

CS 798

# Privacy in Computation and Communication

Module 6

Privacy in Communication: Protecting Metadata

Spring 2024

# Internet communication

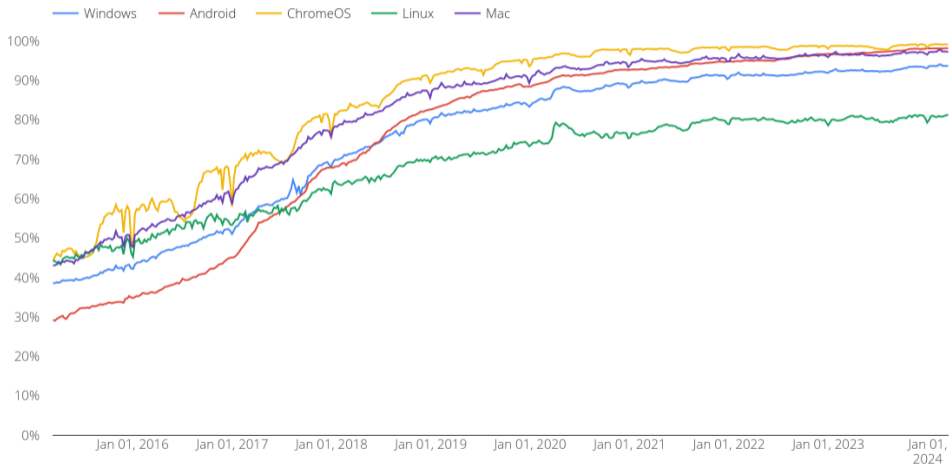
- Recall from Module 2 how information gets transmitted over the Internet
- Data streams are broken into *packets*
  - Headers (e.g., source and destination IP addresses)
  - Body (may or may not be encrypted)
- Each packet traverses a number of ASes to get from its source to its destination
  - BGP sets the path to use, but BGP can be subverted
  - Each AS can potentially view, copy, modify, or drop the packets that go through it

# Encryption on the Internet

- When the Internet was designed, the main concern was reliability
  - The data should get transmitted successfully
- There was very little in the way of security or privacy at the beginning
- When the Internet “exploded” in the mid-1990s, we started to see development and deployment of encrypted protocols
  - Primarily for financial transactions at first
  - Then a wider range of communications
- Big change in the mid-2010s

# Encryption on the Internet

Percentage of pages loaded over HTTPS in Chrome by platform



<https://transparencyreport.google.com/https/overview?hl=en>

# Encryption on the Internet

- You certainly use encryption on the Internet
  - You're using it right now!
- You may also be familiar with “end-to-end” encrypted *messaging* apps
  - e.g., iMessage, WhatsApp, Signal
  - Even the servers facilitating the communication cannot read **the contents of** the messages

# Metadata

- Protecting data with encryption is necessary for security and privacy, but not sufficient
- It does not protect:
  - Who is talking to whom
  - With what frequency
  - At what times
  - With what volume
  - Is a particular user communicating *at all*?

# Protecting metadata

- Why is protecting metadata important?
- “We kill people based on metadata”  
— Gen. Michael Hayden, former director of NSA and CIA
- Just knowing who your friends are gives a lot of information about you
  - Subcultures
  - Political leanings
  - 2SLGBTQ+
  - This can be a big problem in some countries

# Protecting metadata from whom

- From whom might you want to protect the metadata?
- That is, what is the threat model? Who might be adversarial?
- ASes the network traffic passes through
  - Note that this could include governments
  - In the extreme, a *global adversary*
- The servers facilitating the communications
  - Even if they're compelled by law or force
- Sometimes even the person you're talking to
  - More limited in what you can protect



# Mix networks

- Early version proposed by David Chaum in a 1981 paper
- Deployed in the 1990s
- A network of *mix nodes* are run by independent parties
- Aims to defend against global adversaries who may also control or compromise some number (but not all) of the mix nodes
  - Distributed trust

# Mix networks

- A client picks a *path* through the mix network, consisting of some number  $k$  of mix nodes
  - The client can choose the number, but typically there's an upper bound
  - In some networks, the client can choose the nodes in the path freely; in others, there are restrictions; e.g., you have to choose the first node in the path from one set, the second node from another set, etc.
  - Let the nodes be  $N_1, N_2, \dots, N_k$
- The nodes have public encryption keys  $E_1, E_2, \dots, E_k$
- The client has a message  $M$  she wants to send to a recipient  $R$

## Mix networks

- The client takes  $(R, M)$ , and encrypts it with the encryption key of the *last* node in the path, to get  $E_k(R, M)$ , and attaches the id (or address)  $N_k$  of that node, to get  $(N_k, E_k(R, M))$
- The client encrypts that with the encryption key of the *previous* node in the path, to get  $E_{k-1}(N_k, E_k(R, M))$ , and attaches  $N_{k-1}$ , to get  $(N_{k-1}, E_{k-1}(N_k, E_k(R, M)))$
- And so on until the first node in the path, yielding  $E_1(N_2, E_2(N_3, E_3(\dots E_{k-1}(N_k, E_k(R, M)) \dots)))$
- The client sends this message to  $N_1$

# Mix networks

- When a node receives a message, it decrypts it:  
 $(N_2, E_2(N_3, E_3(\dots E_{k-1}(N_k, E_k(R, M)) \dots)))$
- This will be a pair where the first element is an address, and the second is a (usually encrypted) message
- The obvious thing to do next would be to send that message to that address
- Why is this insecure in the mix network's threat model?

# Mix networks

- Instead, the mix node waits to collect “a bunch” of incoming messages
  - For example, wait until receiving a certain number of them, or wait for a certain time, or a combination
- The mix node shuffles the messages into a random order and then sends them (or sometimes just some of them) to their destinations
- What advantage does this bring?

# Things to watch out for

- Early mix networks had several problems that limited the protection they could provide:
  - Message sizes chosen by clients
  - No defence against replay attacks
  - Latencies in the range of *hours*
  - Primarily for protecting email metadata

## Current mixnets

- Mix networks mostly went away for a while
- Recent renewed interest in the technology
- A number of recent designs
- At least one (Nym) being actively deployed

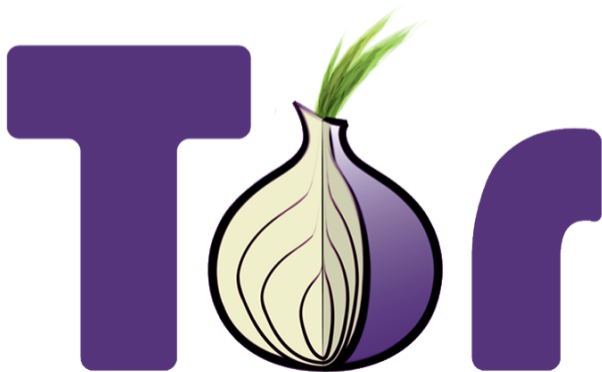
# Interactivity

- Latencies of hours are maybe barely OK for email
- When the WWW started becoming popular in the mid-to-late 1990s, you really didn't want to click on a link and wait hours for a response!
- A new technology for privacy for interactive communications (like WWW) was required



# Onion routing

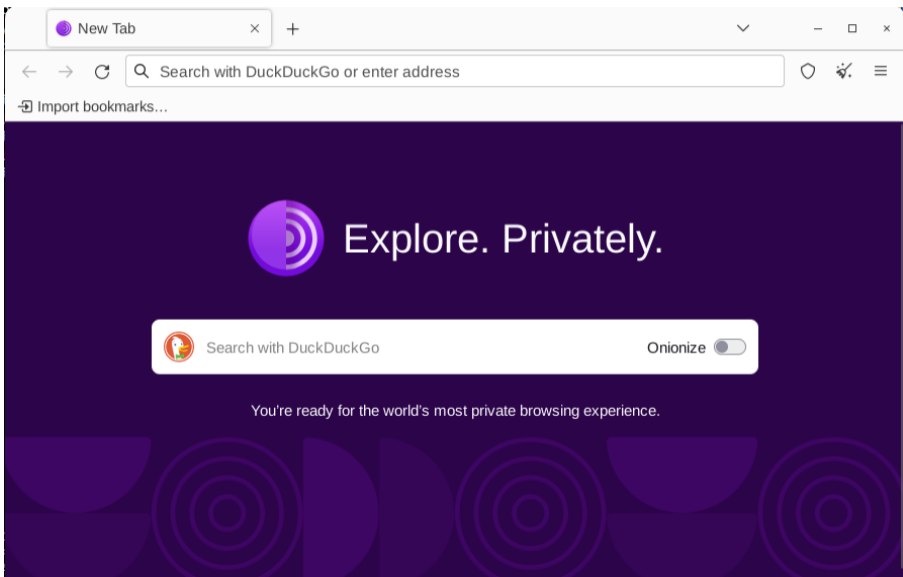
- This new technology was *onion routing* (Syverson, Goldschlag, and Reed, 1997), and its successor, *Tor* (Dingledine, Mathewson, Syverson, 2004)

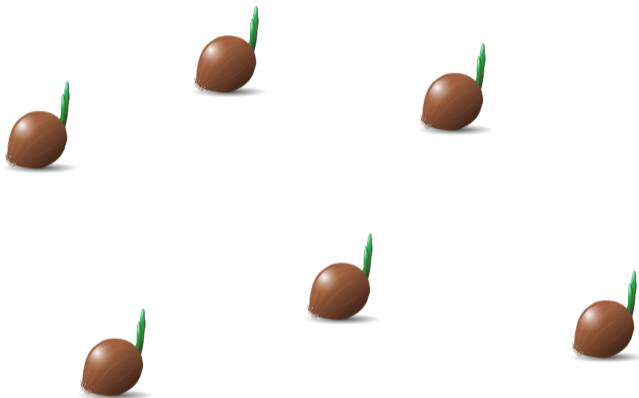


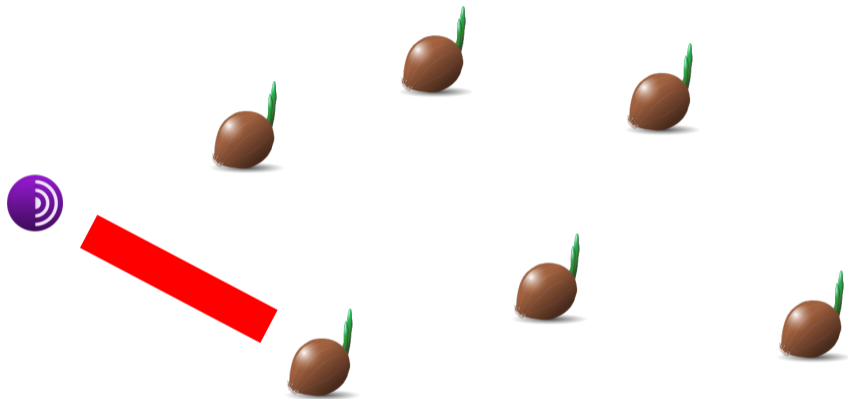
# Onion routing

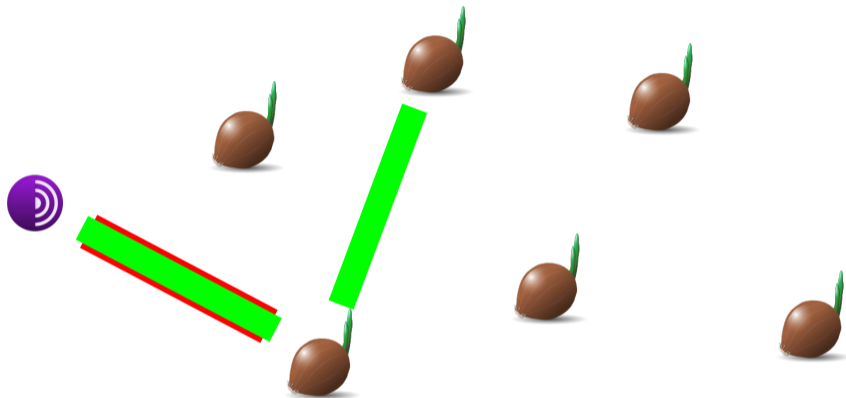
Two big differences from mix networks:

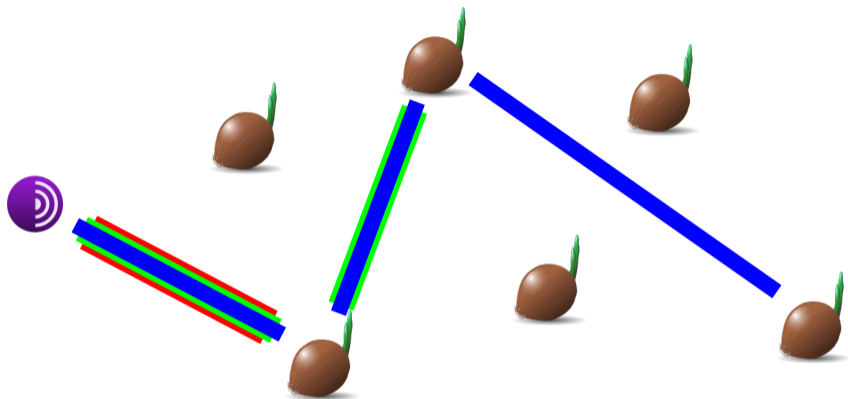
- No mixing (and so much lower delays)! Messages are processed as soon as they arrive
- In mix networks, you just send one message, and it comes out the other side a while later; public key operations are performed on each message
- In onion routing, you first have to set up a *circuit*, which performs public key operations
- Then you can send messages over that circuit, which only use (faster) symmetric key operations



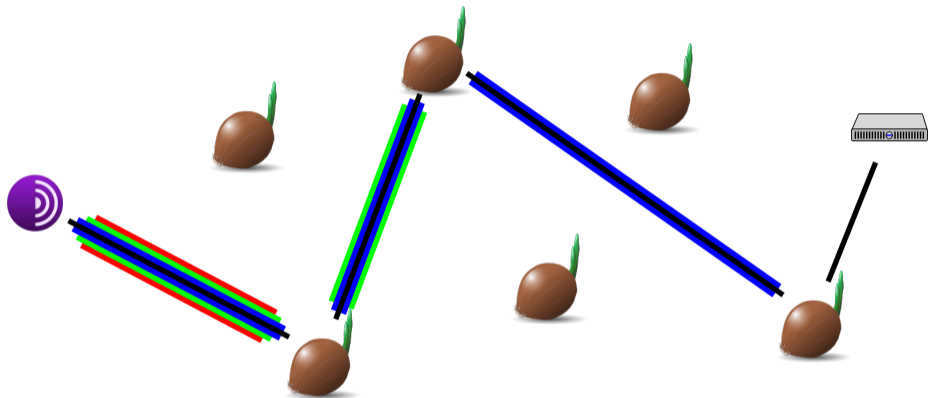






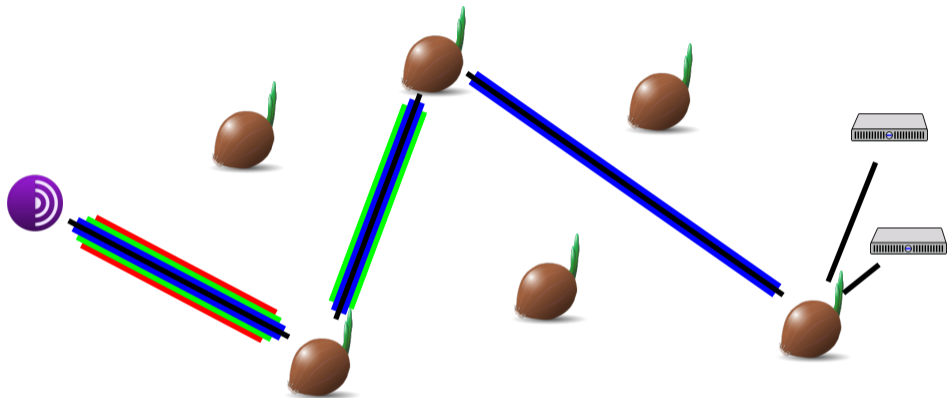


# Tor











# Tor



# Tor onion services

The New York Times - Br x +

← → ↻   <https://www.nytimesn7cgmftshazwhfgzm37qxb44r64ytbb2dj3x62d2lljsciyyd.onion> ☆   

 **The New York Times**

Friday, June 21, 2024

## Heat and Climate Extremes Are Hitting Billions

People all over the world are facing severe heat, floods and fire, aggravated by the use of fossil fuels. The year isn't halfway done.


4 MIN READ

---

## A Ride in a Chemical-Sniffing Van Shows How Heat Amps Up Pollution

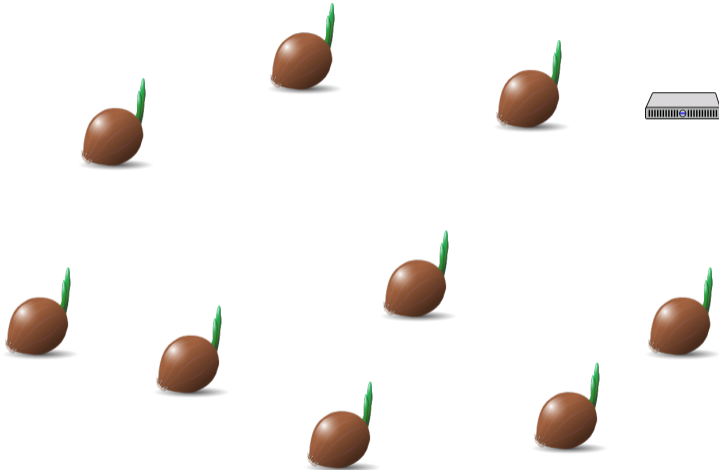
In heat waves, chemicals like formaldehyde and ozone can form more readily in the air, according to researchers driving mobile labs in New York City.

3 MIN READ

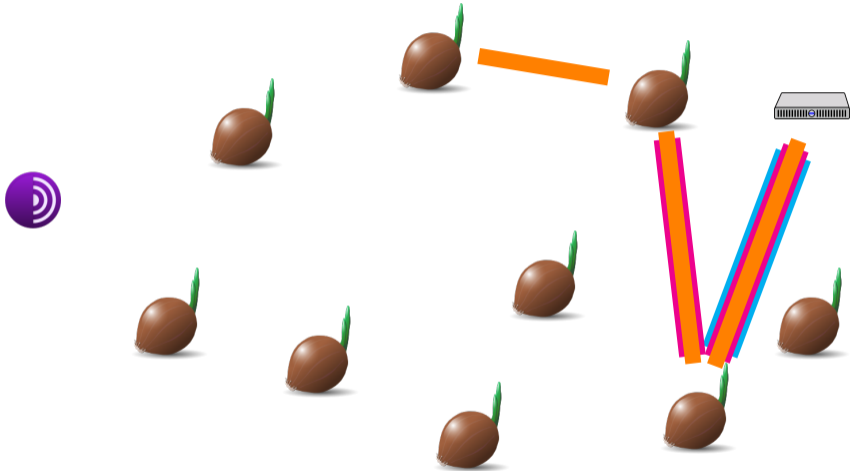


Fadel Senna/Agence France-Presse — Getty Images

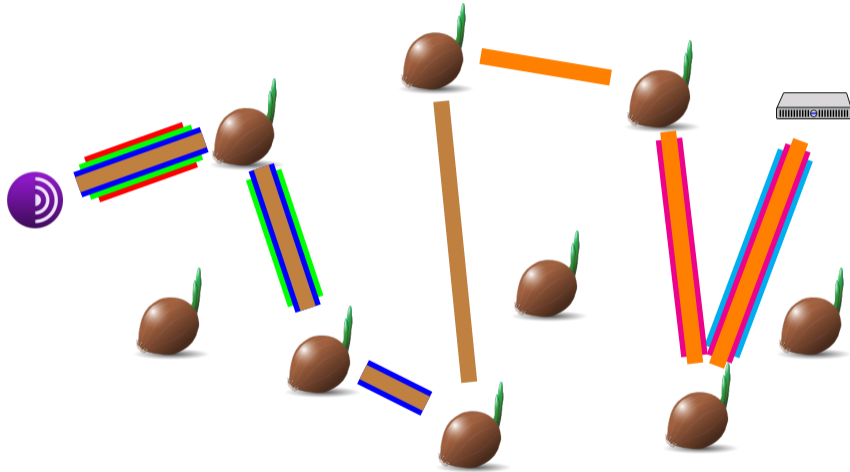
# Tor onion services



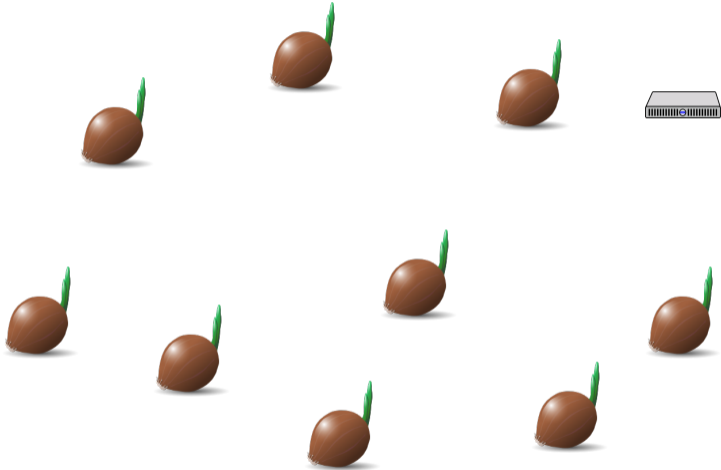
# Tor onion services



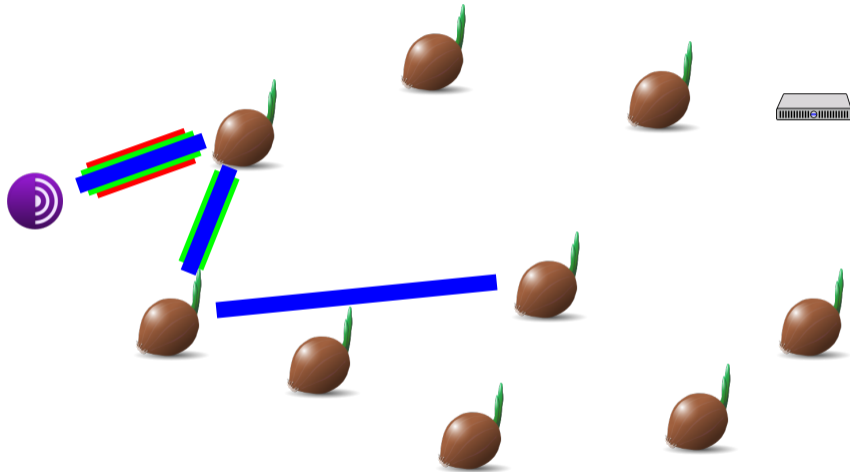
# Tor onion services



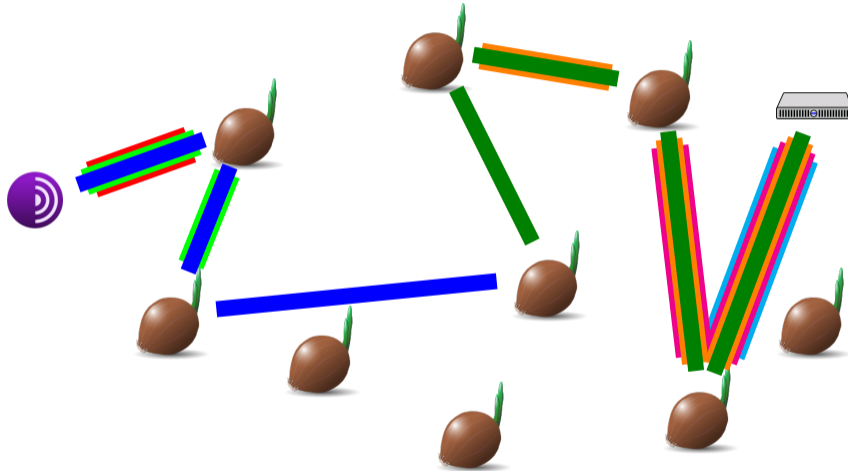
# Tor onion services



# Tor onion services

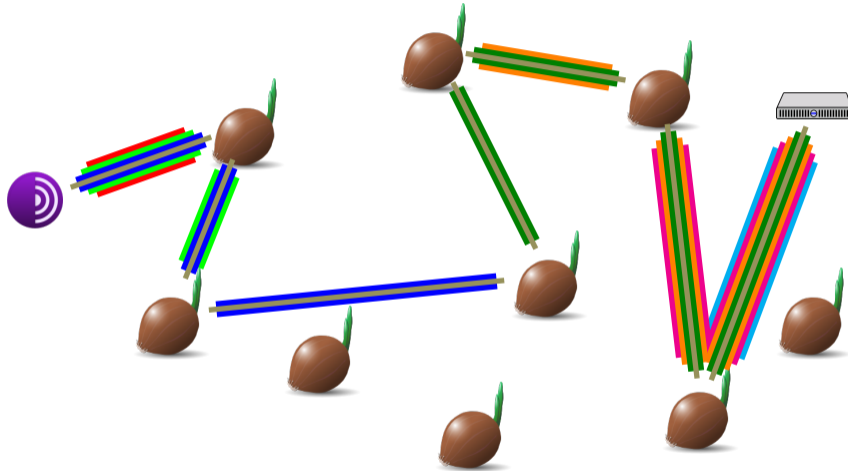


# Tor onion services





# Tor onion services



# Properties of Tor

- Tor is actually useful for interactive traffic
  - WWW, ssh are very common
  - But (at least currently) TCP only, not UDP, so some audio/video protocols, some games, etc., do not work easily
- Millions of people use Tor every day
- What does Tor protect, and from whom?
  - That is, what is Tor's threat model?

# The threat model of Tor

Who might the adversaries be?

- The server you're talking to (if you're the client)
  - Or the client you're talking to (if you're an onion service)
- Some (but not too many) of the Tor nodes
- Some ASes
- Global adversary?
  - Probably not

# What Tor tries to protect

- *Which* Tor user is doing *what* over Tor
- Tor does *not* hide that *some* Tor user is visiting a particular website
  - The website operator will see the HTTP request come from a Tor exit node
- Tor does *not* hide (typically, but see Module 7) that a particular person is using Tor for *something*
  - That person's ISP can (again typically) tell that they are connecting to Tor entry/guard nodes and using the Tor protocol

## But the protection is sometimes fragile

- Support you are a Tor user in, say, Germany, and you are using Tor to access a website also in Germany (hosted by the same AS/ISP as you)
- That AS can see both your traffic to the Tor network and also the traffic from the Tor network to the web server
- Even if your Tor circuit goes through several other countries along the way

# Metadata-protecting communication systems

- There is a desire for systems that allow for metadata-protected communications, but are secure even in the previous scenario, or in the presence of a global adversary
- Metadata-protecting communication systems (MPCS) is an umbrella term for many styles of such system, with different goals and different security properties

# MPCS systematization

- There are dozens of academic papers proposing designs for different kinds of MPCS
- They're not directly comparable, however, since they don't all even try to accomplish the same things
- 2 functionality goals
- 4 privacy goals
- 6 broad approaches to accomplishing these goals

# MPCS systematization

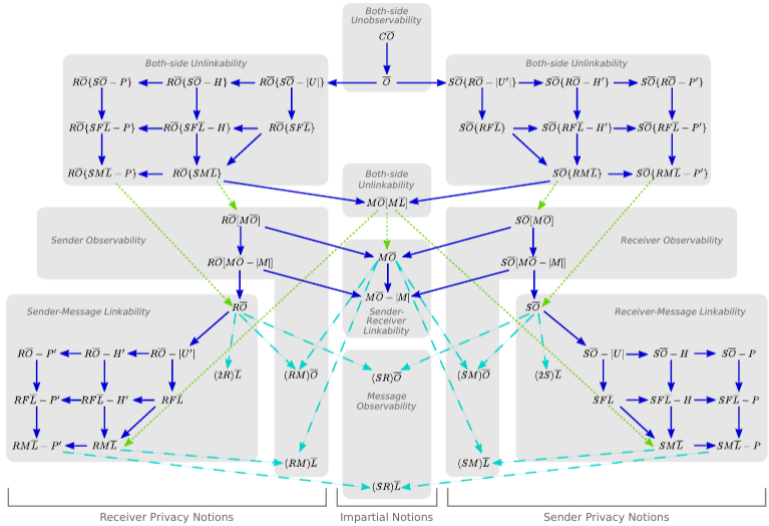
System	Year	Type	Family	Assumption	Robust	AS Protection	RE Resistant	Disruption Resistant	Disconnection Impact	Setup	Parallel Conversations	Asynchronous	Low Latency	Horizontally Scalable	Audit Complexity	Client Overhead	Message (compute if different) Complexity	Messages
					Protections					Usability				Performance				
Vuvuzela [71]	2015	CUS (Mix)	DP	(1 : m)-S: DP	○	○	●	●	▼	☺	○	○	○	○	×	m	$n \cdot m + n \cdot m$	n
Pung [7]	2016	CUS (RUMS)	PIR	RLWE	○	○	●	●	▼	☺	●	●	●	$k^2$	×	n	$n^2 + n$	1
AnonPOP [31]	2016	CUS (Mix)	M	f-S: DP	○	○	●	●	≈	☹	●	○	○	k	×	ℓ	$n \cdot \ell$	n
Stadium [69]	2017	CUS (Mix)	DP	f-S: DP	○	○	●	●	▼	☺	○	○	○	k	×	ℓ	$(n + m_d) \cdot m_d + n \cdot \ell + m^3$	n
MCMix [4]	2017	CUS	SMC	(1 : 3)-S	○	○	●	●	▼	☺	○	○	○	×	1	$n \cdot \log n + n \cdot \log n$	n	
SealPIR [5]	2018	CUS (RUMS)	PIR	RLWE	○	○	●	●	▼	☺	●	●	●	$k^2$	×	n	$n^2 + 1 [n + n]$	1
Karaoke [42]	2018	CUS (Mix)	DP	f-S: DP	○	○	●	●	▼	☺	○	○	●	k	×	ℓ	$(n + m_d) \cdot m_d + n \cdot \ell + m^3$	n
XRD [40]	2019	CUS	M	f-S	○	○	●	●	▼	☺	○	○	○	$k^2$	×	$\sqrt{m}$	$n^2 + n \cdot \sqrt{m}$	1
Clarion [29]	2021	CUS	SMC	(2 : 3)-MA	○	○	●	●	⊗	☺	○	○	○	×	1	$n \cdot \log n + n$	n	
Groove [8]	2022	CUS (Mix)	DP	f-S: DP	○	○	●	●	≈	☺	●	●	●	k	×	ℓ	$(n + m_d) \cdot m_d + n \cdot \ell + m^3$	n
DC-nets [14]	1988	CUS/SUBS	DC	None	○	●	●	○	⊗	☹	○	○	○	×	n	$n^2 [n]$	1	
Dissent [22]	2010	CUS/SUBS	DC	None	○	○	●	○	⊗	☹	○	○	○	$n^3$	n	$n^2 + n^2 [n^2 + n]$	1	
D3/Dissentv2 [76, 77]	2012	CUS/SUBS	DC	(1 : m)-S	○	○	●	●	▼	☹	○	○	○	$n^2 \cdot m$	m	$n \cdot m + n \cdot m$	1	
Verdict [23]	2013	CUS/SUBS	DC	(1 : m)-S	○	○	●	●	▼	☹	○	○	○	$n^2 \cdot m$	m	$n \cdot m + n \cdot m$	1	
Riposte [21]	2015	SUBS	RPIR	(2 : 3)-S	○	○	○	●	▼	×	⊗	●	●	$k^2$	n	$\sqrt{n}$	$\sqrt{n} [n]$	1
Riffle [39]	2016	SUBS	M	(1 : m)-S	○	○	●	●	▼	×	⊗	○	○	○	×	m	$n \cdot m$	n
eMix [16]	2016	SUBS	M	(1 : m)-S	○	○	●	●	▼	×	⊗	○	○	○	×	m	$n \cdot m$	n
Atom [38]	2017	SUBS	M	f-S	●	●	●	●	▼	×	⊗	●	○	k	×	1	$n \cdot m$	n
AsynchroMix [45]	2019	SUBS	SMC	(2n/3 : n)-M	●	●	●	●	▼	×	⊗	○	○	×	m	$m \cdot n^2 \cdot \log^2 n + m \cdot n \cdot \log^2 n$	n	
PowerMix [45]	2019	SUBS	SMC	(2n/3 : n)-M	●	●	●	●	▼	×	⊗	○	○	×	m	$n^2 + m \cdot n [n^2 + n^3 + m]$	n	
Blinder [2]	2020	SUBS	SMC	(3n/4 : n)-M	○	○	●	●	▼	×	⊗	●	○	○	$m \cdot \sqrt{n} m \cdot \sqrt{n} m \cdot n \cdot \log n + m \cdot n [m \cdot n \cdot \log n + n^2]$	n		
Spectrum [51]	2021	SUBS	RPIR	(1 : m)-S	○	○	●	●	▼	×	⊗	○	○	1	m	$m [m \cdot b]$	1	
Trellis [41]	2022	SUBS	M	f-M	●	●	●	●	⊗	×	⊗	○	○	k	×	$\log m$	$n \cdot \log^2 m + n \cdot \log m [n \cdot m]$	n
RPM [44]	2022	SUBS	SMC	(3n/4 : n)-M	●	●	●	●	▼	×	⊗	●	○	○	×	m	$m \cdot n^{1.5} + n [m \cdot n^2 + n^{1.5}]$	n
Sabre [70]	2022	SUMS/SUBS	RPIR	(1 : 2)-S	○	○	●	●	▼	☹	●	●	●	$k^2$	$\log n$	$\log n$	$1 [n]$	1
Express [30]	2021	SUMS	RPIR	(1 : 2)-S	○	○	○	○	▼	☹	●	●	●	$k^2$	n	$\lambda$	$1 [n]$	1
Mixnets [15]	2003	RUS (Mix)	M	(1 : m)-S	○	○	●	●	⊗	×	○	○	○	×	m	$n \cdot m$	n	
Pynchon Gate [61]	2005	RUS (RUMS)	PIR	(1 : m)-S: TTP	○	○	●	●	▼	☹	●	●	●	×	n·m	$n \cdot m$	1	
Loopix [56]	2017	RUS (Mix)	M	f-S: TTP	○	○	●	●	▼	☹	●	●	●	k	×	1	ℓ	1
Talek [17]	2020	RUS (RUMS)	PIR	(1 : m)-S	○	○	●	●	▼	☹	●	●	○	×	m	$n \cdot m$	1	



# Functionality goals

- All of these systems have one of two functionality goals:
- Anonymous broadcast
  - e.g., microblogging, forums, some kinds of social media
  - The messages end up public, but the metadata of *who wrote them* is protected
- End-to-end (E2E) messaging
  - e.g., messaging apps, direct messages
  - The contents of the messages are protected from the adversary
  - The metadata of *who is talking to whom* and sometimes even *the existence of a message* are also protected

# Privacy goals



# Privacy goals

- All of the systems have one of four privacy goals:
- Sender-message unlinkability ( $SM\bar{L}$ )
  - The adversary could tell that a particular party received a particular message, but cannot tell who sent it
- Receiver-message unlinkability ( $RM\bar{L}$ )
  - The adversary could tell that a particular party sent a particular message, but cannot tell who they sent it to

# Privacy goals

- Relationship unobservability ( $M\bar{O}[M\bar{L}]$ )
  - The adversary could see which users are sending or receiving messages, how many, and at what times, but cannot tell which pairs of parties are communicating with each other
- Communication unobservability ( $C\bar{O}$ )
  - The adversary can't even tell how many messages are being sent, or even if any are being sent at all

# Combined functionality and privacy goals

- With two functionality goals, and four privacy goals, you might think there would be 8 combinations of a functionality goal with a privacy goal
- However, some of the combinations do not make sense
- In particular, when targeting the functionality goal of “anonymous broadcast”, there is no specific receiver, and the message is public
  - Which privacy goals do and do not make sense with this functionality goal?
- We can call the combined functionality and privacy goal of an MPCs its *category*

# MPCS categories

- Sender-unlinkable broadcast systems (SUBS)
  - Senders submit messages over the course of an *epoch*
  - At the end of each epoch, the collected messages are published, with the connection of which sender submitted which message broken
- Sender-unlinkable messaging systems (SUMS)
  - Receivers typically have “mailboxes”
  - Senders *obliviously* deposit their messages into the receiver’s mailbox
  - But no one can tell whether any given mailbox even received a message or not, let alone which mailbox received a particular sender’s message
  - Each receiver (non-obliviously) just downloads their mailbox

# MPCS categories

- Receiver-unlinkable messaging systems (RUMS)
  - Senders (non-obliviously) put messages into some data structure
  - Everyone can see which message in the data structure was put there by which sender, and when
  - Receivers obviously read from the data structure to retrieve the messages meant for them
  - No one can tell which messages the receiver read

# MPCS categories

- Relationship unobservable systems (RUS)
  - A messaging system where, as before, the adversary can see which users are sending or receiving messages, how many, and at what times, but cannot tell which pairs of parties are communicating with each other
  - A SUMS or RUMS can often be “boosted” to a RUS by adding delays or epochs: receivers only download their messages after a lot of senders have uploaded their messages
  - A RUS can also be built from mix networks



# MPCS categories

- Communication unobservable systems (CUS)
  - A messaging system where, as before, the adversary can't even tell how many messages are being sent, or even if any are being sent at all
  - A RUS can often be “boosted” to a CUS by adding *dummy* messages: each user sends a fixed number of messages in each epoch, and then each user receives a fixed number of messages
  - A lot of those messages will be dummy messages, and not actually shown to the user in the app user interface, for example

# MPCS properties

- There are many different properties that an MPCS may or may not have
  - See the previous big table
- We're going to mainly focus on some properties that impact *usability*
  - A system with poor usability will have few users
  - A metadata-protecting system with few users has a small *anonymity set*

# MPCS properties

- Low latency
  - How long does it take from the time a sender uploads a message to the time the receiver downloads it?
  - Remember that 1990s-era mixnets had latencies of hours
  - Tor has (usually) sub-second latencies (but is not secure against a global adversary)
  - For messaging, latencies around the same order of time it takes to type the message in the first place (a few seconds) is a reasonable target
  - The latency may depend on the number of users of the system

# MPCS properties

- High throughput
  - How many messages can the system deliver per second?
  - This will depend on the number of users, the latency of the system for that many users, and whether the system is *simplex* or *multiplex*
  - In a simplex system, only one user can send a message at a given time (in a given epoch)
  - In a multiplex system with  $n$  users, each user might be able to send one or more messages in a given epoch
  - So if the latency is  $L$  seconds, a simplex system will have a throughput of  $\frac{1}{L}$  messages per second, while a multiplex system (with  $n$  users each able to send  $f$  messages per epoch) will have a throughput of  $\frac{nf}{L}$  messages per second

# MPCS properties

- Asynchronicity
  - In a *synchronous* system, the sender and receiver have to both be online and actively using the system at the same time
  - Like a phone call
  
  - In an *asynchronous* system, the sender can upload a message and the receiver can download it later
  - Maybe an arbitrary time later, maybe there's some limit
  - Like email or texting

# MPCS properties

- Scalability

- If the system becomes popular, and the number of people using it increases by a factor of  $k$ , you'll need to add more server resources in order to keep the latency the same
- Naively, you could try to just make  $k$  copies of your system, and put  $\frac{1}{k}$  of your users on each copy
- But this partitions the anonymity set!
- Some systems only scale *vertically*: you have to replace your servers with larger servers
- Better is to scale *horizontally*: add more servers, but in a way that doesn't partition the anonymity set
- A key question is how many more servers do you need in order to serve  $k$  times as many users

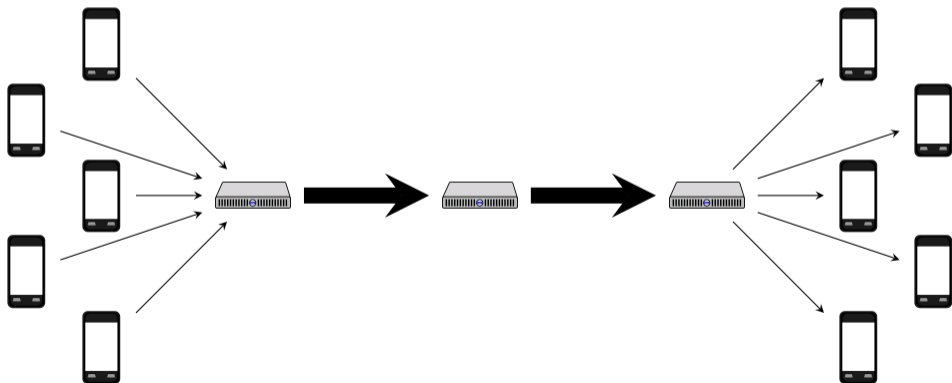
# MPCS families

Work in this area can be broadly grouped into six *families* based on their primary approach to protecting metadata

- Mixnets
- DC nets
- PIR
- Reverse PIR
- Differential privacy
- Secure multiparty computation

# Mixnets

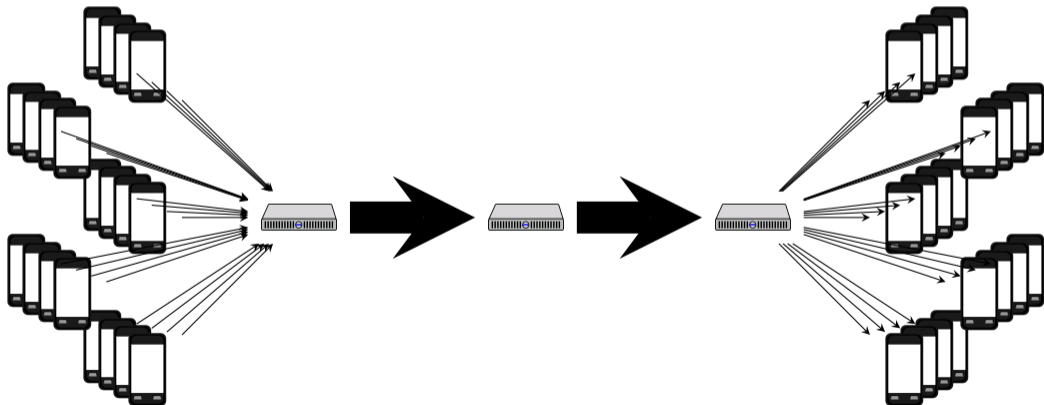
- We've covered this before, but in its simplest form:



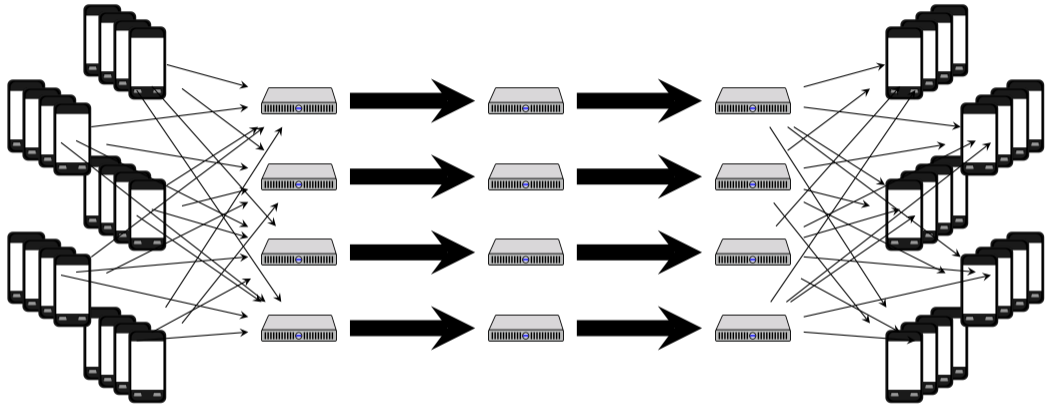


# Mixnets

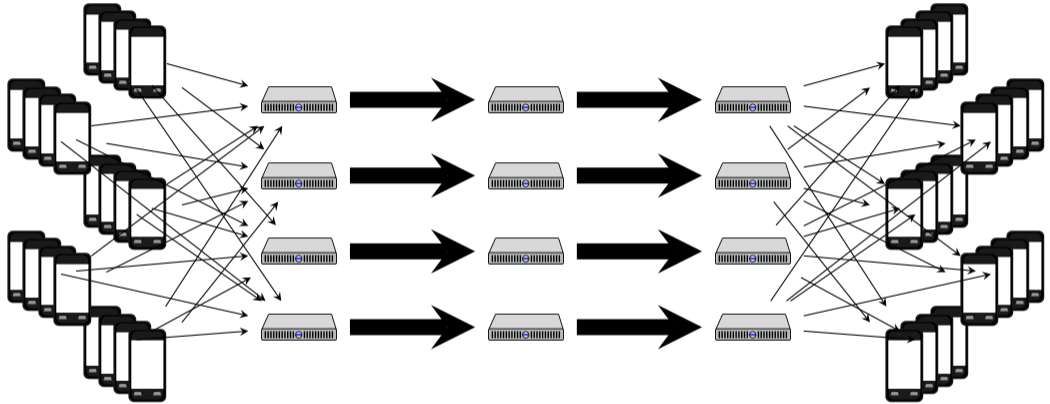
- We've covered this before, but in its simplest form:



# Scaling mixnets

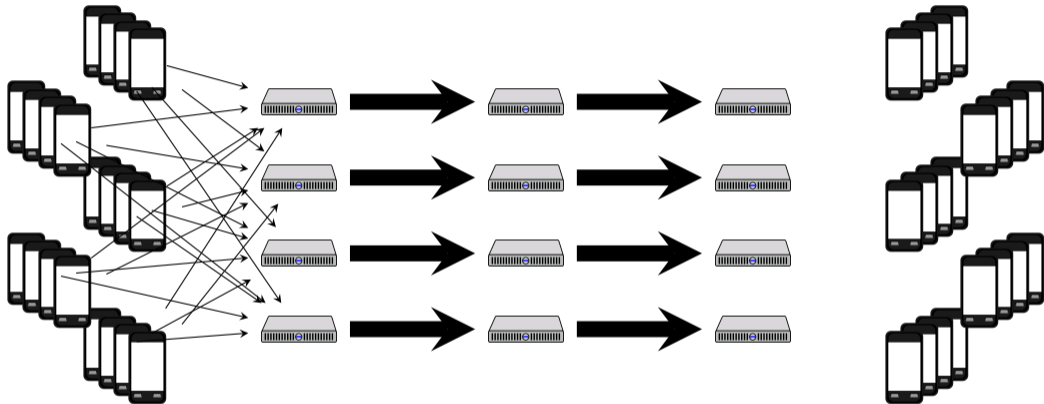


# Scaling mixnets

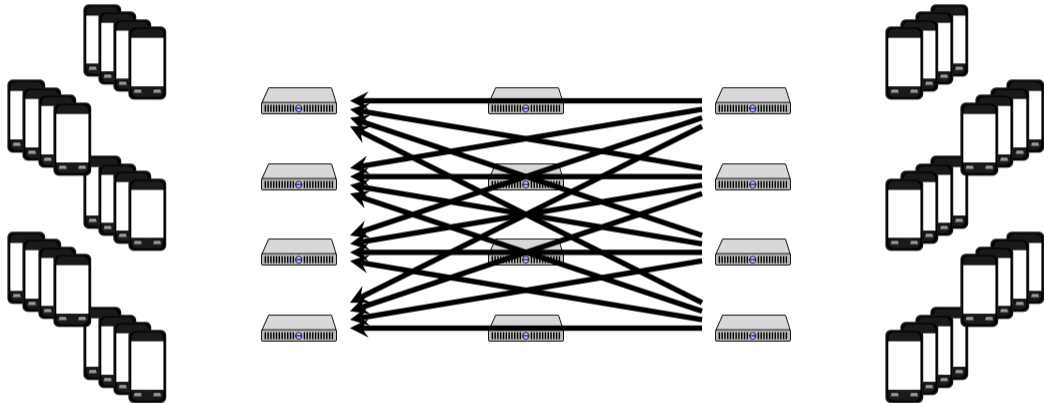


- Problem: partitioned anonymity set

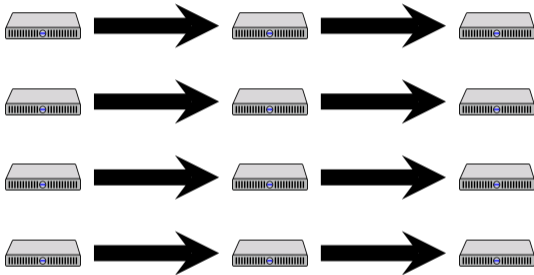
# Scaling mixnets



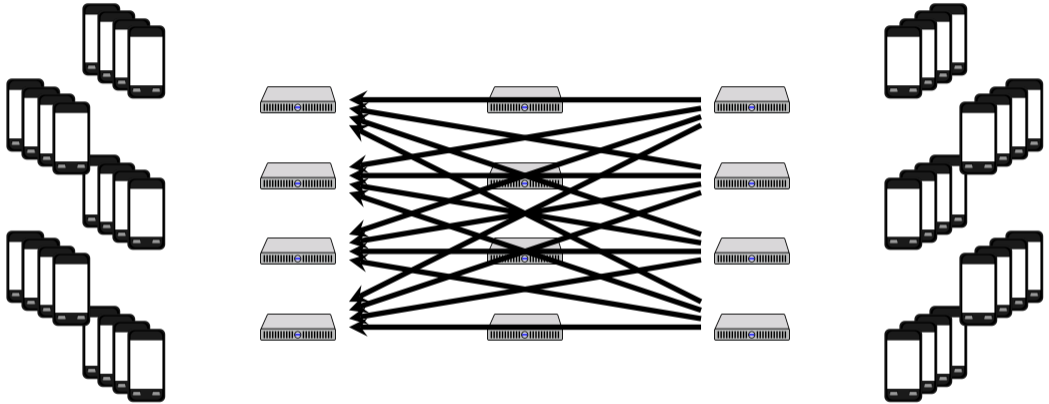
# Scaling mixnets



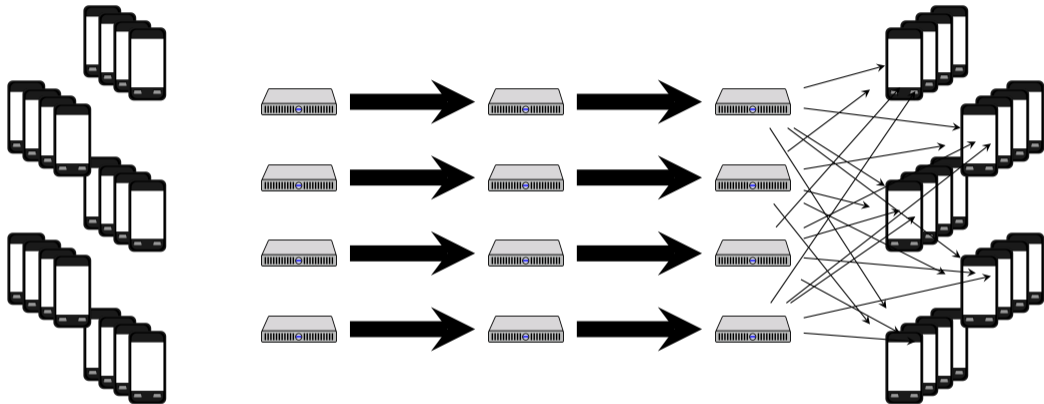
# Scaling mixnets



# Scaling mixnets



## Scaling mixnets



- Atom (2017) was the first mixnet to explicitly target horizontal scalability without partitioning the anonymity set



# Properties of Atom

- Low latency
  - No: 28 *minutes* to mix a batch of 1,000,000 messages using 1024 servers
- High throughput
  - No: about 0.6 messages per second per server
- Asynchronicity
  - Yes: once mixed, messages can just be published (SUBS) or stored in a recipient's mailbox (SUMS)
- Scalability
  - Horizontal scalability: to handle  $k$  times as many clients, you can use  $k$  times as many servers, with each server doing the same amount of work as before
  - Latency also decreases linearly with the number of servers

- Loopix (2017) is a mixnet with a somewhat different design
- Each client is signed up to a dedicated *service provider*
  - This service provider stores the client's mailbox for incoming messages, authenticates the client
  - But: if the service provider is compromised, some of the client's privacy is lost!
- The mix network in Loopix operates *continuously*, not in rounds
  - When the client adds each encryption layer to the message, she specifies how long the message should be delayed at each node
  - If a packet arrives at time  $T$  to a mix node, it is decrypted (and checked for replay attacks); if the packet says to delay by  $d$ , the packet is forwarded to the next mix node at time  $T + d$

- Clients pick the delays  $d$  from a Poisson distribution
- Clients and mixes also send dummy packets *to themselves* (“loops”), and also dummy packets that get dropped, according to Poisson distributions
  - This helps defeat passive attacks and detect active attacks
- Loopix is the basis for the deployed Nym mix network we mentioned earlier

# Properties of Loopix

- Low latency
  - Depends on the Poisson parameters; tradeoff between latency and privacy
- High throughput
  - Yes: hundreds of messages per second per server
- Asynchronicity
  - Yes, but only because of the trusted service provider
- Scalability
  - Horizontal scalability

- DC nets (“dining cryptographer networks”) are a primitive for sender-unlinkable broadcast
- So systems built from DC nets will typically be SUBS
- There are  $n$  clients; time is divided into “slots”
- In each slot, at most one client has a message to send
- Messages are of fixed length (say  $b$  bits)

# DC nets

A

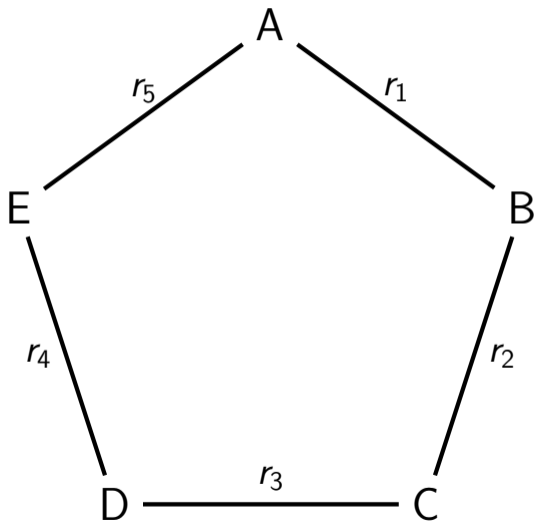
E

B

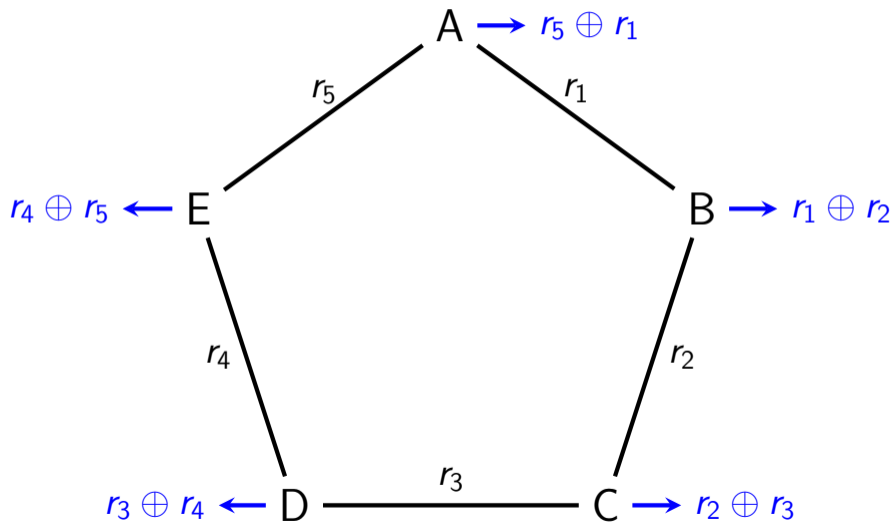
D

C

# DC nets

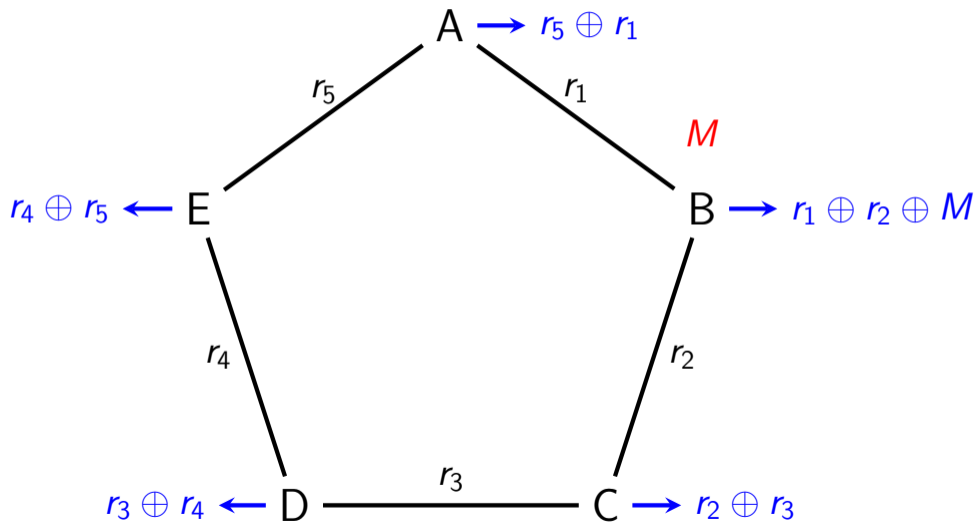


# DC nets

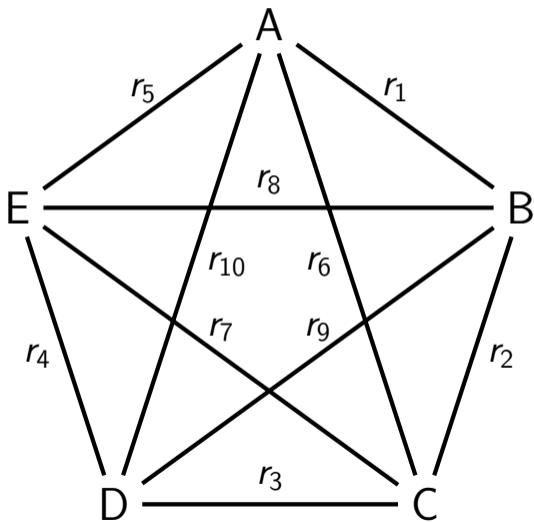




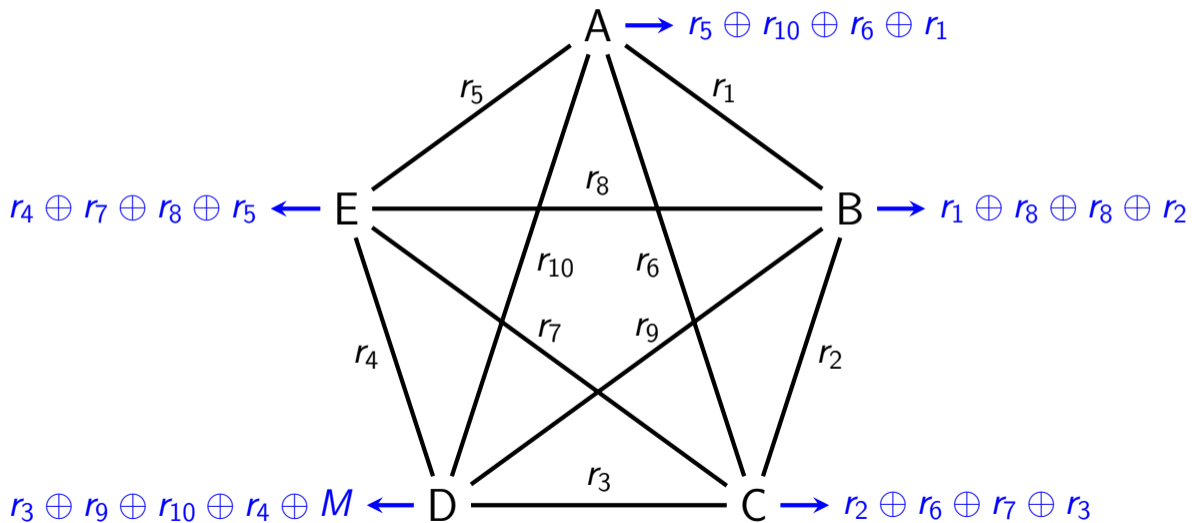
# DC nets



# DC nets



# DC nets



- Original DC net paper (1988)
- Dissent (2010)
  - Provides a way to privately assign players to slots
  - Allows for variable-length messages
- Dissent v2 (2012)
  - $n$  clients submit shares of their messages to  $m$  servers
  - Lower complexity, but at least one of the servers must be honest
  - Detects disrupting clients
- Verdict (2013)
  - Prevents disrupting clients
  - Lots of (somewhat expensive) zero-knowledge proofs

# Properties of the DC net family

- Low latency
  - 1 second to tens of seconds if no party misbehaves, longer if someone does (for 1000 clients, which is not that many)
- High throughput
  - No: each slot can only transmit one message
- Asynchronicity
  - Not really applicable to broadcast systems (there is no receiver)
- Scalability
  - Vertical scalability: to handle  $k$  times as many clients, each DC net party (regardless of how many there are) has to do  $k$  times as much work

# PIR family

- The next family of MPCs is based on PIR
- These systems are generally RUMS, but can be boosted to higher levels as before
- Which message in the system was sent by which sender and when is *not* hidden by the system
- Which message in the system is read by which receiver *is* protected

- General strategy: sender inserts (encrypted) message into a database, receiver uses PIR to read the desired item from the database
- How does the receiver know there's a message waiting?
  - ⇒ Dialling protocol
- How does the receiver know which message in the database to retrieve?

- All participants have public keys
  - Each participant knows (at least) their friends' keys
  - Can also learn public keys during the dialling protocol
- Use Diffie-Hellman to compute the shared secrets between your private key and your friends' public keys
  - There will be one such shared secret between you and each of your friends
- In each round of the protocol, compute a pair of per-round secrets: one for you sending a message to your friend and one for your friend sending a message to you
  - e.g.,  $s_{r,A,B} = \text{MAC}_{k_{AB}}(r, \text{"Alice"})$  and  $s_{r,B,A} = \text{MAC}_{k_{AB}}(r, \text{"Bob"})$



# PIR family

- When Alice wants to send a message to Bob in round  $r$ , she encrypts the message, and inserts it into the database with keyword  $s_{r,A,B}$ 
  - This is a (keyword,value) database
- Bob does a keyword PIR query on the keyword  $s_{r,A,B}$  to retrieve Alice's message
  - The PIR server can tell Alice sent a message, but not which user retrieved that message

- The details of the systems differ among MPC schemes in this family
- Which PIR scheme to use?
  - Pynchon Gate (2005) uses multi-server PIR
  - Pung (2016), SealPIR (2018) use homomorphic encryption PIR
- How to retrieve messages from multiple friends in a round?
  - Pung uses batch codes (recall Assignment 2)
  - SealPIR uses plaintext packing (recall Module 5)

# Properties of the PIR family

- Low latency
  - Yes: a couple of seconds for a million users (SealPIR)
- High throughput
  - No: the server must process a PIR query for each message
- Asynchronicity
  - Not typically: the dialling protocol only completes when the recipient is online
- Scalability
  - Bad horizontal scalability: to handle  $k$  times as many clients, the PIR server(s) must do  $k^2$  times as much work

# Reverse PIR

- We've talked about PIR (private information retrieval) a lot in this course
- Here, we look at a related topic, "Reverse PIR", also called "PIR in reverse" or "PIS" (private information storage)
- Goal: *write* (or sometimes *update*) an entry in a database without revealing either the index (or keyword) of the entry, or the value written

# Reverse PIR

- One technique uses the DPFs we saw in Module 3
- We set up the database in a similar way as we would for 2-server Chor PIR, but here, the database is *itself* XOR-shared
  - In PIR, each server just has a copy of the database
  - So in Reverse PIR, each server itself can't see the contents of the database (why is this vital?)
- Server 0 holds share  $D_0$  of the database  $D$ , and server 1 holds share  $D_1$  such that  $D_0 \oplus D_1 = D$ 
  - That is,  $D_0[i] \oplus D_1[i] = D[i]$  for all  $i$

# Reverse PIR

- A client wants to update entry  $a$  of the database by the value  $b$
- That is, the client wants server 0 to end up with a database share  $D'_0$  and server 1 to end up with  $D'_1$  such that:

$$D'_0[i] \oplus D'_1[i] = \begin{cases} D_0[i] \oplus D_1[i] & i \neq a \\ D_0[i] \oplus D_1[i] \oplus b & i = a \end{cases}$$

- Without either server learning  $a$  or  $b$

- Recall DPFs:

$$\text{GEN}(r, a, b) \rightarrow (\text{key}_0, \text{key}_1)$$

$$\text{EVAL}(0, \text{key}_0, i) \oplus \text{EVAL}(1, \text{key}_1, i) = \begin{cases} 0 & i \neq a \\ b & i = a \end{cases}$$

- And we want (from the previous slide):

$$D'_0[i] \oplus D'_1[i] = \begin{cases} D_0[i] \oplus D_1[i] & i \neq a \\ D_0[i] \oplus D_1[i] \oplus b & i = a \end{cases}$$

- What do we do?

# Reverse PIR family

- We can make an MPCCS from Reverse PIR
- Each recipient has a “mailbox” associated with them
  - It is not a secret as to which recipient has which mailbox
- When a sender wants to send a message, she uses Reverse PIR to write the message into the mailbox
  - The server(s) do not know which mailbox received the message—*all* mailboxes appear to have potentially changed
- Recipients can download their mailboxes at their leisure



# Reverse PIR family

- Benefit: No advance coordination of sender and recipient (no dialling protocol)
- Challenge: Malicious clients can send malformed DPFs that overwrite everyone's mailbox with randomness!
  - Servers need to be able to check that their purported DPF shares are in fact shares of a point function
  - Riposte (2015), Express (2021), Sabre (2022) have progressively more efficient protocols for this
- Why was this not a problem in the PIR family?

# Properties of the Reverse PIR family

- Low latency
  - Yes: 0.05 seconds for a quarter million users (Sabre)
- High throughput
  - No: the servers must expand a DPF for each message
- Asynchronicity
  - Yes: recipients can download their mailboxes whenever they like
- Scalability
  - Bad horizontal scalability: to handle  $k$  times as many clients, the server(s) must do  $k^2$  times as much work

## DP family

- The differential privacy (DP) family of MPCs aims to build a CUS from mixnets
- The construction requires parties to communicate synchronously, and requires dummy traffic, which is chosen using *differential privacy*
- DP is a mathematical technique by which you can add noise to an *observable* to provably bound the probability that an adversary can learn the true value of the observable

## DP family

- For example, an observable might be “how many messages Alice is sending this round”
- Without protection, an adversary seeing this observable can trivially distinguish the cases of “Alice is in a conversation” from “Alice is not in a conversation” (contrary to the privacy requirements of a CUS)
- To protect this observable, Alice might send in each round a number of dummy messages picked from a particular distribution with mean, say, 10

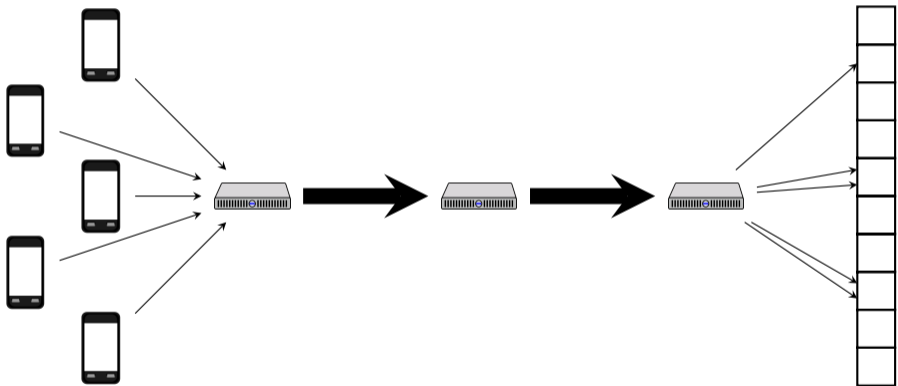
## DP family

- Then the adversary will see Alice sending, for example, 7 messages in a round, or 12 in the next round, and it should be hard for the adversary to tell whether Alice is actually sending real messages in addition to the dummy messages or not
- **But:** each use of DP “spends” a little bit of a *privacy budget*; when you’ve spent too much, the protections DP offers falter
- In our example, if the adversary watches Alice for a long time, and finds the number of messages has mean close to 10, then Alice is probably not actually communicating very much, but if the mean is close to 11, the Alice is probably communicating often

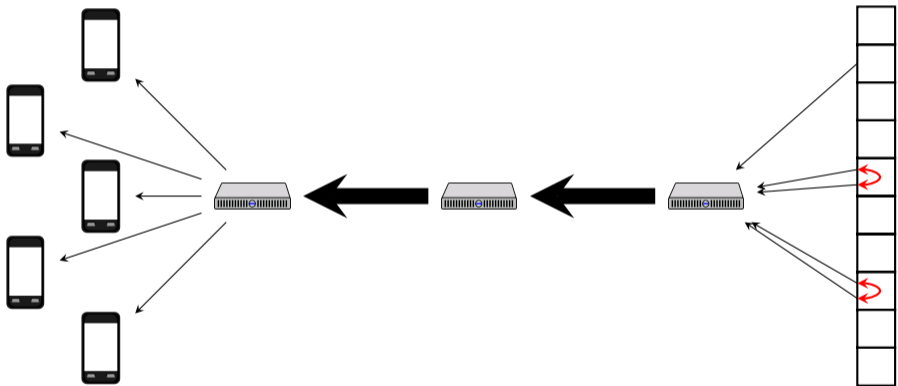
## DP family

- In this family of MPCs, clients again use a dialling protocol to indicate to each other that they want to start a conversation
- If Alice and Bob are in a conversation, then they compute a shared value for each communication round
  - Very much like the PIR family, but in PIR, there was one value for messages Alice sends to Bob, and a different one for messages Bob sends to Alice
  - Here, Alice and Bob use the *same* value for sending messages to each other
- Alice and Bob *both* send a message in each round (even if it's a blank message), addressed to a *dead drop* whose address is the shared value

# DP family



# DP family





# DP family

- Vuvuzela (2015) was the first system to use this approach
  - It used a single chain of mix nodes, and so was only vertically scalable
- Stadium (2017) and Karaoke (2018) added horizontal scalability
  - Similar in intent to Atom (but the details are different)
- Groove (2022) added the ability for parties to go offline briefly
  - They delegate their sending and receiving to a service provider while offline
  - Unlike Loopix, the service provider need not be trusted

# Properties of the DP family

- Low latency
  - Not really: 80 seconds for 3 million users (Groove)
- High throughput
  - Yes: millions of messages can be sent in that same 80 seconds
- Asynchronicity
  - No (except only temporarily with Groove)
- Scalability
  - Horizontal scalability (except Vuvuzela)

- Recall the mechanism we just talked about in the DP family:
  - Each party (whether they're in a conversation or not) inserts a message tagged with a dead drop address
  - Parties not in a conversation use a dummy message and a random dead drop address
  - Parties in a conversation send real or dummy messages, but the same dead drop address
  - The system collects all the messages, swaps any pairs that have the same dead drop address, and returns them to the senders

# MPC family

- The observation of MCMix (2017) was that you can view this as a private *computation* problem:
- There are  $n$  clients. Each client  $i$  has an input  $(a_i, m_i)$ . Client  $i$  gets the output 
$$o_i = \begin{cases} m_j & \text{if there is a } j \neq i \text{ with } a_j = a_i \\ m_i & \text{otherwise} \end{cases}$$
- We can then use any of the private computation techniques we looked at earlier in the course to implement this function!

- MCMix used the MPC (distributed trust) technique
- High-level approach:
  - Use oblivious sort to sort the input pairs  $(a_i, m_i)$  by the address  $a_i$  (obviously remember which client input which pair)
  - For each consecutive pair in the sorted array, obviously swap them if the addresses are the same
  - Reverse the above oblivious sort to send the (possibly having been swapped) items back to the client that input them
- The oblivious sort can be implemented as an oblivious shuffle followed by a non-oblivious sort
  - Like we saw in Module 4

# MPC family

- Note the difference to the mixnet family:
- With mixnets, each server (one at a time) shuffles the batch of messages, and gives some kind of proof of correctness of its shuffle
- In the MPC family, the servers *collectively* perform a single verifiable oblivious shuffle, secure in the honest minority setting

# MPC family

- A simpler version of this technique also works to achieve the anonymous broadcast functionality
- Basically, you only need to do the oblivious shuffle, not the (non-oblivious) sort or the oblivious swaps
  - But the oblivious shuffle was the bottleneck, so it's not *that* much cheaper
- Various subsequent schemes aimed to improve the efficiency of this bottleneck MPC oblivious shuffle
  - Asynchromix (2019), PowerMix (2019), Clarion (2022), RPM (2022)

# MPC family

- PowerMix's approach was significantly different; it used the following mathematical observation:
- If you have messages  $m_1, m_2, \dots, m_n$  (say in  $\mathbb{Z}_p$ ), and you publish:

$$s_1 = m_1 + m_2 + \dots + m_n$$

$$s_2 = m_1^2 + m_2^2 + \dots + m_n^2$$

$$s_3 = m_1^3 + m_2^3 + \dots + m_n^3$$

$\vdots$

$$s_n = m_1^n + m_2^n + \dots + m_n^n$$

- Then anyone can solve for the values of the  $n$  messages
  - But not their order!



# MPC family

- PowerMix's approach was significantly different; it used the following mathematical observation:
- If you have messages  $m_1, m_2, \dots, m_n$  (say in  $\mathbb{Z}_p$ ), and you publish:

$$s_1 = m_1 + m_2 + \dots + m_n$$

$$s_2 = m_1^2 + m_2^2 + \dots + m_n^2$$

$$s_3 = m_1^3 + m_2^3 + \dots + m_n^3$$

⋮

$$s_n = m_1^n + m_2^n + \dots + m_n^n$$

Try it?

$$m_1 + m_2 = 10 \quad m_1^2 + m_2^2 = 52$$

- Then anyone can solve for the values of the  $n$  messages
  - But not their order!

# Properties of the MPC family

- Low latency
  - Not typically
- High throughput
  - No
- Asynchronicity
  - Not when used for messaging functionality (a dialling protocol is required); not really applicable when used for broadcast functionality
- Scalability
  - Vertical scalability