Alice, Bob, and Charlie want to know who is the richest of them all. They could announce their net worth in public. That may arouse unwanted suspicion from scammers or the tax authorities. Alternatively, they could ask their trusted friend Jimmy to arbitrate. Alice, Bob, and Charlie each secretly tell Jimmy their net worth. Jimmy then announces the identity of the richest person.

Trusted entities like Jimmy perform many important functions in society. Auditors verify the compliance of companies with tax laws without revealing their internal secrets. Banks certify the solvency of their customers while ensuring that their money doesn't disappear. Election officials guarantee that all votes are counted yet voters' preferences remain secret.

What happens when the trusted party is a crook? Societies have financial and legal incentives that are meant to prevent large-scale breaches of trust. As we all know well these incentives are far from perfect. Corruption of persons and institutions that are meant to be trusted happens everywhere.

In the 1980s cryptographers came up with a novel perspective on trust. Their premise was that processes like audits, bank transactions, and elections are *computations* of a special kind. They involve multiple entities with competing interests. In a bank transaction the computation entails consistently updating the customer's and the bank's ledger. Consistency is important for both the customer and the bank to not have their money stolen. In an audit, the output of the computation might be the amount of tax that the company owes the government. In addition to ensuring that taxation was fair, the company may want to avoid revealing extraneous information like its suppliers or customers. In an election, what is being computed is the vote tally. Voters want their votes to be counted but their preferences to remain anonymous.

A *secure multiparty computation* is a jointly executed algorithm whose outcome is "indistinguishable" from what it would be had the computation been performed by a trusted party. To understand what this means let's go back to Alice, Bob, and Charlie. Their respective inputs are their net fortunes x, y, and z. The output $f(x, y, z)$ should identify the richest person, namely

$$f(x,y,z) = \begin{cases} \texttt{Alice}, & \text{if } x > y, z, \\ \texttt{Bob}, & \text{if } y > z, x, \\ \texttt{Charlie}, & \text{if } z > x, y. \end{cases}$$

This is the argmax function. Trusted Jimmy would collect Alice's, Bob's, and Charlie's input and announce the winner without revealing any extraneous information as in Figure 1 (a).

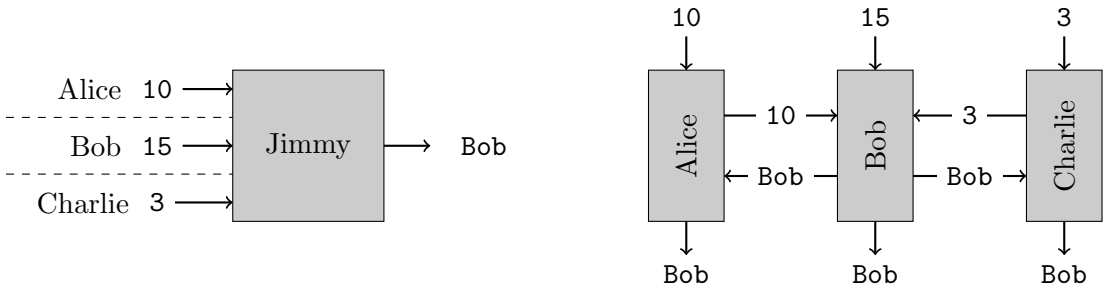


FIGURE 1: Joint computation of f = argmax. (a) Computation by a trusted party: Alice, Bob, and Charlie find out nothing beyond the value $f(x, y, z)$. (b) In a joint computation, they should come up with the same output $f(x, y, z)$. This implementation is insecure because Bob finds out everyone's net worth.

In a multiparty computation, Jimmy is replaced by algorithms through which Alice, Bob, and Charlie exchange messages. These algorithms must be *functional*: The output produced by Alice, Bob, and Charlie

at the end of the interaction must be the same as that produced by Jimmy in the trusted party setting. Functionality by itself is not difficult to satisfy. Bob, for example, could take up Jimmy's trusted party role, perform the computation himself, and forward the output to Alice and Charlie, as in Figure 1 (b).

This algorithm is not "secure". Bob learns not only who is the richest among the three but also finds out Alice's and Charlie's net worth. A secure multiparty computation eliminates this unintended disclosure of information. Alice's, Bob's, and Charlie's algorithms will ensure that Bob learns nothing about Alice's and Charlie's input *except* what is implied by the value of his output, namely that his input y is larger than both of theirs x and z. The same will be true for Alice and Charlie.

There are several algorithms for secure multiparty computation. The one we will describe was discovered independently by Ben-Or, Goldwasser, and Wigderson, and Chaum, Crépeau, and Damgård in 1988. In this algorithm Alice, Bob, and Charlie "share" their inputs and then cooperatively evaluate f step by step on their shares. Let's start with the sharing.

1 Secret sharing

A secret sharing scheme is a mechanism by which a dealer distributes a secret piece of information among several parties so that the secret can be reconstructed, but subsets of parties learn no information about the secret.

Before we define the concept let us give an example. Suppose the secret information consists of a single bit $s \in \{0, 1\}$ that the dealer Dave wants to split among the parties Alice, Bob, and Charlie. Dave samples *shares* x_1, x_2, x_3 for the three parties as follows:

$$\begin{aligned} &\text{if } s = 0, \quad \text{sample } x_1x_2x_3 \text{ uniformly from } \{000, 011, 101, 110\} \\ &\text{if } s = 1, \quad \text{sample } x_1x_2x_3 \text{ uniformly from } \{111, 100, 010, 001\}. \end{aligned}$$

After sampling Dave communicates the shares x_1, x_2, x_3 to Alice, Bob, and Charlie through secret channels. If the three of them get together they can reconstruct the secret uniquely from the shares: s is the modulo 2 sum of x_1, x_2, and x_3. On the other hand, Alice learns nothing about the secret: Her share x_1 is a random bit (0 with probability half and 1 with probability half) that *does not depend* on the value of s. By observing x_1 in isolation, Alice finds out nothing about the secret. The same is true for Bob and Charlie.

More is true: If Alice and Bob get together, they also find out nothing about the secret. The pair of bits x_1x_2 that they observe jointly is equally likely to match each of the four outcomes $\{00, 01, 10, 11\}$, regardless of the value of the secret. The same is true for any two out of the three parties. This example illustrates an important principle: Randomness is essential for hiding information.

A complete *secret sharing scheme* is a pair of algorithms: A sharing algorithm *Share* and a reconstruction algorithm *Reconstruct* where

- *Share* that takes a secret s as input and outputs randomized shares $x_1, x_2, \ldots, x_3$, one per party
- *Reconstruct* that takes shares $x_1, x_2, \ldots, x_3$ as inputs and outputs a secret.

The secret sharing scheme is *functional* if reconstruction always works: $\mathit{Reconstruct}(\mathit{Share}(s)) = s$. The scheme we just described is clearly functional. The reconstruction algorithm is $x_1 + x_2 + x_3 \bmod 2$, and this always equals the secret s.

Functionality by itself can be achieved in many ways. Dave could simply reveal s to Alice, Bob, and Charlie. Then there is no secrecy. To specify secrecy we need another definition.

Definition 1. A complete secret sharing scheme is *secure* if for every proper subset S of parties and for every two secrets s, s' the random variable sequences $(x_i : i \in S)$ and $(x'_i : i \in S)$ are identically distributed, where $(x_1, x_2, \ldots, x_3)$ and $(x'_1, x'_2, \ldots, x'_3)$ are the outputs of $\mathit{Share}(s)$ and $\mathit{Share}(s')$, respectively.

The notation $(x_i : i \in S)$ stands for the sequence of random variables indexed by elements of S under some fixed ordering of S. For example, $(x_i : i \in \{2, 3, 5\}) = x_2x_3x_5$.

In our bit sharing example there are only two possible secrets: the bits $s = 0$ and $s' = 1$. Let's call the corresponding shares $x_1x_2x_3$ and $x'_1x'_2x'_3$, respectively. x_1 and x'_1 are identically distributed because they are both random bits. So are x_2 and x'_2, as well as x_3 and x'_3. Moreover, x_1x_2 is identically distributed to $x'_1x'_2$: They are both pairs of random bits, equally likely to take the four values 00, 01, 10, and 11. So are x_1x_3 and $x'_1x'_3$, as well as x_2x_3 and $x'_2x'_3$. This covers all proper subsets of parties (except the empty set for which the claim is vacuous), so our scheme is secure.

Security implies that no proper subset of parties can reconstruct the secret. Since their joint shares are identically distributed, any "reconstruction" algorithm that Alice and Bob apply on their joint shares produces the same *distribution* of outcomes when the secret is 0 as when the secret is 1. In particular, their algorithm cannot reliably output 0 in the former case 1 in the latter case. The outcomes 0 and 1 must have the same probability in both cases. Not only can Alice and Bob not reconstruct the secret completely; they cannot even guess its value with probability exceeding 50% (in the absence of prior information).

This construction generalizes to n parties. The sharing algorithm $Share(s)$ outputs random bits $x_1, \ldots, x_n$ conditioned on $x_1 + \cdots + x_n = s \pmod 2$. Share x_i is given to party i. The reconstruction algorithm adds all the shares modulo 2. This scheme is both functional and secure. Now that we have a scheme for single-bit secrets for any number of parties, we can obtain one for many-bit secrets by applying it independently to every bit of information. It can be used to secretly share arbitrary secrets.

Threshold secret sharing

A client can use this scheme to store a k-bit file on a collection of $n \geq 2$ servers with the following guarantee. No proper subset of the servers obtains any information about the file, but the client can reconstruct the file after communicating with all the servers. What if some of the servers break down? It is desirable that the client can still reconstruct her file from the surviving ones. A solution that comes to mind is a variant of secret sharing in which not all n parties are required participate in the reconstruction, but some lower threshold r is sufficient.

To account for this generalization, we need to change the syntax and the relevant requirements. The reconstruction algorithm now takes two inputs: A subset R consisting of at least r parties, and their respective shares $(x_i : i \in R)$. Functionality should hold for all such subsets R, not merely for the complete set of parties. The functionality requirement says that for every set of r parties with shares $x_i : i \in R$ can determine the secret.

As any r parties can now reconstruct the secret, we can only hope for secrecy against smaller sets.

Definition 2. A r-out-of-n secret sharing scheme is *secure* if it satisfies Definition 1 but only when applied to subsets S of size less than r.

Now that we have the definitions in place we can describe a solution that meets all requirements.

2 Shamir's secret sharing

In 1979 Shamir came up with a secret sharing scheme for any number of parties n and any value of the threshold r (in the range 1 to n).

In Shamir's scheme, both the secret and the shares are not bits but elements of a finite field. A finite field is a finite number structure that admits addition, subtraction, multiplication, and division (except by zero). One important example is arithmetic modulo q when q is a prime number. More generally, a finite field of size q exists (and is unique) whenever q is the power of a prime number. Arithmetic in these prime power fields is somewhat more complicated. In our discussion we'll assume that the "alphabet size" q is a prime number and all arithmetic is modulo q.

As our first example let's take $r = 2$ and $n = 4$. We will identify the four parties with the numbers 1, 2, 3 and 4 modulo $q = 5$. A *line* ℓ is a function like $\ell(x) = 2 + 3x$ modulo q.

To share the secret s, Dave picks a random line ℓ passing through s at 0, i.e. conditioned on $\ell(0) = s$, uniformly at random among all such lines. Party i is then given the share $y_i = \ell(i)$. As $\ell(0) = s$ the line must have the form $\ell(x) = s + ax$. There are five such lines: s, $s + x$, $s + 2x$, $s + 3x$, and $s + 4x$. If $s = 2$ and Dave had picked the line $2 + 3x$ among these five, then the share values would be $Y_1 = 0$, $y_2 = 3$, $Y_3 = 1$, and $y_4 = 4$ (see Figure 2 (a)).

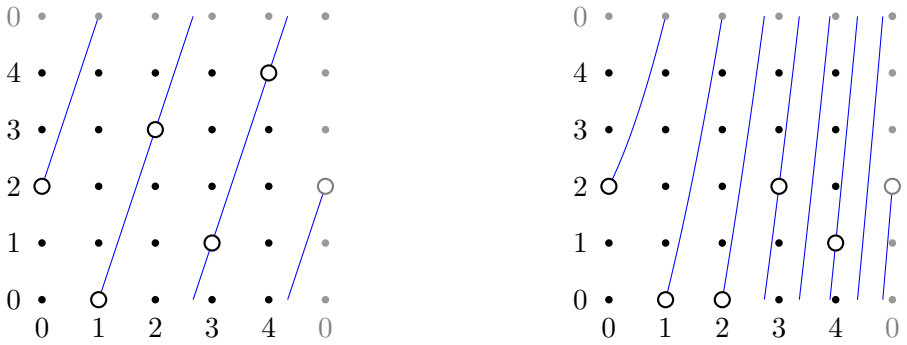


FIGURE 2: Shamir secret sharing for $n = 4$ parties modulo 5. Example shares with secret $s = 2$ and threshold (a) $r = 2$ (b) $r = 3$.

To reconstruct the secret, any two parties i and j can uniquely interpolate a line ℓ though their shares y_i and y_j. They can find this line by solving the equations $\ell(i) = y_i$ and $\ell(j) = y_j$. For examples, if parties 2 and 4 have shares $y_2 = 3$ and $y_4 = 4$ as in Figure 2 (a), Dave's line $\ell(x) = s + ax$ must satisfy

$$\begin{aligned} s + a \cdot 2 &\equiv 3 \pmod 5 \quad \text{and} \\ s + a \cdot 4 &\equiv 4 \pmod 5. \end{aligned}$$

This is a system in two equations and two unknowns modulo 5. It solves uniquely to $a = 3$ and $s = 2$, so the line must be $\ell(x) = 2 + 3x$ and the secret must be $s = 2$.

The share y_i of any individual party i, on the other hand, is the evaluation at i of the random line ℓ passing through s at 0: $y_i = s + a \cdot i$. In a finite field, both multiplication and addition by a fixed number *permute* the field elements. As a is equally likely to equal any of the five, so are $a \cdot i$ and $s + a \cdot i$, regardless of the value of the secret s. Therefore y_i is identically distributed—in fact, uniformly random—for every secret. The scheme is secure.

What happens if the reconstruction threshold is $r = 3$? Now instead of a line, Dave samples a quadratic $p(x) = s + a_1 x + a_2 x^2$ with random a_1, a_2 (see Figure 2 (b)). Reconstruction by any three parties now entails solving a system of three equations in three unknowns. For example, if parties 1, 2 and 4 wanted to reconstruct from their shares in Figure 2 (b) they would be solving the system

$$\begin{aligned} s + a_1 + a_2 &= 0 \\ s + a_1 \cdot 2 + a_2 \cdot 2^2 &= 0 \\ s + a_1 \cdot 4 + a_2 \cdot 4^2 &= 1 \end{aligned}$$

obtaining the solution $a_1 = 2$, $a_2 = 1$, and $s = 2$. We will see shortly that the solution to this type of system is always unique. As for secrecy, it turns out that the pair of shares (y_i, y_j) given to every pair of parties $i \neq j$ is equally likely to take up any of the 5^2 possible pairs of values modulo 5, regardless of the secret. Thus pairs of shares are identically distributed and the scheme is secure.

Shamir Secret Sharing for n parties and threshold r.

Algorithm *Share*:
Input: A secret s modulo q, where $q > n$.
1 Sample a random polynomial p of degree at most $r-1$ modulo q conditioned on $p(0) = s$.
2 For every i from 1 to n:
3 Send share $y_i = p(i)$ to party i.

Algorithm *Reconstruct*:
Input: A subset R of size r and shares $y_i : i \in R$.
1 Find a polynomial $\tilde{p}$ of degree at most $r-1$ such that $\tilde{p}(i) = y_i$ for all $i \in R$.
2 Output $\tilde{p}(0)$.

In step 1 of *Share*, the polynomial p can be sampled by choosing independent random coefficients $a_1, \ldots, a_{r-1}$ modulo q and setting p to equal $p(x) = s + a_1 x + a_2 x^2 + \cdots + a_{r-1} x^{r-1}$. As there are q choices for each coefficient, there are q^{r-1} possible polynomials.

Theorem 3. *Shamir's r-out-of-n Secret Sharing Scheme is functional and secure assuming $q > n$.*

Functionality and security rely on a key fact about polynomials:

Fact 4. *For every set of r pairs (i, y_i), where the is are distinct, there is a unique polynomial p of degree at most $r-1$ so that $p(i) \equiv y_i$ for all pairs.*

Proof of Theorem 3. A polynomial $\tilde{p}$ of degree $r-1$ satisfying the reconstruction constraints $\tilde{p}(i) = y_i$ for all $i \in R$ must exist because p is such a polynomial. Fact 4 says that this $\tilde{p}$ is unique, so p and $\tilde{p}$ must be the same polynomial. In particular $p(0) = \tilde{p}(0) = s$, so reconstruction is correct.

For security, we will show that for every secret s, the shares $y_i : i \in S$ are equally likely to take any one of the q^{r-1} possible $(r-1)$-tuples of possible values. In particular, the shares resulting from any two secrets are identically distributed.

The parties in S observe the values $y_i = p(i)$ for all $i \in S$, where p is a random degree-$(r-1)$ polynomial subject to $p(0) = s$. By Fact 4, there is exactly one choice of p that satisfies the conditions $p(0) = s$ and $p(i) = y_i$ for all $i \in S$. As there are q^{r-1} choices for $(a_1, \ldots, a_{r-1})$, each possible tuple of values is taken with the same probability $1/q^{r-1}$. □

One implementation detail we did not specify is how to find the polynomial $\tilde{p}$ in step 1 of *Reconstruct*. This is a system of modular linear equations. It can be solved by Gaussian elimination. Recall that this entails about $O(rn^2)$ arithmetic operations. The reconstruction algorithm, however, doesn't really need to know a_1 up to a_{r-1}. It only cares for the constant coefficient $s = \tilde{p}(0) = p(0)$.

The Lagrange interpolation formula gives a more direct and faster way to find $p(0)$ from $p(i)$ for $i \in R$. It says that for every degree-$(r-1)$ polynomial p, and every set R of r inputs,

$$p(0) \equiv \sum_{i \in R} p(i) \prod_{j \in R \setminus \{i\}} \frac{j}{j-i}. \tag{1}$$

It can be used to implement reconstruction like this:

Algorithm *Reconstruct*:
Input: A subset R of size r and shares $y_i : i \in R$.
Output: $\sum_{i \in R} y_i \prod_{j \in R \setminus \{i\}} j/(j-i)$ modulo q.

For example, in Figure 2(b), parties 1, 2, and 4 would reconstruct to

$$0 \cdot \frac{2}{2-1} \cdot \frac{4}{4-1} + 0 \cdot \frac{1}{1-2} \cdot \frac{4}{4-2} + 1 \cdot \frac{1}{1-4} \cdot \frac{2}{2-4} \equiv 2^{-1} \cdot 2 \cdot 3^{-1} \equiv 2.$$

Proof of Fact 4. The following polynomial has degree at most $r-1$ and evaluates to y_i at i for every $i \in R$:

$$p(x) = \sum_{i \in R} y_i \prod_{j \in R \setminus \{i\}} \frac{x-j}{i-j}.$$

The product terms are set up so that only the i-th term survives (and evaluates to y_i) when $x = i$ and all the others vanish. The degree is at most $r-1$ because each product has $r-1$ factors. (Formula (1) is the special case when $x = 0$.)

It is unique because if there were two such polynomials then their difference would also have degree at most $r-1$ and would evaluate to zero on all of R. By Fact 5 this is impossible. □

Fact 5. *A nonzero degree-d polynomial over a field can have at most d zeros.*

Proof of Fact 5. If p has degree d and $p(b)$ equals zero then $p(x) = p(x) - p(b)$ has the form $a_1(x-b) + a_2(x^2-b^2) + \cdots + a_d(x^d - b^d)$. As $x-b$ factors into all the terms, p itself can be written as $(x-b)$ times a lower degree polynomial. If p has more than d zeros and we repeat this argument more than d times, we obtain the absurd conclusion that one of the factors of p has degree less than zero. □

3 Secure multiparty computation

In a secure multiparty computation, several parties want to perform a joint computation on their private inputs without revealing any information. Let's consider the case of three parties Alice, Bob, and Charlie. Unlike for secret sharing, there are several reasonable definitions of security for multiparty computation. We will focus on this one:

> After participating in the joint computation, no party (Alice, Bob, or Charlie) finds out any new information about the other two's inputs, beyond what is implied by its own output.

Suppose that in the joint computation of $f = \text{argmax}$, Bob submitted input 5 and it was announced that $f(x, 5, z) = \text{Charlie}$. From this Bob can deduce that $z > 5$ and $z > x$. The requirement is that he can deduce nothing else.

To make sense of this definition, it is helpful to compare two potential runs of the joint computation of $f = \text{argmax}$. In both runs we insist that Bob's input remains the same but Alice's and Charlie's may change. Suppose in the first run, the three parties' inputs are $(3, 5, 8)$. In the second run, the inputs are $(7, 5, 9)$. Both outputs are $f(3, 5, 8) = f(7, 5, 9) = \texttt{Charlie}$. In the absence of prior knowledge, after engaging in the computation, Bob should conclude that Alice's and Charlie's inputs are equally likely to have been $(3, 8)$, $(7, 9)$, or any two values (x, y) for which $f(x, 5, y) = \texttt{Charlie}$.

The phrase "Bob should conclude" has a precise mathematical meaning. The *view* of Bob is the collection of random variables consisting of his randomness and the sequence of all the messages he receives from Alice and Charlie.

Definition 6. A joint computation is secure against Bob if for every pair of executions in which Bob's input and output are the same (but Alice's and Charlie's inputs may be different), Bob's views are identically distributed. The joint computation is secure if it is secure against all of Alice, Bob, and Charlie.

The BGW algorithm assumes the availability of authenticated and secure communication infrastructure. Specifically, any two parties have a dedicated channel that is private (the other parties obtain no information except that communication occurred) and authenticated (they can be sure that they are talking to one another).

Let's first discuss the special but important case in which Bob wants to privately calculate the sum of Alice's input x and Charlie's input z. (Alice and Charlie calculate nothing.) To do this, Alice and Charlie first jointly sample a random number r modulo q. Alice sends Bob the value $x + r$, Charlie sends Bob $z - r$, and Bob outputs the sum of the two messages. Alice's and Charlie's views consist of a single random

number so the algorithm is secure against them. Bob observes two random numbers that add up to his output. The algorithm is secure because the joint distribution of these two numbers only depends on his output and not on Alice's and Charlie's input (see Figure 3).

Procedure *ThreeSum*
Private Inputs x for Alice and z for Charlie.
1 Alice chooses a random r modulo q and shares it with Charlie.
2 Alice sends $x' = x + r$ to Bob.
Charlie sends $z' = z - r$ to Bob.
3 Bob privately outputs $x' + z'$.

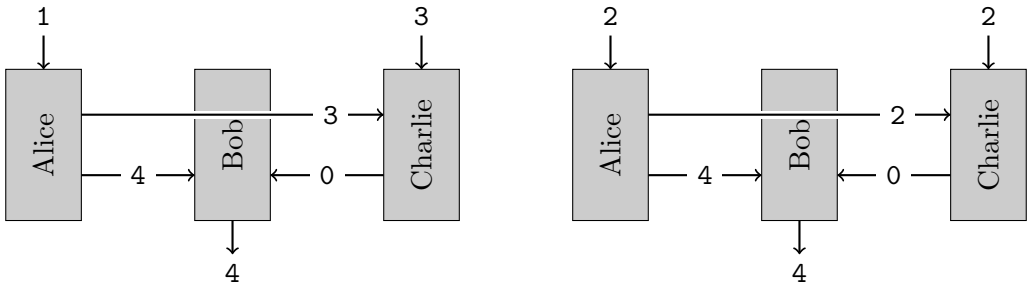


FIGURE 3: Security of *ThreeSum*. The modulus is $q = 5$. In the two executions, Bob's output is the same but Alice's and Charlie's inputs differ. The probability that Bob's view equals $(4, 0)$ is $1/5$ in both executions.

4 The BGW algorithm

The Ben-Or, Goldwasser, Wigderson (BGW) algorithm takes as its input a representation of $f(x, y, z)$ as an arithmetic circuit. This is a circuit with addition gates that take an arbitrary number of inputs and multiplication gates that take two inputs. It outputs instructions for a secure multiparty computation of f by Alice, Bob, and Charlie on their private inputs x, y, and z.

The arithmetic circuit operates over a finite field that is suitably large to allow Shamir Secret Sharing among Alice, Bob, and Charlie. The circuit can have any number of inputs, but these are partitioned among Alice (the x-inputs), Bob (the y-inputs), and Charlie (the z-inputs). Functions like argmax have reasonably small representations by arithmetic circuits.

The BGW algorithm consists of three phases. In the setup phase, secret shares of each party's inputs are distributed to the other parties. In the computation phase, parties compute shares representing the values at all the wires in the order of circuit evaluation. In the reconstruction phase, the shares representing the output wire(s) are combined to obtain the desired output(s).

The underlying secret sharing scheme is Shamir's with 3 parties and reconstruction threshold 2. We'll work modulo $q = 5$. Alice's, Bob's, and Charlie's shares are the values of the line $\ell(t) = \text{secret} + rt$ at $t = 1$, 2, and 3.

In the setup phase each party shares their input via this scheme. For example Alice sends the values $x + r$, $x + 2r$, and $x + 3r$ to herself, Bob, and Charlie, respectively, for each of her inputs x. In the reconstruction phase, the output(s) are reconstructed from their share(s). For example, Alice can uniquely determine her output from hers and Bob's shares of it.

It remains to describe the computation phase. Suppose Alice, Bob, and Charlie want to evaluate the shares representing the output of a plus gate $p(0) + q(0)$. Each party knows their share $p(t)$ of $p(0)$ and $q(t)$ of $q(0)$. Each declares $p(t) + q(t)$ to be their share of the output $p(0) + q(0)$. As p and q are linear

functions, so is $p + q$ and each party ends up with a valid share for the output. This step doesn't involve communication so it preserves security.

In the case of a times gate $p(0) \cdot q(0)$, each party multiplies its shares $p(t)$ and $q(t)$. If p and q are linear functions describing the parties shares, then $v = pq$ is a *quadratic* polynomial whose value at zero equals the gate output $p(0) \cdot q(0)$. The values $v(1)$, $v(2)$, and $v(3)$ still uniquely specify the desired output $v(0)$. Reconstruction is in principle possible if all three parties participate.

However, shares can be multiplied only once. Any subsequent multiplication would result in a cubic (or higher degree) representation of the gate value by its shares and reconstruction would no longer be possible. To overcome this issue, the BGW algorithm applies an interactive *degree reduction* procedure after every multiplication.

Degree reduction

The inputs to degree reduction are shares $v(1)$, $v(2)$, and $v(3)$ of the secret value $v(0)$ specified by a quadratic polynomial v. The outputs are 2-out-of-3 Shamir secret shares of $v(0)$, namely evaluations $\ell(1)$, $\ell(2)$, and $\ell(3)$ of a linear function ℓ that is random conditioned on $\ell(0) = v(0)$. Each party should learn nothing other than its new share $\ell(i)$. As each individual share is independent of the secret, no information is leaked.

The starting point of degree reduction is the relation between the secret $v(0)$ and the quadratic shares $v(1)$, $v(2)$, and $v(3)$ given by the Lagrange interpolation formula (1):

$$v(0) = \frac{2 \cdot 3}{(1-2)(1-3)} v(1) + \frac{1 \cdot 3}{(2-1)(2-3)} v(2) + \frac{1 \cdot 2}{(3-1)(3-2)} v(3) = 3v(1) + 2v(2) + v(3). \qquad (2)$$

Even though v is a quadratic polynomial, the secret is a linear function of the shares. (For example, the shares in Figure 2 (b) satisfy (2).) Alice, Bob, and Charlie want to re-share $v(0)$ amongst themselves so that their shares are now on a random line ℓ. How can they do that?

Let's consider Alice's perspective. Alice's output $\ell(1)$ should be a random value r modulo 5. This r, together with the secret $v(0)$, determine $\ell(2)$ and $\ell(3)$ as the shares on the line $\ell(0) = v(0)$, $\ell(1) = r$:

$$\ell(t) = (1-t)v(0) + tr.$$

But (2) expresses $v(0)$ in terms of $v(1)$, $v(2)$, and $v(3)$. After simplifying, we get an expression for $\ell(t)$ in terms of r, $v(1)$, $v(2)$, and $v(3)$:

$$\ell(t) = (3(1-t)v(1) + tr) + 2(1-t)v(2) + (1-t)v(3).$$

This function is linear. Bob's and Charlie's new shares $\ell(2)$ and $\ell(3)$ are linear combinations of r, $v(1)$, $v(2)$, and $v(3)$:

$$\begin{aligned} \ell(2) &= (2r + 2v(1)) + 3v(2) + 4v(3) \\ \ell(3) &= (3r + 4v(1)) + v(2) + 3v(3). \end{aligned} \qquad (3)$$

These can be communicated securely using the *ThreeSum* algorithm.

Procedure *ReduceDegree*
Private Inputs Shares $v(1)$ for Alice, $v(2)$ for Bob, $v(3)$ for Charlie satisfying (2).

1 Alice replaces her share $v(1)$ by a random r modulo 5.
2 Alice and Charlie communicate $2r + 2v(1)$ plus $4v(3)$ to Bob via *ThreeSum*.
Bob replaces his share $v(2)$ by $3v(2) + ThreeSum(2r + 2v(1), 4v(3))$.
3 Alice and Bob communicate $3r + 4v(1)$ plus $v(2)$ to Charlie via *ThreeSum*.
Charlie replaces his share $v(3)$ by $3v(3) + ThreeSum(3r + 4v(1), v(2))$.

Bob's and Charlie's new shares are linear functions of the old shares of all three parties. They can learn them (and nothing else) by running algorithm *ThreeSum*.

The algorithm

The BGW algorithm applies the circuit operations on the shares in forward order. Every multiplication is followed by a degree reduction. The degree-reduction step enforces the invariant "$g(1)$, $g(2)$, and $g(3)$ are 2-out-of-3 shares of the output of g" for every gate g in the circuit. This invariant underlies the functionality of BGW.

Algorithm *BGW* for three parties modulo 5.
Common Input: An arithmetic circuit f with inputs partitioned among Alice, Bob, and Charlie.
Private Inputs for Alice, Bob, and Charlie modulo 5.

Setup phase: For each input u,
1 The party owning u 2-out-of-3 shares it among Alice, Bob, and Charlie.

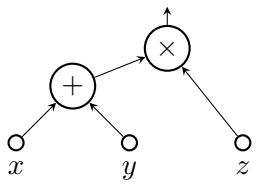
Computation phase: For each gate g in forward order of computation,
1 Each party creates a share for g by applying operation $[g]$ to the shares of its incoming wires.
2 If g is a times gate, apply *ReduceDegree* to $g(1)$, $g(2)$, and $g(3)$.

Reconstruction phase: For each output v,
1 The shares $v(1), v(2), v(3)$ are revealed to the party/parties that owns it.
2 The party reconstructs its value from the shares.

The algorithm extends to more than three parties. If there are n parties, Shamir's scheme with reconstruction threshold $\lceil n/2 \rceil$ should be used. This ensures gate outputs are determined by their shares even after multiplication. The n-party BGW algorithm is secure not only against individual participants, but against any joint coalitions of fewer than $n/2$ of them. For example if there are 7 parties, no 3 parties can deduce anything about the other 4 parties' inputs beyond what is implied by their respective outputs, even if these 3 parties collaborate.

Example

We illustrate BGW on the arithmetic circuit $f(x, y, z) = (x + y) \times z$ modulo 5:



The secret inputs are $x = 1$ for Alice, $y = 3$ for Bob, and $z = 4$ for Charlie.

Setup Alice chooses random slope 3 and shares her input x using the line $\ell_x(t) = 1 + 3t$. The resulting shares are

$$\ell_x(1) = 4, \qquad \ell_x(2) = 2, \qquad \ell_x(3) = 0.$$

Bob chooses random slope 0 and shares his input y using $\ell_y(t) = 3$. The resulting shares are

$$\ell_y(1) = 3, \qquad \ell_y(2) = 3, \qquad \ell_y(3) = 3.$$

Charlie chooses random slope 2 and shares his input z using $p_z(t) = 4 + 2t$. The resulting shares are

$$\ell_z(1) = 1, \qquad \ell_z(2) = 3, \qquad \ell_z(3) = 0.$$

Computation The first gate is an addition gate. Each party adds its x and y shares locally:

$$\ell_+(1) = \ell_x(1) + \ell_y(1) \equiv 2 \qquad \ell_+(2) = \ell_x(2) + \ell_y(2) \equiv 0 \qquad \ell_+(3) = \ell_x(3) + \ell_y(3) \equiv 3.$$

The wire carrying $x + y$ is represented by the shares $(\ell_+(1), \ell_+(2), \ell_+(3)) = (3, 0, 3)$.

The second gate is a multiplication gate. Each party multiplies its shares locally:

$$v_\times(1) = \ell_+(1) \cdot \ell_z(1) = 2 \cdot 1 \equiv 2 \qquad v_\times(2) = \ell_+(2) \cdot \ell_z(2) = 0 \cdot 3 \equiv 0 \qquad v_\times(3) = \ell_+(3) \cdot \ell_z(3) = 3 \cdot 0 \equiv 0.$$

These values are evaluations of a quadratic polynomial v satisfying $v_\times(0) = (x + y)z$.

To reduce the degree, Alice samples a fresh random value $r = 2$. Applying Equation (3) gives

$$\begin{aligned}
\ell_\times(1) &= r = 2, \\
\ell_\times(2) &= 2r + 2v_\times(1) + 3v_\times(2) + 4v_\times(3) = 2 \cdot 2 + 2 \cdot 2 \equiv 3, \\
\ell_\times(3) &= 3r + 4v_\times(1) + v_\times(2) + 3v_\times(3) = 3 \cdot 2 + 4 \cdot 2 \equiv 4.
\end{aligned}$$

After degree reduction, the output wire is represented by the linear sharing

$$\ell_\times(1) = 2, \qquad \ell_\times(2) = 3, \qquad \ell_\times(3) = 4.$$

Reconstruction To reconstruct, any two out of three parties reveal their shares, e.g., $\ell_\times(1) = 2$, $\ell_\times(2) = 3$. The output is determined by Lagrange interpolation:

$$2 \cdot \frac{2}{2-1} + 3 \cdot \frac{1}{1-2} \equiv 1$$

This is indeed the value $f(1, 3, 4) = (1 + 3) \cdot 4$ modulo 5.

5 Variants

One quality of the BGW algorithm is its universality. It can take an arbitrary circuit representing computation by a trusted party and compile it into a multiparty computation without one. Another feature is its perfect security. Under security Definition 6 there is zero leakage of unintended information.

The main drawback of BGW is its communication complexity. The communication between the parties is proportional to the circuit size of f. In common applications the inputs to f could be gigabyte-sized databases resulting in prohibitively expensive communication. A related drawback is its round complexity. Even if the algorithm is implemented in parallel, the number of interaction rounds would be proportional to the depth of the circuit.

Yet another weakness is that the BGW algorithm is meaningful only for a minimum of three parties. Many interesting functionalities involve only two parties, like a bank and a customer.

Other secure multiparty computation algorithms mitigate these shortcomings. Yao's garbled circuit algorithm terminates after three rounds of interaction, though its communication complexity is still proportional to the size of the circuit for f. It works for as few as two parties, and is secure against coalitions consisting of all but one of the participants.

Secure multiparty computation can also be realized using fully homomorphic encryption. The resulting algorithm is asymmetric. In the two-party setting, the amount of communication is proportional only to Alice's input size; Bob's plays no role. This can be advantageous in client-server settings where the client's input is typically much shorter than the server's. Fully homomorphic encryption is difficult to implement and still impractical at large scale.

One common weakness of these algorithms is that they are susceptible to "malicious" attacks. Security Definition 6 assumes that all parties participating in the joint computation follow the intended algorithm. In many situations, there is nothing that prevents a party from running its own algorithm that is designed to pry for unintended information disclosure or even derail the joint execution.

There is a variant of the BGW algorithm that is resilient even against such attacks, as long as fewer than a third of all participants are corrupted. That variant is obtained by augmenting the algorithm we described with "certification messages" through which each party proves that it is following the prescribed instructions. These certifications reveal no additional information about that party's inputs. Algorithms that prove the validity of a statement and reveal nothing else are called zero-knowledge proofs.