{1}\{\tilde{p} > \tilde{q}\}, \text{ where } \tilde{x} = x + r \bmod N$$

Proof. First of all, observe that $P(x)$ evaluates an interval containment on x, and the expression of $P(x)$ derived from this lemma is being used as it is in Fig. 2.
We first look at the RHS of the expression we want to prove, i.e. $P(x) = \mathbf{1}\{\tilde{x} \leqslant \tilde{q}\} - \mathbf{1}\{\tilde{x} < \tilde{p}\} + \mathbf{1}\{\tilde{p} > \tilde{q}\}$.

$$\begin{aligned}
&\mathbf{1}\{\tilde{x} \leqslant \tilde{q}\} - \mathbf{1}\{\tilde{x} < \tilde{p}\} + \mathbf{1}\{\tilde{p} > \tilde{q}\} \\
&\qquad = \mathbf{1}\{\tilde{x} \leqslant \tilde{q}\} - \mathbf{1}\{\tilde{x} < \tilde{p}\} + 1 + \mathbf{1}\{\tilde{p} > \tilde{q}\} - 1 \\
&\qquad = \mathbf{1}\{\tilde{x} \leqslant \tilde{q}\} + (1 - \mathbf{1}\{\tilde{x} < \tilde{p}\}) - (1 - \mathbf{1}\{\tilde{p} > \tilde{q}\}) \\
&\qquad = \mathbf{1}\{\tilde{x} \leqslant \tilde{q}\} + \mathbf{1}\{\tilde{x} \geqslant \tilde{p}\} - \mathbf{1}\{\tilde{p} \leqslant \tilde{q}\}
\end{aligned}$$

Next, we look at the statement of the predicate $P(x)$. $P(x)$ can be rewritten as $\mathbf{1}\{(x \geqslant p) \wedge (x \leqslant q)\}$. We take this predicate from being over x to $\tilde{x}$.

$$\begin{aligned}
\mathbf{1}\{(x \geqslant p) \wedge (x \leqslant q)\} &= \mathbf{1}\{(x + r \geqslant p + r) \wedge (x + r \leqslant q + r)\} \\
&= \mathbf{1}\{\big(\tilde{x} + w_x N \geqslant \tilde{p} + w_p N\big) \wedge \big(\tilde{x} + w_x N \leqslant \tilde{q} + w_q N\big)\}
\end{aligned}$$

where $w_x = \mathbf{1}\{x + r > N - 1\}$, $w_p = \mathbf{1}\{p + r > N - 1\}$ and $w_q = \mathbf{1}\{q + r > N - 1\}$.
Therefore, the primary expression that we want to prove becomes:

$$\mathbf{1}\{(\tilde{x} + w_x N \geqslant \tilde{p} + w_p N) \wedge (\tilde{x} + w_x N \leqslant \tilde{q} + w_q N)\} = \mathbf{1}\{\tilde{x} \leqslant \tilde{q}\} + \mathbf{1}\{\tilde{x} \geqslant \tilde{p}\} - \mathbf{1}\{\tilde{p} \leqslant \tilde{q}\} \tag{3}$$

An important observation (*Obs. 1*) to make before going forward is the following: Consider $a, b \in \mathbb{U}_N$, and their corresponding wraps when added with $r \in \mathbb{U}_N$ are $w_a = \mathbf{1}\{a + r > N - 1\}$ and $w_b = \mathbf{1}\{b + r > N - 1\}$, respectively. If $w_a > w_b$, then $\tilde{a} < \tilde{b}$, where $\tilde{a} = a + r \bmod N$ and $\tilde{b} = b + r \bmod N$. This is true because when $w_a > w_b$: $\tilde{a} = a + r - N = r - (N - a) < r$, while $\tilde{b} = b + r \geqslant r$ (over $\mathbb{Z}$).
We proceed by analyzing 2 cases: **Case A** being $\tilde{p} \leqslant \tilde{q}$ and **Case B** being $\tilde{p} > \tilde{q}$.

Case A $\tilde{p} \leqslant \tilde{q}$: In this case, from Obs. 1 and Lemma 6, $w_p = w_q$. We get 3 subcases depending on the value of w_x.

- If $w_x = w_p = w_q$, Equation 3 becomes: $\mathbf{1}\{(\tilde{x} \geqslant \tilde{p}) \wedge (\tilde{x} \leqslant \tilde{q})\} = \mathbf{1}\{\tilde{x} \leqslant \tilde{q}\} + \mathbf{1}\{\tilde{x} \geqslant \tilde{p}\} - 1$. Here, if $\tilde{x} \geqslant \tilde{p}$, LHS = RHS, and if $\tilde{x} < \tilde{p}$, LHS = 0 and RHS = $1 + 0 - 1 = 0$ = LHS.
- If $w_x \neq w_p = w_q$ and $w_x = 0$, Equation 3 becomes: $\mathbf{1}\{(\tilde{x} \geqslant \tilde{p} + N) \wedge (\tilde{x} \leqslant \tilde{q} + N)\} = \mathbf{1}\{\tilde{x} \leqslant \tilde{q}\} + \mathbf{1}\{\tilde{x} \geqslant \tilde{p}\} - 1$. Here, LHS = 0, and RHS = $0 + 1 - 1$ = LHS (because $\tilde{x} > \tilde{q} \geqslant \tilde{p}$ from Obs. 1).
- If $w_x \neq w_p = w_q$ and $w_x = 1$, Equation 3 becomes: $\mathbf{1}\{(\tilde{x} + N \geqslant \tilde{p}) \wedge (\tilde{x} + N \leqslant \tilde{q})\} = \mathbf{1}\{\tilde{x} \leqslant \tilde{q}\} + \mathbf{1}\{\tilde{x} \geqslant \tilde{p}\} - 1$. Here also, LHS = 0, and RHS = $1 + 0 - 1$ = LHS (because $\tilde{x} < \tilde{p} \leqslant \tilde{q}$ from Obs. 1).

Case B $\tilde{p} > \tilde{q}$: Since $p \leqslant q$ and $\tilde{p} > \tilde{q}$, we have that $w_p = 0$ and $w_q = 1$.

- If $w_x = 0$, Equation 3 becomes: $\mathbf{1}\{(\tilde{x} \geqslant \tilde{p}) \wedge (\tilde{x} \leqslant \tilde{q} + N)\} = \mathbf{1}\{\tilde{x} \leqslant \tilde{q}\} + \mathbf{1}\{\tilde{x} \geqslant \tilde{p}\}$. LHS = $\mathbf{1}\{\tilde{x} \geqslant \tilde{p}\}$ and from Obs. 1, we have $\tilde{x} > \tilde{q}$, and therefore, RHS = $\mathbf{1}\{\tilde{x} \geqslant \tilde{p}\}$ = LHS.
- If $w_x = 1$, Equation 3 becomes: $\mathbf{1}\{(\tilde{x} + N \geqslant \tilde{p}) \wedge (\tilde{x} + N \leqslant \tilde{q} + N)\} = \mathbf{1}\{\tilde{x} \leqslant \tilde{q}\} + \mathbf{1}\{\tilde{x} \geqslant \tilde{p}\}$. LHS = $\mathbf{1}\{\tilde{x} \leqslant \tilde{q}\}$ and from Obs. 1, we have $\tilde{x} < \tilde{p}$, and therefore, RHS = $\mathbf{1}\{\tilde{x} \leqslant \tilde{q}\}$ = LHS.

□

G.2 Proof of Lemma 1

In the following unless explicitly stated using mod N, all expressions and equations are over $\mathbb{Z}$. We consider the natural embedding of $\mathbb{U}_N$ into $\mathbb{Z}$.

Proof. Let $w_a = \mathbf{1}\{a + r > N - 1\}$ and $w_b = \mathbf{1}\{b + r > N - 1\}$, then $(b - a)$ can be written as:

$$\begin{aligned}(b - a) &= (b + r - w_b \cdot N) - (a + r - w_b \cdot N)\\ &= \tilde{b} - \tilde{a} + (w_b - w_a) \cdot N \end{aligned} \tag{4}$$

We start by looking at the statement of the predicate $Q(x)$.

$$\begin{aligned}&x + (b - a) \bmod N < \tilde{b}\\ &x + (b - a) \bmod N < r + b \bmod N\\ &x + (b - a) \bmod N < (r + a) + (b - a) \bmod N\\ &x + (b - a) \bmod N < \tilde{a} + (b - a) \bmod N\end{aligned} \tag{5}$$

Now, we use the fact that $b > a$ and lift the predicate inequality (Equation 5) to $\mathbb{Z}$.

$$\begin{aligned}x + (b - a) - e_x \cdot N &< \tilde{a} + (b - a) - e_a \cdot N \quad \text{over } \mathbb{Z}\\ x + (\tilde{b} - \tilde{a} + (w_b - w_a) \cdot N) - e_x \cdot N &< \tilde{a} + (b - a) - e_a \cdot N \qquad \text{(From Equation 4)}\\ x + (\tilde{b} - \tilde{a} + (w_b - w_a) \cdot N) - e_x \cdot N &< (a + r - w_a \cdot N) + (b - a) - e_a \cdot N\\ x + (\tilde{b} - \tilde{a} + (w_b - w_a) \cdot N) - e_x \cdot N &< (b + r - w_a \cdot N) - e_a \cdot N\\ x + (\tilde{b} - \tilde{a}) - e_x \cdot N &< (b + r - w_b \cdot N) - e_a \cdot N\\ x + (\tilde{b} - \tilde{a}) - e_x \cdot N &< \tilde{b} - e_a \cdot N\\ x &< \tilde{a} + (e_x - e_a) \cdot N\end{aligned}$$

Therefore, we can substitute the statement of predicate $Q(x)$ with $x < \tilde{a} + (e_x - e_a) \cdot N$. Now, we do a case analysis on relation between e_x and e_a to prove that $P(x) = Q(x) + (e_a - e_x)$.

– Case $e_a = e_x$: In this case, $Q(x) + (e_a - e_x) = Q(x) = (x < \tilde{a}) = P(x)$.
– Case $e_x < e_a$: In this case, $x < \tilde{a}$ by Lemma 6 and $e_x = 0, e_a = 1$. Now, $Q(x) = (x < \tilde{a} - N) = 0$ because $0 \leqslant x, \tilde{a} < N$. Hence, $Q(x) + (e_a - e_x) = 1 = P(x)$.
– $e_x > e_a$: In this case, $x \geqslant \tilde{a}$ by Lemma 6 and $e_x = 1, e_a = 0$. Now, $Q(x) = (x < \tilde{a} + N) = 1$. Hence, $Q(x) + (e_a - e_x) = 0 = P(x)$.

This proves $P(x) = Q(x) + (e_a - e_x)$.

In order to prove $P'(x) = Q'(x) + (e_a - e_x)$, we make use of the following:

$$P'(x) = P(x) + \mathbf{1}\{x = \tilde{a}\} \tag{6}$$

$$Q'(x) = Q(x) + \mathbf{1}\{x + (b-a) \bmod N = \tilde{b}\} \tag{7}$$

It is straightforward to see why this is true. Let's look at the first equation $P'(x) = P(x) + \mathbf{1}\{x = \tilde{a}\}$. The only value of x where $P(x)$ and $P'(x)$ differ is $x = \tilde{a}$, $P(\tilde{a})$ outputs 0, while $P'(\tilde{a})$ outputs 1. For $x = \tilde{a}$, according to the equation, $P'(\tilde{a}) = P(\tilde{a}) + 1 = 1$, which matches the expected output of $P'(\tilde{a})$. Similarly, Equation 7 can be proven. Using Equation 4, the term $\mathbf{1}\{x + (b-a) \bmod N = \tilde{b}\}$ in Equation 7 can also be rewritten as $\mathbf{1}\{x = \tilde{a}\}$. Therefore,

$$Q'(x) = Q(x) + \mathbf{1}\{x = \tilde{a}\} \tag{8}$$

Replacing the value of $P(x)$ and $Q(x)$ from Equations 6 and 8 in $P(x) = Q(x) + (e_a - e_x)$, we get

$$P'(x) - \mathbf{1}\{x = \tilde{a}\} = Q'(x) - \mathbf{1}\{x = \tilde{a}\} + (e_a - e_x)$$
$$P'(x) = Q'(x) + (e_a - e_x)$$

□

G.3 Proof of Lemma 2

In the following unless explicitly stated using $\bmod\ N$, all expressions and equations are over $\mathbb{Z}$. We consider the natural embedding of $\mathbb{U}_N$ into $\mathbb{Z}$.

Proof. Using the basic wrap around property in $\mathbb{U}_N$, we have: $c' = c + 1 - wN$ over $\mathbb{Z}$, where $w = \mathbf{1}\{c = N-1\}$. Therefore, $c = c' - 1 + wN$. Replacing c in the statement of $R(x)$, we get:

$$x \leqslant c' - 1 + wN$$
$$x < c' + wN$$

When $w = 0$, i.e. $c \neq N - 1$, the statement of $R(x)$ becomes $x < c'$, which is the same as predicate $S(x)$. On the other hand, when $w = 1$, we have $c' = 0$, and $R(x)$ becomes $x < N$ which always outputs 1. In this case, $S(x)$ becomes $x < 0$ and therefore, always outputs 0. Hence, $R(x) = S(x) + \mathbf{1}\{c = N-1\}$ is correct. □

G.4 Interval Containment using $f^{<}_{(N-1)+\mathsf{r}^{\mathsf{in}},N-1}$ and $f^{\leqslant}_{(N-1)+\mathsf{r}^{\mathsf{in}},1}$

Here we show a construction of FSS gate for $g_{\mathsf{IC},n,p,q}$ that reduces any public interval $[p,q]$ to $f^{<}_{(N-1)+\mathsf{r}^{\mathsf{in}},N-1}$ and $f^{\leqslant}_{(N-1)+\mathsf{r}^{\mathsf{in}},1}$ in Fig. 13. This construction relies on Lemma 1 and builds over the simple interval containment from Fig. 2.

G.5 FSS Gate for Multiple Interval Containments

Fig. 14 shows our FSS gate construction for multiple interval containments.

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: $(k_0^{(L)}, k_1^{(L)}) \leftarrow \mathsf{Gen}^{<}_n(1^\lambda, \gamma, N-1, \mathbb{U}_N)$.
 3: $(k_0^{(R)}, k_1^{(R)}) \leftarrow \mathsf{Gen}^{\leqslant}_n(1^\lambda, \gamma, 1, \mathbb{U}_N)$.
 4: Set $\alpha^{(p)} = p + \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$ and $\alpha^{(q)} = q + \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$.
 5: Sample random $z_0, z_1 \leftarrow \mathbb{U}_N$ s.t. $z_0 + z_1 = \mathsf{r}^{\mathsf{out}} + \mathbf{1}\{\alpha^{(p)} > \alpha^{(q)}\} - \mathbf{1}\{\alpha^{(p)} > p\} + \mathbf{1}\{\alpha^{(q)} > q\}$.
 6: For $b \in \{0,1\}$, let $k_b = k_b^{(L)} || k_b^{(R)} || z_b$.
 7: return $(k_0, k_1)$.

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

Fig. 13: Construction for FSS Gate for Interval Containment $\mathcal{G}_{\mathsf{IC}}$ using $f^{<}_{(N-1)+\mathsf{r}^{\mathsf{in}},N-1}$ and $f^{\leqslant}_{(N-1)+\mathsf{r}^{\mathsf{in}},1}$, b refers to party id.

Multiple Interval Containment Gate $(\mathsf{Gen}^{\mathsf{MIC}}_{n,m,\{p_i,q_i\}_i}, \mathsf{Eval}^{\mathsf{MIC}}_{n,m,\{p_i,q_i\}_i})$

```
$\mathsf{Gen}^{\mathsf{MIC}}_{n,m,\{p_i,q_i\}_i}(1^\lambda, \mathsf{r}^{\mathsf{in}}, \{\mathsf{r}_i^{\mathsf{out}}\}_i)$:
 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: for $i = \{1, \ldots, m\}$ do
 4:     Set $q_i' = q_i + 1 \in \mathbb{U}_N$, $\alpha_i^{(p)} = p_i + \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$, $\alpha_i^{(q)} = q_i + \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$ and $\alpha_i^{(q')} = q_i + 1 + \mathsf{r}^{\mathsf{in}} \in \mathbb{U}_N$.
 5:     Sample random $z_{i,0}, z_{i,1} \leftarrow \mathbb{U}_N$ s.t.
    $z_{i,0} + z_{i,1} = \mathsf{r}_i^{\mathsf{out}} + \mathbf{1}\{\alpha_i^{(p)} > \alpha_i^{(q)}\} - \mathbf{1}\{\alpha_i^{(p)} > p_i\} + \mathbf{1}\{\alpha_i^{(q')} > q_i'\} + \mathbf{1}\{\alpha_i^{(q)} = N-1\}$.
 6: end for
 7: For $b \in \{0,1\}$, let $k_b = k_b^{(N-1)} || \{z_{i,b}\}_i$.
 8: return $(k_0, k_1)$.

$\mathsf{Eval}^{\mathsf{MIC}}_{n,m,\{p_i,q_i\}}(b, k_b, x)$:
 1: Parse $k_b = k_b^{(N-1)} || \{z_{i,b}\}_i$.
 2: for $i = \{1, \ldots, m\}$ do
 3:     Set $q_i' = q_i + 1 \bmod N$.
 4:     Set $x_i^{(p)} = x + (N-1-p_i) \in \mathbb{U}_N$ and $x_i^{(q')} = x + (N-1-q_i') \in \mathbb{U}_N$.
 5:     Set $s_{i,b}^{(p)} \leftarrow \mathsf{Eval}^{<}_n(b, k_b^{(N-1)}, x_i^{(p)})$.
 6:     Set $s_{i,b}^{(q')} \leftarrow \mathsf{Eval}^{<}_n(b, k_b^{(N-1)}, x_i^{(q')})$.
 7:     $y_{i,b} = b \cdot (\mathbf{1}\{x > p_i\} - \mathbf{1}\{x > q_i'\}) - s_{i,b}^{(p)} + s_{i,b}^{(q')} + z_{i,b}$.
 8: end for
 9: return $\{y_{i,b}\}_i$.
```

Fig. 14: FSS Gate for Multiple Interval Containment $\mathcal{G}_{\mathsf{MIC}}$. b refers to party id.

H Applications of Public Intervals

In this section, we present our FSS gate construction for Most Significant Non-Zero Bit function which is heavily used in many scientific computing applications [4,5]. Later in the section, we formally describe our construction for bit decomposition and prove its correctness.

H.1 Most Significant Non-Zero Bit (MSNZB)

Securely computing mathematical functions such as division (with a secret divisor), square root, trigonometric functions, logarithms and exponentials are crucial for many scientific computation tasks. Secure computation protocols for these functions has received much attention [4,5,27]. One of the most important sub-functions that is used in the secure computation of such functions is that of MSNZB, which is the unit vector indicating the position of the most significant non-zero bit. Mathematically, consider $x \in \mathbb{U}_N \setminus \{0\}$ parsed as $x = x_{n-1}||\dots||x_0$. For $j \in \{0, \dots, n-1\}$, define $\boldsymbol{e}_j \in \mathbb{U}_N^n$ to be the unit vector with 1 at the j^{th} location and 0 elsewhere. Then, $\mathsf{MSNZB}(x) = \boldsymbol{e}_j$ such that $x_j = 1$ and $x_k = 0, \forall k > j$. For completeness, $\mathsf{MSNZB}(0)$ is defined to be 0^n.

We now present a construction of an FSS gate for MSNZB. The MSNZB gate $\mathcal{G}_{\mathsf{MSNZB}}$ is the family of functions $\{g_{\mathsf{MSNZB},n} : \mathbb{U}_N \to \mathbb{U}_N^n\}$ parameterized by input and output groups $\mathbb{G}^{\mathsf{in}} = \mathbb{U}_N, \mathbb{G}^{\mathsf{out}} = \mathbb{U}_N^n$ and given by

$$\mathcal{G}_{\mathsf{MSNZB}} = \{g_{\mathsf{MSNZB},n} : \mathbb{U}_N \to \mathbb{U}_N^n\}, g_{\mathsf{MSNZB},n}(x) = \mathsf{MSNZB}(x)$$

We denote the corresponding offset gate class by $\hat{\mathcal{G}}_{\mathsf{MSNZB}}$ and the offset functions by $\hat{g}_{\mathsf{MSNZB},n}^{[\mathsf{r}^{\mathsf{in}},\{\mathsf{r}_i^{\mathsf{out}}\}]}(x) = g_{\mathsf{MSNZB},n}(x - \mathsf{r}^{\mathsf{in}}) + (\mathsf{r}_{n-1}^{\mathsf{out}}, \dots, \mathsf{r}_0^{\mathsf{out}})$, where the addition is done element-wise.

First, observe that $\mathsf{MSNZB}(x) = \boldsymbol{e}_j$ if and only if $x \in [2^j, 2^{j+1} - 1]$. Using this, we construct an FSS gate for $\mathcal{G}_{\mathsf{MSNZB}}$, by reducing it to the problem of multiple *public* interval containments that we constructed in Section 4.2. The theorem below captures our result formally.

```
MSNZB Gate ($\mathsf{Gen}_n^{\mathsf{MSNZB}}, \mathsf{Eval}_n^{\mathsf{MSNZB}}$)
$\mathsf{Gen}_n^{\mathsf{MSNZB}}(1^\lambda, \mathsf{r}^{\mathsf{in}}, \{\mathsf{r}_i^{\mathsf{out}}\}_i)$:
 1: For $i \in \{0, 1, \dots, n-1\}$, define $p_i = 2^i$ and $q_i = 2^{i+1} - 1$.
 2: return $(k_0, k_1) \leftarrow \mathsf{Gen}_{n,n,\{p_i,q_i\}_i}^{\mathsf{MIC}}(1^\lambda, \mathsf{r}^{\mathsf{in}}, \{\mathsf{r}_i^{\mathsf{out}}\}_i)$.

$\mathsf{Eval}_n^{\mathsf{MSNZB}}(b, k_b, x)$:
 1: For $i \in \{0, 1, \dots, n\}$, define $p_i = 2^i$ and $q_i = 2^{i+1} - 1$.
 2: return $\{z_{i,b}\}_i \leftarrow \mathsf{Eval}_{n,n,\{p_i,q_i\}_i}^{\mathsf{MIC}}(b, k_b, x)$
```

Fig. 15: FSS Gate for Most Significant Non-Zero Bit $\mathcal{G}_{\mathsf{MSNZB}}$, b refers to party id.

Theorem 14. *There is an FSS gate $(\mathsf{Gen}_n^{\mathsf{MSNZB}}, \mathsf{Eval}_n^{\mathsf{MSNZB}})$ for $\mathcal{G}_{\mathsf{MSNZB}}$ that has key size equivalent to $\mathcal{G}_{\mathsf{MIC}}$ over $\mathbb{U}_N$ computing n public intervals and evaluates it only once.*

Proof. Our construction of FSS Gate for MSNZB is given in Fig. 15. The security of this construction follows from the security of FSS gate for $\mathcal{G}_{\mathsf{MIC}}$. For correctness, we first use the fact that $\mathsf{MSNZB}(x) = \boldsymbol{e}_j$ if and only if $x \in [2^j, 2^{j+1} - 1]$. Next, let $z_i = z_{i,0} + z_{i,1} \in \mathbb{U}_N$. Then, by correctness of FSS gate for $\mathcal{G}_{\mathsf{MIC}}$, it holds that $z_i = \mathbf{1}\{2^i \leqslant (x - \mathsf{r}^{\mathsf{in}}) \leqslant 2^{i+1} - 1\} + \mathsf{r}_i^{\mathsf{out}}$. Also, note that all intervals $[p_i, q_i]$ for $i \in \{0, \dots, n-1\}$ are disjoint. Hence, for $x \in [2^j, 2^{j+1} - 1]$, $z_j = 1 + \mathsf{r}_j^{\mathsf{out}}$, and $z_i = \mathsf{r}_i^{\mathsf{out}}$ for all $i \in \{0, \dots, n-1\} \setminus j$. It is easy to see that for the corner case of $x = 0$, all $z_i = \mathsf{r}_i^{\mathsf{out}}$. □

H.2 Bit Decomposition

We present the full construction of the FSS gate for $\mathcal{G}_{\mathsf{BIT}}$ in Fig. 16.

Lemma 8. *The FSS gate presented in Fig. 16 for $\mathcal{G}_{\mathsf{BIT}}$ is correct.*

```
$\mathsf{BIT}$ Gate $(\mathsf{Gen}^{\mathsf{BIT}}_{n,w}, \mathsf{Eval}^{\mathsf{BIT}}_{n,w})$
$\mathsf{Gen}^{\mathsf{BIT}}_{n,w}(1^\lambda, \mathsf{r}^{\mathsf{in}}, \{\mathsf{r}^{\mathsf{out}}_i\}_i)$:
 1: $(k_0, k_1) = (\phi, \phi), i = n - 1$.
 2: while $i \geqslant 0$ do
 3:     $n' = i + 1, N' = 2^{n'}, \mathsf{r}^{\mathsf{in}\prime} = \mathsf{r}^{\mathsf{in}}{}_{[0,i+1)} \in \mathbb{U}_{N'}, \gamma' = N' - 1 + \mathsf{r}^{\mathsf{in}\prime} \bmod N' \in \mathbb{U}_{N'}$.
 4:     $(k_0^{(i)}, k_1^{(i)}) \leftarrow \mathsf{Gen}^{<}_{n'}(1^\lambda, \gamma', 1, \mathbb{U}_N)$.
 5:     $j = 0$.
 6:     while $j < w$ & $i - j \geqslant 0$ do
 7:         $p = i - j$.
 8:         $S^{(i,j)} = \{[2^p + k \cdot 2^{p+1}, 2^{p+1} + k \cdot 2^{p+1} - 1], \forall k \in \{0, 1, \ldots 2^{i-p} - 1\}\}$.
 9:         $z^{(i,j)} = \mathsf{r}^{\mathsf{out}}_p \in \mathbb{U}_N$.
10:         for $[a_0, a_1] \in S^{(i,j)}$ do
11:             $\alpha^{(a_0)} = a_0 + \mathsf{r}^{\mathsf{in}\prime}, \alpha^{(a_1)} = a_1 + \mathsf{r}^{\mathsf{in}\prime}, \alpha^{(a_1')} = a_1 + 1 + \mathsf{r}^{\mathsf{in}\prime} \in \mathbb{U}_{N'}$.
12:             $z^{(i,j)} = z^{(i,j)} + \mathbf{1}\{\alpha^{(a_0)} > \alpha^{(a_1)}\} - \mathbf{1}\{\alpha^{(a_0)} > a_0\} + \mathbf{1}\{\alpha^{(a_1')} > a_1 + 1 \bmod N'\} + \mathbf{1}\{\alpha^{(a_1)} = N' - 1\}$.
13:         end for
14:         Sample random $z_0^{(i,j)}, z_1^{(i,j)} \in \mathbb{U}_N$ s.t. $z_0^{(i,j)} + z_1^{(i,j)} = z^{(i,j)}$.
15:         $j = j + 1$
16:     end while
17:     For $b \in \{0, 1\}$, $k_b = k_b || (k_b^{(i)} || \{z_b^{(i,j)}\}_j)$.
18:     $i = i - w$
19: end while
20: return $(k_0, k_1)$.

$\mathsf{Eval}^{\mathsf{BIT}}_{n,w}(b, k_b, x)$:
 1: $k_b' = k_b, i = n - 1$.
 2: while $i \geqslant 0$ do
 3:     Parse $k_b' = (k_b^{(i)} || \{z_b^{(i,j)}\}_j) || t_b$. Set $k_b' = t_b$.
 4:     $n' = i + 1, N' = 2^{n'}, x' = x_{[0,i+1)} \in \mathbb{U}_{N'}$.
 5:     $j = 0$.
 6:     while $j < w$ & $i - j \geqslant 0$ do
 7:         $p = i - j$.
 8:         $S^{(i,j)} = \{[2^p + k \cdot 2^{p+1}, 2^{p+1} + k \cdot 2^{p+1} - 1], \forall k \in \{0, 1, \ldots 2^{i-p} - 1\}\}$.
 9:         $y_b^{(i,j)} = z_b^{(i,j)}$.
10:         for $[a_0, a_1] \in S^{(i,j)}$ do
11:             $a_1' = a_1 + 1 \bmod N', x^{(a_0)} = x' + (N' - 1 - a_0) \in \mathbb{U}_{N'}, x^{(a_1')} = x' + (N' - 1 - a_1') \in \mathbb{U}_{N'}$.
12:             $s_{b,0}^{(i,j,a_0,a_1)} \leftarrow \mathsf{Eval}^{<}_{n'}(b, k_b^{(i)}, x^{(a_0)})$.
13:             $s_{b,1}^{(i,j,a_0,a_1)} \leftarrow \mathsf{Eval}^{<}_{n'}(b, k_b^{(i)}, x^{(a_1')})$.
14:             $y_b^{(i,j)} = y_b^{(i,j)} + b(\mathbf{1}\{x' > a_0\} - \mathbf{1}\{x' > a_1'\}) - s_{b,0}^{(i,j,a_0,a_1)} + s_{b,1}^{(i,j,a_0,a_1)}$.
15:         end for
16:         $j = j + 1$
17:     end while
18:     $i = i - w$
19: end while
20: return $\{y_b^{(i,j)}\}_{i,j}$.
```
Fig. 16: FSS Gate for Bit Decomposition $\mathcal{G}_{\mathsf{BIT}}$. b refers to party id and ϕ refers to the empty string. w refers to a parameter in our construction and which we set as $w = \lceil \log n \rceil$.

Proof. We analyze a particular iteration of the loops for iterators (i, j) in Fig. 16.

$$\begin{aligned} y^{(i,j)} &= y_0^{(i,j)} + y_1^{(i,j)} \\ &= z^{(i,j)} + \sum_{(a_0,a_1)\in S^{(i,j)}} \left(\begin{array}{r} \mathbf{1}\{x' > a_0\} - \mathbf{1}\{x' > a_1'\} - (s_{0,0}^{(i,j,a_0,a_1)} + s_{1,0}^{(i,j,a_0,a_1)}) \\ +(s_{0,1}^{(i,j,a_0,a_1)} + s_{1,1}^{(i,j,a_0,a_1)}) \end{array} \right) \\ &= \mathsf{r}^{\mathsf{out}}_{i-j} + \sum_{(a_0,a_1)\in S^{(i,j)}} \beta^{(i,j,a_0,a_1)} \end{aligned}$$

where

$$\begin{aligned} \beta^{(i,j,a_0,a_1)} &= \mathbf{1}\{x' > a_0\} - \mathbf{1}\{x' > a_1'\} - \mathbf{1}\{x^{(a_0)} < \gamma'\} + \mathbf{1}\{x^{(a_1')} < \gamma'\} \\ &+ \mathbf{1}\{\alpha^{(a_0)} > \alpha^{(a_1)}\} - \mathbf{1}\{\alpha^{(a_0)} > a_0\} + \mathbf{1}\{\alpha^{(a_1')} > a_1 + 1 \bmod N'\} + \mathbf{1}\{\alpha^{(a_1)} = N' - 1\} \end{aligned} \tag{9}$$

By the correctness of IC construction in Fig. 3, we have the following for $x \in \mathbb{U}_N$

$$\begin{aligned} \mathbf{1}\{p \leqslant (x - \mathsf{r}^{\mathsf{in}}) \bmod N \leqslant q\} + \mathsf{r}^{\mathsf{out}} &= \mathbf{1}\{x > p\} - \mathbf{1}\{x > q'\} - \mathbf{1}\{x^{(p)} < \gamma\} + \mathbf{1}\{x^{(q')} < \gamma\} \\ &+ \mathbf{1}\{\alpha^{(p)} > \alpha^{(q)}\} - \mathbf{1}\{\alpha^{(p)} > p\} + \mathbf{1}\{\alpha^{(q')} > q'\} + \mathbf{1}\{\alpha^{(q)} = N - 1\} + \mathsf{r}^{\mathsf{out}} \end{aligned} \tag{10}$$

We use the above equation in Equation 9 by setting $N = N', p = a_0, q = a_1, \mathsf{r}^{\mathsf{in}} = \mathsf{r}^{\mathsf{in}'}, \mathsf{r}^{\mathsf{out}} = 0^{n'}, x = x', \gamma = \gamma', q' = a_1' = a_1 + 1 \bmod N'$ to get

$$\beta^{(i,j,a_0,a_1)} = \mathbf{1}\{a_0 \leqslant (x' - \mathsf{r}^{\mathsf{in}'}) \bmod N' \leqslant a_1\}$$

Therefore,

$$y^{(i,j)} = \mathsf{r}^{\mathsf{out}}_{i-j} + \sum_{(a_0,a_1)\in S^{(i,j)}} \mathbf{1}\{a_0 \leqslant (x' - \mathsf{r}^{\mathsf{in}'}) \bmod N' \leqslant a_1\} \tag{11}$$

We next prove the following claim.

Claim. If $w' = (x' - \mathsf{r}^{\mathsf{in}'}) \bmod N'$ and $w = (x - \mathsf{r}^{\mathsf{in}}) \bmod N$, then $\sum_{(a_0,a_1)\in S^{(i,j)}} \mathbf{1}\{a_0 \leqslant w' \leqslant a_1\} = w_{[i-j]}$.

Proof. Note that since $x' = x_{[0,i+1)}, \mathsf{r}^{\mathsf{in}'} = {\mathsf{r}^{\mathsf{in}}}_{[0,i+1)}$ and $N' = 2^{n'} = 2^{i+1}$, we have $w' = w_{[0,i+1)}$. Also, note that

$$\sum_{(a_0,a_1)\in S^{(i,j)}} \mathbf{1}\{a_0 \leqslant w' \leqslant a_1\} = \sum_{k\in\{0,1,\ldots 2^j-1\}} \mathbf{1}\{2^p + k \cdot 2^{p+1} \leqslant w' \leqslant 2^{p+1} - 1 + k \cdot 2^{p+1}\}$$

where $p = i - j$. Note that each of the intervals in the summations is a disjoint interval and only one of them can ever be 1. We find one such interval in this summation for $w_{[i-j]}$, which would prove our claim.

If $w' = w_{[i-j+1,i+1)}||w_{[i-j]}||w_{[0,i-j)}$ is the bit-representation of w', we have $w' = w_{[i-j+1,i+1)} \cdot 2^{i-j+1} + w_{[i-j]} \cdot 2^{i-j} + w_{[0,i-j)}$. Setting $k' = w_{[i-j+1,i+1)}$ and bounding by the maximum and minimum values of $w_{[0,i-j)}$ as $2^{i-j} - 1$ and 0 respectively, we get

$$k' \cdot 2^{i-j+1} + w_{[i-j]}2^{i-j} \leqslant w' \leqslant k' \cdot 2^{i-j+1} - 1 + (w_{[i-j]} + 1)2^{i-j}$$

Clearly, now $w_{[i-j]} = 1 \iff \mathbf{1}\{k' \cdot 2^{i-j+1} + 2^{i-j} \leqslant w' \leqslant k' \cdot 2^{i-j+1} - 1 + 2^{i-j+1}\} = 1$. The claim follows. □

From Equation 10 and the above claim, we have

$$y^{(i,j)} = \mathsf{r}^{\mathsf{out}}_{i-j} + w_{[i-j]}, \text{where } w = (x - \mathsf{r}^{\mathsf{in}}) \bmod N$$

□

I Proofs and Supplemental Material for Section 6

In this section, we first describe the FSS gate constructions for addition and multiplication from [21]. We then present the proof of Lemma 3 over which we build our LRS FSS gate. Finally, we provide the proof of Lemma 4 used in our ARS construction.

I.1 Addition

The addition gate $\mathcal{G}_+$ is the family of functions $g_{+,n} : \mathbb{U}_N \times \mathbb{U}_N \to \mathbb{U}_N$ parameterized by input group $\mathbb{G}^{\mathsf{in}} = \mathbb{U}_N \times \mathbb{U}_N$ and output group $\mathbb{G}^{\mathsf{out}} = \mathbb{U}_N$, and given by $g_{+,n}(x_1, x_2) := x_1 + x_2$. We denote the corresponding offset gate class by $\hat{\mathcal{G}}_+$ and the offset functions by $\hat{g}_{+,n}^{[\mathsf{r}_1^{\mathsf{in}}, \mathsf{r}_2^{\mathsf{in}}, \mathsf{r}^{\mathsf{out}}]}(x_1, x_2) = g_{+,n}(x_1 - \mathsf{r}_1^{\mathsf{in}}, x_2 - \mathsf{r}_2^{\mathsf{in}}) + \mathsf{r}^{\mathsf{out}} = (x_1 - \mathsf{r}_1^{\mathsf{in}}) + (x_2 - \mathsf{r}_2^{\mathsf{in}}) + \mathsf{r}^{\mathsf{out}}$.

Proposition 1 (Addition gate)**.** *There is an FSS Gate* $(\mathsf{Gen}_n^+, \mathsf{Eval}_n^+)$ *for* $\mathcal{G}_+$ *that has a total key size of* n *bits.*

Proof. We present our construction of FSS Gate for + formally in Fig. 17. □

```
Addition Gate (Gen_n^+, Eval_n^+)
$\mathsf{Gen}_n^+(1^\lambda, \mathsf{r}_1^{\mathsf{in}}, \mathsf{r}_2^{\mathsf{in}}, \mathsf{r}^{\mathsf{out}})$:
 1: Sample random $R_0, R_1 \leftarrow \mathbb{U}_N$ s.t. $R_0 + R_1 = \mathsf{r}^{\mathsf{out}} - (\mathsf{r}_1^{\mathsf{in}} + \mathsf{r}_2^{\mathsf{in}})$.
 2: For $b \in \{0,1\}$, let $k_b = R_b$.
 3: return $(k_0, k_1)$.

$\mathsf{Eval}_n^+(b, k_b, x_1, x_2)$:
 1: Parse $k_b = R_b$.
 2: return $x_1 + x_2 + R_b$.
```

Fig. 17: FSS Gate for Addition $\mathcal{G}_+$, b refers to party id.

I.2 Multiplication

Multiplication of two values $x_1, x_2 \in \mathbb{U}_N$ refers to the multiplication of the two values x_1 and x_2 carried out in the group $\mathbb{U}_N$. The multiplication gate $\mathcal{G}_\times$ is the family of functions $g_{\times,n} : \mathbb{U}_N \times \mathbb{U}_N \to \mathbb{U}_N$ parameterized by input group $\mathbb{G}^{\mathsf{in}} = \mathbb{U}_N \times \mathbb{U}_N$ and output group $\mathbb{G}^{\mathsf{out}} = \mathbb{U}_N$, and given by $g_{\times,n}(x_1, x_2) := x_1 \cdot x_2$. We denote the corresponding offset gate class by $\hat{\mathcal{G}}_\times$ and the offset functions by $\hat{g}_{\times,n}^{[\mathsf{r}_1^{\mathsf{in}}, \mathsf{r}_2^{\mathsf{in}}, \mathsf{r}^{\mathsf{out}}]}(x_1, x_2) = g_{\times,n}(x_1 - \mathsf{r}_1^{\mathsf{in}}, x_2 - \mathsf{r}_2^{\mathsf{in}}) + \mathsf{r}^{\mathsf{out}} = (x_1 - \mathsf{r}_1^{\mathsf{in}}) \cdot (x_2 - \mathsf{r}_2^{\mathsf{in}}) + \mathsf{r}^{\mathsf{out}}$.

Proposition 2 (Multiplication gate)**.** *There is an FSS Gate* $(\mathsf{Gen}_n^\times, \mathsf{Eval}_n^\times)$ *for* $\mathcal{G}_\times$ *that has a total key size of* $3n$ *bits.*

Proof. We present the construction of FSS Gate for $\times$ formally in Fig. 18. □

```
Multiplication Gate (Gen_n^×, Eval_n^×)
$\mathsf{Gen}_n^\times(1^\lambda, \mathsf{r}_1^{\mathsf{in}}, \mathsf{r}_2^{\mathsf{in}}, \mathsf{r}^{\mathsf{out}})$:
 1: Sample random $P_0, P_1 \leftarrow \mathbb{U}_N$ s.t. $P_0 + P_1 = \mathsf{r}_1^{\mathsf{in}}$.
 2: Sample random $Q_0, Q_1 \leftarrow \mathbb{U}_N$ s.t. $Q_0 + Q_1 = \mathsf{r}_2^{\mathsf{in}}$.
 3: Sample random $R_0, R_1 \leftarrow \mathbb{U}_N$ s.t. $R_0 + R_1 = \mathsf{r}_1^{\mathsf{in}} \cdot \mathsf{r}_2^{\mathsf{in}} + \mathsf{r}^{\mathsf{out}}$.
 4: For $b \in \{0,1\}$, let $k_b = P_b || Q_b || R_b$.
 5: return $(k_0, k_1)$.

$\mathsf{Eval}_n^\times(b, k_b, x_1, x_2)$:
 1: Parse $k_b = P_b || Q_b || R_b$.
 2: return $b \cdot (x_1 \cdot x_2) - x_1 \cdot Q_b - x_2 \cdot P_b + R_b$.
```

Fig. 18: FSS Gate for Multiplication $\mathcal{G}_\times$, b refers to party id.

I.3 Proof of Lemma 3

Proof. Let $x_0 = x_0^{(0)} + 2^s \cdot x_0^{(1)}$ and $x_1 = x_1^{(0)} + 2^s \cdot x_1^{(1)}$, where $x_b^{(0)} = x_{b[0,s)}$ and $x_b^{(1)} = x_{b[s,n)}$. Also, $x = x_0 + x_1 - 2^n \cdot t^{(n)}$. Hence, we can write

$$\begin{aligned} x &= x_0^{(0)} + 2^s \cdot x_0^{(1)} + x_1^{(0)} + 2^s \cdot x_1^{(1)} - 2^n \cdot t^{(n)} \\ &= 2^s \cdot (x_0^{(1)} + x_1^{(1)}) - 2^n \cdot t^{(n)} + x_0^{(0)} + x_1^{(0)} \\ &= 2^s \cdot (x_0^{(1)} + x_1^{(1)} + t^{(s)}) - 2^n \cdot t^{(n)} + \underbrace{x_0^{(0)} + x_1^{(0)} - 2^s \cdot t^{(s)}}_{<2^s} \end{aligned}$$

The sum of the last 3 terms in the above expression is $< 2^s$. Thus,

$$\begin{aligned} (x \gg_L s) &= x_0^{(1)} + x_1^{(1)} + t^{(s)} - 2^{n-s} \cdot t^{(n)} \\ &= (x_0 \gg_L s) + (x_1 \gg_L s) + t^{(s)} - 2^{n-s} \cdot t^{(n)} \end{aligned}$$

□

I.4 Proof of Lemma 4

Proof. All the equations below are over $\mathbb{U}_N$. Substring function on signed integer $x \in \mathbb{S}_N$ is defined as: $x_{[0,i)} = (x \bmod N)_{[0,i)} \in \mathbb{U}_{2^i}$. Similarly, the bit function on signed $x \in \mathbb{S}_N$ is: $x_{[i]} = (x \bmod N)_{[i]}$. Using mathematical definitions of logical right shift and arithmetic right shift, it is easy to see that:

$$(x \gg_A s) \equiv_{\mathsf{s}} (x_{[0,n-1)} \gg_L s) + \sum_{i=0}^{s} 2^{n-s-1+i} \cdot x_{[n-1]} \tag{12}$$

Next, we observe the following:

$$\begin{aligned} \underbrace{x_{[n-1]} \dots x_{[n-1]} \dots x_{[n-1]}}_{n} &= 2^n - x_{[n-1]} \\ \underbrace{x_{[n-1]} \dots x_{[n-1]}}_{s+1}\underbrace{0 \dots 0}_{n-s-1} &= 2^{n-s-1} \cdot (2^n - x_{[n-1]}) \\ \underbrace{x_{[n-1]} \dots x_{[n-1]}}_{s+1}\underbrace{0 \dots 0}_{n-s-1} &= 2^n - 2^{n-s-1} \cdot x_{[n-1]} \end{aligned}$$

Now, we can rewrite the LHS of the above equation as:

$$\sum_{i=0}^{s} 2^{n-s-1+i} \cdot x_{[n-1]} = 2^n - 2^{n-s-1} \cdot x_{[n-1]}$$

Using the above in Equation 12, we get

$$(x \gg_A s) \equiv_{\mathsf{s}} (x_{[0,n-1)} \gg_L s) - 2^{n-s-1} \cdot x_{[n-1]} \tag{13}$$

Next, note that $x_{[0,n-1)} = x_{0\,[0,n-1)} + x_{1\,[0,n-1)}$ over $\mathbb{U}_{2^{n-1}}$. Hence, using Lemma 3, following holds over $\mathbb{Z}$ and hence, also over $\mathbb{U}_N$.

$$(x_{[0,n-1)} \gg_L s) = (x_{0\,[0,n-1)} \gg_L s) + (x_{1\,[0,n-1)} \gg_L s) + t^{(s)} - 2^{n-s-1} \cdot t^{(n-1)}$$

Replacing this in Equation 13, we get our final expression. □