EX4 RDS 2026
Asynchronous MPC for omission failures
Version 1

Submission options

This exercise consists of a theory component and two system projects.

You are required to complete *one* of these options.

Completing more than one option is allowed and will be considered for bonus credit, but may require substantial additional work.

System model for the theory exercise and the first system project. Assume a system of n parties communicating over an asynchronous network. The adversary is adaptive and computationally unbounded.

Assume $n = 3f+1$. The adversary may corrupt up to f parties. A corrupt party reveals all its internal state to the adversary, and the adversary may cause arbitrary omission failures on behalf of corrupt parties. Parties never send incorrect messages, they may only omit sending messages.

All computation is over a large finite field $\mathbb{F}$.

Randomness beacon. Parties have access to a randomness beacon. For every index $i \geq 1$, once $f+1$ parties request the ith beacon value, the beacon sends to all parties an independently sampled uniformly random element $\rho_i \in \mathbb{F}$. The value ρ_i is unpredictable to the adversary before it is released.

1 Asynchronous MPC with omission failures

The goal in this question is to implement an MPC protocol for a function a that can be represented as an arithmetic circuit C_a over $\mathbb{F}$ with addition and

multiplication gates.

1.1 The function

We consider a function that computes the outcome of a second price auction.

Each party starts with an input $x_i \in \mathbb{F}$. For simplicity, you can assume all inputs are unique.

Assume bids are unique integers in the range $[0, 2^k)$. Inputs are interpreted as integers embedded into the field $\mathbb{F}$. To compute comparison based operations such as maximum and second maximum using arithmetic circuits over $\mathbb{F}$, parties may use a standard bit decomposition subprotocol to represent each input x_i as a vector of k secret shared bits.

The function a takes as input a set $I \subset [n]$ of bids $B = \{x_i \mid i \in I\}$ with $|I| = n - f$. Let $w_{id} = \text{argmax}_i\{x_i \in B\}$ be the winning party and let $w_{val} = \max\{x_i \in B \setminus \{x_{w_{id}}\}\}$ be the second highest bid.

The function returns a vector of outputs $o_1, \ldots, o_n$ where $o_{w_{id}} = w_{val}$ and all other values are 0.

Hint. Arithmetic circuits over a field do not natively support comparison. A standard approach is to decompose inputs into bits and express comparison logic using addition and multiplication gates.

Remark. Even in the ideal world, the adversary gets to choose the set I of parties used to compute the outcome of a. It gets to do that without knowing the content of the bids.

1.2 Tasks

- Explain in words how to transform this function to a circuit C_a. Using big O notation as a function of n and k, how many additions and multiplications are required.
- Explain how to mask the output so that only party i learns o_i. Since a is deterministic, this may require extra interaction, or the use of the beacon.

1.3 Complete secret sharing

We define *Complete Secret Sharing* as a strengthening of verifiable secret sharing that guarantees that any party terminating the sharing protocol holds a valid share of the uniquely bound secret.

Consider a secret sharing protocol for $n = 3f + 1$ parties with omission failures only, in the asynchronous network model above. There is a designated dealer holding an input value $s \in \mathbb{F}$, and two protocols, *share* and *recover*.

Termination

- If all non faulty parties invoke *share*, then all non faulty parties eventually terminate.
- If all non faulty parties invoke *recover* after terminating *share*, then all non faulty parties eventually terminate.

Hiding

- If the dealer is non faulty, then for any set F of at most f parties, there exists an efficient simulator $\mathcal{S}$ that produces a distribution indistinguishable from the joint view of the adversary during *share*, independently of s.

Binding

- If the dealer is faulty, then there exists an efficient extractor $\mathcal{E}$ that, given the views of all non faulty parties at termination of *share*, outputs a polynomial p of degree at most f such that any subsequent invocation of *recover* outputs $p(0)$.

Completeness

- If a party terminates the *share* protocol, then it holds a valid share consistent with the polynomial p extracted by the Binding property.

Validity

- If the dealer is non faulty and *recover* is invoked after *share* terminates, then the output of *recover* equals the dealer input s.

Task. Design a complete secret sharing protocol satisfying the above properties in this model. Analyze its message complexity and expected termination time.

1.4 Agreeing on the input set

In this section we formalize and use an agreement on a common set (ACS) primitive.

Goal. Each party i proposes a value v_i (in our setting, an identifier for a complete secret sharing instance that binds its bid). Parties should eventually agree on a set V of proposals, such that V contains $n - f$ proposals.

ACS interface. An ACS protocol has an input for each party i, denoted v_i, and an output set $V \subseteq \{v_1, \ldots, v_n\}$.

Termination

- If all non faulty parties invoke ACS, then all non faulty parties eventually terminate with probabilty 1.

Agreement

- All non faulty parties that terminate output the same set V.

Validity

- Every element of V is the input of some party.

Quality

- If a non faulty party outputs a set V, then $|V| = n - f$.
- Moreover, V contains the inputs of at least $n - 2f$ non faulty parties.

Remark. In a fully asynchronous model, deterministic consensus style primitives cannot guarantee termination against even one omission fault. In this exercise, randomized termination with probability 1 is enabled by the beacon.

Task. Explain how to implement *Agreeing on the input set.*

1.5 Adding two secret sharings

1.6 Multiplying two secret sharings

Hint, a core subtask is multiplication. Suppose parties hold degree f sharings of two secrets a and b, meaning there exist degree at most f polynomials p_a and p_b with $p_a(0) = a$ and $p_b(0) = b$, and party i holds shares $a_i = p_a(i)$ and $b_i = p_b(i)$. The goal is for parties to end with a degree f sharing of $c = ab$.

BGW idea, each party locally computes $d_i = a_i b_i$. Then the points $\{(i, d_i)\}$ lie on the polynomial $p_d(x) = p_a(x)p_b(x)$, which has degree at most $2f$, and satisfies $p_d(0) = ab$. So local multiplication produces a degree $2f$ sharing of the correct product.

The next step is degree reduction. One standard approach is to pick a set T of at least $2f+1$ parties, interpolate $p_d(0)$ from their shares using Lagrange coefficients, and then reshare the resulting value so that the new sharing has degree f. Concretely, with Lagrange coefficients $\{\lambda_i\}_{i\in T}$ for interpolation at 0 from the points in T, we have the identity

$$c = p_d(0) = \sum_{i\in T} \lambda_i d_i.$$

In this asynchronous omission model, a key challenge is choosing the set T. Different parties may receive different subsets of the d_i values at different times, and some parties may omit sending. The protocol therefore needs a subroutine that causes all non faulty parties to agree on a common set T of parties whose d_i values are considered complete for this multiplication gate, so that everyone uses the same interpolation coefficients. This is exactly the type of set selection that can be driven using the ACS machinery described earlier.

Task. Explain how your multiplication protocol separates (1) producing the degree $2f$ sharing via local products, from (2) agreeing on a common set T and using interpolation coefficients to drive a degree reduction step back to degree f, without revealing c.

1.7 Combining the primitives

Explain how to combine all primitives to implement the auction.

Bonus: Beaver multiplication triplets (theory)

This bonus focuses on the theoretical role of Beaver multiplication triplets in secure multiparty computation.

A Beaver triplet is a triple (a, b, c) of field elements such that $c = ab$, where each of a, b, c is secret shared among the parties. Triplets are generated in an offline preprocessing phase, independently of the actual inputs to the computation.

Tasks.

- Explain how Beaver triplets enable multiplication of two secret shared values using only local computation and a constant number of rounds of interaction, assuming the triplets are already available.
- Prove correctness: show that the resulting shares reconstruct to the correct product of the inputs.
- Prove privacy: argue that no additional information about the multiplied secrets is revealed beyond what follows from the output.
- Explain how Beaver triplets can be generated using the randomness beacon and complete secret sharing primitives defined earlier in the exercise.
- Compare the round complexity and message complexity of multiplication with and without Beaver triplets in the asynchronous omission failure model. How many rounds are saved in the online phase? What is the overhead in the offline phase?

2 System project 1: implementing the auction

- Implement this auction for $n = 4$ parties and $f = 1$. Assume bids are 5-bit integers, that is, values in $[0, 32)$, and that all bids are unique.

Comparison and max operations should be implemented using bit decomposition and arithmetic circuits, or treated as a clearly specified black box subroutine.

- The implementation should be fully asynchronous and event driven. You may not assume rounds or synchrony.
- The adversary may cause one party to omit arbitrary messages and may arbitrarily delay messages.
- Build an extensive test suite that covers honest executions, executions with one omitting party, and executions with random message delays.
- Measure and report the total number of messages sent and the number of beacon invocations.

Bonus: Beaver multiplication triplets

As an optional extension, learn about Beaver multiplication triplets and how they enable efficient secure multiplication.

Add an offline preprocessing stage in which the parties jointly generate a supply of Beaver triplets (a, b, c), where $c = ab$, each value being secret shared among the parties. You may assume the randomness beacon and the complete secret sharing primitives from the exercise.

Modify your multiplication protocol to use pre generated Beaver triplets instead of performing degree reduction from scratch for each multiplication gate.

Benchmark and compare the system with and without Beaver triplets. In particular, measure the total number of messages, the number of beacon invocations, and the time to complete the auction, and explain the observed differences.

3 System project 2: adding unlinkable payments to EX3

This project builds on EX3 and focuses on implementing a simplified replicated payment system *with unlinkability*. Unlike EX3, unlinkability is an explicit requirement in this project.

Like EX3, the system consists of n servers and multiple clients. Where $n = 2f + 1$, and the system tolerates up to f omission failures.

- Implement a version of the payment scheme from EX3 for $n = 5$ servers tolerating $f = 2$ omission failures, and 5 clients.
- Assume the servers already hold shares of a threshold signing key corresponding to a system wide public key. Concretely, assume a Distributed Key Generation (DKG) protocol has been run successfully before any client operations begin.
- Assume a fixed denomination of value 1. There is only a single token type. There are no multi input payments and no change tokens.
- Support the following operations:
  - *Mint*: a client requests the servers to mint a single token of value 1 by consuming 1 unit of its un minted balance.
  - *Pay*: a client spends exactly one valid unspent token to transfer value to another client. The recipient may be specified either as a fresh public key or via a long term identity public key.
- The servers must ensure authorization unforgeability, no double spend, liveness in a constant number of rounds, and conservation of value, as in EX3.
- **Unlinkability.** The system must ensure unlinkability, which is *not* provided by the baseline scheme of EX3. Informally, a passive adversary observing all server state and all messages should not be able to link a *Pay* operation to the *Mint* operation that created the spent token, except with negligible probability.
- Specify a clear threat model for unlinkability, including what the adversary observes, what is considered auxiliary information, and what constitutes a link.
- Build a test suite demonstrating correctness, no double spend, and unlinkability across multiple executions. Include at least one test that mints many tokens and then pays with a random one, and checks that transcripts do not expose which minted token was spent.

Bonus: Distributed Key Generation

Instead of assuming a pre-existing shared signing key, implement a Distributed Key Generation (DKG) protocol for the servers, following the DKG construction of EX3. Here you can assume access to the same randomness beacon.

Your DKG should ensure that, at termination, servers hold shares of a uniformly random secret signing key and that the corresponding public key is known to all parties.

Analyze the round complexity and message complexity of your DKG