Secure Multi-Party Computation

Yanheng Wang

Table of contents

1 Preliminaries

1.1 Goals

Suppose n players $\{1,\dots,n\}$ live in a network where every pair can communicate securely. Each player i holds a private input $x_i \in \mathbb{F}$ at the outset. The players want to cooperate in a protocol so that they ultimately compute $y := \varphi(x_1,\dots,x_n)$ for a prescribed function φ: $\mathbb{F}^n \to \mathbb{F}$. The setting is called *multi-party computation*, or MPC for short.

Sounds trivial, isn't it? Yes, now comes the complication: some players in the game may be malicious. Our goal is to design protocols that resist different levels of misbehaviours:

- *Passive attack.* Some hidden spies $C \subset \{1,\dots,n\}$ follow the protocol exactly, but under the table they conspire to pool all information they have received. They collect their own inputs $\{x_i\}_{i\in C}$, the output y, plus intermediate results transmitted to them by the protocol. The first two types of leakage are uncontrollable anyway. Still we hope that the third type – the intermediate results – do not reveal extra information to what can be inferred from $\{x_i\}_{i\in C}$ and y. That is, our protocol does its best to safeguard privacy of the other players.
- *Active attack.* Some unknown players $C \subset \{1,\dots,n\}$ not only engage in passive attack but also actively deviate from the protocol. For example, when the protocol instructs "everyone send his value to others", a player in C may withhold it or send inconsistent/wrong values to different players, either consciously or unconsciously (say due to system failure). Our protocol should be robust enough to detect/counter the effect and strive for correctness and privacy.

Henceforth we call C the "cheaters" and $D := \{1,\dots,n\} \setminus C$ the "defenders". Such distinction is for analyses only; we don't know a priori which players are cheating. In the notes we will construct protocols that are passive/active secure against any $C \subset \{1,\dots,n\} : |C| \leqslant t$, where the constant t is the "cheating level" we could tolerate (typically $n/3$).

Until the final sections, we always assume a *synchronous* infrastructure. That is, all players share a clock and proceed in rounds, and all sent messages arrive instantly.

1.2 Polynomials

Polynomials are indispensable in constructing secure protocols. For our purpose we always assume a field $\mathbb{F} := \mathbb{Z}_p$ for some prime $p \gg n$. A function $f : \mathbb{F} \to \mathbb{F}$ is a *polynomial* if

$$f(x) = \sum_{i=0}^{d} \alpha_i x^i \quad \text{for some } \alpha_1,\dots,\alpha_d \in \mathbb{F} \text{ where } d < p.$$

Remark 1. Our "polynomial" is called "polynomial function" in standard literature, due to some subtlety that we will not encounter in the notes.

For the moment, it is unclear if a polynomial admits unique expansion. Would it be possible that we can expand a polynomial in multiple ways? Our discussion below excludes such possibility.

Lemma 2. Let f be a polynomial that can be expanded as $f(x) = \sum_{i=0}^{d} \alpha_i x^i$ with $\alpha_d \neq 0$. Then f has at most d roots. In particular $f \not\equiv 0$ because $d < p$.

Proof. By induction on d. For $d = 0$, clearly $f \equiv \alpha_0 \neq 0$ has no root. For $d \geqslant 1$, let $x^\star$ be a root if there is any. Using Euclidean division, we can find $f(x) = (x - x^\star) \cdot q(x) + r$ where $r \in \mathbb{F}$ and $q(x) = \sum_{i=0}^{d-1} \beta_i x^i$ with $\beta_{d-1} \neq 0$. Plugging in $x = x^\star$ we see $r = 0$, thus $f(x) = (x - x^\star) \cdot q(x)$. By induction hypothesis q has at most $d-1$ roots, so f has at most d roots. □

Corollary 3. Every polynomial $f \not\equiv 0$ has unique expansion $f(x) = \sum_{i=0}^{d} \alpha_i x^i$ with $\alpha_d \neq 0$.

Proof. Suppose $\sum_{i=0}^{d} \alpha_i x^i = f(x) = \sum_{i=0}^{e} \beta_i x^i$ where $d \leqslant e < p$. We agree that $\alpha_i := 0$ for all $d < i \leqslant e$ and consider

$$0 \equiv \sum_{i=0}^{e} (\beta_i - \alpha_i) x^i =: g(x).$$

By Lemma 2, this is impossible *unless* $\beta_i = \alpha_i$ for all i. □

With this result, we can now safely identify a polynomial $f \not\equiv 0$ with its unique expansion coefficients $(\alpha_1, \ldots, \alpha_d)$ where $\alpha_d \neq 0$. The integer d is called the *degree* of f and denoted as $\deg(f)$. As a convention, $\deg(0) := -\infty$.

This encoding allows us to sample a polynomial of degree $\leqslant d$ uniformly at random: It suffices to sample the coefficients $(\alpha_1, \ldots, \alpha_d) \in \mathbb{F}^d$ uniformly.

1.3 Lagrange interpolation

Definition 4. For a function $g: L \to \mathbb{F}$ where $L \subseteq \mathbb{F}$, we define its *Lagrange interpolation* as

$$\Lambda_g(x) := \sum_{\ell \in L} g(\ell) \cdot \frac{\Delta_{L-\ell}(x)}{\Delta_{L-\ell}(\ell)} \qquad \text{where} \qquad \Delta_{L-\ell}(x) := \prod_{\ell' \in L \setminus \{\ell\}} (x - \ell').$$

By definition $\Delta_{L-\ell}$ is a polynomial of degree $\leqslant |L| - 1$. Moreover,

$$\Delta_{L-\ell}(x) \begin{cases} = 0 & \text{if } x \in L \setminus \{\ell\}, \\ \neq 0 & \text{if } x = \ell. \end{cases}$$

Therefore Λ_g is a polynomial of degree $\leqslant |L| - 1$, and for all $x \in L$ we have $\Lambda_g(x) = g(x)$.

Lemma 5. Let f be any polynomial of degree d, and $L \subseteq \mathbb{F}$ be any set with $|L| \geqslant d + 1$. Then $f \equiv \Lambda_{f|L}$ where $f|L$ denotes the restriction of f on L.

Proof. Note that $\Lambda_{f|L}(x) = f(x)$ for all $x \in L$. That is,

$$(f - \Lambda_{f|L})(x) = 0, \quad \forall x \in L.$$

Recall that $\deg(f) = d$ and $\deg(\Lambda_{f|L}) \leqslant |L| - 1$. So the polynomial $f - \Lambda_{f|L}$ has degree at most $\max\{d, |L| - 1\} = |L| - 1$. But we have already found $|L|$ roots. So by Lemma 2, this polynomial must be constantly zero, hence $f \equiv \Lambda_{f|L}$. □

Lemma 5 gives us a another encoding of a degree-d polynomial: just take arbitrary $d + 1$ (or more) evaluations of the polynomial. The exercise below offers another persepective.

Exercise 1. Given evaluations of a degree-d polynomial f at $d + 1$ points, find a formula for the coefficients of f. (*Hint: use Vandemonde matrix.*)

We could already mention how Lagrange interpolation plays a role in secure protocols. A player can represent his secret by a polynomial f of degree t and distribute its evaluations to others. Each player holds only one evaluation, which is insufficient to recover f. On the other hand, if $\geqslant t + 1$ players meet together, they can apply Lemma 5 to assemble f.

2 Passive MPC

Now we embark on the journey to MPC protocols. This section is devoted to a short and elegant protocol that resists passive cheaters. Recall that passive cheaters can pool their information together, but will not actively deviate from the protocol.

2.1 Shamir sharing

The *Shamir sharing* realises the idea that we introduced in Section 1.3. When a player wants to share a secret s with others, he chooses a polynomial f such that $\deg(f) \leqslant t$ and $f(0) = s$. That is, he "hides" the secret in the 0-slot of the polynomial. Then he distributes the evaluation $f(i)$ to player i, for $i = 1, \dots, n$. Each player now holds a small piece or "share". The shares together determine the secret (in fact, $\geqslant t+1$ shares would already do), but knowing only $\leqslant t$ of them is insufficient.

For perfect security, f must be chosen at random. To be precise we denote $F_d(s) := \{f : \deg(f) \leqslant d, f(0) = s\}$. When a player shares s, he should draw a fresh f uniformly from $F_t(s)$. This can be implemented by generating its coefficients $(\alpha_0, \alpha_1, \dots, \alpha_t) \in \{s\} \times \mathbb{F}^t$ uniformly.

Protocol SHAMIR-DISTRIBUTE

goal: player k distributes shares of secret s to others

the distributer k	**each player i**
sample $f \in F_t(s)$ uniformly	
send $s_i := f(i)$ to each player i	receive my share

Protocol SHAMIR-ASSEMBLE

premise: every player holds a share of the same secret s
goal: player k collects the shares and assembles s

the assembler k	**each player i**
receive $s_1, \dots, s_n$	send my share to k
define function $g : i \mapsto s_i$	
$s := \Lambda_g(0)$	

It's important to understand that SHAMIR-DISTRIBUTE transforms the secret s into a vector $\boldsymbol{s} = (s_1, \dots, s_n)$ scattered over all players. Any individual i holds the i-th component s_i only. But all players communally hold the whole vector and thus the secret.

Lemma 6. Any $\leqslant t$ players together cannot gain any information about s from their shares. Formally, for all $C \subset \{1, \dots, n\}$ with $|C| \leqslant t$, the variables $\{s_i\}_{i \in C}$ and s are independent. In information theory language, their mutual information is zero.

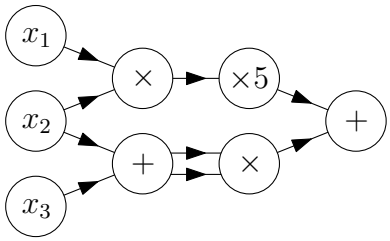
Proof. It suffices to prove that, for $|C| = t$, the variables $\{s_i\}_{i \in C}$ are uniform over $\mathbb{F}^t$ under whatever condition s. To this end, observe that $f \in F_t(s)$ one-one corresponds to the values $\{s_i\}_{i \in C} \in \mathbb{F}^t$. (The converse mapping is via Lagrange interpolation.) Since we sampled $f \in F_t(s)$ uniformly, the distribution of $\{s_i\}_{i \in C}$ must be uniform as well. □

A few more words to avoid misconception: When we say "someone SHAMIR-DISTRIBUTE something", the player *isn't* doing it alone. Instead, *all* players take part in the protocol, just with this particular player acting as the distributer.

2.2 Computation

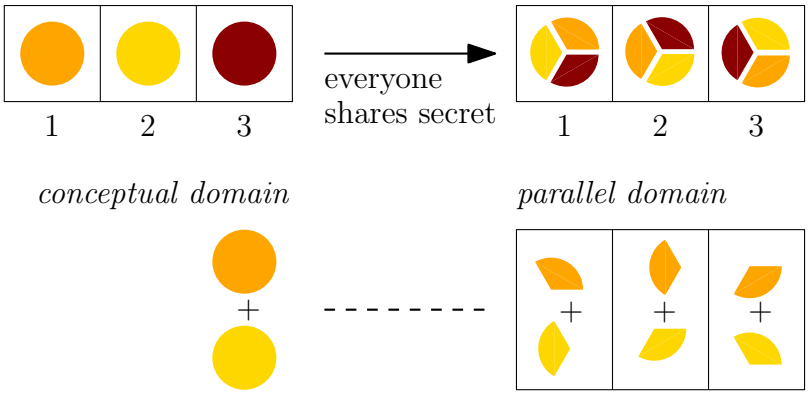
Recall that the players want to compute $y := \varphi(x_1, \ldots, x_n)$. For convenience, the function φ is given as an arithmetic circuit consisting of three types of gates: addition, scaling, and multiplication. The gates are evaluated in a topological order. We do not need division gates because $x^{-1} = x^{p-2}$ by Fermat's theorem and the latter can be computed via the well-known "square and multiply" method.

Example 7. The function $\varphi(x_1, x_2, x_3) := 5\,x_1x_2 + (x_2 + x_3)^2$ can be given as



Now we outline the idea of the protocol. Basically, all computations are sandwiched between SHAMIR-DISTRIBUTE and SHAMIR-ASSEMBLE:

- Every player SHAMIR-DISTRIBUTE his private input to others. After this phase, everyone grabs a share of each other's input.
- Now they forget about the inputs and just work on the shares. Following the same topological order of the circuit, every individual directly operates the shares at hand (though as if manipulating meaningless random numbers!). But as we will show, this corresponds implicitly to operating the actual inputs.
- When the players finish the entire circuit, they SHAMIR-ASSEMBLE the final output.



Let's fix some shorthands. $s \rightsquigarrow \boldsymbol{s}$ indicates that the owner of s SHAMIR-DISTRIBUTE it as shares $\boldsymbol{s}$. Conversely, $s \leftsquigarrow \boldsymbol{s}$ means that (say) player 1 SHAMIR-ASSEMBLE value s from shares $\boldsymbol{s}$, and then notifies everyone of the value s.

The vector representation also allows us to describe local computations as vector operations. For example, $\boldsymbol{a}+\boldsymbol{b}$ means that each player i locally adds the i-th components of $\boldsymbol{a}$ and $\boldsymbol{b}$.

Protocol PASSIVE-MPC

for $i=1\ldots n$ **do**
  $x_i \rightsquigarrow \boldsymbol{x}_i$
foreach gate γ **in** circuit φ
  case γ is an addition gate
    assume $\boldsymbol{a},\boldsymbol{b}$ the results from parents
    compute $\boldsymbol{a}+\boldsymbol{b}$ as the result of γ
  case γ is a scaling gate of factor λ
    assume $\boldsymbol{a}$ the result from parent
    compute $\lambda\,\boldsymbol{a}$ as the result of γ
  case γ is a multiplication gate
    assume $\boldsymbol{a},\boldsymbol{b}$ the results from parents
    compute $\boldsymbol{a}*\boldsymbol{b}=:\boldsymbol{c}=(c_1,\ldots,c_n)$
    for $i=1\ldots n$ **do**
      $c_i \rightsquigarrow \boldsymbol{c}_i$
    compute

$$\boldsymbol{d}:=\sum_{i=1}^{n}\frac{\Delta_{\{1,\ldots,n\}-i}(0)}{\Delta_{\{1,\ldots,n\}-i}(i)}\boldsymbol{c}_i$$

    as the result of γ
assume $\boldsymbol{y}$ the result from the final gate
$y \leftsquigarrow \boldsymbol{y}$

For the analysis, we identify a vector $\boldsymbol{a}=(a_1,\ldots,a_n)$ with the natural mapping $i\mapsto a_i$. Hence $\Lambda_{\boldsymbol{a}}$ means the Lagrange interpolation of $i\mapsto a_i$.

Lemma 8. For any vectors $\boldsymbol{a}=(a_1,\ldots,a_n)$ and $\boldsymbol{b}=(b_1,\ldots,b_n)$,

- $\Lambda_{\boldsymbol{a}+\boldsymbol{b}}=\Lambda_{\boldsymbol{a}}+\Lambda_{\boldsymbol{b}}$.
- $\Lambda_{\lambda\boldsymbol{a}}=\lambda\,\Lambda_{\boldsymbol{a}}$.
- $\Lambda_{\boldsymbol{a}*\boldsymbol{b}}=\Lambda_{\boldsymbol{a}}\cdot\Lambda_{\boldsymbol{b}}$ iff $\deg(\Lambda_{\boldsymbol{a}})+\deg(\Lambda_{\boldsymbol{b}})<n$.

Proof. Here we only show the first statement; the other two can be proved similarly. For all $k\in\{1,\ldots,n\}$ we have $\Lambda_{\boldsymbol{a}+\boldsymbol{b}}(k)=a_k+b_k=\Lambda_{\boldsymbol{a}}(k)+\Lambda_{\boldsymbol{b}}(k)$. Since both sides have degree $\leqslant n-1$ (recall the degree bound of Lagrange interpolation!) but we have found n agreements, the two polynomials must be equal by Lemma 2. □

Theorem 9. Assume $t:=\lfloor(n-1)/2\rfloor$. Let $\gamma\in\varphi$ be any gate. If the correct value of γ is v, and the protocol assigns to it a result $\boldsymbol{v}$, then $\Lambda_{\boldsymbol{v}}\in F_t(v)$.

Proof. By induction on the topological order of circuit φ. If γ is an input gate then the statement is trivially true. If γ is an arithmetic gate then we distinguish three types:

- *Addition gate.* Assume a,b are the correct values from the parents. By induction hypothesis we have $\Lambda_{\boldsymbol{a}}\in F_t(a)$ and $\Lambda_{\boldsymbol{b}}\in F_t(b)$. By Lemma 8, we have $\Lambda_{\boldsymbol{a}+\boldsymbol{b}}=\Lambda_{\boldsymbol{a}}+\Lambda_{\boldsymbol{b}}\in F_t(a+b)$.
- *Scaling gate.* Similar.

- *Multiplication gate.* Again sssume a, b are the correct values from the parents. By hypothesis, $\Lambda_{\boldsymbol{a}} \in F_t(a)$ and $\Lambda_{\boldsymbol{b}} \in F_t(b)$. By Lemma 8, we have $\Lambda_{\boldsymbol{c}} = \Lambda_{\boldsymbol{a}} \cdot \Lambda_{\boldsymbol{b}} \in F_{2t}(ab)$ because $\deg(\Lambda_{\boldsymbol{a}}) + \deg(\Lambda_{\boldsymbol{b}}) \leqslant 2t < n$. But the problem is that the degrees pile up. This is why the protocol does not take $\boldsymbol{c}$ as the result, for otherwise our induction cannot proceed. Instead it uses a trick to circumvent the issue. From $\Lambda_{\boldsymbol{c}} \in F_{2t}(ab)$ we see $\Lambda_{\boldsymbol{c}}(0) = ab$. But on the other hand,

$$\Lambda_{\boldsymbol{c}}(0) = \sum_{i=1}^{n} \frac{\Delta_{\{1,\dots,n\}-i}(0)}{\Delta_{\{1,\dots,n\}-i}(i)} c_i$$

  by definition of Lagrange interpolation. Therefore,

$$ab = \sum_{i=1}^{n} \frac{\Delta_{\{1,\dots,n\}-i}(0)}{\Delta_{\{1,\dots,n\}-i}(i)} c_i$$

  is a linear combination of the values $\{c_i\}_{i=1}^n$. Keep in mind that player i owns c_i. Security will break if he simply publishes this information. The trick is to wrap c_i by yet another Shamir sharing as the protocol does. Basically, we treat $c_1, \dots, c_n$ as private inputs and evaluate a linear combination of them. The problem then reduces to the addition/scaling cases, which we already handled. In the end we have

$$\Lambda_{\boldsymbol{d}} \in F_t\left(\sum_{i=1}^{n} \frac{\Delta_{\{1,\dots,n\}-i}(0)}{\Delta_{\{1,\dots,n\}-i}(i)} c_i \right) = F_t(ab). \qquad \square$$

Corollary 10. The protocol PASSIVE-MPC is correct.

2.3 Passive security

You should have an intuition that the protocol PASSIVE-MPC resists passive attack. After all, the players always use fresh Shamir sharings to distribute their secrets, which leaks zero information by Lemma 6. This sort of argument is rigorous enough for us, but we still want to provide a formalization that clarifies the meaning of security.

Definition 11. Let P be a protocol. Denote τ_i the set of variables player i can see during its execution. We say P is *passively secure* with respect to C if there is an algorithm that

- takes $\{x_i\}_{i \in C}$ and y as inputs; and
- generates a transcript that has the same distribution as $\{\tau_i\}_{i \in C}$.

We justify our definition as follows. Suppose the algorithm in the definition exists, then we can generate all intermediate results in $\{\tau_i\}_{i \in C}$ blindly from $\{x_i\}_{i \in C}$ and y, without even running the protocol. So anyone who tries to extract information from $\{\tau_i\}_{i \in C}$ is not advantageous to a person who knows only $\{x_i\}_{i \in C}$ and y. In other words, we cannot blame the protocol for leaking information.

Exercise 2. Prove that PASSIVE-MPC is passively secure with respect to any $C \subset \{1,...,n\}$ where $|C| \leqslant t := \lfloor (n-1)/2 \rfloor$.

Exercise 3. Consider an alternative subprotocol implementing the multiplication gates, and argue about its correctness and security:

Let $\boldsymbol{a}, \boldsymbol{b}$ be the results from parents and compute $\boldsymbol{c} := \boldsymbol{a} * \boldsymbol{b}$. Each player i samples a secret random value $r_i \in \mathbb{F}$ and distributes it twice: $r_i \rightsquigarrow \boldsymbol{r}_i$ and $r_i \rightsquigarrow \boldsymbol{r}_i^+$. The second distribution is a bit special by using a polynomial of degree up to $2t$ (rather than the usual t). Compute $\boldsymbol{r} := \sum_{i=1}^n \boldsymbol{r}_i$ and $\boldsymbol{r}^+ := \sum_{i=1}^n \boldsymbol{r}_i^+$. Compute $\boldsymbol{\delta} := \boldsymbol{c} - \boldsymbol{r}^+$ and immediately reconstruct $\delta \rightsquigarrow \boldsymbol{\delta}$. Finally, take $\boldsymbol{r} + (\delta, \dots, \delta)$ as the result.

3 Broadcast and Consensus

Before getting into actively secure MPC protocols we develop two key tools, *broadcast* and *consensus*, that help counter active cheating. Even with the presence of active cheaters, the broadcast protocol can deliver a consistent value from the broadcaster to the others, while the consensus protocol can reach agreement among defenders.

SPECIFICATION OF BROADCAST

$(\varnothing, \ldots, \varnothing, x_k, \varnothing, \ldots, \varnothing) \mapsto (y_1, \ldots, y_n)$

Consistent: $y_i = y_j$ for all $i, j \in D$.

Sound: if $k \in D$, then $y_i = x_k$ for all $i \in D$.

SPECIFICATION OF CONSENSUS

$(x_1, \ldots, x_n) \mapsto (y_1, \ldots, y_n)$

Consistent: $y_i = y_j$ for all $i, j \in D$.

Persistent: if $x_i = b$ for all $i \in D$, then $y_i = b$ for all $i \in D$.

Let's consider a super simple broadcast protocol: the broadcaster k just sends x_k to every player. Well, if $k \in D$ then everything goes smoothly. But if $k \in C$ then he can intentionally send different values to different players, leading to inconsistency. This should give you an idea why designing such protocols are non-trivial.

Lemma 12. One could use consensus to achieve broadcast, and vice versa.

Proof. Consensus ⇒ broadcast:

Protocol BROADCAST	
the broadcaster k	**each player $i \neq k$**
send x_k to all players	
	receive x_k from k
run CONSENSUS	run CONSENSUS

Broadcast ⇒ consensus is left as an exercise. □

3.1 Without setup

In this section we design a consensus protocol that allows up to $t := \lfloor (n-1)/3 \rfloor$ cheaters. For simplicity, we assume the inputs are $0/1$. After understanding the proof, you are invited to extend the protocol to arbitrary field.

The protocol consists of multiple, cascading stages; each stage strengthens the output from the previous one. Ultimately the output is strong enough and meets our specification.

Protocol CONSENSUS
for $k = 1 \ldots t+1$ **do**
run WEAK-CONSENSUS
run GRADED-CONSENSUS
run KING-CONSENSUS$_k$ with k being the king

- In WEAK-CONSENSUS stage, everybody exchange their preferences. A player adjusts his preference to $b \in \{0, 1\}$ if he sees it trending *substantially*; otherwise he abtains. The idea is to play safe: follow the trend whenever possible, but if the trend is unclear then step aside to avoid hindering the agreement.

- In Graded-Consensus stage, everybody exchange their (new) preferences. A player then adapts to the majority. In addition, he grades his decision as "strong" if the lead is substantial.
- In King-Consensus stage, a designated king gives an advisory value to others. A player ignores the advice if he already gained a "strong" belief in the previous stage; otherwise he takes the advice. In some sense, the king breaks ties and accelarates the formation of agreement.

The three stages are fleshed out below:

Protocol Weak-Consensus

each player i
send my x_i to all players
receive $x_1, \dots, x_n$ from all players
let c_0 count the 0's among them
let c_1 count the 1's among them
adjust

$$x_i := \begin{cases} 0 & c_0 \geqslant n-t \\ 1 & c_1 \geqslant n-t \\ \varnothing & \text{otherwise} \end{cases}$$

Protocol Graded-Consensus

each player i
send my x_i to all players
receive $x_1, \dots, x_n$ from all players
let d_0 count the 0's among them
let d_1 count the 1's among them
adjust $x_i := \mathbb{1}\{d_1 \geqslant d_0\}$
grade $\sigma_i := \mathbb{1}\{d_{x_i} \geqslant n-t\}$

Protocol King-Consensus

the king k	**each player $i \neq k$**
send my x_k to all players	receive x_k from the king
	if $\sigma_i = 0$ **then** adjust $x_i := x_k$

Lemma 13. The protocol Consensus is persistent.

Proof. Suppose $x_i = b$ for all $i \in D$. We analyse the protocol in stages.

- In Weak-Consensus, every player receives at least $|D| \geqslant n-t$ copies of b. So any defender $i \in D$ stays $x_i = b$.
- Then in Graded-Consensus, every player receives at least $n-t$ copies of b, and at most t copies of $\bar{b}$ (all sent by cheaters). Note that $n-t \geqslant t$ by our choice of t. Therefore, any defender i shall stay $x_i = b$ and grade himself $\sigma_i = \mathbb{1}\{d_b \geqslant n-t\} = 1$.
- Hence in King-Consensus, no defender shall listen to the word of the king. All of them insist on b.

- And the agreement b is preserved inductively throughout the loop. In the end, all defenders output b. □

Lemma 14.

1. When WEAK-CONSENSUS finishes, if some defender $i \in D$ holds value $x_i = b \in \{0,1\}$ then all defenders hold either b or $\varnothing$.
2. When GRADED-CONSENSUS finishes, if some defender $i \in D$ holds value $x_i = b$ and grade $\sigma_i = 1$ then all defenders hold value b.
3. When KING-CONSENSUS$_k$ finishes, if $k \in D$ holds $x_k = b$ then all defenders hold value b.
4. The protocol CONSENSUS is consistent.

Proof.

1. Suppose $i \in D$ holds $x_i = b \in \{0,1\}$, thus he had received $c_b \geqslant n - t$ copies of b. Among them at most t copies came from cheaters, thus at least $n - 2t$ copies came from defenders. So at least $n - 2t$ defenders held b prior to this stage. It implies that all players shall receive $\geqslant n - 2t$ copies of b, and hence $\leqslant 2t < n - t$ copies of $\bar{b}$. So no defender shall output $\bar{b}$.
2. Suppose $i \in D$ holds $x_i = b$ and $\sigma_i = 1$. From his perspective $d_b \geqslant n - t$. Again, at least $n - 2t$ copies of b that he has received are reliable, so at least $n - 2t$ defenders held the value b. We make two parallel observations:
   - Any defender is able to count $d_b \geqslant n - 2t$.
   - No defender held $\bar{b}$ due to point 1. So any received $\bar{b}$ must come from a cheater, thus $d_{\bar{b}} \leqslant t$.

   Therefore $d_b > d_{\bar{b}}$, so any defender ends up with value b.
3. Suppose $k \in D$ holds $x_k = b$. For any other defender $i \in D$ we consider two cases.
   - If his grade is $\sigma_i = 0$, then he will take the king's value b.
   - If his grade is $\sigma_i = 1$, then he will insist on his own value x_i. But from point 2 we know that all defenders – including the king – held the same value prior to the current stage, thus $x_i = x_k = b$.
4. Since $|C| \leqslant t$, at least one of the $t + 1$ rounds would designate a defender as the king. For that particular round we can achieve consistent output among defenders, by point 3. From then on the output preserves by Lemma 13. □

3.2 With setup

In this section we describe a signature-based broadcast protocol that allows up to $t := n - 1$ cheaters. The superior bound comes at price:

- We have to introduce cryptographic assumptions (e.g. dlog, RSA) underlying the digital signature scheme.
- Also, the network must provide some public key infrastrutures for key delivery.
- The cheaters have non-zero success probability of sabotaging the protocol by breaking the signature scheme. The probability is negligible though, if the cheaters are polynomially bounded.

The basic idea of the protocol is simple: All players forward whatever new value(s) they received from others, so that everyone can observe inconsistency if there is any. In that case, they output some default value, say 0.

But be careful! We could not afford running a protocol indefinitely, so a maximum number of rounds, r, must be imposed. Now consider the following situation. All players follow the protocol in the first $r-1$ rounds perfectly and all see a consistent value, say 35. Until in the final round a cheater sends a defender i some new value, say 22. Even though i sees inconsistency, he has no more chance to inform others. So i outputs the default value 0 while all other defenders output 35.

This "last round" problem is resolved by a smart construction due to Dolev and Strong. Suppose each player i has a secret/public key pair $(\mathrm{sk}_i, \mathrm{pk}_i)$, where the pubic keys are accessible by everybody. The high level idea is the following. Any value circulating in the network must be signed at each hop so that every player can confirm its history. A value will be discarded if its history is too short. So basically a cheater cannot suddenly come up with a new value in the last round.

Protocol Dolev-Strong

the broadcaster *k*
sign $\sigma := \mathrm{sgn}(\mathrm{sk}_k, x_k)$
send $(x_k, \{\sigma\})$ to all players

then, each player *i*
for $r = 1 \ldots t+1$ **do**
  receive (x, Σ)
  if x not yet accepted, and Σ contains valid signatures from k and from $r-1$ other distinct players **then**
    accept x
    sign $\sigma := \mathrm{sgn}(\mathrm{sk}_i, x)$
    send $(x, \Sigma \cup \{\sigma\})$ to all players
if have accepted exactly one x **then** output x
else output default value 0

Lemma 15. The protocol Dolev-Strong is consistent and sound.

Proof. (Consistency) Consider any message (x, Σ) accepted by any defender $i \in D$ in any round r.

- If $r \leqslant t$ is not the last round, then i is able to forward $(x, \Sigma \cup \{\sigma\})$ in the next round. By then every defender will accept the value x.
- If $r = t+1$ is the last round, then i cannot forward the value. Fortunately, since Σ in this case contains signatures from $r = t+1 > |C|$ distinct players[1], at least one of them, say j, is a defender. This j had accepted the value x in some *previous* round and had already informed others. Therefore every defenders have accepted the value x.

Hence, whenever some defender accepts x, all defenders accept it too. So the defenders output consistently.

(Soundness) Exercise. □

1. Here we used the unforgeability of secure digital signatures. We observe r distinct *signatures*, but we conclude that there are r distinct *signers*. This makes sense only if the signatures are unforgeable.

4 Active MPC

In the active attack setting, cheaters can deviate from our protocol arbitrarily and mess it up. As protocol designers, we must build mechanism to detect and correct possible deviations. We let the players run our old PASSIVE-MPC protocol, but on top of that we ask them to monitor each other and raise a complaint when they observe misbehaviour. If player i blames player j, then one of them must be a cheater. To figure out who, j proves his innocence by showing a conclusive evidence. If j fails to do so then we eliminate him from the game; otherwise we eliminate i. The game is then restarted with one player less.

How can a player tell misbehaviour without knowing others' internal states? And, why can a defender present a proof of innocence, while a cheater cannot? The high level idea is as follows. Everybody encrypt his initial state and broadcast the cypher. Due to some structural property of our encryption scheme, calculus on the cyphers corresponds to calculus on the states behind the scenes. Therefore, every players can do cypher calculus in the public domain, which essentially "tracks" the state evolution of the protocol. At some point a player must "open" his current state to another player for verifification. If it does not match the (publicly known) cypher, then something must be wrong.

The "encryption" here is more precisely called "commitment". We will next define its properties before we embed it into our protocol. Unless otherwise stated, we allow cheating level $t := \lfloor (n-1)/3 \rfloor$.

4.1 Commitment Function

A *commitment function* $H\colon (x,r) \mapsto \sigma$ locks value x into "commitment" σ, using a uniform random "key" r to protect x. We impose three semantic requirements:

- *Binding*: Given any (x,r), it is hard[2] to find $x' \neq x$ and r' such that $H(x',r') = H(x,r)$.
- *Hiding*: Given σ, it is hard to tell which value x gave rise to σ.
- *Homomorphic*: For all (x,r) and (x',r') we have $H(x,r) \cdot H(x',r') = H(x+x', r+r')$.

The binding property implies impossibility to alter preimage – once you have committed to x and broadcasted the commitment σ, you cannot lie when others request a proof. The hiding property means you can safely broadcast σ, as others cannot infer x from it. The homomorphic property says commitment calculus mirrors preimage calculus.

Remark 16. Sometimes we can impose even stronger properties. For instance, the "perfect binding" property requires that there is no $x' \neq x$ and r' with $H(x,r) = H(x',r')$. The "perfect hiding" property requires that $H(x,\cdot)$ is bijective for any x. In particular, provided r is uniform, $\sigma = H(x,r)$ is always uniformly distributed independent of the choice of x.

Below we collect two concrete commitment functions based on the discrete logarithm assumption.

Example 17. **(Pederson)** Let $(\mathbb{G}, +)$ be a group of order p with generators g and h. We define $H\colon \mathbb{Z}_p \times \mathbb{Z}_p \to \mathbb{G}$ by $H(x,r) := g^x h^r$. It is perfect hiding because h is a generator and thus $H(x,\cdot)$ is bijective. It is binding because any algorithm spotting a collision $g^x h^r = g^{x'} h^{r'}$ with $x \neq x'$ can be transformed into an attacker for discrete logarithm problem (how?). Finally, it is homomorphic for obvious reason.

Example 18. **(ElGammal)** This is essentially Pederson commitment tagged with the key. That is, we define $H\colon \mathbb{Z}_p \times \mathbb{Z}_p \to \mathbb{G}^2$ by $H(x,r) := (g^x h^r, g^r)$. It is hiding, perfectly binding, and homomorphic. The arguments are left as exercise.

2. As per cryptography convention, "hard" means that any polynomial time algorithm can only succeed with negligible probability.

Next we show the use case of a commitment function H. Notation:

- "$\sigma \asymp (x, r)$" indicates that "all players had locally stored σ, which *should* equal $H(x, r)$".
- The command "let `expr` $\asymp (x, r)$" instruct all players to locally evaluate expression `expr` and store the result σ, which *should* equal $H(x, r)$. Here `expr` shall involve public information only, so everyone can execute it without trouble. Upon finish of the command, we will have $\sigma \asymp (x, r)$.

Protocol LOCK

goal: all players learn the commitment by player k to his secret s

the owner k	**each player**
sample r uniformly	
broadcast $\sigma := H(x, r)$	$\blacktriangleright\sigma$
	let $\sigma \asymp (x, r)$

Protocol OPEN

premise: $\sigma \asymp (x, r)$
goal: player i transfers (x, r) to player j

the owner i	**the receiver j**
send (x, r) to j	receive (x, r)
	if $H(x, r) = \sigma$ **then** broadcast "accept"
	else broadcast "reject"

when hearing "reject",

the owner i	**each player**
broadcast (x, r)	$\blacktriangleright(x, r)$
	if $H(x, r) = \sigma$ **then**
	eliminate player j and reset
	else
	eliminate player i and reset

4.2 Committed sharing

We are now ready to adapt Shamir sharing to the active setting, by adding commitments and verifications to the main skeleton.

Protocol COMMITTED-DISTRIBUTE

premise: $\sigma_0 \asymp (s, \rho_0)$

the distributer k	**each player i**
sample $f \in F_t(s)$ uniformly	
expand $f(x) =: \sum_{j=0}^{t} \alpha_j x^j$	
for $j = 1 \ldots t$ **do**	**for** $j = 1 \ldots t$ **do**
lock α_j (with key ρ_j)	$\blacktriangleright\sigma_j \asymp (\alpha_j, \rho_j)$
	let $\prod_{j=0}^{t} \sigma_j^{i^j} \asymp (f(i), \sum_{j=0}^{t} \rho_j i^j)$
open $(f(i), \sum_{j=0}^{t} \rho_j i^j)$ to each i	$\blacktriangleright(s_i, r_i)$
	keep (s_i, r_i) as my share/key

The work of constructing polynomial f is partly handed to the public. The constructed f has degree $\leqslant t$ and is consistently determined by any $t+1$ shares. It is not guaranteed, though, that f is random.

Protocol COMMITTED-ASSEMBLE

premise: $\sigma_i \asymp (s_i, r_i)$ where $i = 1, \ldots, n$

the assembler k	**each player i**
receive $(s_1, r_1), \ldots, (s_n, r_n)$	send my share/key (s_i, r_i) to k
exclude indices i: $H(s_i, r_i) \neq \sigma_i$	
define function g: $i \mapsto s_i$	
$s := \Lambda_g(0)$	

4.3 State tracking

- $s \rightsquigarrow (\boldsymbol{s}, \boldsymbol{r})$ means that the players run COMMITTED-DISTRIBUTE. When they finish, the owner of s has distributed the shares $\boldsymbol{s}$ along with keys $\boldsymbol{r}$. As before $\boldsymbol{s}$ and r are vectors; player i knows i-th component only.
- $s \leftsquigarrow (\boldsymbol{s}, \boldsymbol{r})$ means that the players assemble s by running COMMITTED-ASSEMBLE where $k = 1, \ldots, n$ take turns as the assembler. Note that it is improper to assemble centrally and then broadcast, as the broadcaster might alter the value that he learns.
- "$\boldsymbol{\sigma} \asymp (\boldsymbol{s}, \boldsymbol{r})$" and "let $\texttt{expr} \asymp (\boldsymbol{s}, \boldsymbol{r})$" should be interpreted coordinatewise.

Protocol ACTIVE-MPC

for $i = 1 \ldots n$ **do**
  let γ be the input gate for player i
  player i commits to x_i
  $x_i \rightsquigarrow (\gamma.\boldsymbol{s}, \gamma.\boldsymbol{r})$
foreach gate γ **in** circuit φ
  let μ and ν be the parents of γ
  $\boldsymbol{\sigma} \asymp (\mu.\boldsymbol{s}, \mu.\boldsymbol{r})$
  $\boldsymbol{\pi} \asymp (\nu.\boldsymbol{s}, \nu.\boldsymbol{r})$
  case γ is an addition gate
    $\gamma.\boldsymbol{s} := \mu.\boldsymbol{s} + \nu.\boldsymbol{s}$
    $\gamma.\boldsymbol{r} := \mu.\boldsymbol{r} + \nu.\boldsymbol{r}$
    let $\boldsymbol{\sigma} * \boldsymbol{\pi} \asymp (\gamma.\boldsymbol{s}, \gamma.\boldsymbol{r})$
  case γ is a multiplication gate
    $\boldsymbol{c} = (c_1, \ldots, c_n) := \mu.\boldsymbol{s} * \nu.\boldsymbol{s}$
    for $i = 1 \ldots n$ **do**
      player i locks c_i
      player i proves $c_i = a_i \cdot b_i$
      $c_i \rightsquigarrow (\boldsymbol{c}_i, \boldsymbol{r}_i)$
      $\boldsymbol{\eta}_i \asymp (\boldsymbol{c}_i, \boldsymbol{r}_i)$
    $\gamma.\boldsymbol{s} := \sum_{i=1}^{n} w_i \boldsymbol{c}_i$
    $\gamma.\boldsymbol{r} := \sum_{i=1}^{n} w_i \boldsymbol{r}_i$
    let $\prod_{i=1}^{n} \boldsymbol{\eta}_i^{w_i} \asymp (\gamma.\boldsymbol{s}, \gamma.\boldsymbol{r})$
let γ be the final gate
$y \leftsquigarrow (\gamma.\boldsymbol{s}, \gamma.\boldsymbol{r})$

For brevity, in the protocol we omitted the scaling gates, and abbreviated

$$w_i := \frac{\Delta_{\{1,\dots,n\}-i}(0)}{\Delta_{\{1,\dots,n\}-i}(i)}.$$

In the case of addition gates, the players locally add the shares (like before) as well as the associated keys. At the same time, the public multiply the commitments accordingly to track the change by homomorphism.

In the case of multiplication gates, the idea is the same except for one more technicality: There is no apparent way to derive a commitment to c_i from what the public already knows (e.g. the commitments to a_i and b_i). Homomorphism just doesn't apply. Hence, as the protocol does, player i has to establish a *fresh* commitment to c_i that has no connection to prior public knowledge. How can we bridge the gap? Here a MULTIPLICATION-PROVE is invoked to convince other players that i did the multiplication properly. The protocol is based on polynomials too, but in a special way.

Protocol MULTIPLICATION-PROVE

premise: the prover k locked v, v', v'' and $\sigma_0 \asymp (v, \rho_0)$
goal: he convinces others that $v = v' \cdot v''$

$v' \rightsquigarrow (\boldsymbol{v}', \boldsymbol{r}')$, discard keys
$v'' \rightsquigarrow (\boldsymbol{v}'', \boldsymbol{r}'')$, discard keys

the prover k	**each player i**
$f := \Lambda_{\boldsymbol{v}'} \Lambda_{\boldsymbol{v}''}$	
expand $f(x) =: \sum_{j=0}^{2t} \alpha_j x^j$	
for $j = 1 \dots 2\,t$ **do**	**for** $j = 1 \dots 2\,t$ **do**
lock α_j (with key ρ_j)	$\blacktriangleright \sigma_j \asymp (\alpha_j, \rho_j)$
	let $\prod_{j=0}^{2t} \sigma_j^{i^j} \asymp (f(i), \sum_{j=0}^{2t} \rho_j i^j)$
open $(f(i), \sum_{j=0}^{2t} \rho_j i^j)$ to each i	$\blacktriangleright (f(i), r_i)$
	if $f(i) = v_i' \cdot v_i''$ **then** broadcast "match"
	else broadcast "mismatch"

when hearing "mismatch" from player i,

the prover k	**each player**
broadcast $(f(i), r)$	$\blacktriangleright (f(i), r)$
	if $H(f(i), r) = \prod_{j=0}^{2t} \sigma_j^{i^j}$ **then**
	eliminate player i and reset
	else
	eliminate player k and reset

discard everything generated in this protocol

We remind the readers that, in the first two steps we implicitly use the (already known) commitments of v' and v''.

Suppose the prover lies (i.e. locked v, v', v'' such that $v \neq v' \cdot v''$). Note that the public construct three kits of commitments, which respectively correspond to evaluation points on polynomials g, g', g''. Here $\deg(g'), \deg(g'') \leqslant t$ and $\deg(g) \leqslant 2\,t$. Besides $g(0) = v \neq v' \cdot v'' = g'(0) \cdot g''(0)$. Since the different polynomials g and $g' \cdot g''$ can agree on at most $2\,t$ evaluations, there are at least $n - 2\,t > t \geqslant |C|$ mismatches. So at least one defender shall notice a mismatch and broadcast it, and the prover will be eliminated.

4.4 Strengthening*

Due to limitation of multiplication proof, our protocol tolerates up to $t = \lfloor (n-1)/3 \rfloor$ cheaters. Using the theory of *zero knowledge proofs*, it is possible to design a multiplication proof protocol that allows up to $t = \lfloor (n-1)/2 \rfloor$ cheaters. We will not pursue that direction in the notes.

Another direction of strengthening calls for redesigning COMMITTED-DISTRIBUTE and COMMITTED-ASSEMBLE so as to get rid of commitment functions while preserving their functionality. The distribution is now achieved by first sending a bivariate polynomial and then projecting it to a univariate polynomial. The players exchange (limited amount of) information to check consistency of the bivariate polynomial, and raise a complaint if not. After resolving all complaints, we can guarantee that the projected univariate polynomial has degree $\leqslant t$ and shares the secret s. The redesign lifts the entire protocol to information-theoretical security, i.e. resisting $t = \lfloor (n-1)/3 \rfloor$ cheaters of *unbounded* resources.

5 Efficiency

Multiplication gates are the performance bottleneck of our protocols.

- In PASSIVE-MPC, each multiplication gate calls for a fresh SHAMIR-DISTRIBUTE, which communicates n^2 messages.
- In ACTIVE-MPC, each multiplication gate invokes a fresh COMMITTED-DISTRIBUTE and n MULTIPLICATION-PROVE. Each of these needs $\Theta(n)$ LOCK. Each LOCK requires one broadcast, which in turn incurs $\Theta(t \cdot n^2) = \Theta(n^3)$ messages. Therefore, a multiplication gate amounts to $\Theta(n^5)$ messages!

Can we do better? Surprisingly, the message complexity of a multiplication gate be cut down drastically to amortized $\Theta(n)$ in both passive and active protocols!

Let us begin with the passive setting. Recall the alternative multiplication scheme that we have seen in Exercise 3:

> Let $\boldsymbol{a}, \boldsymbol{b}$ be the results from parents and compute $\boldsymbol{c} := \boldsymbol{a} * \boldsymbol{b}$. Each player i samples a secret random value $r_i \in \mathbb{F}$ and distributes it twice: $r_i \rightsquigarrow \boldsymbol{r}_i$ and $r_i \rightsquigarrow \boldsymbol{r}_i^+$. The second distribution is a bit special by using a polynomial of degree up to $2t$ (rather than the usual t). Compute $\boldsymbol{r} := \sum_{i=1}^n \boldsymbol{r}_i$ and $\boldsymbol{r}^+ := \sum_{i=1}^n \boldsymbol{r}_i^+$. Compute $\boldsymbol{\delta} := \boldsymbol{c} - \boldsymbol{r}^+$ and immediately reconstruct $\delta \leftsquigarrow \boldsymbol{\delta}$. Finally, take $\boldsymbol{r} + (\delta, \dots, \delta)$ as the result.

Basically, each player contributes a random value r_i to the "secure randomness" $r := \sum_{i=1}^n r_i$ which no $\leqslant t$ players can ever infer. The price is that we need n randomness (thus n^2 messages) to piece together only one secure randomness!

In fact, summing $t+1$ individual randomness already provides a secure r. Still in the worst case $t+1 = \Theta(n)$, so the improvement is insubstantial. Well, if there is no way to compress the "input size", how about expanding the "output size"? That is, can we extract from $r_1, \dots, r_n$ *multiple* secure randomness and save the excess for future use? If we manage to extract $\Theta(n)$ many, then we win in the amortized sense.

5.1 Hyperinvertible map

Definition 19. Assume $\Psi: (y_1, \dots, y_n) \mapsto (y_{n+1}, \dots, y_{n+q})$ maps from $\mathbb{F}^n$ to $\mathbb{F}^q$. If for all $I \subset \{1, \dots, n+q\}$ of size $|I| = n$ and all assigenments to $\{y_i\}_{i \in I}$, there is a unique assignment to $\{y_i\}_{i \notin I}$ satisfying $\Psi(y_1, \dots, y_n) = (y_{n+1}, \dots, y_{n+q})$, then we call Ψ *hyperinvertible*.

In other words, Ψ is hyperinvertible if fixing in total n values (inputs and/or outputs) would uniquely determine the remaining values. Such property reminds us of polynomial evaluation and interpolation. For example, the map

$$\Psi(y_1, \dots, y_n) := (g(n+1), \dots, g(n+q)) \quad \text{where} \quad g := \Lambda_{(y_1, \dots, y_n)}$$

is hyperinvertible. To see this, notice that $\Psi(y_1, \dots, y_n) = (y_{n+1}, \dots, y_{n+q})$ if and only if all points $(1, y_1), \dots, (n+q, y_{n+q})$ lie on a polynomial of degree $\leqslant n-1$. (Prove it!) Hence, fixing any n values would uniquely determine the other values. Moreover the map is linear.

Lemma 20. Let $\Psi: \mathbb{F}^n \to \mathbb{F}^q$ be a hyperinvertible map where $n \geqslant q$. Then no matter how we fix $n - q$ many inputs, the restricted map is a bijection.

Proof. Denote $\Psi\colon (y_1,\dots,y_n)\mapsto(y_{n+1},\dots,y_{n+q})$. Fix any $I_0\subset\{1,\dots,n\}$ with $|I_0|=n-q$ and any condition imposed on inputs $\{y_i\}_{i\in I_0}$. They together with the outputs uniquely determine all other inputs, establishing an inverse mapping from $\{y_i\}_{i\in\{n+1,\dots,n+q\}}$ to $\{y_i\}_{i\in\{1,\dots,n\}\setminus I_0}$. Hence the restricted map is bijective. □

5.2 Faster passive MPC

Let $\Psi\colon\mathbb{F}^n\to\mathbb{F}^{n-t}$ be a linear hyperinvertible map. and apply it on individual randomness $r_1,\dots,r_n$ from the players. Among these at most t are cheating, but Lemma 20 ensures a bijection from the remaining (i.e. good) inputs and the outputs. Since the good inputs are uniformly distributed, the output must be uniform as well, regardless of misbehaviour of cheaters.

Protocol REFILL

for $i=1\dots n$ **do**
  player i samples r_i uniformly
  $r_i \rightsquigarrow \boldsymbol{r}_i$ using degree up to t
  $r_i \rightsquigarrow \boldsymbol{r}_i^+$ using degree up to $2t$
$(\boldsymbol{\rho}_1,\dots,\boldsymbol{\rho}_{n-t}) := \Psi(\boldsymbol{r}_1,\dots,\boldsymbol{r}_n)$
$(\boldsymbol{\rho}_1^+,\dots,\boldsymbol{\rho}_{n-t}^+) := \Psi(\boldsymbol{r}_1^+,\dots,\boldsymbol{r}_n^+)$
keep random pairs $(\boldsymbol{\rho}_1,\boldsymbol{\rho}_1^+),\dots,(\boldsymbol{\rho}_{n-t},\boldsymbol{\rho}_{n-t}^+)$ for future use

The line $(\boldsymbol{\rho}_1,\dots,\boldsymbol{\rho}_{n-t}) := \Psi(\boldsymbol{r}_1,\dots,\boldsymbol{r}_n)$, for example, means that each player i locally computes $(\rho_{1,i},\dots,\rho_{n-t,i}) := \Psi(r_{1,i},\dots,r_{n,i})$. Here *linearity* of Ψ is crucial, because otherwise we will lose the connection between local computation on the shares and the conceptual computation on the randomness.

Following the lines, a faster passive MPC protocol is immediate:

Protocol PASSIVE-MPC

…
foreach gate γ **in** circuit φ
  …
  case γ is a multiplication gate
    assume $\boldsymbol{a},\boldsymbol{b}$ the results from parents
    compute $\boldsymbol{a}*\boldsymbol{b} =: \boldsymbol{c} = (c_1,\dots,c_n)$
    if randomness depleted **then**
      REFILL
    take the next randomness pair $(\boldsymbol{\rho},\boldsymbol{\rho}^+)$
    $\boldsymbol{\delta} := \boldsymbol{c} - \boldsymbol{\rho}^+$
    $\delta \rightsquigarrow \boldsymbol{\delta}$
    take $\boldsymbol{\rho} + (\delta,\dots,\delta)$ as the result of γ
…

5.3 Faster active MPC

The protocol REFILL does not work in the active setting: a cheater $i\in C$ might distribute $\boldsymbol{r}_i$ and/or $\boldsymbol{r}_i^+$ via polynomials of wrong degree, or introduce inconsistencies between the two. We amend the protocol with a checking phase. Let $\Psi\colon\mathbb{F}^n\to\mathbb{F}^n$ be a linear hyperinvertible map.

Protocol RELIABLE-REFILL

for $i = 1 \ldots n$ **do**
  player i samples r_i uniformly
  $r_i \rightsquigarrow \boldsymbol{r}_i$ using degree up to t
  $r_i \rightsquigarrow \boldsymbol{r}_i^+$ using degree up to $2t$
$(\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_n) := \Psi(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n)$
$(\boldsymbol{\rho}_1^+, \ldots, \boldsymbol{\rho}_n^+) := \Psi(\boldsymbol{r}_1^+, \ldots, \boldsymbol{r}_n^+)$
for $i = 1 \ldots 2t$ **do**
  player i collects $\boldsymbol{\rho}_i$ from others and interpolates $f := \Lambda_{\boldsymbol{\rho}_i}$
  player i collects $\boldsymbol{\rho}_i^+$ from others and interpolates $g := \Lambda_{\boldsymbol{\rho}_i^+}$
  abort **if** player i claims $(\deg(f) > t) \vee (\deg(g) > 2t) \vee (f(0) \neq g(0))$
keep randomness $(\boldsymbol{\rho}_{2t+1}, \boldsymbol{\rho}_{2t+1}^+), \ldots, (\boldsymbol{\rho}_n, \boldsymbol{\rho}_n^+)$ for future use

Simply put, we "open" $2t$ randomness to different players for checking. Among them at least $2t - t = t$ players are defenders, who will check faithfully. If no one announces an abnormality, then we have at least t "good samples". Recall that we also have at least $n - t$ "good inputs". They together determine the rest of inputs/outputs linearly by assumption on Ψ. We must then conclude that *all* shares obey the correct degree constaint. The unopened $n - 2t$ randomness can thus serve as the desired secure randomness.

Next we explain secret distribution. The key idea is to mask the secret by a secure randomness and broadcast the result. All other players can then undo the mask locally.

Protocol RELIABLE-DISTRIBUTE

premise: the next available randomness pair is $(\boldsymbol{\rho}, \boldsymbol{\rho}^+)$
goal: k distributes the shares of secret s; they do lie on a degree-t polynomial

the distributer k	**each player i**
receive $\rho_1, \ldots, \rho_n$	send my randomness share ρ_i to k
find $f : \deg(f) \leqslant t$ with $\lvert\{i : f(i) = \rho_i\}\rvert > 2t$	
$\rho := f(0)$	
broadcast $\delta := s - \rho$	receive δ
	compute my share as $s_i := \rho_i + \delta$

In the first half (highlighted in yellow), k reconstructs the secure randomness ρ from shares $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_n)$. Of course, any cheater can send an incorrect share; but we still have $|D| \geqslant n - t > 2t$ unaltered shares – which lie on a degree t polynomial by guarantee of RELIABLE-REFILL. So the desired f always exists. On the other hand, if some f satisfies the condition, then $> 2t - t = t$ shares lying on it came from defenders, which enforces f to be *the unique* correct polynomial. In summary, the first phase can always figure out the right ρ.

In the second half, the distributer masks his secret by ρ and broadcasts it. This does not expose any secret since ρ is random. Each player then locally removes the mask and obtain a share for s. Note that the shares $\boldsymbol{s} = (s_1, \ldots, s_n)$ inherits the degree from the shares $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_n)$, which is by assumption correct.

Remark 21. The search for f can be implemented efficiently by Berlekamp-Welch decoder. One can regard it as a robust version of Lagrange interpolation in SHAMIR-ASSEMBLE that resists up to t outliers. This unveils curious connections to coding theory and, indeed, the trick was inspired by error correction codes.

The final missing piece, RELIABLE-ASSEMBLE, is in charge of reconstructing secrets from shares. What could be simpler? Just Berlekamp-Welch decodes the secret towards each player!

But there is space for improvement. The following protocol allows us to assemble multiple (up to $n-2t=\Theta(n)$) secrets in one batch for the sake of efficiency.

Protocol RELIABLE-ASSEMBLE

premise: $\ell+1$ secrets $s^{(0)},\dots,s^{(\ell)}$, each shared by degree-d polynomial
goal: all players learn all these secrets

for $i=1\dots n$ **do**
  $\boldsymbol{\sigma}^{(i)} := \sum_{j=0}^{\ell} \boldsymbol{s}^{(j)}\, i^j$
  player i collects $\sigma_1^{(i)},\dots,\sigma_n^{(i)}$ from others
  player i finds $f\colon \deg(f)\leqslant d$ with $\left|\left\{k\colon f(k)=\sigma_k^{(i)}\right\}\right| > d+t$
  if f does not exists **then** abort
  else player i sends $\sigma^{(i)} := f(0)$ to all players

then, each player locally
find $g\colon \deg(g)\leqslant \ell$ with $|\{i\colon g(i)=\sigma^{(i)}\}| > \ell+t$
if g does not exists **then** abort
else regard the coefficients of g as secrets $s^{(0)},\dots,s^{(\ell)}$

Here is what happens behind the scenes. Before protocol ever starts, we *mentally* embed the secrets in the coefficients of a polynomial

$$g(x) := \sum_{j=0}^{\ell} s^{(j)}\, x^j.$$

We also have in mind evaluations $\sigma^{(i)} := g(i)$ for all $i\in[n]$. Now the protocol starts, which somehow works backwards:

- we derive the shares $\boldsymbol{\sigma}^{(i)}$ from shares $\boldsymbol{s}^{(0)},\dots,\boldsymbol{s}^{(\ell)}$ via linear combination;
- then we can recover $\sigma^{(i)}$ from $\boldsymbol{\sigma}^{(i)}$, via the robust Berlekamp-Welch decoder;
- that in turn allows us to recover g from $\sigma^{(1)},\dots,\sigma^{(n)}$, again via Berlekamp-Welch;
- so finally we can recover the coefficients, i.e. secrets.

The analysis is identical to the first phase of RELIABLE-DISTRIBUTE – just repeat it twice. Provided $d,\ell<n-2t$, the protocol always succeeds and, in that case, assembles correctly. The message complexity is $\Theta(n^2/\ell)$, and the main saving comes from imposing polynomial structure on the otherwise independent secrets.

Exercise 4. Prove the correctness rigorously. Why is it necessary that each $\sigma^{(i)}$ is assembled towards different players?

Exercise 5. If the underlying field is large (in specific, having more than $n+\ell$ elements), then one can design an alternative RELIABLE-ASSEMBLE. This time, we embed the secrets not as coefficients of g, but the *evaluations* of g. We mentally define $g := \Lambda_{i\mapsto s^{(i)}}$ and the values $\sigma_i := g(\ell+i)$ for $i\in[n]$. Note that each σ_i is a linear combinations of the secrets. Please complete the description of this protocol and argue about its correctness.

Now we put every pieces together and yield an efficient active MPC protocol:

Protocol ACTIVE-MPC

for $i = 1 \ldots n$ **do**
  if randomness depleted **then**
    RELIABLE-REFILL
  player i RELIABLE-DISTRIBUTE his input

foreach gate γ **in** circuit φ
  assume $\boldsymbol{a}, \boldsymbol{b}$ the results from parents
  case γ is an addition gate
    take $\boldsymbol{a} + \boldsymbol{b}$ as the result of γ
  case γ is a multiplication gate
    compute $\boldsymbol{a} * \boldsymbol{b} =: \boldsymbol{c} = (c_1, \ldots, c_n)$
    if randomness depleted **then**
      RELIABLE-REFILL
    take the next randomness pair $(\boldsymbol{\rho}, \boldsymbol{\rho}^+)$
    $\boldsymbol{\delta} := \boldsymbol{c} - \boldsymbol{\rho}^+$
    RELIABLE-ASSEMBLE δ from $\boldsymbol{\delta}$
    take $\boldsymbol{\rho} + (\delta, \ldots, \delta)$ as the result of γ

RELIABLE-ASSEMBLE the output

There are two worthy comments:

- For simplicity we did not exploit the batch processing ability of RELIABLE-ASSEMBLE – what a shame! In real implementation, we should of course parallelize multiplications and do their corresponding δ-assembling in one batch.
- The protocol might abort in RELIABLE-REFILL and in RELIABLE-ASSEMBLE. The latter is caused by the possibly high degree (namely $2t$) of shares $\boldsymbol{\delta}$. Using a trick called *circuit randomization*, it is possible to drop the degree and reduce the abortion issue to RELIABLE-REFILL.

  Since RELIABLE-REFILL involves no secret inputs, we can do it before the MPC even starts and prepare sufficient random pairs for future. If it aborts, we ask the problematic player to publish his entire transcript, allowing a judge to pinpoint the cheater. Then we eliminate the cheater and start all over again.

6 Asynchronous Broadcast and Consensus

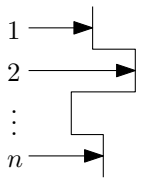
6.1 The asynchronous model

So far we have assumed the idealized synchronous model, a fair approximation to small local networks. But if the network spans across the globe, communications will take longer, which shakes our "instant delivery" assumption.

The *asynchronous model* takes communication delays into account. Imagine a *scheduler* in full charge of messages delivery. Any message sent by the players is handed to the scheduler. He can freely decide the arrival time of each message at hand (but of course he cannot deliver to the past or wait till the end of the world...). For the players it means: (i) each message can be delayed for an unbounded finite time; (ii) a message sent early may arrive late.

Let's investigate what happens if we run our protocols in this model. Recall that they have rigid time frames. A player *need to* receive required messages before the next round strikes; otherwise he gets stuck, lags behind, and will be considered malicious by the other players. One might patch the issue by broadcasting "Sorry, I'm stuck now, please wait me for another round." But then a cheater could take advantage of this mechanism and keep everyone waiting forever. It turns out that any easy patch will tear an immediate hole.

Asynchronous protocols come and fix just this. They abandon the concept of rounds but are purely driven by messages. The arrival of a message at a player will activate him to make some progress. Because messages may pop up asynchronously and irregularly, different players may have made different progresses so far. Of course, their progresses are not independent but instead interconnected (otherwise the protocol can't be correct). The hardest part of asynchronous protocol design is to enforce these interconnections.



6.2 Asynchronous broadcast

As a warm-up, we present an asynchronous broadcast protocol known as Bracha's protocol.

Protocol ASYNC-BROADCAST

rule	player	on event	do
(0)	k	started	send $(\texttt{input}, x)$ to all
(1)	any	got $(\texttt{input}, x)$ from broadcaster k	send $(\texttt{echo}, x)$ to all
(2)	any	got $(\texttt{echo}, x)$ from $n-t$ players for some x	send $(\texttt{ready}, x)$ to all
(3)	any	got $(\texttt{ready}, x)$ from $t+1$ players for some x	send $(\texttt{ready}, x)$ to all
(4)	any	got $(\texttt{ready}, x)$ from $n-t$ players for some x	output x and halt

Lemma 22. Assume $t := \lfloor (n-1)/3 \rfloor$. The protocol ASYNC-BROADCAST is

- **Consistent:** defenders either *all* get stuck, or *all* terminate and output the same value.
- **Sound:** if $k \in D$ then all defenders eventually terminate and output k's input.

Proof. We argue about soundness first. Assume $k \in D$ honestly sends $(\texttt{input}, x)$ to all players, which eventually triggers (1) of each defender. On the other hand, the cheaters could by no means trigger (1), so each defender shall echo only the correct x. That creates $n-t$ copies of $(\texttt{echo}, x)$, enough to trigger (2); the cheaters cannot trigger (2) with false values since $|C| \leqslant t < n-t$. This results in $n-t$ copies of $(\texttt{ready}, x)$, enough to trigger rule (4); again the cheaters can't trick defenders into wrong values.

Next we move on to consistency. Suppose $i \in D$ terminates with value x. We will show that all defenders terminate and output the same x. By (4), defender i has observed $n-t$ copies of $(\texttt{ready}, x)$, among which at least $n-2t \geqslant t+1$ copies came from defenders and will eventually and identically deliver to everyone, triggering (3). That would spawn the number of $(\texttt{ready}, x)$ messages, ensuring that every defender eventually sees $n-t$ copies, enough to trigger (4).

Are we finished? Not quite. We used the cunning word "eventually" to indicate that the asserted behaviour *will occur at some point if the players never terminate*. The worry is, if the "eventuality" arrives too late, a defender might have terminated (due to other causes) and will not execute the rules as usual! Hence we must also show that the conspiring cheaters and scheduler cannot trick the defenders to terminate before our "eventuality" actually happens.

Let's now explore *the past* so that no subtlety about the future shall tangle with our argument. As we noted, player i saw $n-t$ copies of $(\texttt{ready}, x)$. Well, the message must be *initiated* by someone from the first place. They cannot be all invented out of air by cheaters, otherwise rule (3) shall reject to spawn them. The only possible origin is that *some* defender $j \in D$ did invoke rule (2) and disseminated the message. Alright, we travel back and see that j got $n-t$ copies of $(\texttt{echo}, x)$. We collect the senders into a set X.

Now we assume, for the sake of contradiction, that some other $i' \in D$ outputs an inconsistent value $x' \neq x$. Applying the same argument, we conclude a set X' of $\geqslant n-t$ players that have sent $(\texttt{echo}, x')$. Note that $|X \cap X'| = |X| + |X'| - |X \cup X'| \geqslant 2(n-t) - n > t$, so the intersection must contain a defender. But how can he send both $(\texttt{echo}, x)$ and $(\texttt{echo}, x')$ to others? No way if you look at the rules! □

The consistency here is weaker than its synchronous counterpart; it could happen that none of the defenders terminate. But the flaw is inherent in asynchronous model. If the broadcaster k – a centralized role – is cheating and does not send a message at all, then everyone else will of course wait forever!

6.3 Asynchronous consensus: first version

In the decentralized setting of consensus, the story becomes different. It turns out that a "king" is not necessary and can be replaced by a communal effort, thus ensuring termination. We will first present a preliminary version that, in practical sense, achieves the desired properties. Later we will study a refined version that satisfies the properties perfectly and terminates with probability 1. Throughout we assume $t := \lfloor (n-1)/4 \rfloor$.

Protocol Async-Weak-Consensus

rule	on event	do
(0)	started	Async-Broadcast my value
(1)	collected $n-t$ values	output $\begin{cases} 0 & \#0 \geqslant n-2t \\ 1 & \#1 \geqslant n-2t \\ \varnothing & \text{otherwise} \end{cases}$

Lemma 23. The protocol ASYNC-WEAK-CONSENSUS terminates; moreover, it is

- **Consistent:** if some $i \in D$ outputs $b \in \{0,1\}$ then no defender outputs $\bar{b}$.
- **Persistent:** if all defenders input the same value b, then they all output b.

Proof. Termination follows obviously from soundness in Lemma 22. Now let us inspect persistency, assuming all defenders have input x. Any $i \in D$ would have gathered $n-t$ inputs when it invokes rule (1). At least $(n-t)-t = n-2t$ inputs came from defenders and read x; at most t inputs came from cheaters and read $\bar{x}$. Hence i shall output x.

We proceed to check consistency. Suppose some $i \in D$ outputs $b \in \{0,1\}$, then i has received $\geqslant n-2t$ copies of b from broadcast. So by consistency of broadcast (Lemma 22), there can be at most $2t$ players who had broadcast $\bar{b}$. Therefore, every defender $i' \in D$ shall observe at most $2t < n-2t$ copies of $\bar{b}$, and thus would not output $\bar{b}$. □

To boost the result to full consensus, we resort to the power of randomness. The preliminary idea cannot be simpler: Any player who ends up with value $\varnothing$ will flip a fair coin $c \in \{0,1\}$ as his new value. Off you go. With probability $\geqslant 2^{-n}$ all defenders happen to agree with each other. Hence, if we repeatedly alternate the WEAK-CONSENSUS and coin flip for 2^{100n} times, defenders arrive at consensus with very good chance!

Needless to say, this is intolerably slow. The problem is that different players are using *independent* coins. Can we couple their coins in hope of raising the agreement probability? In the synchronous setting there is a trivial solution: We may collect n coins from different players via broadcasts, then add them up (modulo 2) to synthesize a shared random coin. But in the asynchronous setting the solution gets problematic, as the broadcasts might not terminate. If the players only wait for some broadcasts to terminate, then different players might decide on different sets of coins to sum up.

Fortunately, the inconsistency can be controlled stochastically. We illustrate the gist in the protocol ASYNC-COIN-TOSS, assuming a large margin $t = \sqrt{n}$. We will show that with constant probability $\varepsilon > 0$, every defenders get the same coin value 1; with another chance of ε they all get 0; it could be a total mess with the remaining probability. Using more sophisticated constructions and analyses, the corruption bound can be lifted to $t = \lfloor (n-1)/4 \rfloor$ without harming the probability guarantees.

Protocol ASYNC-COIN-TOSS

rule	on event	do
(0)	started	broadcast a uniform random $r \in \{-1,1\}$
(1)	collected $n-t$ values	let Z be their sum and output $\mathbb{1}\{Z>0\}$

Suppose $i \in D$ has collected values $r_1, \ldots, r_{n-t}$. Without loss of generality, assume that $r_1, \ldots, r_{n-2t}$ came from defenders. By central limit theorem, the partial sum $Y := \sum_{j=1}^{n-2t} r_j$ roughly follows normal distribution with expectation 0 and variance $n-2t = \Theta(n)$. Hence the event $\{Y > 3\sqrt{n}\}$ happens with probability $\varepsilon > 0$ independent of n. This implies

$$\mathbb{P}(E) := \mathbb{P}(Z > 2\sqrt{n}) \geqslant \varepsilon.$$

Now take any $i' \in D$, who might have collected a different set of values and computed sum Z'. But the collections of i and i' share at least $n-2t$ common values. The remaining

values could bring about at most $2t$ difference, that is $|Z' - Z| \leqslant 2t = 2\sqrt{n}$. Therefore, conditioned on event E, we always have $Z' > 0$, thus i' outputs 1.

To summarise: with probability at least ε, *all* defenders output 1. By a symmetric argument, with probability at least ε, *all* defenders output 0.

Now we can present our asynchronous consensus protocol.

Protocol Async-Consensus

rule	on event	do
$(\top)$	started	enter iteration $h := 1$
$(h0)$	entered iteration h	Async-Weak-Consensus$_h$ Async-Coin-Toss$_h$
$(h1)$	Async-Weak-Consensus$_h \rightsquigarrow b$ and Async-Coin-Toss$_h \rightsquigarrow c$	update my value $x := \begin{cases} b & b \neq \varnothing \\ c & b = \varnothing \end{cases}$ enter iteration $h+1$
$(\bot)$	entered iteration 100	output my value and terminate

Theorem 24. The protocol Async-Consensus terminates and is

- **Consistent:** all defenders output the same value with probability at least $1 - (1-\varepsilon)^{100}$.
- **Persistent:** if all defenders input the same value x, then they all output x.

Exercise 6. Prove the lemma.

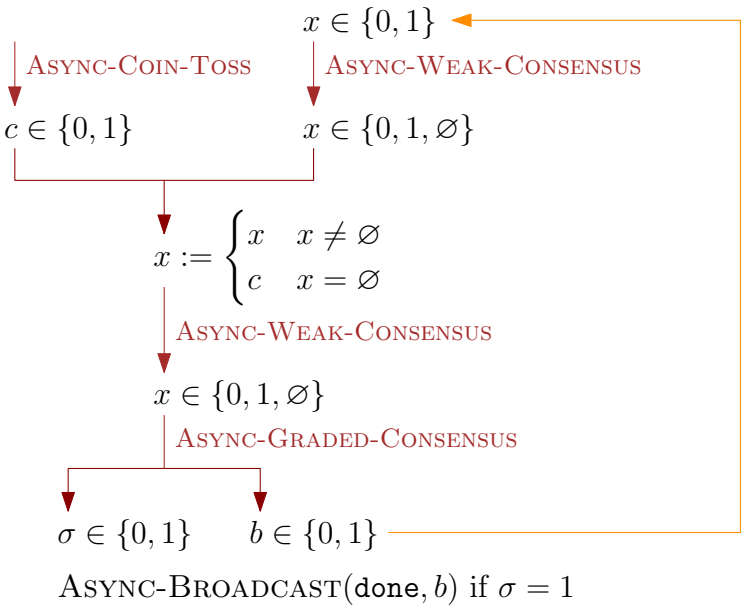
6.4 Asynchronous consensus: second version

In algorithmic terms, our first version of consensus protocol is a Monte Carlo procedure that always terminates but allows some margin of error. We can turn it into a Las Vegas procedure by "repeating until success". The main technical difficulty is the *detection of success*. With cheaters playing all around, how do we tell when consensus is attained? It comes to mind that the "grade" in Graded-Consensus can help us decide. Below is just a port of the protocol to the asynchronous setting.

Protocol Async-Graded-Consensus

rule	on event	do
(0)	started	Async-Broadcast my value
(1)	collected $n-t$ values	let d_0 count the 0's let d_1 count the 1's $b := \mathbb{1}\{d_1 \geqslant d_0\}$ $\sigma := \mathbb{1}\{d_b \geqslant n - 2t\}$ output (b, σ) and halt

Each iteration of Async-Consensus is then fairly straightforward, as shown by the diagram below. The same workflow is repeated over and over, with the new x in the end replacing that in the beginning. Whenever player i have collected $t+1$ copies of $(\texttt{done}, a)$ for some a, it outputs a and terminates.



Lemma 25. For ASYNC-GRADED-CONSENSUS embedded in the protocol, we have:

- If some defender outputs $(b,1)$ then all defenders output either $(b,0)$ or $(b,1)$.
- If all defenders input the same x then all defenders output $(x,1)$.

Proof.

- If some defender outputs $(b,1)$, then from his perspective $d_b \geqslant n-2t$. We make two parallel observations:
  - Due to consistency of ASYNC-BROADCAST, all defenders shall eventually observe $n-2t$ copies of b as well. Of course, they might have terminated earlier. But it's safe to say that they do count $\geqslant n-3t$ copies of b before termination.
  - Note that $d_b \geqslant n-2t > 2t > |C|$, so there is a copy of b came from some defender, say $i \in D$. By consistency of previous stage, ASYNC-WEAK-CONSENSUS, we know that *no* defender had input $\bar{b}$ to ASYNC-GRADED-CONSENSUS. Hence any defender can count at most $|C| \leqslant t$ copies of $\bar{b}$.

  Since $n-3t > t$ by our choice of t, all defenders will output $(b,\cdot)$ as well.
- If all defenders input the same x, then any defender shall count $\geqslant n-2t$ copies of x and $\leqslant t$ copies of $\bar{x}$. So he will output $(x,1)$. □

Now the main theorem easily follows.

Theorem 26. Our new version of ASYNC-CONSENSUS terminates with probability 1 (actually within four phases in expectation). Upon termination, it is

- **Consistent:** all defenders output the same value.
- **Persistent:** if all defenders input the same value x, then they all output x.

Exercise 7. Prove the theorem.

7 Asynchronous MPC

Assume $t := \lfloor (n-1)/4 \rfloor$. With the help of asynchronous broadcast and consensus, we can design asynchronous MPC protocol. It roughly follows the tricks in Section 5, which we now review.

- RELIABLE-REFILL
  - each player generates a random value and shares it with degrees $\leqslant t$ and $\leqslant 2t$.
  - expand these randomness via hyperinvertible map;
  - assemble some of the expanded randomness, abort if any inconsistency is observed;
  - keep the remaining expanded randomness.
- RELIABLE-DISTRIBUTE
  - the distributer assembles a fresh randomness via Berlekamp-Welch, masks his secret, and broadcasts the result;
  - each player unmasks it locally.
- RELIABLE-ASSEMBLE reconstructs a secret towards every player, using Berlekamp-Welch decoder.
- MULTIPLY
  - players locally perform multiplication $\boldsymbol{c} := \boldsymbol{a} * \boldsymbol{b}$ and mask $\boldsymbol{\delta} := \boldsymbol{c} - \boldsymbol{\rho}^+$;
  - players reliably assemble δ;
  - players locally unmask $\boldsymbol{\rho} + (\delta, \dots, \delta)$.

Let's address the tricky points that we have highlighted. (i) Berlekamp-Welch decoder has the potential of abortion, but luckily $|D| \geqslant n - t > 3t \geqslant d + t$ for $d \in \{t, 2t\}$ under our assumption, so we are safe. (ii) RELIABLE-REFILL is susceptible to abortion too, and this time there's no lucky fix. (iii) RELIABLE-DISTRIBUTE calls for broadcast which might not terminate; but the problem is unsolvable since the distributer is a centralized role.

In the following we suppress problem (ii) by redesigning both RELIABLE-REFILL and RELIABLE-DISTRIBUTE. Originally, the latter relies on the former, but now we will turn the dependency around.

7.1 A new distribution protocol

Our new distribution protocol makes use of 2D polynomials; the added dimension allows other players to verify the polynomial degree without compromising privacy. After the check completes, the 2D polynomial is projected back to the familiar 1D case.

Let us introduce some terminology. A 2D polynomial $f(x, y)$ of degree at most d is a function that, for all fixed $x, y \in \mathbb{F}$, both $f(\cdot, y)$ and $f(x, \cdot)$ are polynomials of degree at most d. Define $F_d^2(s) := \{f : \deg(f) \leqslant d \wedge f(0,0) = s\}$.

Protocol ASYNC-DISTRIBUTE

#	player	on event	do
(0)	k	started	sample $f \in F_t^2(s)$ uniformly send $f(i, \cdot), f(\cdot, i)$ to each i
(1)	any i	got $f(i, \cdot), f(\cdot, i)$ of degree $\leqslant t$	send $v_{ij} := f(i, j)$ to each j
(2)	any j	got v_{ij} from i	**if** $v_{ij} = f(i, j)$ **then** ASYNC-BROADCAST "j approves i"
(3)	any j	observed $n - t$ players who approve each other	record these players in my set K
(4)	any j	got $2t + 1$ many v_{ij} $(i \in K)$ s.t. $\exists g : \deg(g) \leqslant t$ with $g(i) = v_{ij}$	take my share as $s_j := g(0)$

Remark 27. A defender j might not be able to make himself into K.

Lemma 28. ASYNC-DISTRIBUTE has similar guarantees to that of ASYNC-BROADCAST:

- Defenders either *all* get stuck, or *all* terminate and their shares lie on a polynomial of degree at most t.
- If $k \in D$ then all defenders eventually terminate and their shares lie on the polynomial $f(\cdot, 0) \in F_t(s)$.

Proof. If $k \in D$ then obviously all defenders terminate. Next we consider the general case. Assume defender $j \in D$ terminates, so that he saw a set K of $n-t$ mutually approving players. All defenders will eventually receive these (broadcast) approvals by Lemma 22 and activate (3). Since $|D \cap K| \geqslant n - 2t > 2t$, there are always enough values to trigger (4), the termination condition.

We move on to show the semantic claims. Suppose that a defender $j \in D$ observed the mutual-approval set K, and gathered $\{v_{ij}\}_{i \in I}$ for some $I \subseteq K : |I| = 2t+1$ with $g(i) = v_{ij}$ for polynomial g of degree $\leqslant t$. We make several observations:

a) Note that $D \cap K$ contains mutually approving defenders. So for $a, b \in D \cap K$, there is no ambiguity to write v_{ab}. Moreover, due to condition in rule (1), the polynomials $f(a, \cdot)$ and $f(\cdot, b)$ have degree $\leqslant t$, thus the map $(D \cap K)^2 \ni (a, b) \mapsto v_{ab}$ determines a 2D polynomial $\Lambda_{(D\cap K)^2}$ of degree $\leqslant t$ via Lagrange interpolation.

b) Similarly, the subset $D \cap I$ determines a 2D polynomial $\Lambda_{(D\cap I)^2}$ of degree $\leqslant t$ via Lagrange interpolation.

c) Then observe that $|D \cap I| \geqslant t+1$. But $\Lambda_{(D\cap I)^2}$ and $\Lambda_{(D\cap K)^2}$ agree on $(D \cap I)^2$, which is enough to ensure $\Lambda_{(D\cap I)^2} = \Lambda_{(D\cap K)^2}$.

d) Finally and apparently, $g(\cdot) = \Lambda_{(D\cap I)^2}(\cdot, 0)$. Therefore $s_j := g(0) = \Lambda_{(D\cap I)^2}(0, 0)$.

Now we are ready to put things together. Take arbitrary defenders $j, j' \in D$. Applying the above observations individually to j and j', we yield

$$\begin{aligned} s_j &= \Lambda_{(D\cap I)^2}(0,0) = \Lambda_{(D\cap K)^2}(0,0) \\ s_{j'} &= \Lambda_{(D\cap I')^2}(0,0) = \Lambda_{(D\cap K')^2}(0,0). \end{aligned}$$

But $|D \cap K \cap K'| \geqslant n - 3t \geqslant t+1$, so $\Lambda_{(D\cap K)^2} = \Lambda_{(D\cap K\cap K')^2} = \Lambda_{(D\cap K')^2}$. This implies $s_j = s_{j'}$ and concludes our proof. □

Remark 29. Diagrammatically, the proof couples two defenders by the chain $I \leftrightarrow K \leftrightarrow K' \leftrightarrow I'$. The detour to K and K' is crucial: It is not possible to couple I and I' by a direct counting argument.

7.2 Finding core set

Suppose every player is taking part in n parallel protocols $P_1, \ldots, P_n$. The protocol P_k was initiated by player k and satisfies the following termination conditions:

- If $k \in D$ then every defender terminate in P_k;
- If $k \in C$ then the defenders either all get stuck in P_k, or all terminate in P_k.

Clearly ASYNC-BROADCAST is a valid example.

The following protocol makes sure that the defenders agree on a set $S \subseteq \{1, \ldots, n\}$ of size $\geqslant n - t$ such that all $\{P_k\}_{k \in S}$ terminate.

Protocol ASYNC-CORE			
#	**player**	**on event**	**do**
(0)	any i	terminated in protocol P_k	input $0 \to$ ASYNC-CONSENSUS$_k$ if i hasn't input to it yet
(1)	any i	$\{k : \text{ASYNC-CONSENSUS}_k \to 0\}$ reaches size $n-t$	$\forall k$, input $1 \to$ ASYNC-CONSENSUS$_k$ if i hasn't input to it yet
(2)	any i	finished in all ASYNC-CONSENSUS$_k$	output $\{k : \text{ASYNC-CONSENSUS}_k \to 0\}$

Lemma 30. Every defender $i \in D$ terminates in ASYNC-CORE (almost surely) and outputs the same set $S \subseteq \{1, \ldots, n\}$. Moreover $|S| \geqslant n-t$ and every defender eventually terminates in protocols $\{P_k\}_{k \in S}$.

Proof. First we show that every defender shall terminate in ASYNC-CORE. The argument is arranged in steps:

- We claim that there exists a defender that invokes (1). Suppose to the contrary that no defender ever invokes it. Let's take $k \in D$. By assumption, all defenders terminate in P_k, so they will input 0 to ASYNC-CONSENSUS$_k$ via rule (0). (They cannot have input 1 already since (1) is never invoked by assumption!) Now by Theorem 26, ASYNC-CONSENSUS$_k$ almost surely outputs 0. This works for all $k \in D$, and $|D| \geqslant n-t$ is enough to trigger (1), a contradiction.
- Once the very first defender invokes (1), we know $\{k : \text{ASYNC-CONSENSUS}_k \to 0\}$ has size $n-t$. Due to consistency of ASYNC-CONSENSUS, all defenders can spot at least $n-t$ ASYNC-CONSENSUS's that output 0, eventually. So in fact all defenders will invoke (1).
- Rule (1) instructs the defenders to fill in all missing inputs. Again by Theorem 26, all ASYNC-CONSENSUS$_k$ ($k = 1, \ldots, n$) shall terminate almost surely and kick off rule (2).

Now that they terminate, clearly they output the same set S by consistency. As we have pointed out, one (and thus every) defender fires rule (1), implying $|S| \geqslant n-t$.

It remains to argue that every defender shall terminate in $\{P_k\}_{k \in S}$. Let's consider any $k \in S$, so that ASYNC-CONSENSUS$_k \to 0$. Observe there is at least one defender, say $i \in D$, who had input $0 \to$ ASYNC-CONSENSUS$_k$; otherwise by persistency the output would be 1. Why did i input 0? Because i terminated in protocol P_k. But then all defenders eventually terminate in P_k by the guarantee of protocol P_k. □

7.3 A new refill protocol

Hopefully, the refill protocol manifests itself at this point.

Protocol ASYNC-DISTRIBUTE			
#	**player**	**on event**	**do**
(0)	any i	started	ASYNC-DISTRIBUTE (denoted P_i) a random r_i and ASYNC-CORE on $P_1, \ldots, P_n$
(1)	any i	ASYNC-CORE $\to S$ and $P_j \to \boldsymbol{r}_j$ for all $j \in S$	compute (component i of) $\boldsymbol{r} := \sum_{j \in S} \boldsymbol{r}_j$

7.4 Debate of the model

Surely, you can debate that the scheduler in the asynchronous model is too strong. Maybe in reality, the scheduler (aka nature) can only delay a message up to 10 seconds. If that is true, then we can use synchronous protocols with 10 seconds per round. Everything should go perfectly. Why bother using asynchronous protocols?

Well, from a pragmatic view, don't forget the following. A delay of 10 seconds is rare. Say 95% messages can arrive within a second and only 1% messages need 10 seconds. But the synchronous model must prepare for the worst case and spend 10 seconds per round! In contrast, an asynchronous model is much more flexible and adapts automatically to the delivery pace of messages, thus statistically saving plenty of waiting time.