UC DAVIS
COLLEGE OF ENGINEERING

ECS265: Distributed Database Systems

Paper Presentation

- Shyam Varahagiri, Kiransingh Pal,
  Sachin Balasubramanyam, Anish Kataria

Asynchronous Secure Computations with Optimal Resilience

- Michael Ben-Or, Boaz Kelmer, Tal Rabin

Secure Multiparty Computation

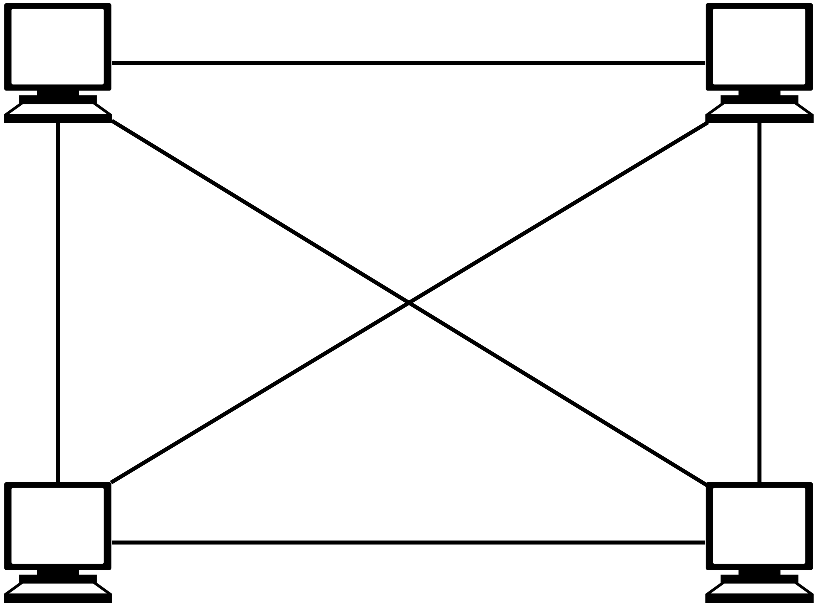
- Secure Multiparty Computation is a versatile and very powerful tool in the design of cryptographic protocols.
- It allows multiple parties to collaboratively compute a function over their private data without revealing the individual inputs to each other, ensuring the privacy of all participants while still achieving a shared result.
- Essentially, it enables "black box" calculations where only the final output is visible, not the underlying data used to compute it.

Secure Multiparty Computation

Let's say we have *n* players (processors).

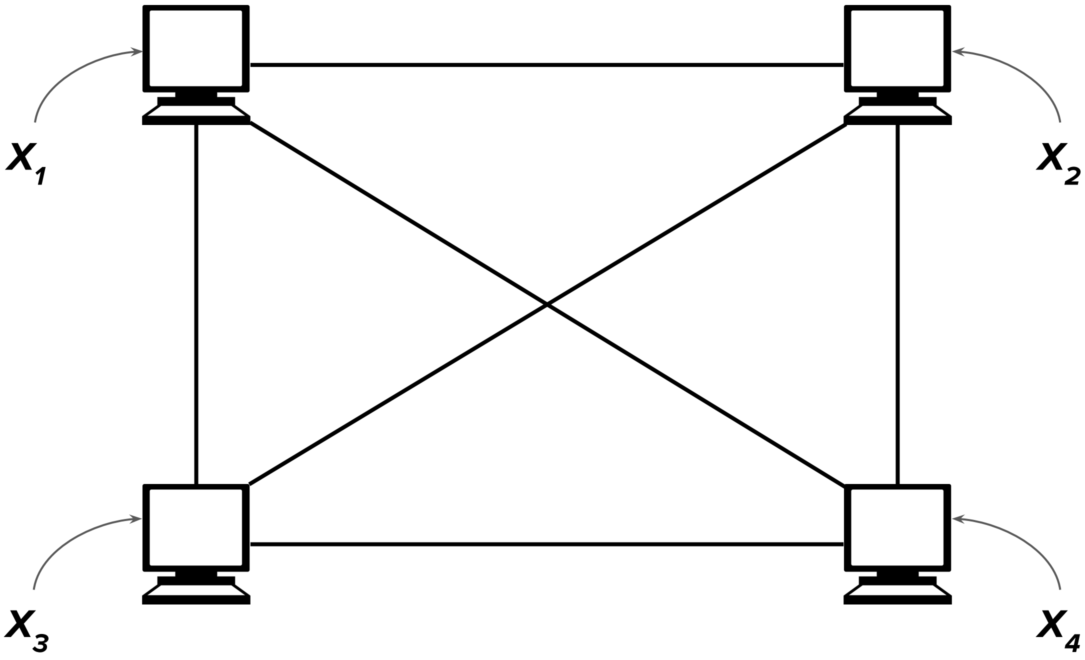
Secure Multiparty Computation

And any 2 players are connected via a secure and reliable communication channel.



Secure Multiparty Computation

Each player has a private input value X_i, and the goal is to collectively compute some function $F(X_1, X_2, \ldots\ldots, X_n)$.



Secure Multiparty Computation

Asynchronous multiparty computation generally consists of three steps:

Secure Multiparty Computation

Asynchronous multiparty computation generally consists of three steps:

1. Each player P_i commits himself to his input.

$$X'_i = X_i \text{ for each honest player } P_i.$$

Secure Multiparty Computation

Asynchronous multiparty computation generally consists of three steps:

1. Each player P_i commits himself to his input.

$$X'_i = X_i \text{ for each honest player } P_i.$$

2. Players agree to a common subset called CompSet for size at least $n - t$.

   This CompSet is the same for all honest players.

Secure Multiparty Computation

Asynchronous multiparty computation generally consists of three steps:

1. Each player P_i commits himself to his input.

$$X'_i = X_i \text{ for each honest player } P_i.$$

2. Players agree to a common subset called CompSet for size at least *n* - *t*.

   This CompSet is the same for all honest players.

3. Players now compute the value $F(y_1, y_2, ..., y_n)$ where:

$$y_i = X'_i \text{ for } P_i \in \text{CompSet, else } y_i = 0.$$

Secure Multiparty Computation

- A protocol is t-resilient if every honest player will complete the computation and output the value $F(y_1, y_2, ..., y_n)$.
- The exact conditions under which secure multiparty computation is possible for synchronous distributed systems have been studied quite extensively.
  - Secure error-free computation is possible exactly under the same conditions needed for the Byzantine Agreement Problem, where, $t < n/3$. *[PSL80, BGW88]* .
  - Furthermore, secure computation is possible, with an exponentially small probability of error, even for $t < n/2$ if we add a broadcast channel to the system *[RB89]*.
- For asynchronous systems, *[BCG93]* proved that asynchronous secure error-free computation is possible if and only if the number of faulty players $t < n/4$.

However, the asynchronous Byzantine Agreement problem can be solved by a randomized error-free protocols for $t < n/3$ [Bra84]. So the question arises:

By allowing an exponentially small probability of error, can we achieve asynchronous secure computation with optimal resilience, where $n/4 \leq t < n/3$?

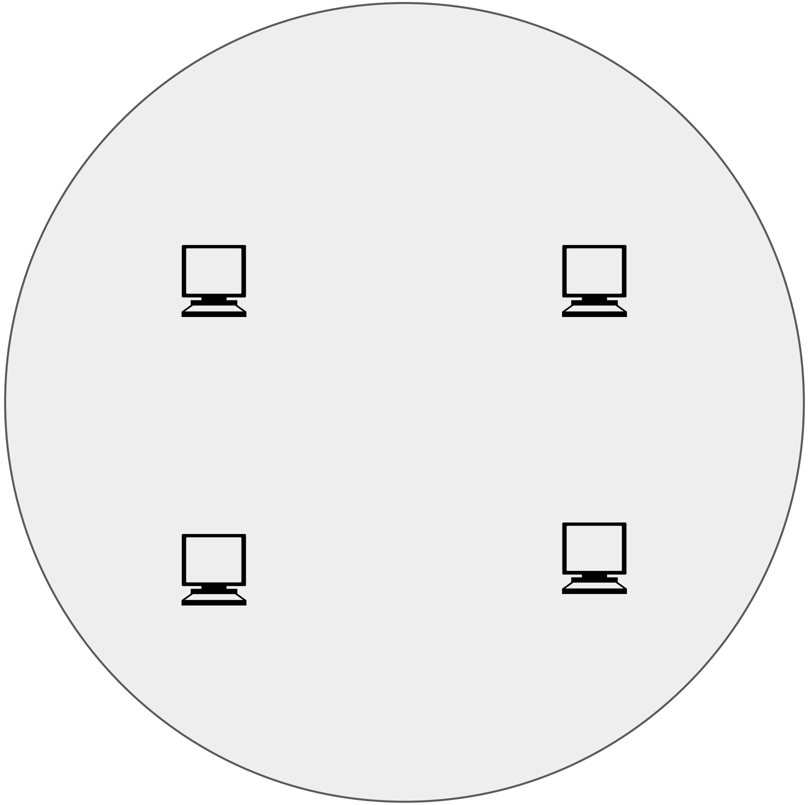
The answer is **yes!** But first we need to understand AVSS (Asynchronous Verifiable Secret Sharing scheme).

AVSS (Asynchronous Verifiable Secret Sharing scheme)

- AVSS is a protocol created by Canetti and Rabin [CR93].
- AVSS allows a dealer to share a secret that can be reconstructed by the players at a later stage.
- AVSS protects the honest players from a faulty dealer, by forcing him to commit to a specific value that is guaranteed to be the reconstructed value.
- The protocol tolerates up to $t < n/3$ faulty players, and has an exponentially small probability of error.
- However, the AVSS by itself is not sufficient for implementing secure multiparty computations. Let us see why.

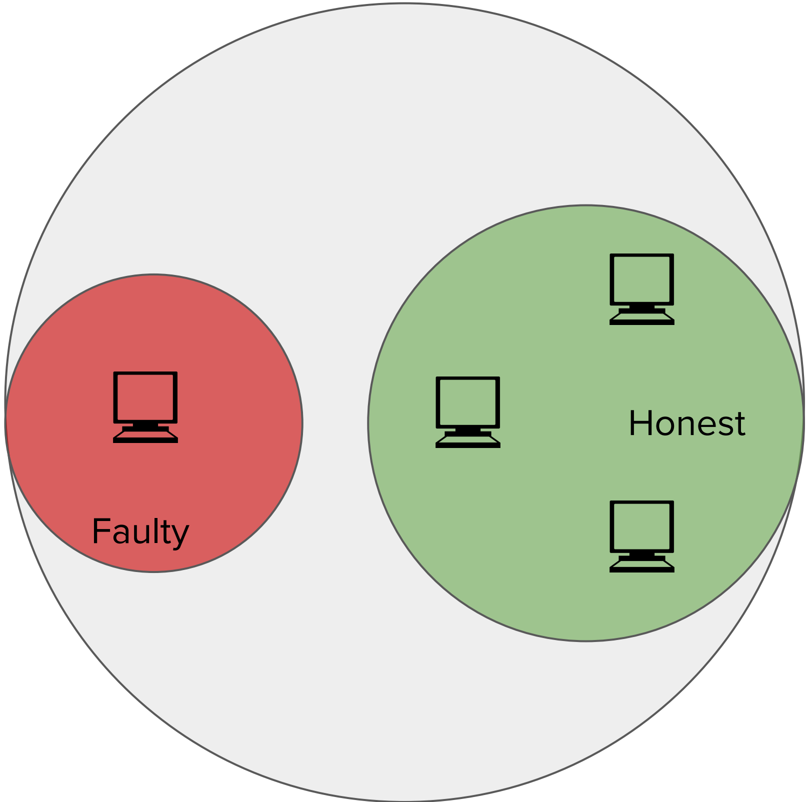
The problem with AVSS.

Let us have a network with $n = 3t + 1$ systems.



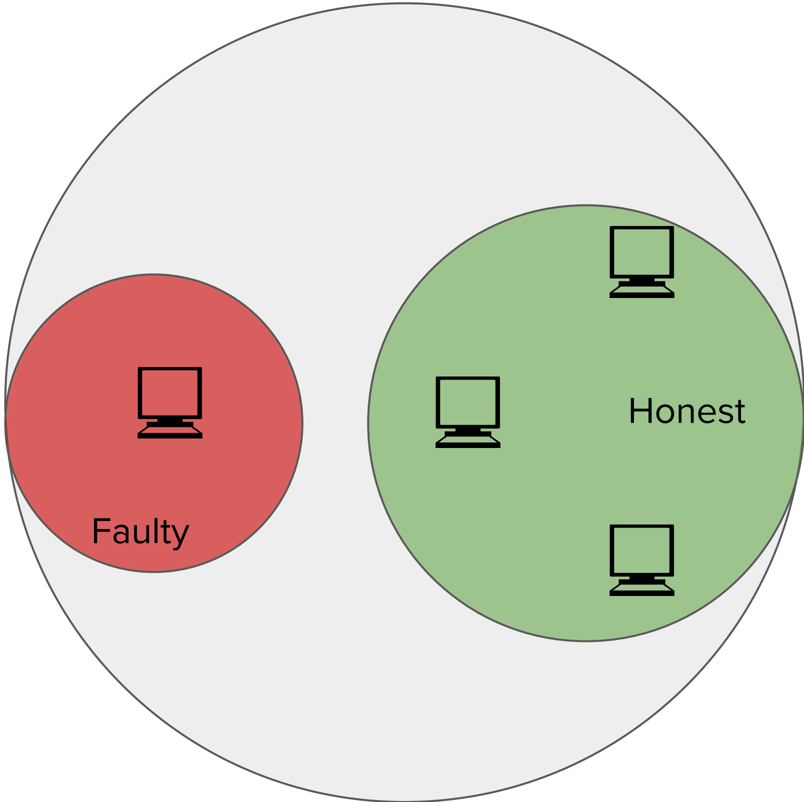
The problem with AVSS.

Of this, we have *t* faulty systems and *2t + 1* honest systems.



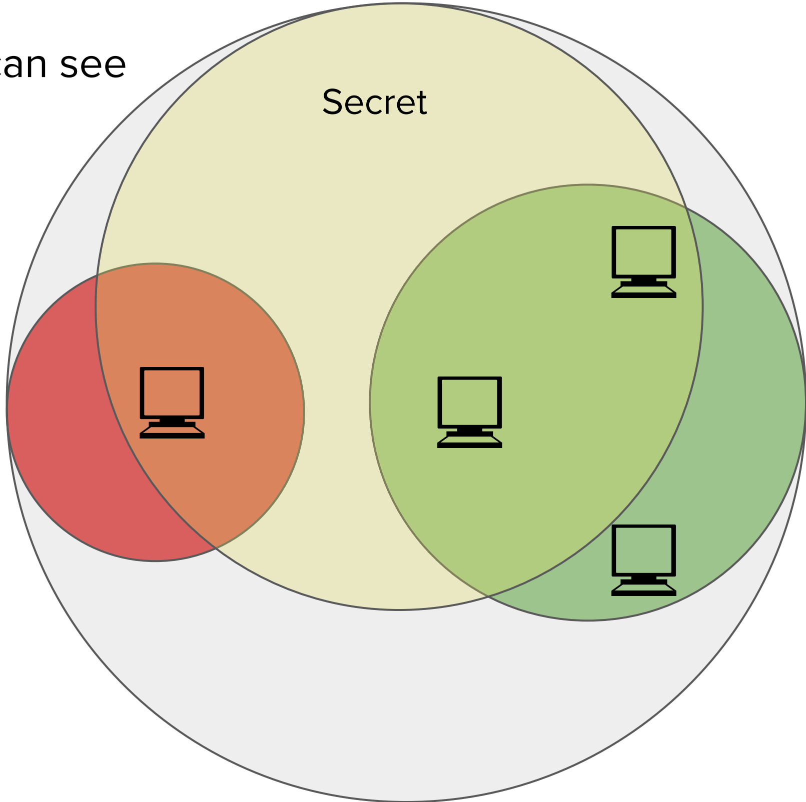
The problem with AVSS.

- AVSS assures that at least $n - t$ players will receive the secret. However, that guarantees that only $n - 2t = t + 1$ players are honest.



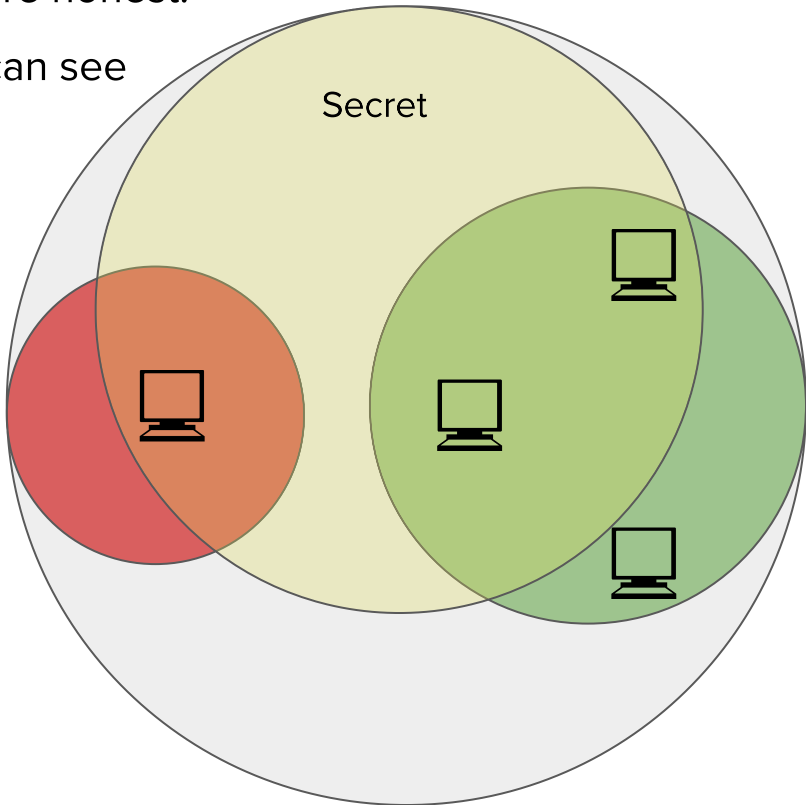
The problem with AVSS.

- AVSS assures that at least $n - t$ players will receive the secret. However, that guarantees that only $n - 2t = t + 1$ players are honest.
- Let the dealer share the secret. Here we can see t faulty players and $t + 1$ honest players have received the secret.



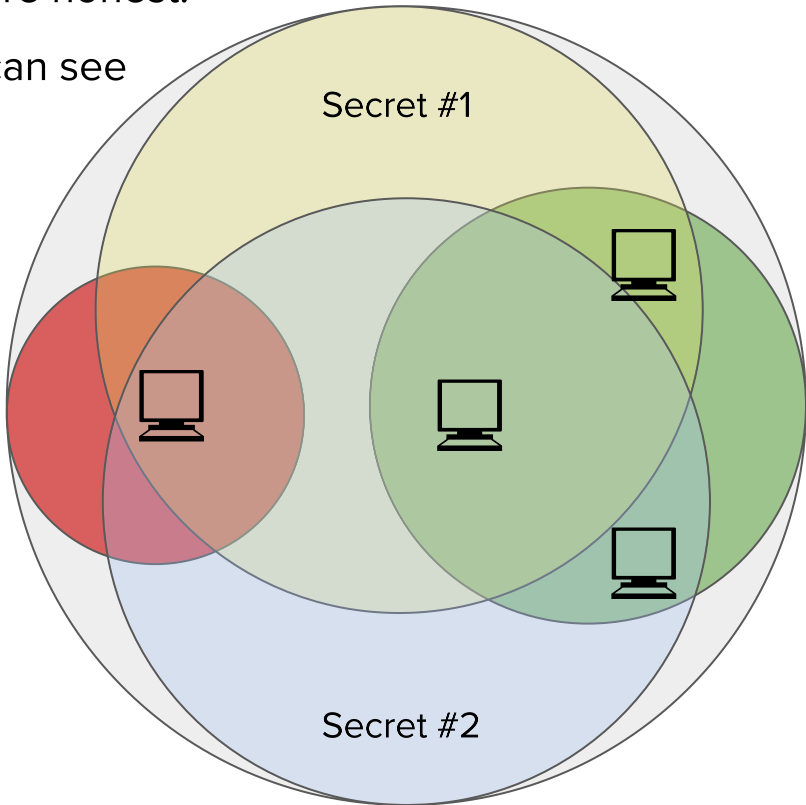
The problem with AVSS.

- AVSS assures that at least $n - t$ players will receive the secret. However, that guarantees that only $n - 2t = t + 1$ players are honest.
- Let the dealer share the secret. Here we can see t faulty players and $t + 1$ honest players have received the secret.
- But what if we want to add another secret?



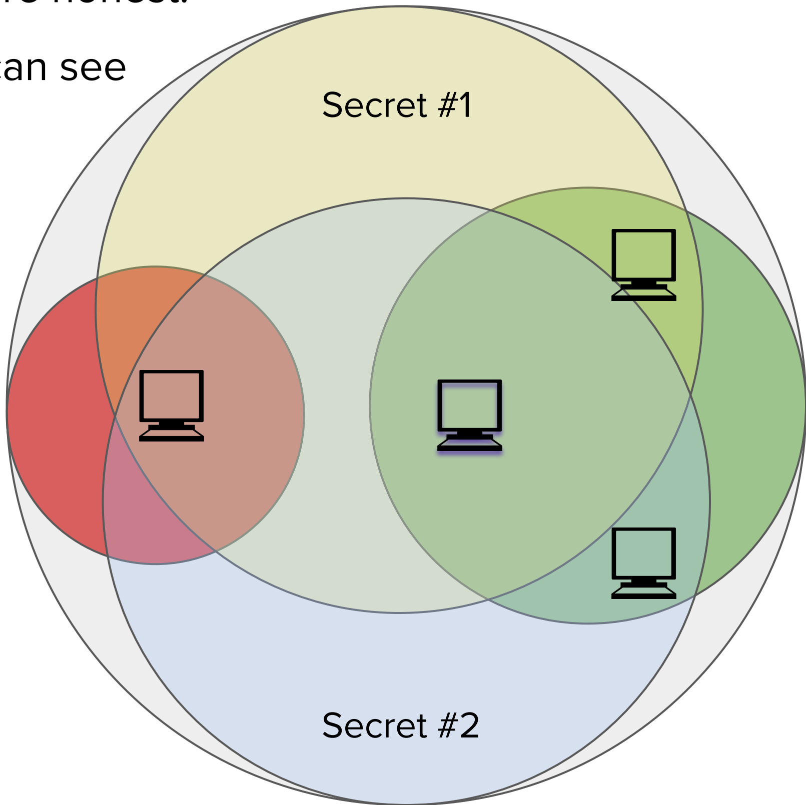
The problem with AVSS.

- AVSS assures that at least $n - t$ players will receive the secret. However, that guarantees that only $n - 2t = t + 1$ players are honest.
- Let the dealer share the secret. Here we can see t faulty players and $t + 1$ honest players have received the secret.
- But what if we want to add another secret?



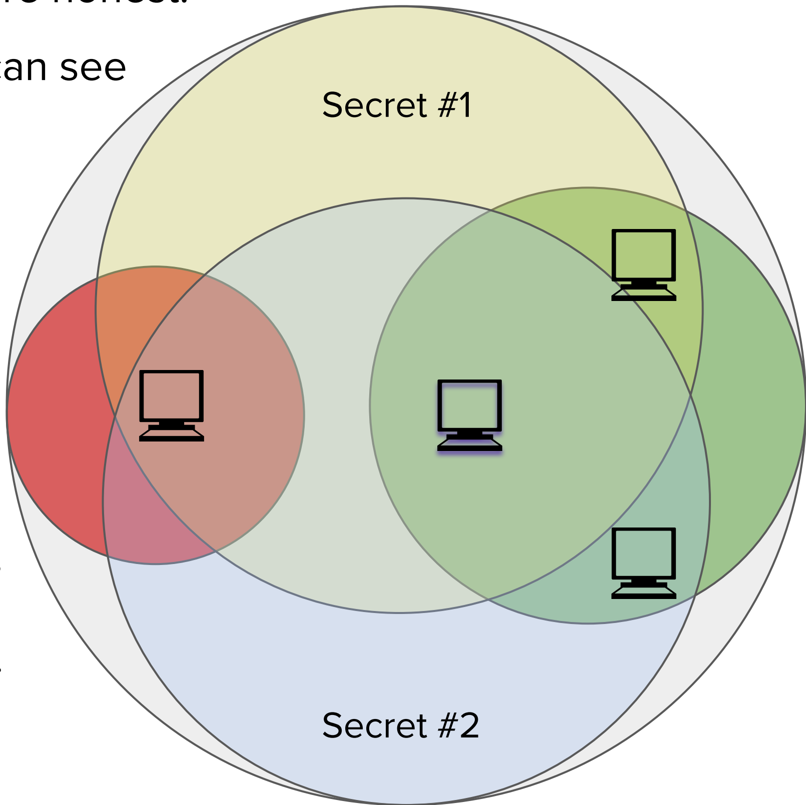
The problem with AVSS.

- AVSS assures that at least $n - t$ players will receive the secret. However, that guarantees that only $n - 2t = t + 1$ players are honest.
- Let the dealer share the secret. Here we can see t faulty players and $t + 1$ honest players have received the secret.
- But what if we want to add another secret?
- We can see that in the worst case only one honest player holds a correct share of both secrets.



The problem with AVSS.

- AVSS assures that at least $n - t$ players will receive the secret. However, that guarantees that only $n - 2t = t + 1$ players are honest.
- Let the dealer share the secret. Here we can see t faulty players and $t + 1$ honest players have received the secret.
- But what if we want to add another secret?
- We can see that in the worst case only one honest player holds a correct share of both secrets.
- To ensure secure computations for $t < n/3$, we must ensure that all honest players eventually obtain their share of the secret.



Ultimate Secret Sharing

- The **AVSS Scheme falls short** of our needs to compute. We need to improve the secret sharing scheme to ensure that **all honest players receive valid shares of each secret** and to incorporate an error detection mechanism to **filter out faulty shares**.

- **USS** is a **cryptographic protocol** which ensures that secrets can be securely shared among players, even in the presence of faulty players, while maintaining the integrity of the shared secret.

Properties of USS Scheme

- Dealer D shares a secret s: For a dealer D, we define two protocols — Sh (for secret sharing) and Rec (for reconstruction).
- **Correctness and Resilience**: We say the scheme is (1-ε) correct and t-resilient if the following conditions hold:
  - **Honest dealer**: If the dealer is honest, every honest player will eventually complete protocol Sh. This ensures the distribution of the secret is correct.
  - **Completion among honest players**: If any honest player completes Sh, then all honest players will eventually complete it. This ensures consistency across all honest players.
  - **Completion of Rec protocol**: Once all honest players have completed Sh and invoke protocol Rec, all honest players will complete Rec. This guarantees the reconstruction of the secret.
  - **Faulty players and secret security**: If the dealer is honest and no honest player has started Rec, faulty players cannot gain any information about the shared secret. This is a critical aspect of security.

Protocol L(i) and G(i)

- **Protocol L(.)** : Ensures that each player can access and maintain their individual share of the secret.

- **Protocol G(.)** : This protocol provides a self-correcting mechanism and outputs null for faulty players. It ensures that faulty players do not influence the reconstruction process.

Definition of an Ultimately Shared Secret

Now, we shift to the idea of a secret that is not just shared by a dealer but is computed collectively by a network of players. This is what we call an **"ultimately shared secret."**

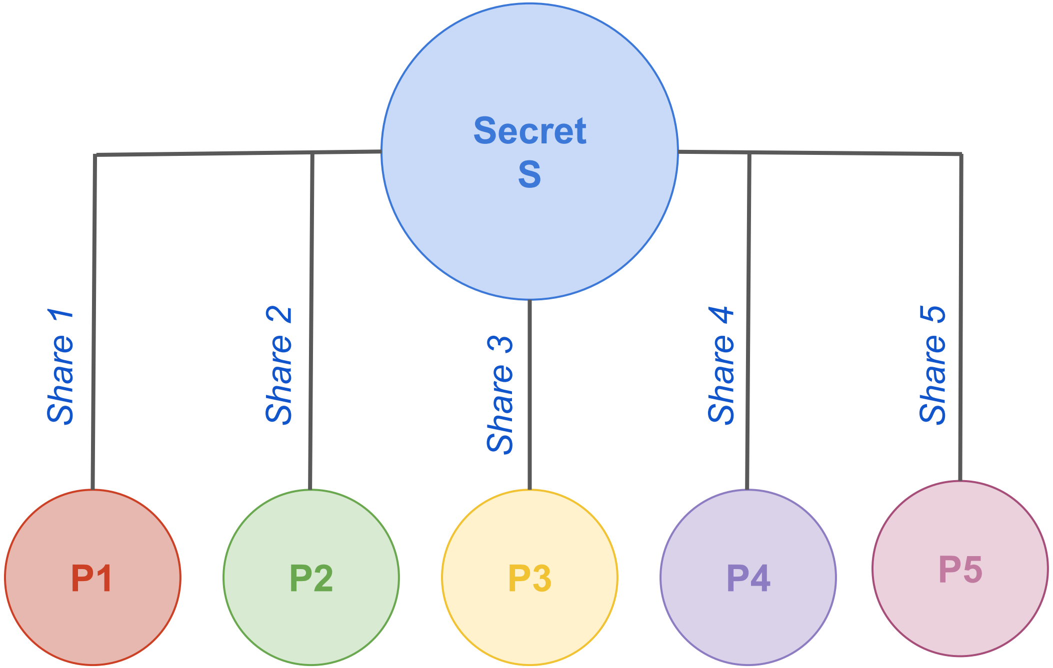
- **Execution of protocol L(i)**: Player P_i invokes L(i) and locally outputs β_i. This means every player knows their own share.
- **Consistency of Protocols:** If G(i) is not null, then G(i) equals L(i) and the share β_i.
- **Protocol Rec**: When all honest players invoke Rec with their inputs from G(1), G(2), ..., G(n), the output is the shared secret r.
- Further, **r = s**, the ultimately shared secret.

Ultimate Secret Sharing

Example

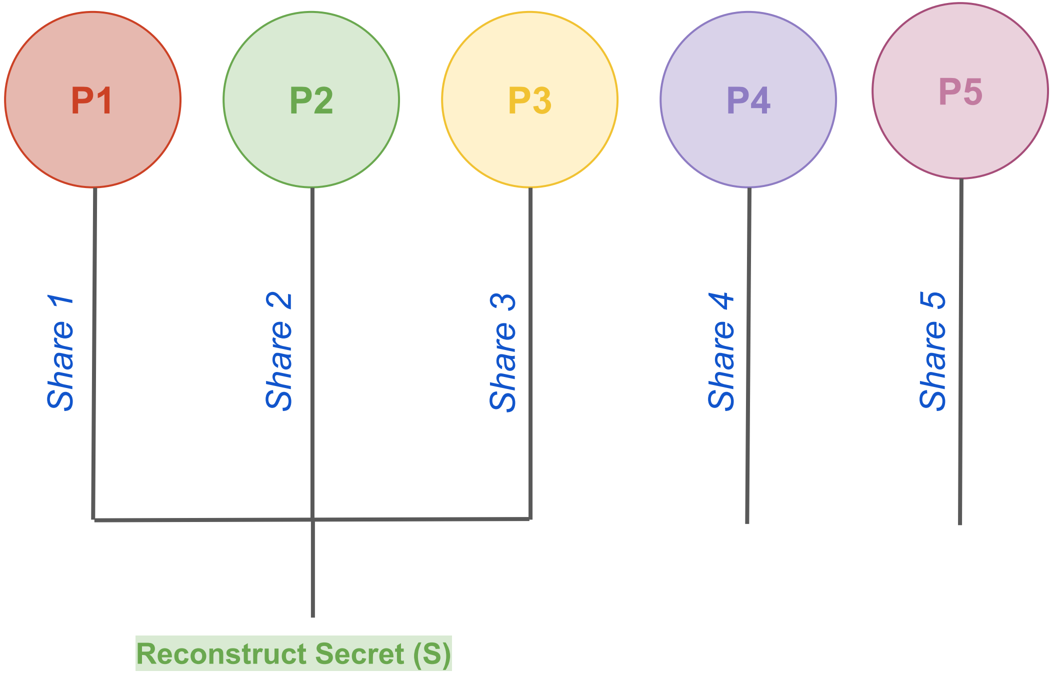
Dealer D hold the secret **S**
Number of Players = **5**
Threshold = **3**

Ultimate Secret Sharing



The dealer shares the secret with players.

Ultimate Secret Sharing

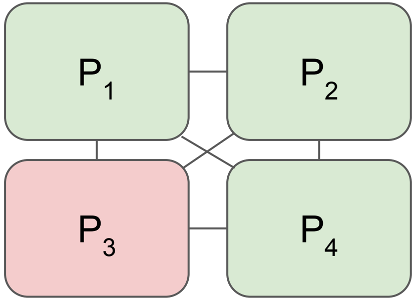


Threshold-based reconstruction - Provides security against collusion or partial disclosure.

Unqualified participants learn nothing - Ensures that unauthorized or unqualified participants cannot gain any information about the secret

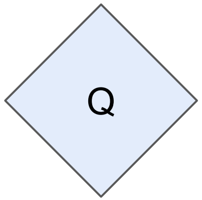
Agreement on a Common Subset

- At least **2t+1** need to satisfy



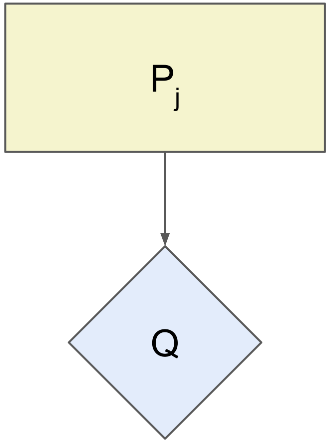
Agreement on a Common Subset

- At least **2t+1** need to satisfy
- Q: Dynamic Predicate



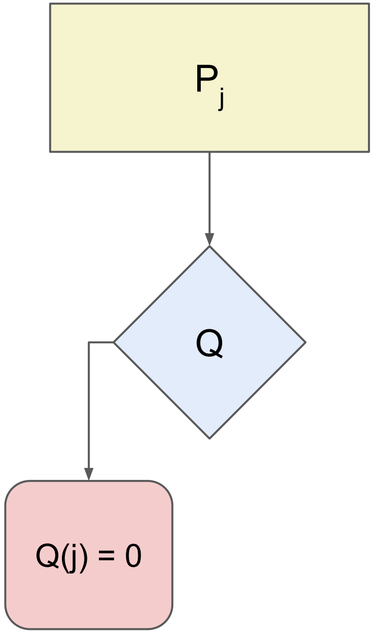
Agreement on a Common Subset

- At least **2t+1** need to satisfy
- Q: Dynamic Predicate



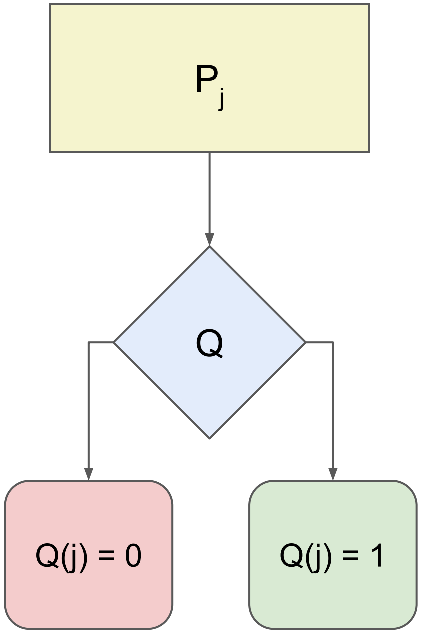
Agreement on a Common Subset

- At least **2t+1** need to satisfy
- Q: Dynamic Predicate



Agreement on a Common Subset

- At least **2t+1** need to satisfy
- Q: Dynamic Predicate



Agreement on a Common Subset

- At least **2t+1** need to satisfy
- Q: Dynamic Predicate

<u>Code for each Player P_i</u>

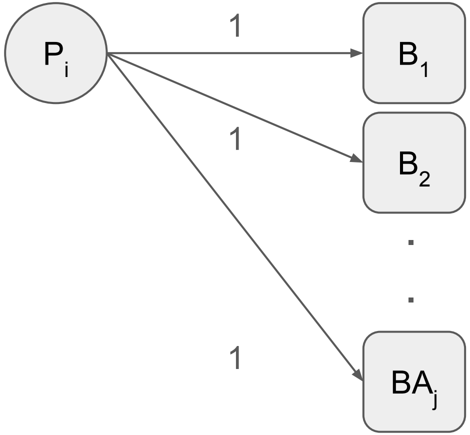
1. For each P_j for whom you know that $Q(j) = 1$, participate in BA_j with input 1.

Agreement on a Common Subset

- At least **2t+1** need to satisfy
- Q: Dynamic Predicate

Code for each Player P_i

1. For each P_j for whom you know that $Q(j) = 1$, participate in BA_j with input 1.

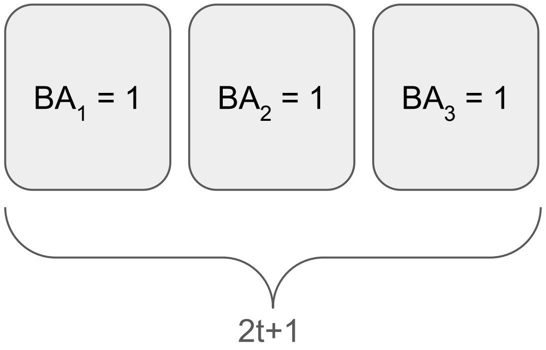


Agreement on a Common Subset

- At least **2t+1** need to satisfy
- Q: Dynamic Predicate

Code for each Player P_i

1. For each P_j for whom you know that $Q(j) = 1$, participate in BA_j with input 1.
2. Upon completing 2t+1 BA protocols with output 1, enter input 0 to all BA protocols for which you haven't entered a value yet.

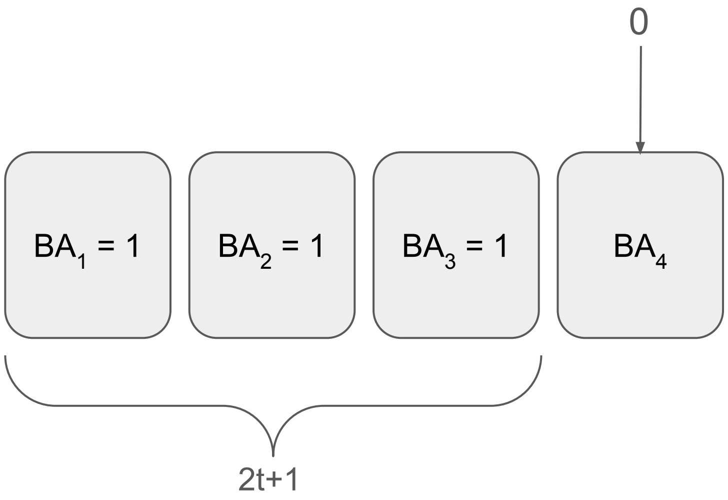


Agreement on a Common Subset

- At least **2t+1** need to satisfy
- Q: Dynamic Predicate

<u>Code for each Player P_i</u>

1. For each P_j for whom you know that $Q(j) = 1$, participate in BA_j with input 1.
2. Upon completing 2t+1 BA protocols with output 1, enter input 0 to all BA protocols for which you haven't entered a value yet.

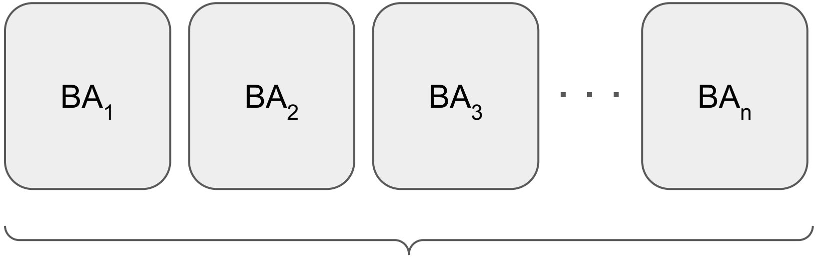


Agreement on a Common Subset

- At least **2t+1** need to satisfy
- Q: Dynamic Predicate

Code for each Player P_i

1. For each P_j for whom you know that $Q(j) = 1$, participate in BA_j with input 1.
2. Upon completing 2t+1 BA protocols with output 1, enter input 0 to all BA protocols for which you haven't entered a value yet.
3. Upon completing all n BA protocols, let your $SubSet_i$ be the set of all indices i for which BA_i had output 1.



Example: $SubSet_i$ = {1, 2, 4, etc}

Agreement on a Common Subset

- **Goal:** Players agree on a subset of size ≥2t+1 where Q(j) = 1 for each player P_i.
- **Proof:**
  - At least 2t+1 BAs terminate with output 1 due to honest players' inputs.
  - All n BAs will terminate because honest players input 0 to the remaining BAs.
  - Result: Honest players agree on a common subset of size ≥2t+1 where Q(j) = 1.

Computation Protocols

Goal:

Compute the function $F(z_1,z_2,\ldots,z_n)$ securely.

1. Input Phase:
Each player P_i commits to their input z_i by USS-Sharing it.
The inputs are now shared among all players.

2. Agreement Phase:
Players use the **Agreement Protocol** to agree on valid shared inputs.
Inputs not properly shared are replaced with **0**.

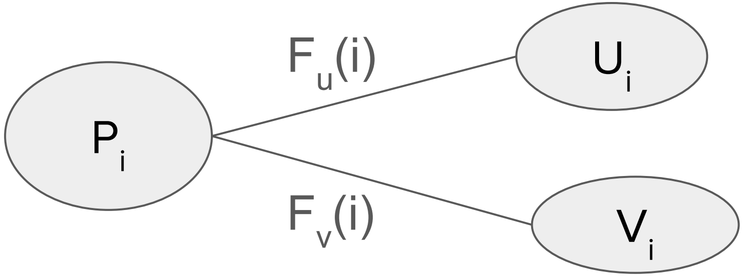
3. Computation & Reconstruction Phase:

- The function F is computed over the agreed inputs.
- Final result is reconstructed using *Rec* (Reconstruction) protocol.

Computing any **Linear** Combination of Secrets

GOAL: Compute a new secret that's a linear combination of secrets u and v.

STEP 1 : **Calculation of Shares**:



- They calculate *weighted shares* by scaling their shares with constants c_1 & c_2.
- These weighted shares correspond to each player's part of the final linear combination $\mathbf{c_1 \cdot u + c_2 \cdot v}$

Computing any **Linear** Combination of Secrets

STEP 2 : **Sharing Using the Sharing Mechanism**

- Players share their weighted shares and the sum of these weighted shares securely with each other.

- This ensures that every player has a piece of the combined result without revealing the original secrets.

Computing any **Linear** Combination of Secrets

STEP 3 : **Verification with Zero-Knowledge Proof (ZKP)**:

- Each player proves that they correctly computed their weighted shares using an Equality Zero-Knowledge Proof. *[Rab 94]*

STEP 4 : **Reconstruction (Rec Protocol)**

- Once all players have their verified shares, they use the reconstruction protocol (Rec) to combine the shares.
- The Rec protocol allows them to jointly reconstruct the linear combination $c_1 \cdot u + c_2 \cdot v$ without any individual learning the full secrets u & v .

Computing the **Multiplication** of Two Secrets

GOAL : Compute a new secret that's a product of secrets u and v.

STEP 1 : **Local Computation of Product Shares**

- Each player P_i starts with shares of secrets u and v, represented as u_i and $v_{i.}$.
- Each player computes:
  - Intermediate shares: $a_iu = u_i$ (share of u) and $a_iv = v_i$ (share of v)
  - Product share: $a_i = a_iu * a_iv$

- **Result**: Each player has a local product share, but the polynomial degree is now 2t.

Computing the **Multiplication** of Two Secrets

STEP 2 : **Sharing and Verification of Product Shares**

- Each player shares their product share ai using the USS-Share protocol.

- Product Zero-Knowledge Proof (ZKP): Each player proves that:

  $a_i = a_iu * a_iv$ to verify correctness without revealing secrets.

Computing the **Multiplication** of Two Secrets

STEP 3 : **Agreement on Valid Shares**

- Each player provides an Equality ZKP to confirm their product shares align with their original shares of u and v.
- Players then run the Protocol Agreement to create a subset (CompSet) of players who completed all steps correctly.
- Result: Only valid shares from compliant players are included in the final computation.

Computing the **Multiplication** of Two Secrets

STEP 4 : **Randomization and Degree Reduction**

- To reduce the polynomial degree from 2t to t, players use a Linear Combination Protocol and a linear transformation. *[BGW88]*

- The result is a randomized polynomial f(x) of degree t with a constant term ***u·v***

- **Outcome**: The product u * v is ultimately shared among players, securely represented by f(x).

THANK YOU !