

Function Secret Sharing Continued

CS 507, Topics in Cryptography: Secure Computation

David Heath

Fall 2026

Recall that last time we introduced the notion of *function secret sharing*. Here, a dealer wishes to “share” some function between two (or more) parties.

Definition 1 (Function Secret Sharing). *Let $\mathcal{F}$ be a family of functions $f : D \rightarrow G$, where G is an abelian group. A two-party FSS scheme for $\mathcal{F}$ is a pair of algorithms $(\text{KeyGen}, \text{Eval})$:*

- $\text{KeyGen}(1^\lambda, f)$ takes (the description of) a function $f \in \mathcal{F}$ and outputs a pair of keys (k_0, k_1) .
- $\text{Eval}(k_b, x)$ takes a key k_b and an input $x \in D$; it outputs a group element $y_b \in G$.

The scheme must satisfy:

- **Correctness.** For every $f \in \mathcal{F}$ and every $x \in D$,

$$\Pr [\text{Eval}(k_0, x) + \text{Eval}(k_1, x) = f(x) \mid (k_0, k_1) \leftarrow \text{KeyGen}(1^\lambda, f)] = 1.$$

- **Security.** There exists a PPT simulator $\mathcal{S}$ such that for every $b \in \{0, 1\}$ and every $f \in \mathcal{F}$,

$$\{ k_b \mid (k_0, k_1) \leftarrow \text{KeyGen}(1^\lambda, f) \} \stackrel{c}{=} \{ \mathcal{S}(1^\lambda, b) \}$$

That is, an FSS scheme gives a mechanism by which to “share” a function f into two keys, such that each key alone reveals nothing about f beyond its public description.

Last time we focused on so-called *distributed point functions* (DPFs), which are function secret sharing schemes for the family of *point functions*:

$$\text{point}_{\alpha, \beta}(x) := \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{otherwise} \end{cases}$$

Today, we will continue our exploration of FSS. In particular we will:

- Explore an additional application of FSS to so-called private aggregation protocols.
- Build on our understanding of DPFs to construct FSS schemes for other function families.
- Use FSS to construct MPC protocols.

Application: Private Aggregation

So far we have used DPFs to *read*: a client compresses a one-hot query, and two servers apply it to a public database. However, the same DPF primitive can be used to allow many clients to *write* into a secret-shared database. The capability is broadly useful, such as for handling a privacy-preserving election.

Concretely, consider two non-colluding servers S_0, S_1 and many clients. For simplicity, we will today assume the adversary is semi-honest. Client i holds a value $v_i \in [n]$ (the item the client wishes to vote for).

The servers wish to learn the *histogram* $H \in \mathbb{Z}^n$, $H_x = |\{i \mid v_i = x\}|$, and nothing about any individual vote v_i .

It is straightforward to use DPFs to achieve the above goal, and the resulting protocol runs in constant rounds and is quite concretely efficient. In particular, each client i samples two DPF keys $(k_{i,0}, k_{i,1}) \leftarrow \text{KeyGen}(\text{point}_{v_i,1})$ over the group $\mathbb{G} = \mathbb{Z}_{2^{32}}$, say, and sends $k_{i,b}$ to S_b . Each server performs a full-domain evaluation of every key it receives and sums the results:

$$h_b := \sum_i (\text{Eval}(k_{i,b}, x))_{x \in [n]}$$

By correctness of the DPF, $h_0 + h_1 = \sum_i \text{hot}(v_i) = H$. The servers exchange their shares h_0, h_1 and reconstruct the histogram.

The above protocol is clearly semi-honest secure. Each server S_b simply obtains one DPF key per client, which hides the individual vote, and obtains shares of the histogram, which are implied by the output of the functionality.

Cost. Each client sends $O(\lambda \log n)$ bits, independent of the number of clients. Each server spends $O(n)$ PRG calls per client. For n in the millions and thousands of clients this is a few seconds of AES, which is why this design has been deployed at scale; see e.g. [BBC⁺21].

Malicious clients. Note that nothing above prevents a client from sending keys for a function that is not a point function: a malicious client might submit a large number of votes for their chosen candidate, or perhaps cleverly vote for several candidates, or even corrupt the entire histogram with random noise. Here, we simply note that it is possible for the servers to efficiently check that client keys are well formed, which allows simulation in the presence of malicious clients; see e.g. [BGI16] and follow-on works.

Distributed Comparison Functions

While we have seen that distributed point functions are flexible, they encode only one type of function family. One might also wish to distribute keys from other families. We next demonstrate another central example, which is called a *distributed comparison function* (DCF).

Definition 2 (Distributed Comparison Function). *For $\alpha \in \{0, 1\}^k$, read as an integer in $[0, 2^k)$, and $\beta \in \mathbb{G}$, the comparison function is*

$$\text{less}_{\alpha, \beta}(x) := \begin{cases} \beta & \text{if } x < \alpha \\ 0 & \text{otherwise.} \end{cases}$$

A distributed comparison function (DCF) is an FSS scheme for the family of comparison functions.

Remarkably, the DPF construction we have already seen can be straightforwardly manipulated to yield a DCF. Indeed, recall that the DPF corresponds to a binary tree, and that tree encodes, at every level j , whether the first j bits of the evaluation point x agree with those of α . Notice that $x < \alpha$ is precisely a statement about where x first leaves α 's path. Writing x and α in terms of their bits, i.e. $x = x_0 \cdots x_{k-1}$ and $\alpha = \alpha_0 \cdots \alpha_{k-1}$, we can observe the following equivalence.

$$(x < \alpha) \Leftrightarrow \sum_{j=0}^{k-1} ((x_{<j} = \alpha_{<j}) \wedge x_j = 0 \wedge \alpha_j = 1),$$

This means that it suffices that, as the servers walk down x 's path through the DPF tree, they obtain at each level j an additive share of $\beta \cdot (x_j = 0 \wedge \alpha_j = 1)$ if the parent node is the active node $\alpha_{<j}$, and a share of 0 otherwise. Summing these k shares gives a share of $\text{less}_{\alpha, \beta}(x)$, and the servers never leave x 's path. This is arranged by having each tree node carry a value in $\mathbb{G}$ alongside its seed and indicator bit, expanded from the seed by the PRG, and by adding to each level's correction word a value correction that fixes the two children of the active node; the correction is applied masked by the shared indicator bit, exactly as for the DPF. Each correction word therefore grows by $O(w)$ bits, where w is the bit-length of a group element.

One can also share the function $\mathbf{greater}_{\alpha,\beta}$ symmetrically.

Hence, there exists an FSS scheme for comparison functions with key length $O(k(\lambda+w))$ bits and incurring $O(k)$ PRG calls to generate/evaluate.

Exercise 1. Write out procedures $\mathbf{KeyGen}$ and $\mathbf{Eval}$ for DCFs. Prove correctness and security.

Other Function Families

DPFs and DCFs are clearly highly related in terms of their constructions: each leverages the same binary-tree-based construction. There are several other function families that can be similarly built up.

Intervals and more. Consider the interval between $[\ell, r)$. I.e. the following family of functions:

$$\mathbf{interval}_{\ell,r,\beta}(x) := \begin{cases} \beta & \text{if } \ell \leq x < r \\ 0 & \text{otherwise} \end{cases}$$

A bit of straightforward reasoning reveals that an FSS scheme for intervals can be constructed from two DCFs. Namely, the dealer shares $\mathbf{less}(r, \beta)$ and $\mathbf{less}(\ell, -\beta)$. To evaluate, the servers evaluate each key, then sum the resulting shares.

The above construction illustrates the compositional nature of certain FSS schemes. Similar compositional approaches can be used to obtain FSS schemes for multi-point functions, splines, and even decision trees; see e.g. [BGI16, BGI19].

All of the above are constructible from any PRG. Most desirable would be an FSS scheme for arbitrary poly-size function classes. Unfortunately, while such constructions are known, they require much stronger (and hence more expensive) forms of cryptography, such as assumptions known to imply fully homomorphic encryption. Finding other function families that admit simple constructions from PRGs remains an interesting open problem.

Secure Computation from FSS

We now use FSS to construct dealer-assisted semi-honest secure two-party computation protocols. Throughout, values are shared additively over a group, typically $\mathbb{Z}_{2^k}$: $\llbracket x \rrbracket$ denotes (x_0, x_1) with $x_0 + x_1 = x$, held by P_0, P_1 respectively. Linear operations on shares are free, and multiplication is handled by Beaver triples, which a dealer can supply. What remains is the non-linear, non-arithmetic gates: comparisons, sign, ReLU, truncation, arbitrary small tables.

Offset functions. Fix a gate $g : \mathbb{G}_{\text{in}} \rightarrow \mathbb{G}_{\text{out}}$ between abelian groups. For $r \in \mathbb{G}_{\text{in}}$ define the *offset function*

$$g'_r(y) := g(y - r).$$

An FSS scheme for the family of functions g' is one whose keys hide the shift r .

The general-purpose FSS gate. Suppose parties hold $\llbracket x \rrbracket$ and wish to compute $\llbracket g(x) \rrbracket$. To achieve this, the dealer samples $r \leftarrow \mathbb{G}_{\text{in}}$, computes $(k_0, k_1) \leftarrow \mathbf{KeyGen}(1^\lambda, g'_r)$, and gives P_b the pair (r_b, k_b) where $r_0 + r_1 = r$. In the online phase, the two parties proceed as follows:

1. Each P_b sends $x_b + r_b$ to the other party. Both now hold $y := x + r$.
2. Each P_b outputs $z_b := \mathbf{Eval}(k_b, y)$.
3. Note that by correctness of FSS, the parties now hold:

$$\llbracket g'_r(y) \rrbracket = \llbracket g(y - r) \rrbracket = \llbracket g((x + r) - r) \rrbracket = \llbracket g(x) \rrbracket$$

So the gate is correct.

Note that this costs only a single round, and each party transmits only a single group element. The idea is that r acts as a one-time pad on x , so it is safe to reveal y ; the FSS keys then let the parties apply g to the padded value while the dealer has, in effect, pre-subtracted the pad.

Exercise 2. Consider the following ideal functionality for a semi-honest FSS gate:

- $\mathcal{F}_g$ interacts with P_0, P_1 , and the adversary.
- Receive $x_b \in \mathbb{G}_{\text{in}}$ from each P_b , and set $x := x_0 + x_1$.
- Compute $z = g(x)$.
- If P_b is corrupt, receive $z_b \in \mathbb{G}_{\text{out}}$ from the adversary; otherwise sample $z_b \leftarrow \mathbb{G}_{\text{out}}$ for one honest party.
- Set $z_{1-b} := z - z_b$ and send z_b to P_b for each b .

Prove the above FSS-based protocol instantiates the above ideal functionality against a semi-honest adversary that statically corrupts any one party by constructing simulators.

It is worth emphasizing that this works for *any* function g for which we can construct FSS keys for the corresponding function family g' . We explore examples of functions g where this becomes possible below:

Small lookup tables. If the gate's input domain is small, then we can compute *every* such g in the above manner. In particular, the offset function g'_r has truth table $\sum_{x \in G_{\text{in}}} g(x) \cdot \text{hot}(x + r)$, i.e. g 's truth table cyclically shifted by r . The dealer therefore shares a single DPF for $\text{point}_{r,1}$; each party performs a full-domain evaluation to obtain its share of $\text{hot}(r)$, shifts that vector by the public y , and takes its inner product with g 's truth table. Correctness: the shifted vector is a sharing of $\text{hot}(y - r) = \text{hot}(x)$, so the inner product is a sharing of $g(x)$. The online cost is one round and k bits regardless of g ; the price is 2^k PRG calls per gate, which confines this to small k .

Comparison. For large k , full-domain evaluation is infeasible, so we must restrict attention to gates g with structure. A basic example is a gate that performs comparison with a public constant. Let $c \in \mathbb{G}_{\text{in}} = \mathbb{Z}_{2^k}$ be a constant with $c \neq 0$ and let $g(x) := x < c$, with x read as an integer in $[0, 2^k)$. Note that this comparison can be phrased as an interval test, i.e. g tests if x lies in the interval $[0, c)$.

Correspondingly, the offset function g'_r needs to test if the padded value $y = x + r$ lies in the interval $[r, r + c)$, written modulo 2^k . Define $p := r$ and $q := r + c \bmod 2^k$, so that the interval is $[p, q)$. The one subtlety is that this interval may wrap around 2^k . The claim is that g'_r is expressed as follows:

$$g'_r(y) = \text{less}_{q,1}(y) - \text{less}_{p,1}(y) + (p > q)$$

Correctness can be shown by case analysis on whether or not values p and q wrap around modulo 2^k .

Note that each term on the right-hand side is something the parties can obtain shares of. The dealer, who knows r and hence p, q , and the bit $p > q$, provides DCF keys for $\text{less}_{p,1}$ and $\text{less}_{q,1}$, together with additive shares of $p > q$. In the online phase, the parties evaluate both keys at y and add the resulting shares. The cost is one round and k bits of communication, with $O(k)$ PRG evaluations per party, for any k . Compare this to GMW, where a k -bit comparison costs $\Theta(k)$ AND gates over $\Theta(\log k)$ rounds.

Note that comparison with a public constant can be used to implement comparison over two secret values. In particular, to compare two secret-shared values x_1, x_2 , we can apply the above gate to $x_1 - x_2$. If inputs are confined to $[0, 2^{k-2})$ so that the difference does not wrap, then $x_1 < x_2$ iff $x_1 - x_2 \bmod 2^k \geq 2^{k-1}$, which is a comparison with the public constant 2^{k-1} .

Instantiating the Dealer. As with Beaver triples, the ultimate goal is to replace the FSS dealer with a cryptographic protocol. For FSS where the function has small input domain (i.e., poly-size domains), there is a communication-efficient protocol: two parties holding $\llbracket \alpha \rrbracket$ generate DPF keys with $O(k)$ OTs and k rounds; see [Ds17]. Unfortunately, for FSS schemes with larger domains, we do not have efficient instantiations of the dealer, and known techniques involve non-black-box evaluation of cryptographic primitives.

References

- [BBC⁺21] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy*, pages 762–776. IEEE Computer Society Press, May 2021.
- [BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016.
- [BGI19] Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 341–371. Springer, Cham, December 2019.
- [Ds17] Jack Doerner and abhi shelat. Scaling ORAM for secure computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 523–535. ACM Press, October / November 2017.