601.445/601.645
Practical Cryptographic Systems

MPC and Private Browsing

Instructor: Alishah Chator

Housekeeping

- New (last!) assignment coming this week
- Will include written and programming portions
- Project Presentations coming up

News?

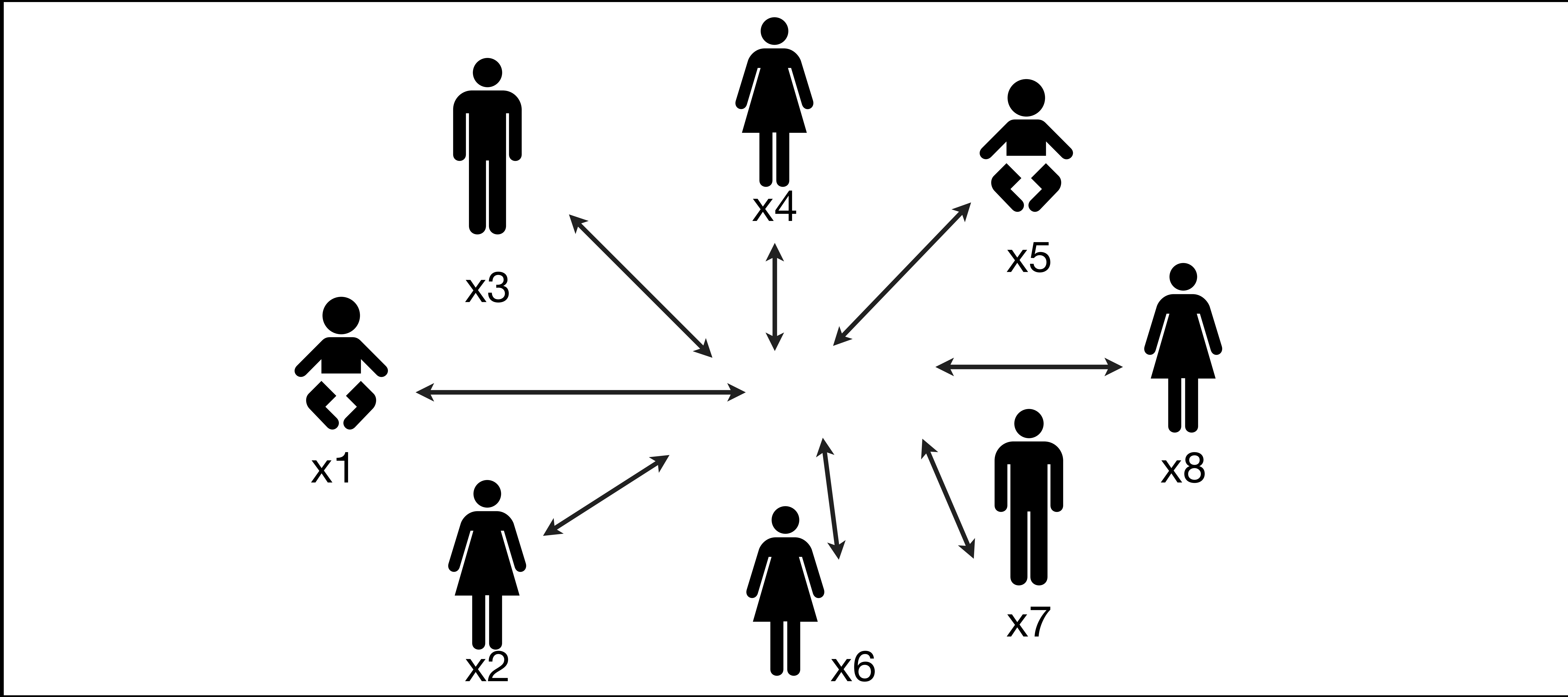
Review

- Today:
  - Finishing up Secret sharing
  - If time: TOR and private browsing

MPC

- Three basic properties we want to achieve
  - Correctness: the output of the function is actually what it should be
  - Privacy: nobody learns anything about <u>honest parties'</u> inputs (other than what they would learn from the function output)
  - Guaranteed output delivery (no dishonest party can prevent honest parties from getting outputs)

Adversarial model



Secret sharing

- Problem:
  - Take a given secret *s* and break it into *N* different pieces (“shares”)
  - Want to recover the original secret from any *M* of the shares, M <= N
  - What is the security goal?

Secret sharing

- Two algorithms:
  - Share(N, M, s): outputs (t1, …., tN)
  - Recover(N, M, t_i1, …, t_iM): outputs s’

Secret sharing

- Correctness?
- Security definition:
  - (Informal) Given any subset of M-1 shares, no adversary learns <u>any information</u> about *s* (other than its size)
  - Alternative definition: Given a set of M-1 shares of *s*, and a set of M-1 shares of some *random value s',* no adversary can tell the difference
  - (E.g., there is no detectable difference between the shares of s and a random element of the same length.)

How do we build secret sharing

- Let’s try to build 2-out-of-2 secret sharing
- We have a bitstring *s*, and wish to compute t1, t2 such that:
  - Neither t1 or t2 (by itself) reveals anything about *s* (other than length)
  - Given both t1, t2 we can recover *s*

How do we build secret sharing

- Let’s try to build 2-out-of-2 secret sharing
- We have a bitstring *s,* and wish to compute t1, t2 such that:
  - Neither t1 or t2 (by itself) reveals anything about *s* (other than length)
  - Given both t1, t2 we can recover *s*
- Solution (share algorithm):
  - Pick a random string t1 such that |t1| = |s|
  - Set t2 = s XOR t1

How do we build secret sharing

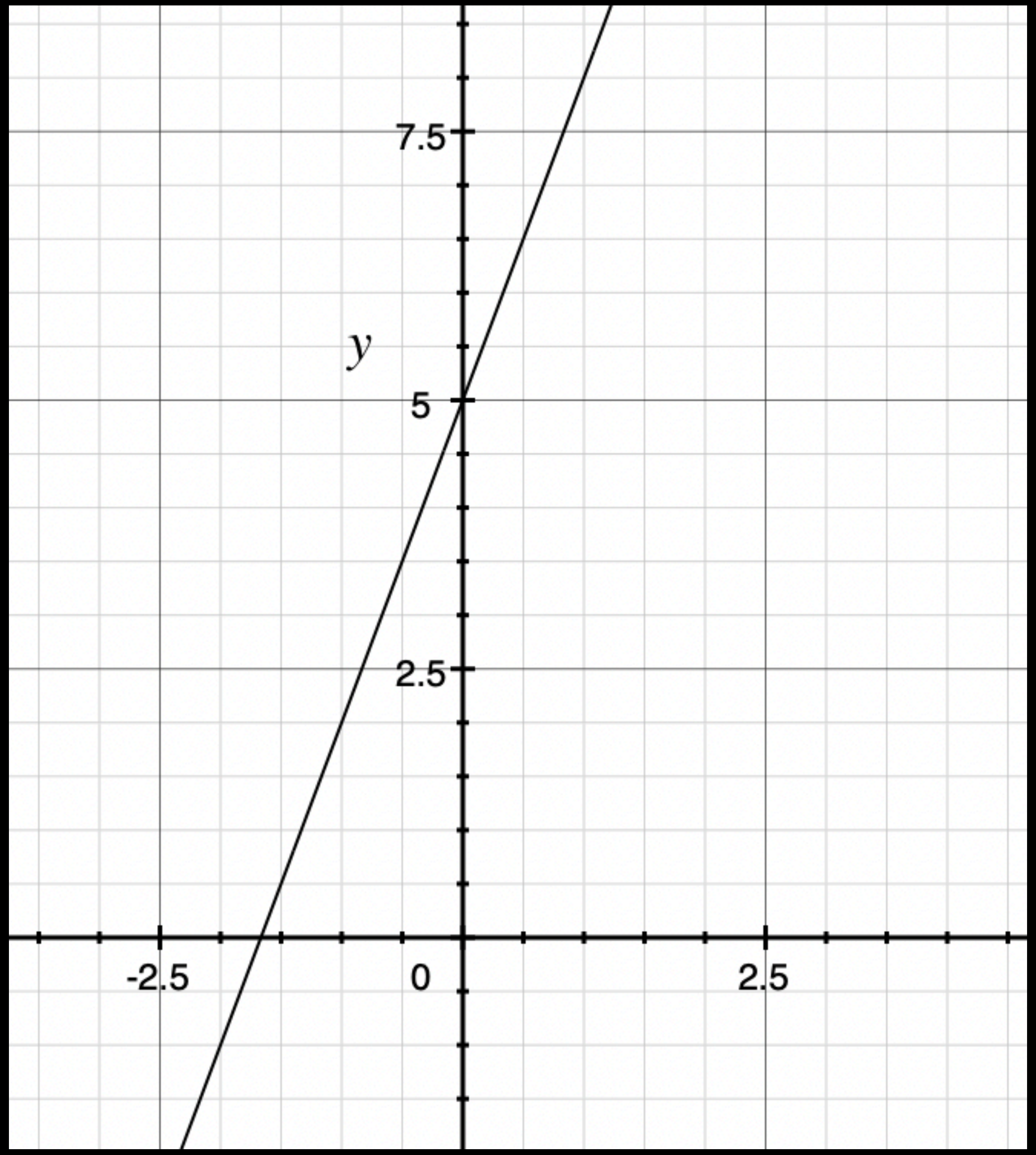
- Let’s try to build 2-out-of-3 secret sharing using the same technique

General secret sharing

- What are the downsides of the XOR approach?
- Can we build a more efficient, general-purpose approach?

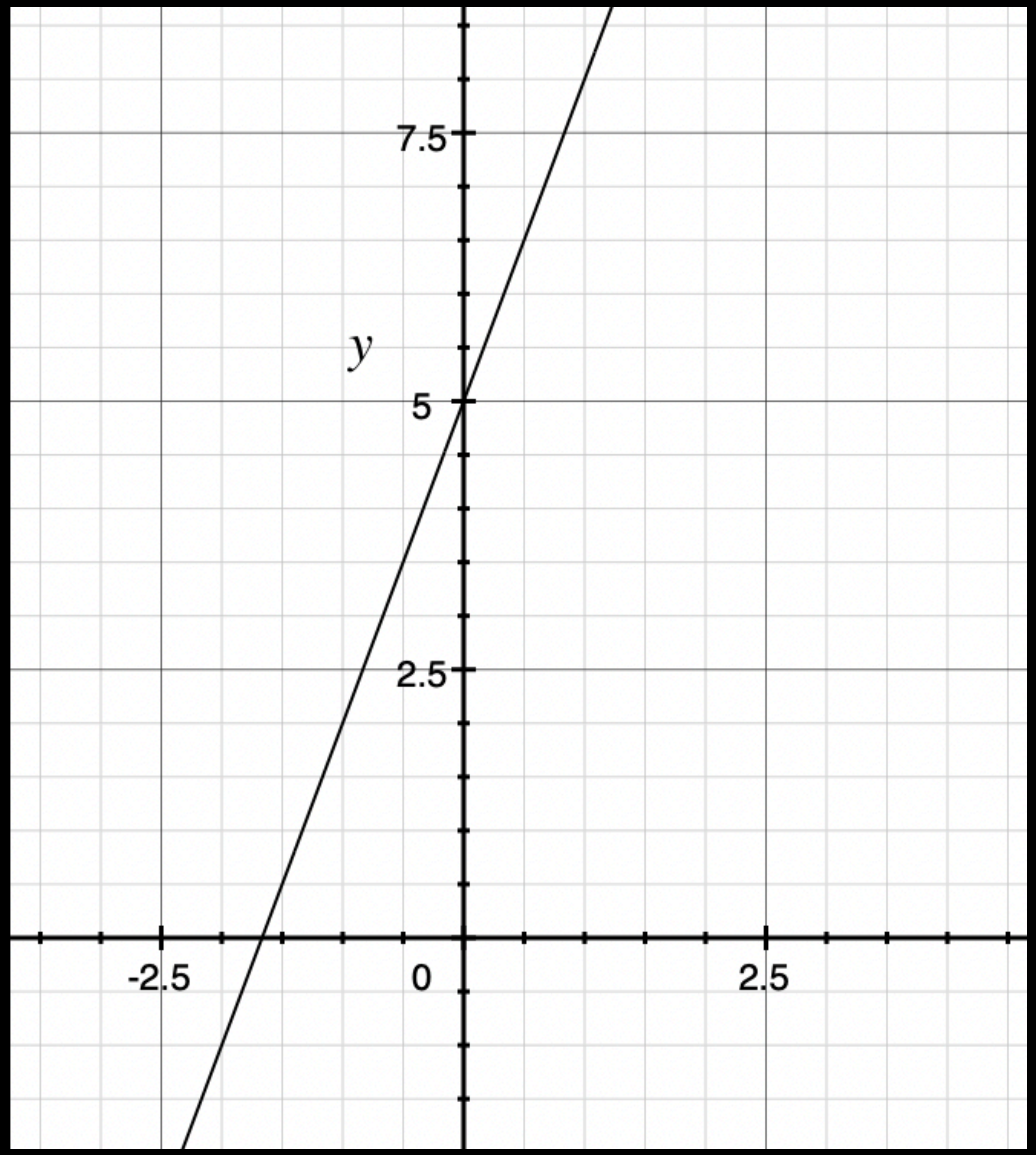
(Simple) linear secret sharing

- Let y=*mx*+*b* be the equation of a line
- Imagine I give you a point (x, y) for x != 0
- What can we learn about *b*?



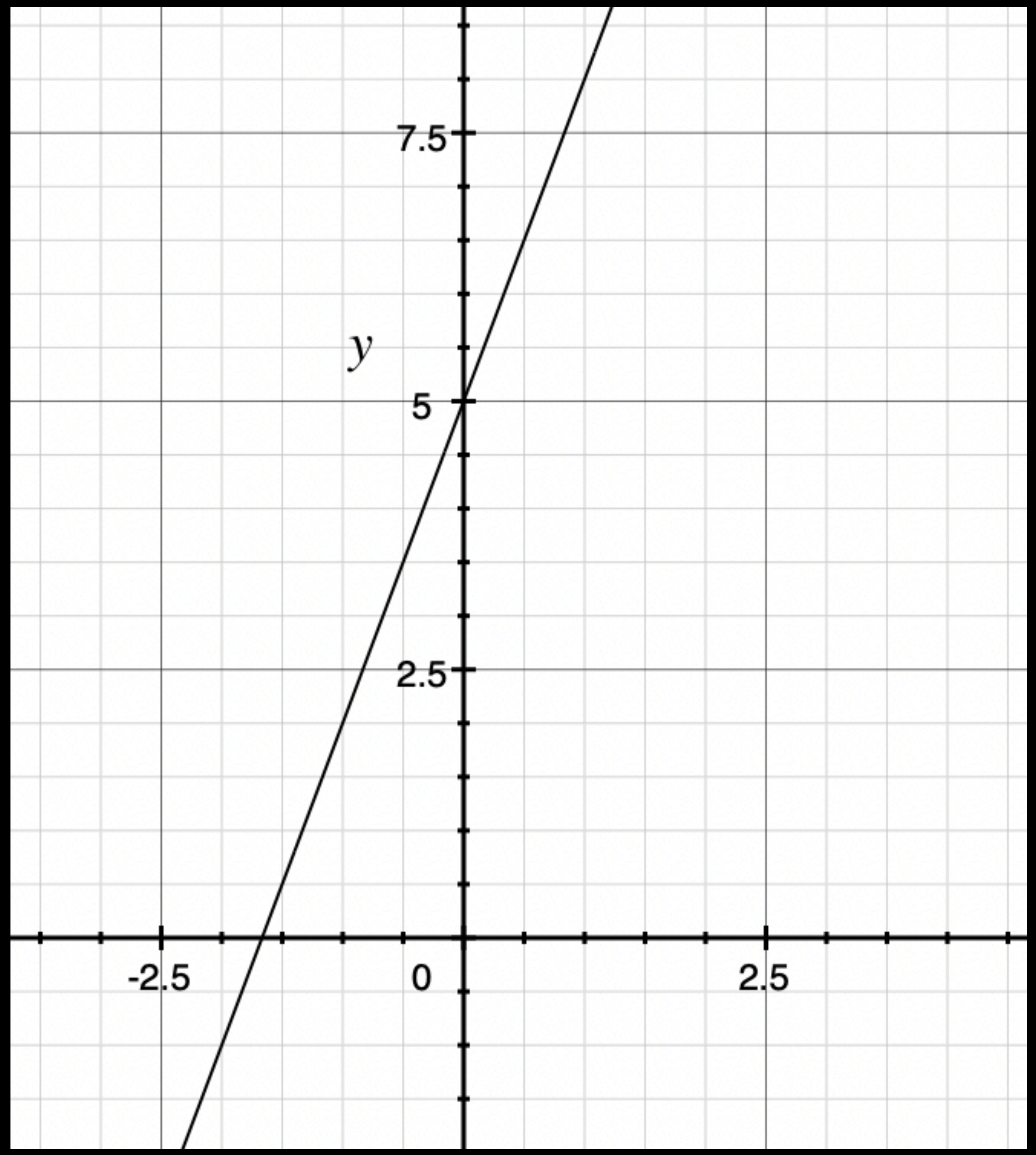
(Simple) linear secret sharing

- Let y=*mx*+*b* be the equation of a line
- Imagine I give you a point (x, y) for x != 0
- What can we learn about *b*?
  - For every b, (x, y) there exists a line that passes through (0, b)



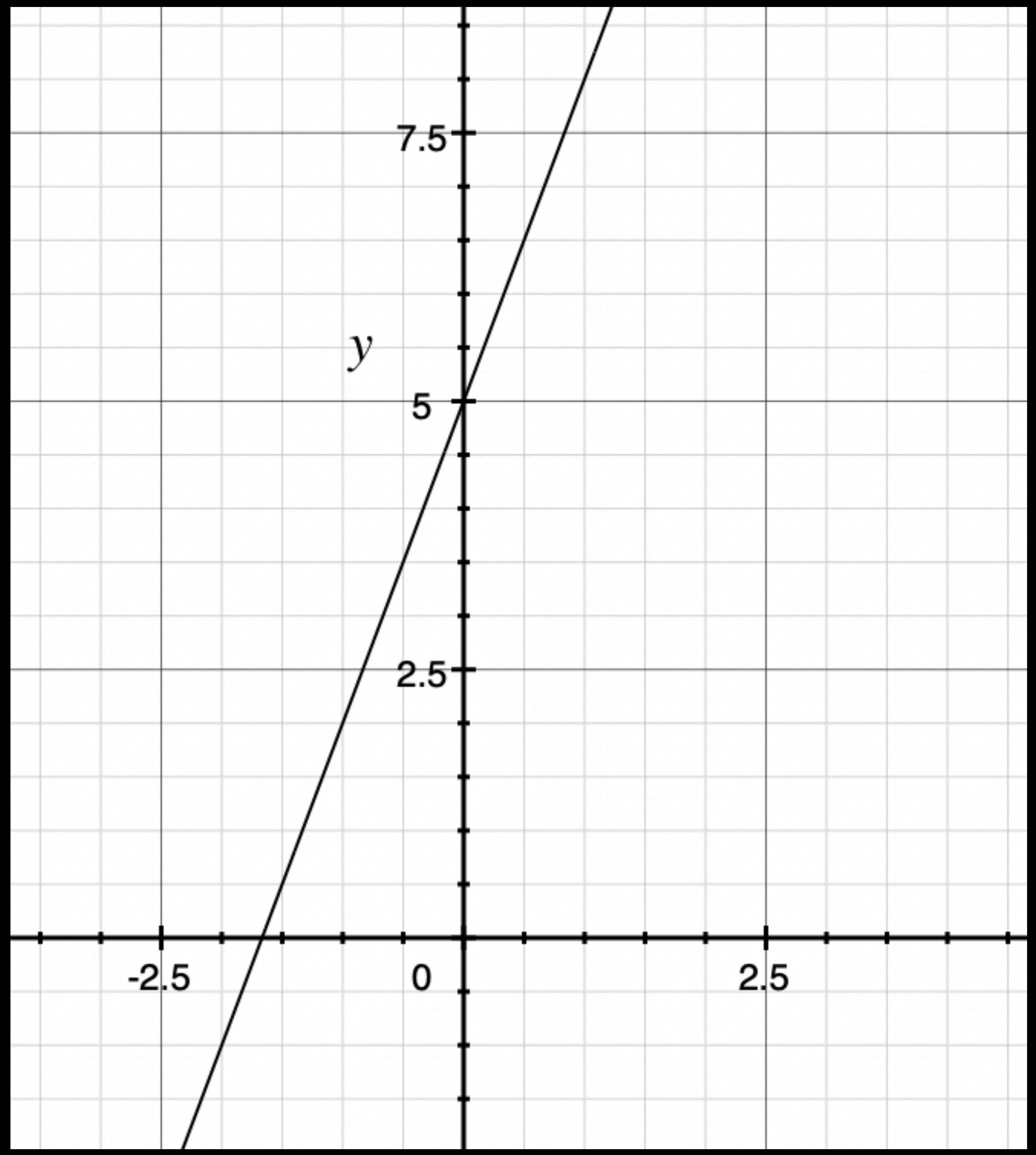
(Simple) linear secret sharing

- Let y=*mx*+*b* be the equation of a line
- Imagine I give you two distinct points (x2, y2), (x1, y1)
- What can we learn about *b*?



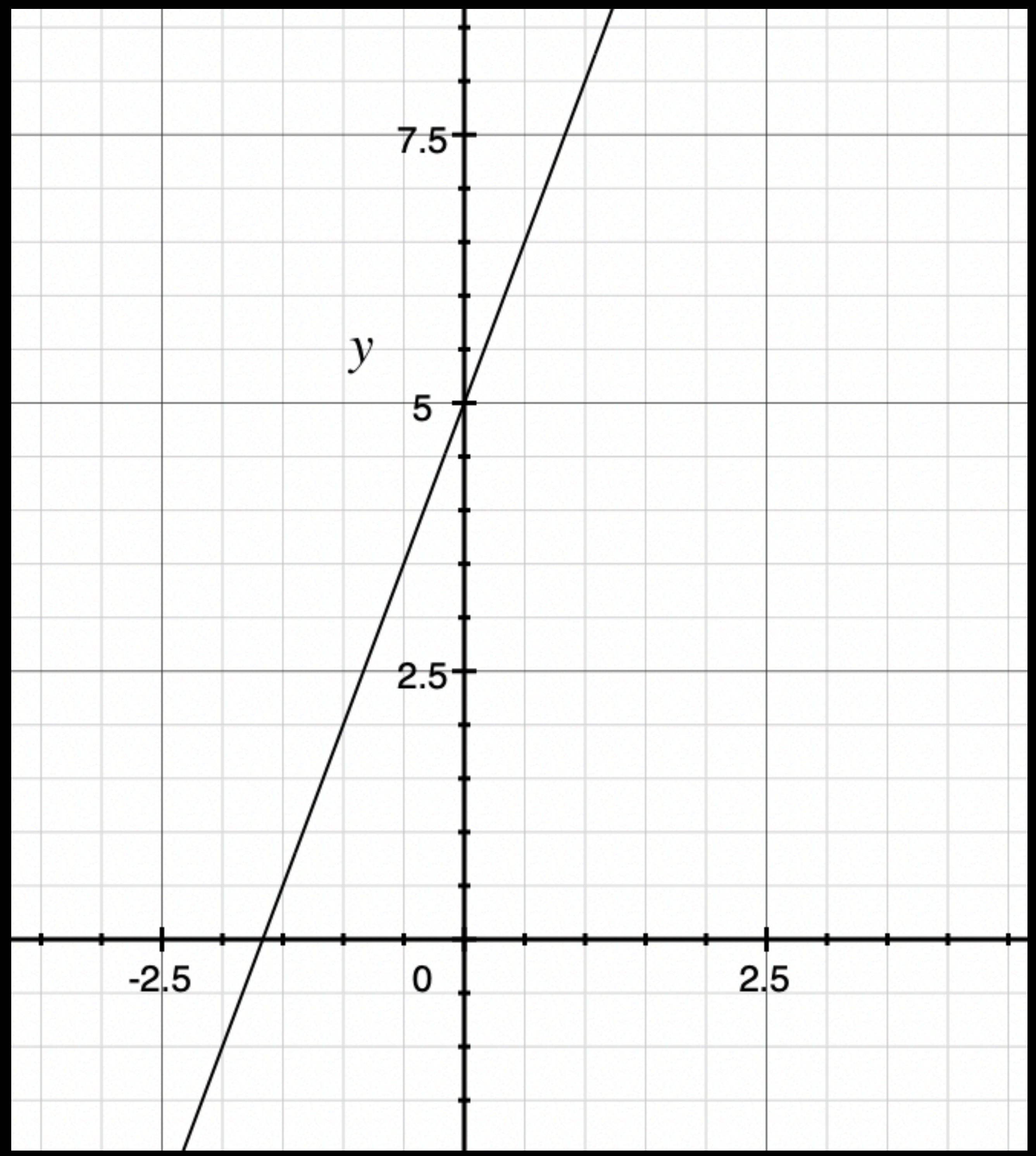
(Simple) linear secret sharing

- Further optimization:
  instead of computing over the real numbers,
  let's compute over the field Zp
- Let y=*mx+b mod p* be the equation of a line
- Same questions



(Simple) linear secret sharing

- This allows us to compute a 2-of-N secret sharing
- Fix some Zp (for largish p)
- Pick a line with constant term (y-intercept) set to *s* and a random coefficient (slope) *m*
- For x=1 to N, output shares:
  - t_i = (x, mx+s) mod p
- Recovery is just linear interpolation



Can we generalize this to $M>2$?

Can we generalize this to M>2?

- Shamir’s observations:
  - Any degree-(M-1) polynomial can be uniquely interpolated given M distinct points (using Lagrangian interpolation)
  - Given only M-1 points (or fewer) the polynomial is not constrained

Can we generalize this to M>2?

- Share(M, N, s):
  - Fix Zp
  - Sample coefficients (a1, …, a_{M-1}), and set P(x) to the polynomial defined by these coefficients, with constant term s
  - Compute shares: (1, P(1)), (2, P(2)), …, (N, P(N))

Other nice facts about secret sharing

- Polynomials can be added easily
  - Given two (random) polynomials F(), G() with constant terms s1, s2
  - The sum of F() + G() has constant term s1+s2
  - Similarly, adding together a vector of secret shares for secrets s1, s2 (respectively) will produce a set of shares for (s1 + s2)

Other nice facts about secret sharing

- Polynomials can be added easily
  - Given two (random) polynomials F(), G() with constant terms s1, s2
  - The sum of F() + G() has constant term s1+s2
  - Similarly, adding together a vector of secret shares for secrets s1, s2 (respectively) will produce a set of shares for (s1 + s2)
  - Better yet, if F() and G() are random polynomials, then their sum will also be a random polynomial

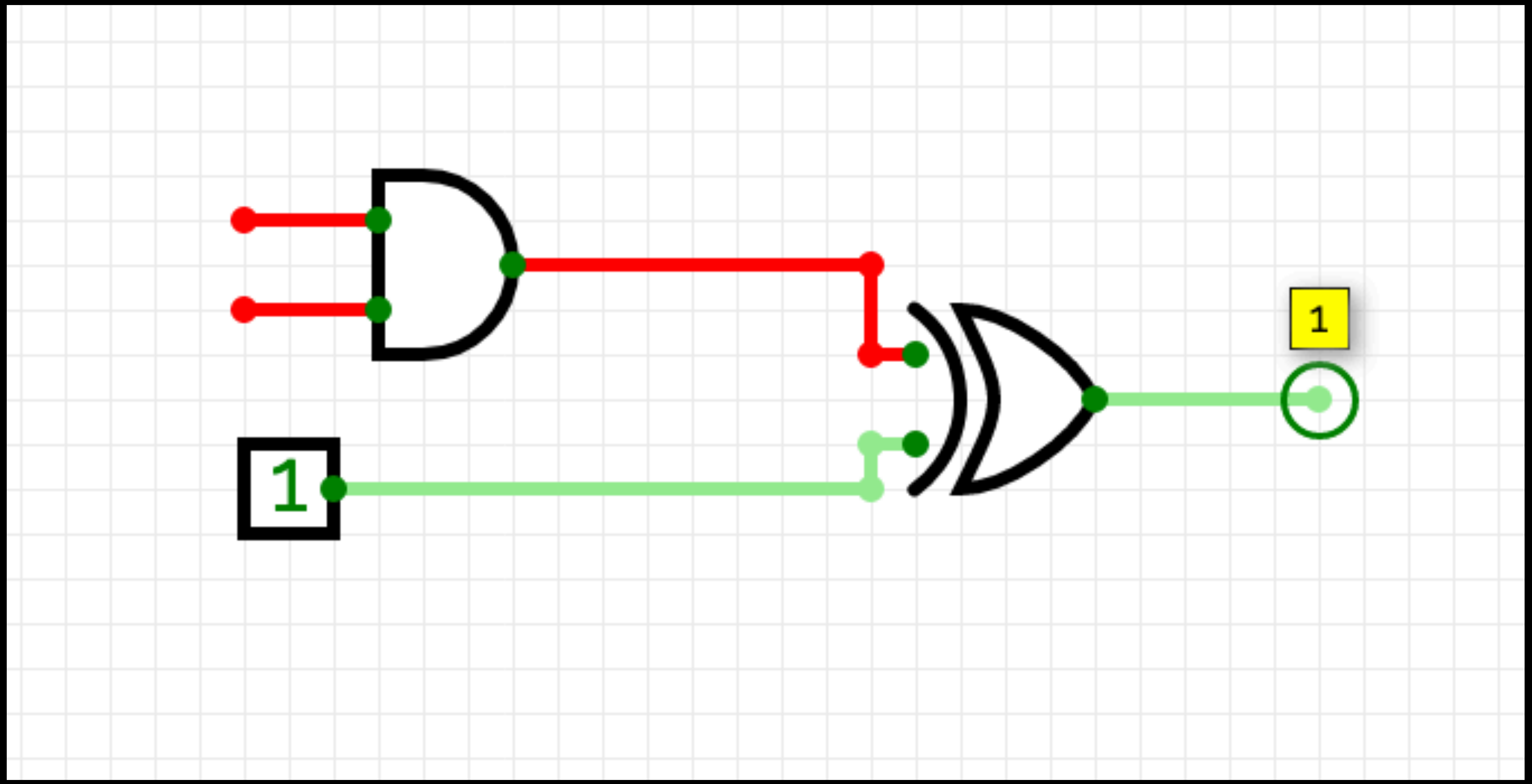
Can we multiply secret shares?

- Not quite as elegantly
  - If we multiply two polynomials of degree d, we get a polynomial of degree 2d. Also it’s not random anymore.
  - This also prevents us from just multiplying shares
  - However, there are *interactive* protocols for multiplying secret shares, then reducing the degree of the resulting polynomial

Recall: Universality

- Some combinations of gates are **Universal**
    - These gates can be combined to build any function
- Examples are {NAND}, {AND,OR,NOT}
- Often in cryptography we are limited to {AND, XOR}

NAND from AND + XOR

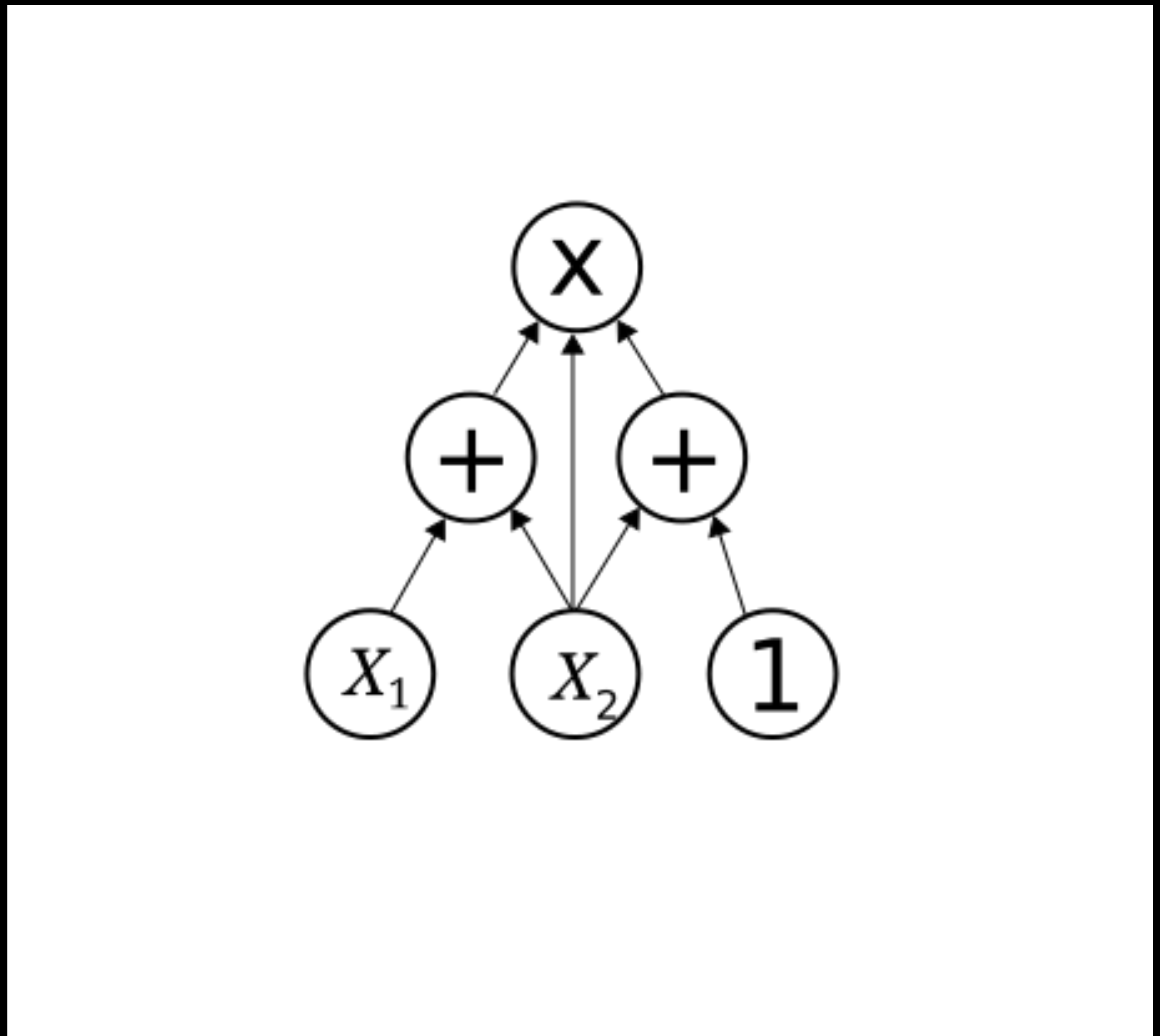


Multi-Party Secure Computation from circuits?

- We can build any function from {AND, XOR}, how does this help us?

Arithmetic Circuits

- Model for complexity of computing polynomials
- Inputs are either variables or fixed field elements
- Gates are Addition or Multiplication



$$x_2(x_1 + x_2)(x_2 + 1)$$

Multi-Party Secure Computation from circuits?

- We can build any function from {AND, XOR}, how does this help us?
- In arithmetic circuits, Multiplication and Addition correspond to AND and XOR
  - Think about the mod 2 case

Multi-Party Secure Computation from circuits?

- We can build any function from {AND, XOR}, how does this help us?
- In arithmetic circuits, Multiplication and Addition correspond to AND and XOR
  - Think about the mod 2 case
- This means {Multiplication, Addition} is universal as well!

Multi-Party Secure Computation from circuits?

- We can build any function from {AND, XOR}, how does this help us?
- In arithmetic circuits, Multiplication and Addition correspond to AND and XOR
    - Think about the mod 2 case
- This means {Multiplication, Addition} is universal as well!
- So to recap:
    - Arithmetic Circuits are universal
    - Arithmetic Circuits model polynomials
    - Polynomials are homomorphic
    - We can secret share using polynomials

BGW-Style MPC Overview

- Represent function as a arithmetic circuit

BGW-Style MPC Overview

- Represent function as a arithmetic circuit
- Secret share the inputs

BGW-Style MPC Overview

- Represent function as a arithmetic circuit
- Secret share the inputs
- Evaluate the circuit gate by gate taking advantage of homomorphic secret sharing

BGW-Style MPC Overview

- Represent function as a arithmetic circuit
- Secret share the inputs
- Evaluate the circuit gate by gate taking advantage of homomorphic secret sharing
- Reconstruct output