Secure Multiparty Computation

and Team Matching

Shamir Secret Sharing

Secret values *a, b*: $a, b \in \mathbb{F}_p$

Shamir Secret Sharing

Secret values *a, b*: $a, b \in \mathbb{F}_p$

To share *a,b* create two random degree *t* polynomials *A(), B()* as follows:

$$A(x) = \sum_{i=0}^{t} \alpha_i x^i, \quad B(x) = \sum_{i=0}^{t} \beta_i x^i$$

Set *A(0)* = *a* and *B(0)* = *b*

Shamir Secret Sharing

Secret values *a, b*: $a, b \in \mathbb{F}_p$

To share *a,b* create two random degree *t* polynomials *A(), B()* as follows:

$$A(x) = \sum_{i=0}^{t} \alpha_i x^i, \quad B(x) = \sum_{i=0}^{t} \beta_i x^i$$

Set *A(0)* = *a* and *B(0)* = *b*

n shares are then computed as: $\forall i \in \{1, \ldots, n\} : a_i = A(i), \quad b_i = B(i)$

A() and *B()* are degree *t* polynomials and can be uniquely defined by *n=t+1* points

Uses:

The obvious: sharing of secrets

The secret value could be a key that is tied to {a bitcoin wallet, disk encryption, etc}

Recovering the key then requires cooperation!

Uses:

The obvious: sharing of secrets

The secret value could be a key that is tied to {a bitcoin wallet, disk encryption, etc}

Recovering the key then requires cooperation!

Tiered key management:

s Threshold = 1 out of 1

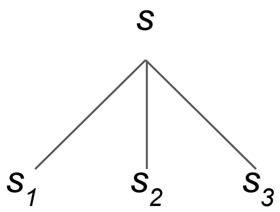
Uses:

The obvious: sharing of secrets

The secret value could be a key that is tied to {a bitcoin wallet, disk encryption, etc}

Recovering the key then requires cooperation!

Tiered key management:



Threshold = 1 out of 1

Threshold = 2 out of 3

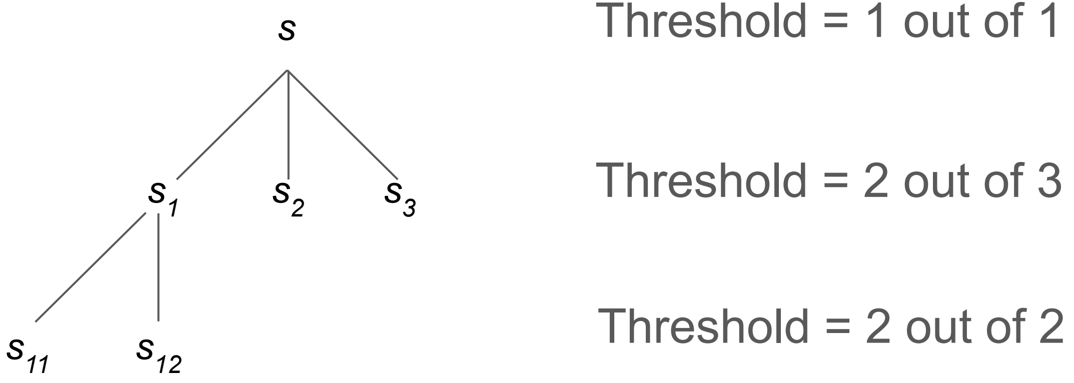
Uses:

The obvious: sharing of secrets

The secret value could be a key that is tied to {a bitcoin wallet, disk encryption, etc}

Recovering the key then requires cooperation!

Tiered key management:



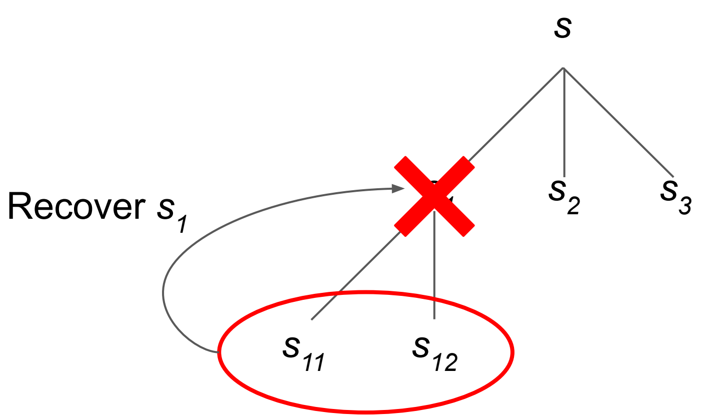
Uses:

The obvious: sharing of secrets

The secret value could be a key that is tied to {a bitcoin wallet, disk encryption, etc}

Recovering the key then requires cooperation!

Tiered key management:



Threshold = 1 out of 1

Threshold = 2 out of 3

Threshold = 2 out of 2

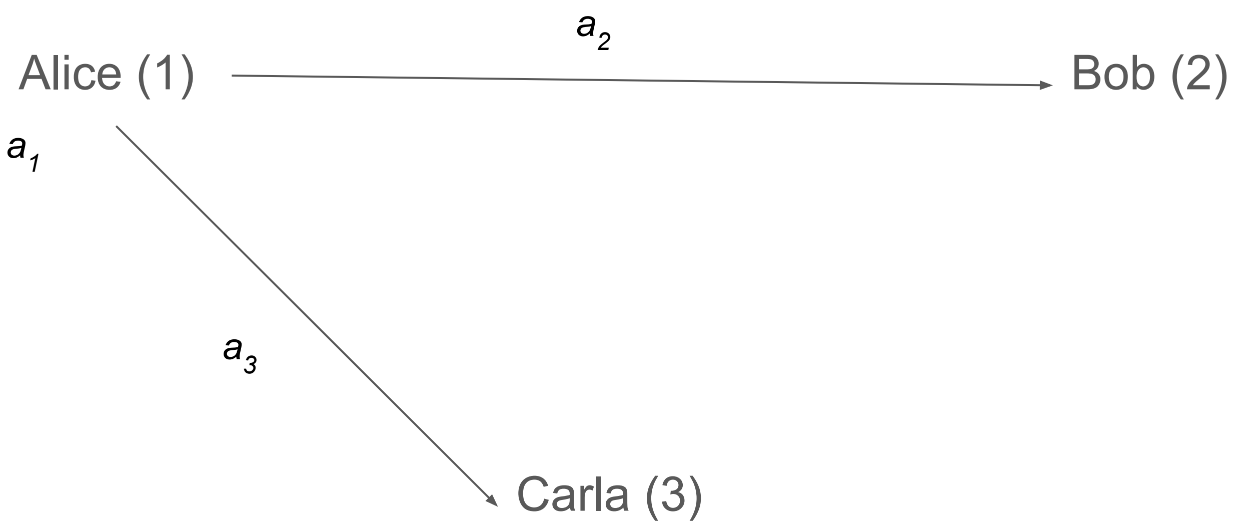
Secure Multiparty Computation (BGW)

Alice, Bob, and Carla all have a secret value.

They want to learn the output of some function on their secret inputs, but they **don’t** want the others to learn their input!

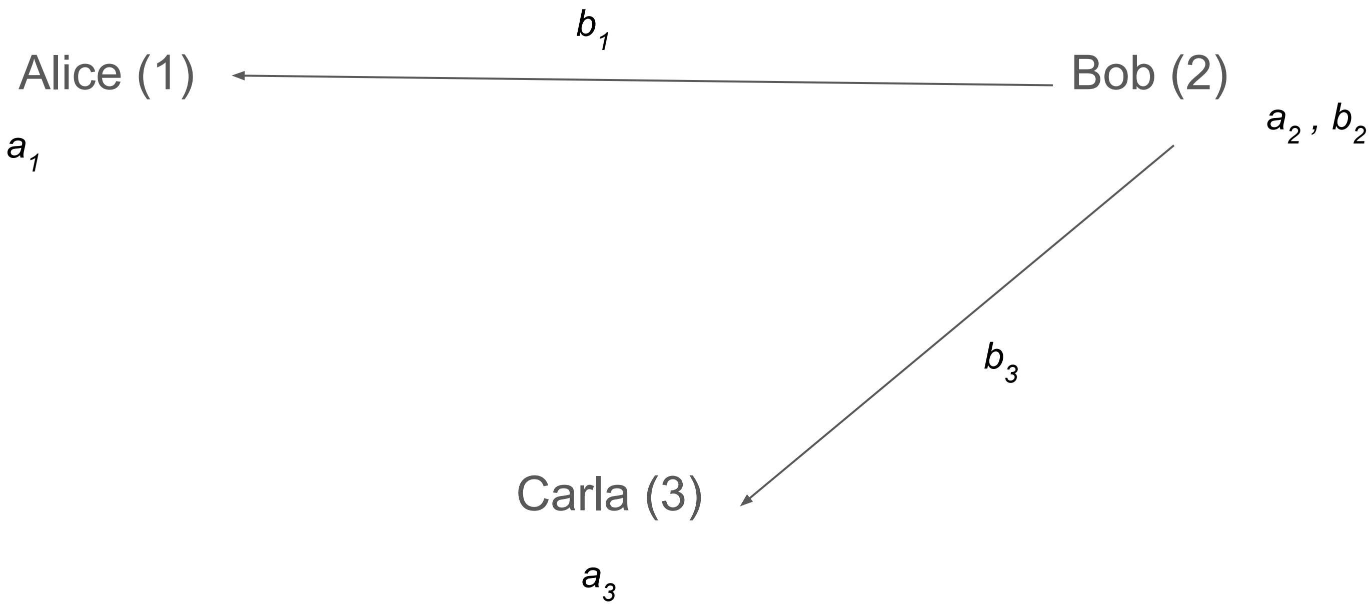
Canonical example of this is salary: several companies may want to learn what the average salary is for a role, but they want to keep their payroll information private.

Secure Multiparty Computation (yay)



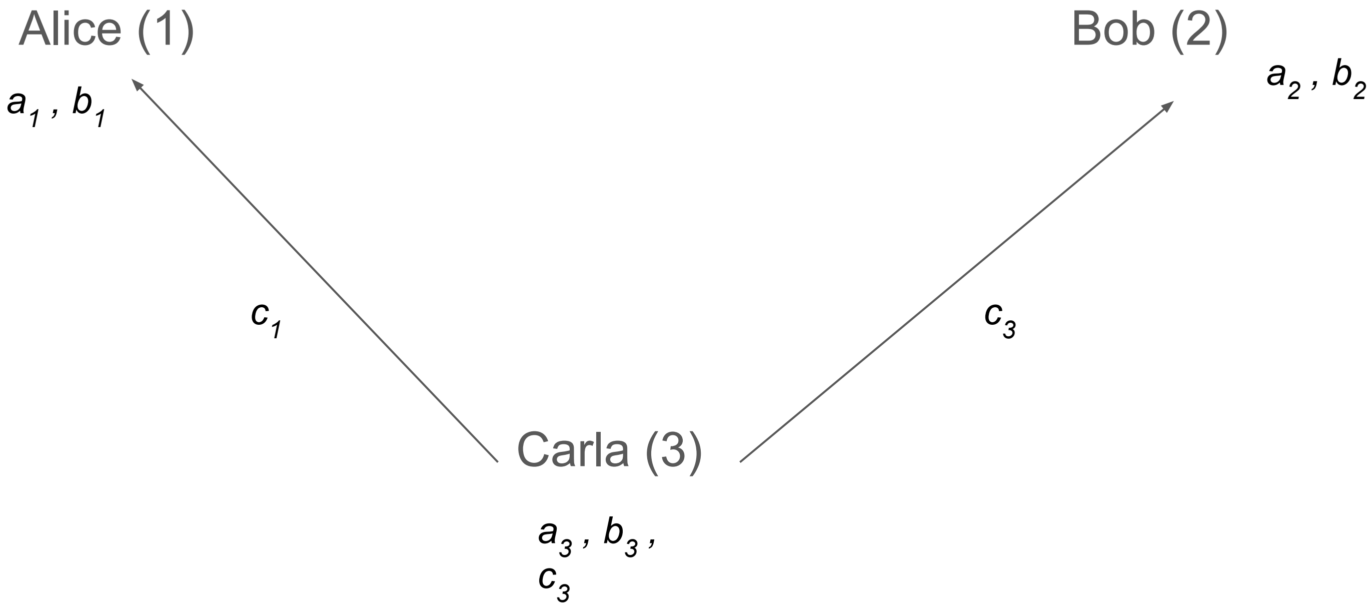
Each party sends Shamir Secret Shares of their input to the other parties

Secure Multiparty Computation (yay)



Each party sends Shamir Secret Shares of their input to the other parties

Secure Multiparty Computation (yay)



Each party sends Shamir Secret Shares of their input to the other parties

Secure Multiparty Computation (yay)

Alice (1)

a_1 , b_1 ,
c_1

Bob (2)

a_2 , b_2 , c_2

Now what?

Carla (3)

a_3 , b_3 ,
c_3

Secure Multiparty Computation (yay)

Alice (1)

a_1 , b_1 ,
c_1

Bob (2)

a_2 , b_2 , c_2

They want to compute
f(a,b,c) = avg(a,b,c) =
sum(a,b,c)/3

Carla (3)

a_3 , b_3 ,
c_3

Secure Multiparty Computation (yay)

Alice (1)

a_1 , b_1 ,
c_1

Bob (2)

a_2 , b_2 , c_2

How to compute
a+b+c?

Carla (3)

a_3 , b_3 ,
c_3

Secure Multiparty Computation (yay)

Alice (1)

$sum(a,b,c)_1 = a_1 + b_1 + c_1$

Bob (2)

$sum(a,b,c)_2 = a_2 + b_2 + c_2$

Add shares locally!

Carla (3)

$sum(a,b,c)_3 = a_3 + b_3 + c_3$

Secure Multiparty Computation (yay)

Alice's secret is A(0) where A() = $\alpha_0 x^0 + \alpha_1 x^1 + \alpha_2 x^2$ (α_1 and α_2 are random! α_0 = a)
(need 3 points to define a degree 2 polynomial: Alice, Bob, and Carla each have one point

Secure Multiparty Computation (yay)

Alice's secret is A(0) where A() = $\alpha_0 x^0 + \alpha_1 x^1 + \alpha_2 x^2$ (α_1 and α_2 are random! α_0 = a)
(need 3 points to define a degree 2 polynomial: Alice, Bob, and Carla each have one point

B() = $\beta_0 x^0 + \beta_1 x^1 + \beta_2 x^2$ (β_1 and β_2 are random! β_0 = b)

C() = $\gamma_0 x^0 + \gamma_1 x^1 + \gamma_2 x^2$ (γ_1 and γ_2 are random! γ_0 = b)

Secure Multiparty Computation (yay)

Alice's secret is A(0) where A() = $\alpha_0 x^0 + \alpha_1 x^1 + \alpha_2 x^2$ (α_1 and α_2 are random! α_0 = a)
(need 3 points to define a degree 2 polynomial: Alice, Bob, and Carla each have one point

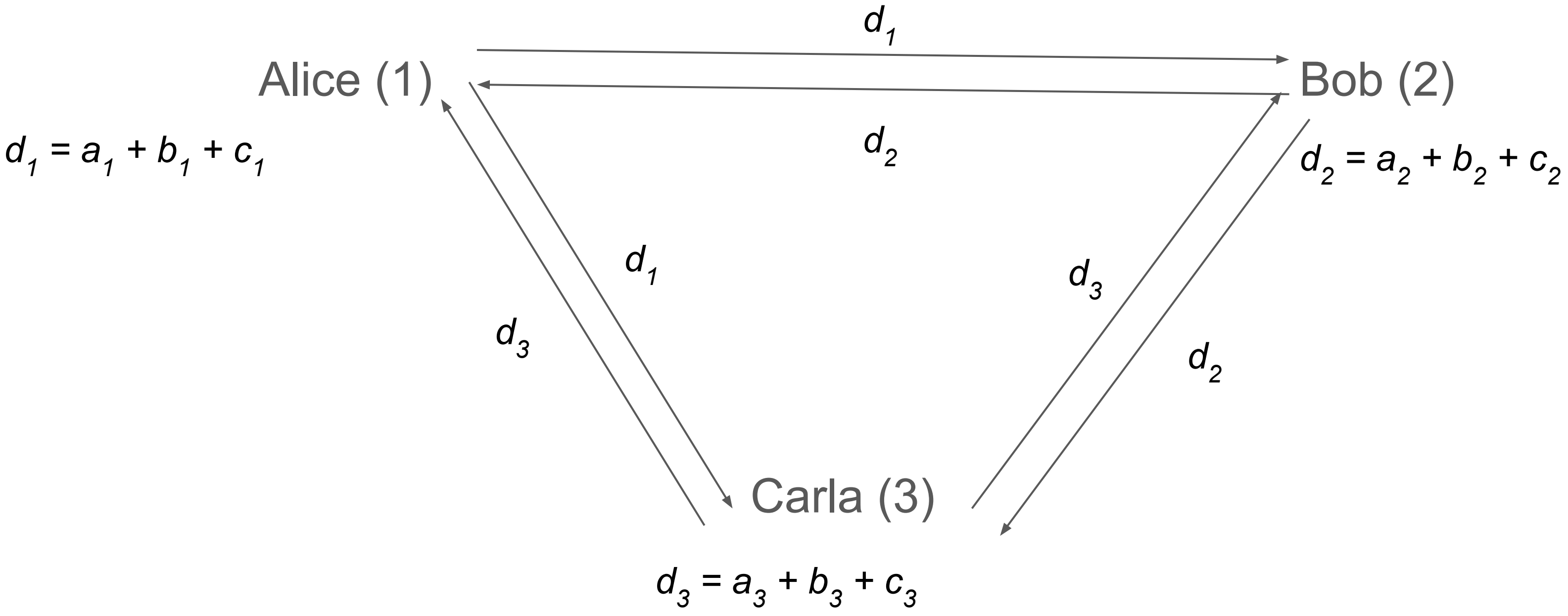
B() = $\beta_0 x^0 + \beta_1 x^1 + \beta_2 x^2$ (β_1 and β_2 are random! β_0 = b)

C() = $\gamma_0 x^0 + \gamma_1 x^1 + \gamma_2 x^2$ (γ_1 and γ_2 are random! γ_0 = b)

A() + B() + C() = D() where D(0) = A(0) + B(0) + C(0) (!!)

When Alice, Bob, and Carla add their shares locally they obtain a share (a point) of D() that can later be interpolated to learn D() and evaluate D(0) to learn the sum of their secret inputs!

Secure Multiparty Computation (yay)



Secure Multiparty Computation (yay)

Alice (1)

Recover: $d = a + b + c$
Divide d by 3 to get average salary

Bob (2)

Recover: $d = a + b + c$
Divide d by 3 to get average salary

Carla (3)

Recover: $d = a + b + c$
Divide d by 3 to get average salary

MPC security model

Alice, Bob, and Carla learn *nothing* from participating in the protocol that could not have also been learned from only their own input and the protocol output.

MPC security model

Alice, Bob, and Carla learn *nothing* from participating in the protocol that could not have also been learned from only their own input and the protocol output.

This is why the computation for average salary outputted the sum of salaries, not the average!

If each participant knows the average salary and the number of participants, they can easily compute the sum of salaries.

MPC security model

Alice, Bob, and Carla learn *nothing* from participating in the protocol that could not have also been learned from only their own input and the protocol output.

This is why the computation for average salary outputted the sum of salaries, not the average!

If each participant knows the average salary and the number of participants, they can easily compute the sum of salaries.

No point in computing a sum in 2pc (you'd learn the other party's input!)

Multiplication in MPC

Multiplication by a constant *c* can be done locally the same way as addition: multiply the shares α_n by *c* to get a share of a polynomial D() where D(0) = $c\alpha_0$

Multiplication of secrets is harder.

Multiplication in MPC

Multiplication by a constant *c* can be done locally the same way as addition: multiply the shares α_n by *c* to get a share of a polynomial D() where D(0) = $c\alpha_0$

Multiplication of secrets is harder.

Firstly, this will raise the degree of the polynomial!

A() = $\alpha_0 x^0 + \alpha_1 x^1 + \alpha_2 x^2$
B() = $\beta_0 x^0 + \beta_1 x^1 + \beta_2 x^2$
A() * B() will have degree 4! You don't have enough shares to recover that…

Multiplication in MPC

Multiplication by a constant *c* can be done locally the same way as addition: multiply the shares α_n by *c* to get a share of a polynomial D() where D(0) = $c\alpha_0$

Multiplication of secrets is harder.

Firstly, this will raise the degree of the polynomial!

A() = $\alpha_0 x^0 + \alpha_1 x^1 + \alpha_2 x^2$
B() = $\beta_0 x^0 + \beta_1 x^1 + \beta_2 x^2$
A() * B() will have degree 4! You don't have enough shares to recover that…

Also, A() * B() will not necessarily be a random polynomial.

Degree reduction and rerandomization is an interactive process that requires communication between Alice, Bob, and Carla.