Aleator: Random Beacon via Scalable Threshold Signatures

Robert Chen
Mentored by Alin Tomescu

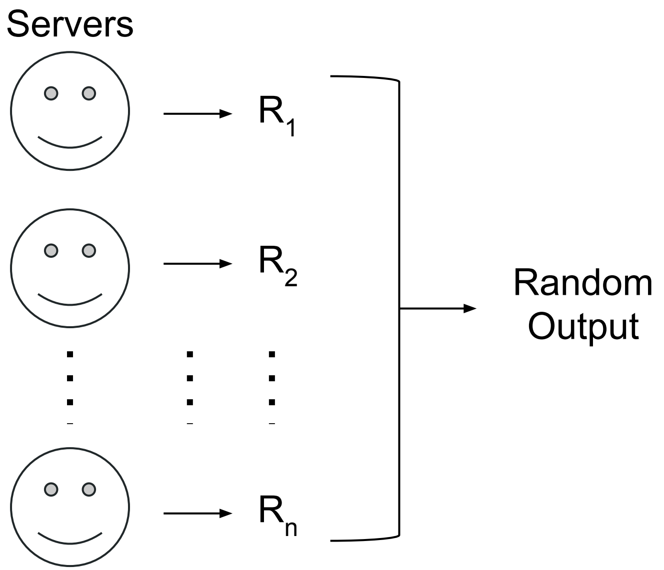
PRIMES Computer Science Conference
10/13/18

Why Scalability?

- **Scalable** threshold signature scheme
    - Increased security
    - Scalable random beacon

What is a Random Beacon?

A set of servers that periodically output a random number.



What is a Random Beacon?

A set of servers that periodically output a random number.

- Some servers could maliciously “bias” the output

What is a Random Beacon?

A set of servers that periodically output a random number.

- Some servers could maliciously “bias” the output
- Need **unbiasability**: servers cannot influence the output in their favor

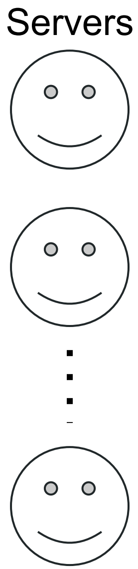
Contributions

- Elegant, scalable random beacon design
- For 100,000 participants, a random output can be produced every 20 seconds with only 3.05 MB of bandwidth (~5 minutes if many dishonest)
- Limiting factor is bandwidth: For 33 outputs × 3.05MB/output ≈ 100 MB, we can produce a random output every 0.6 to 10 seconds

	Participants	Time	Total Time Across System	Bandwidth
Randherd	512	6s	>200s	>100 MB
Aleator	33,000	4s	8s	1 MB

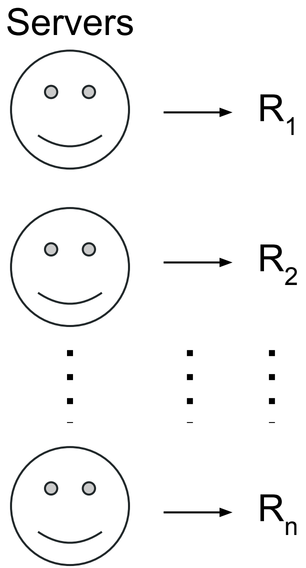
Naive Random Beacon: Combine all

Approach: Combine all *random inputs* to produce random output



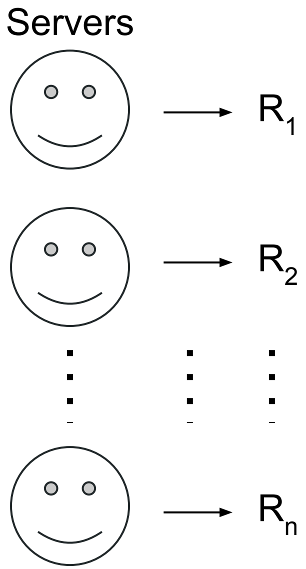
Naive Random Beacon: Combine all

Approach: Combine all *random inputs* to produce random output



Naive Random Beacon: Combine all

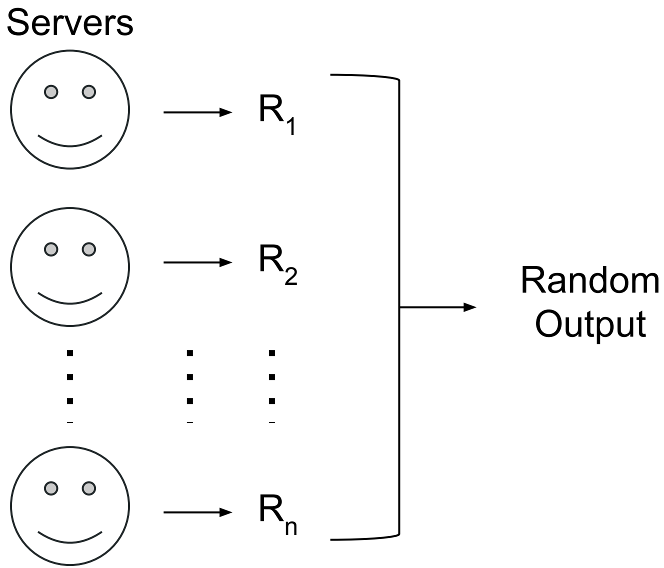
Approach: Combine all *random inputs* to produce random output



Assuming they can agree on everyone’s random inputs

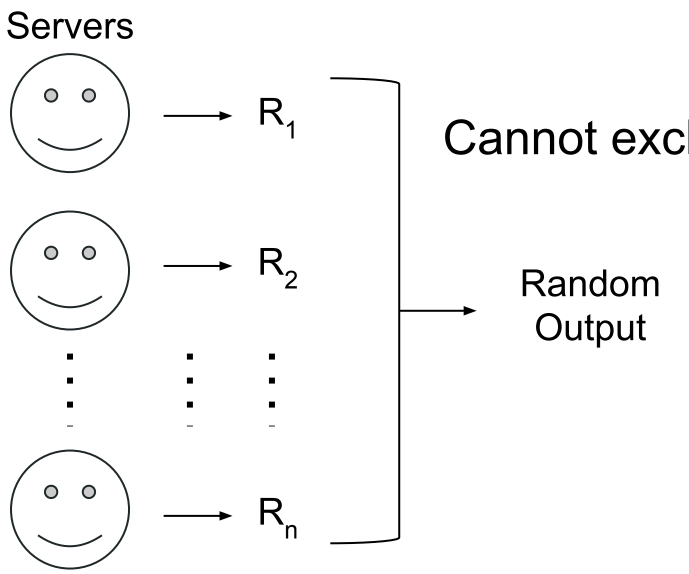
Naive Random Beacon: Combine all

Approach: Combine all *random inputs* to produce random output



Naive Random Beacon: Combine all

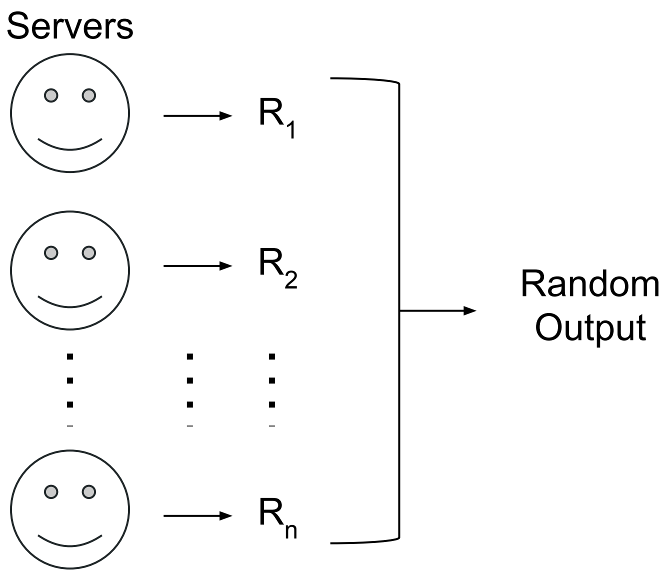
Approach: Combine all *random inputs* to produce random output



Cannot exclude any random inputs

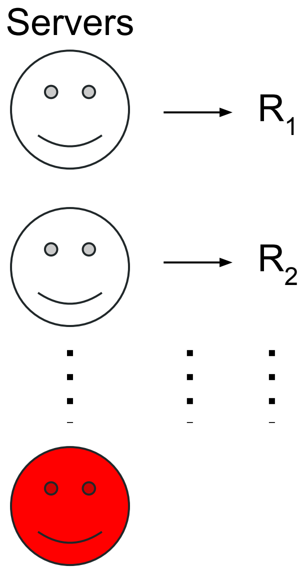
Naive Random Beacon: Combine all

Approach: Combine all *random inputs* to produce random output
Problem: Last participant controls random output



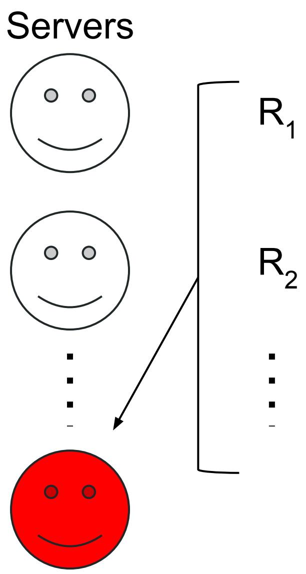
Naive Random Beacon: Combine all

Approach: Combine all *random inputs* to produce random output
Problem: Last participant controls random output



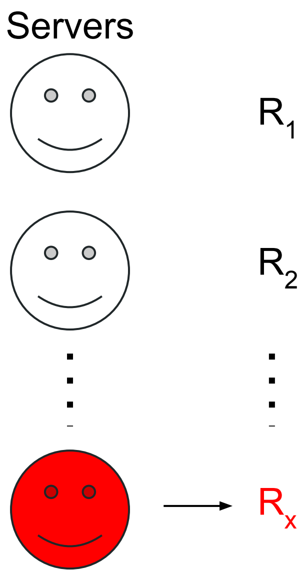
Naive Random Beacon: Combine all

Approach: Combine all *random inputs* to produce random output
Problem: Last participant controls random output



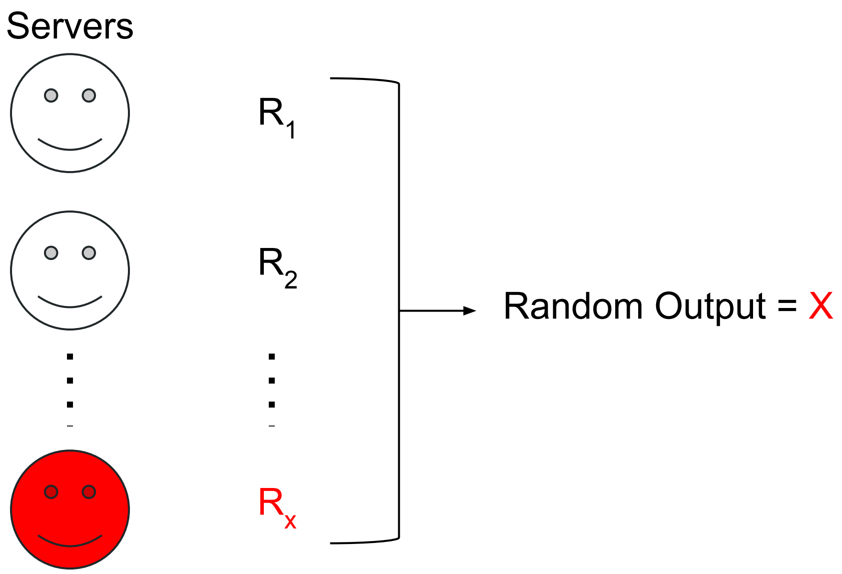
Naive Random Beacon: Combine all

Approach: Combine all *random inputs* to produce random output
Problem: Last participant controls random output



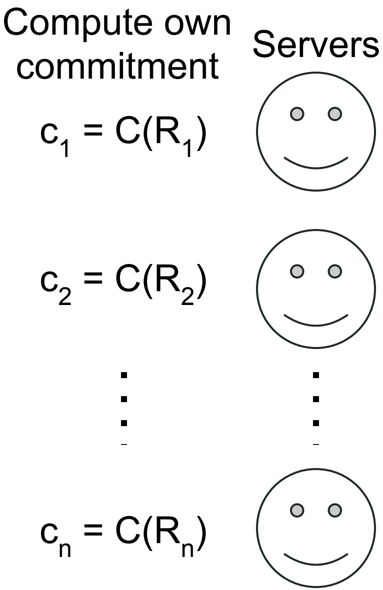
Naive Random Beacon: Combine all

Approach: Combine all *random inputs* to produce random output
Problem: Last participant controls random output



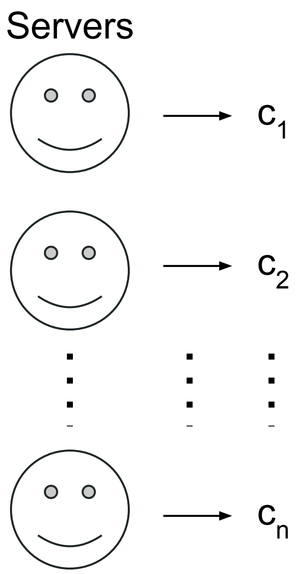
Naive Random Beacon: Commit-then-reveal

Approach: *Commit*-then-reveal random inputs



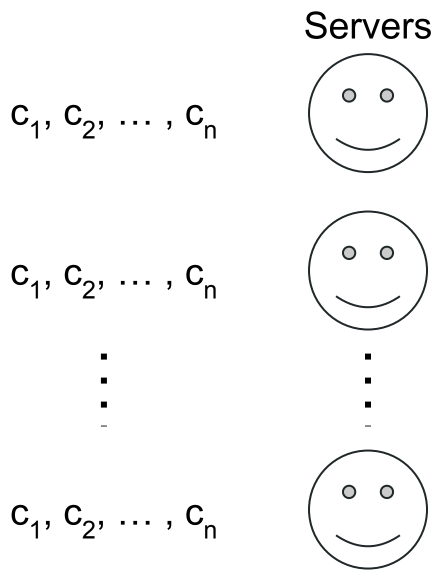
Naive Random Beacon: Commit-then-reveal

Approach: *Commit*-then-reveal random inputs



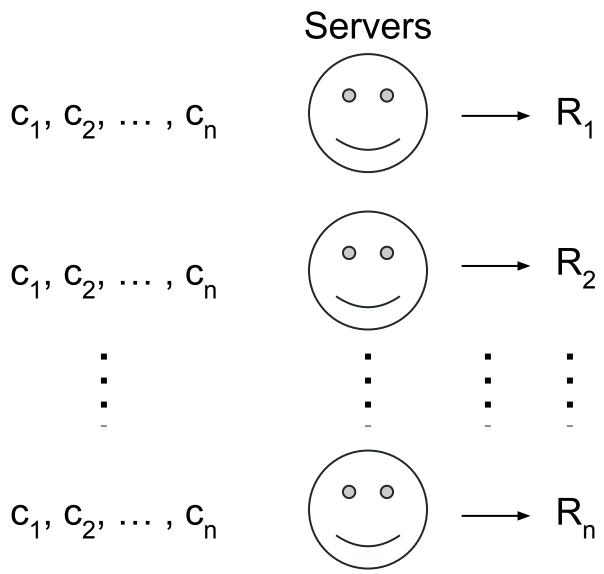
Naive Random Beacon: Commit-then-reveal

Approach: *Commit*-then-reveal random inputs



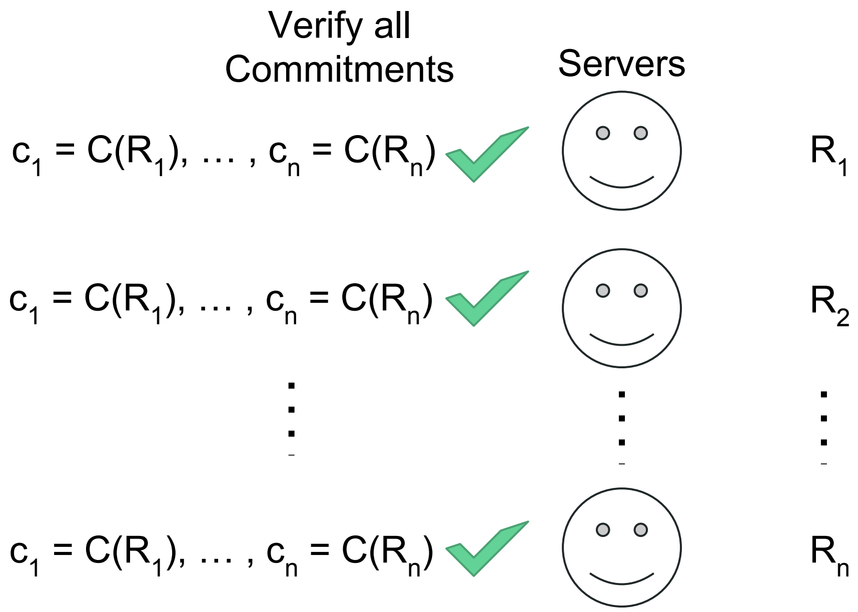
Naive Random Beacon: Commit-then-reveal

Approach: *Commit*-then-reveal random inputs



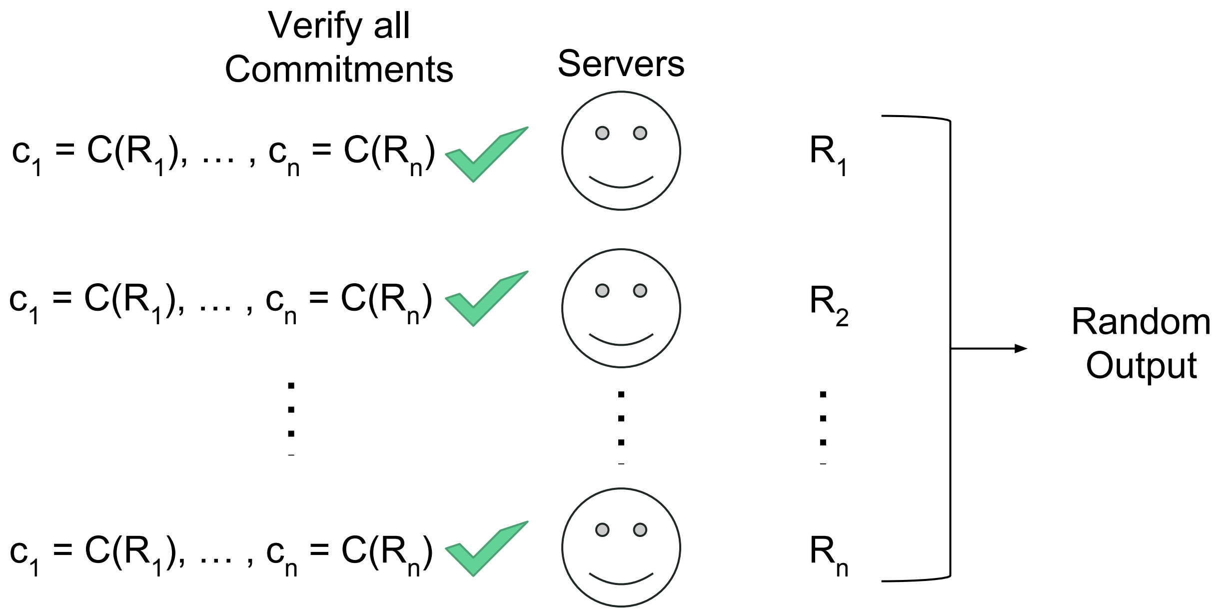
Naive Random Beacon: Commit-then-reveal

Approach: Commit-then-reveal random inputs



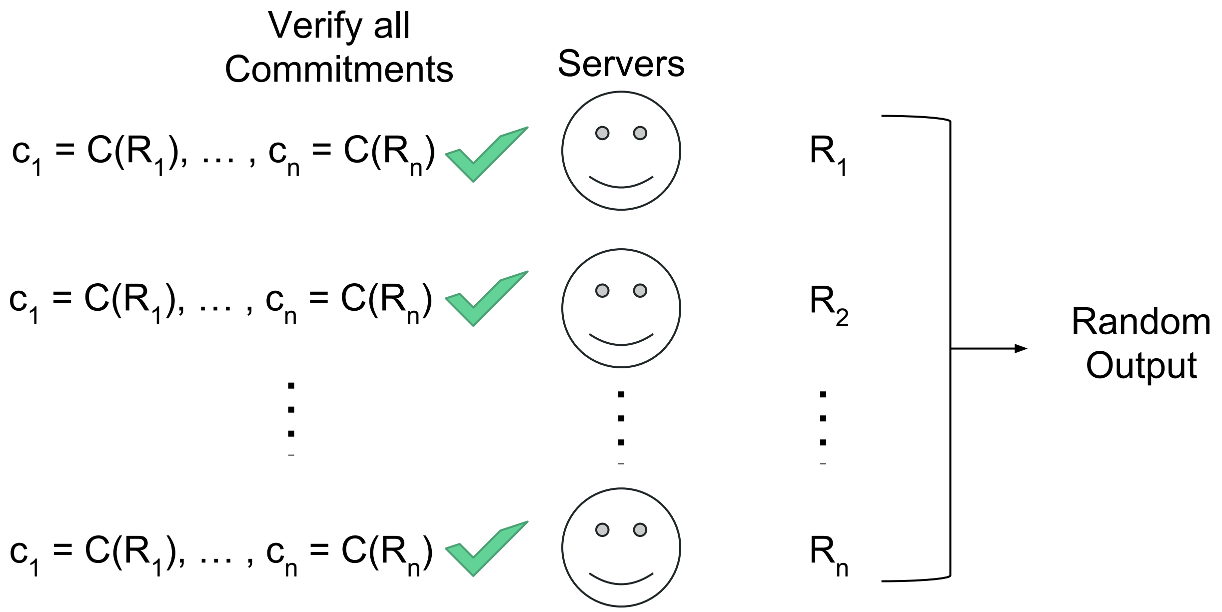
Naive Random Beacon: Commit-then-reveal

Approach: Commit-then-reveal random inputs



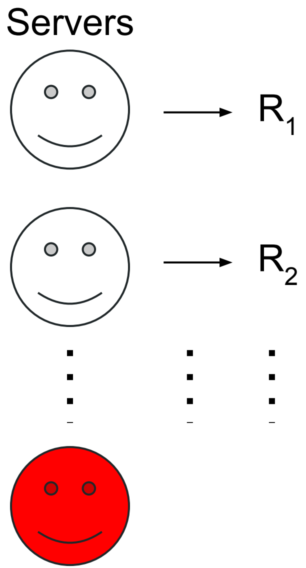
Naive Random Beacon: Commit-then-reveal

Approach: Commit-then-reveal random inputs
Problem: Dishonest participants refuse to reveal



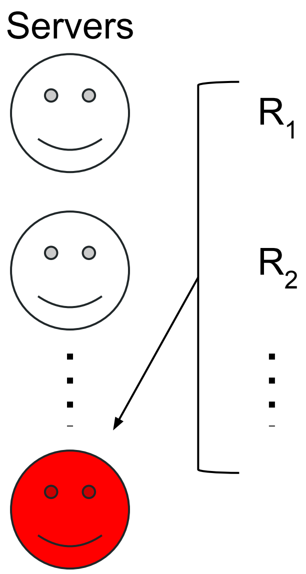
Naive Random Beacon: Commit-then-reveal

Approach: Commit-then-reveal random inputs
Problem: Dishonest participants refuse to reveal



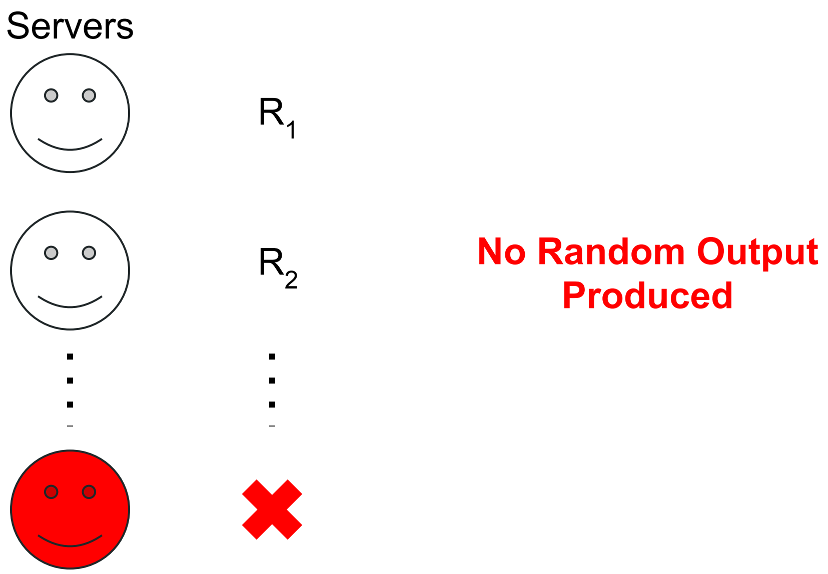
Naive Random Beacon: Commit-then-reveal

Approach: Commit-then-reveal random inputs
Problem: Dishonest participants refuse to reveal

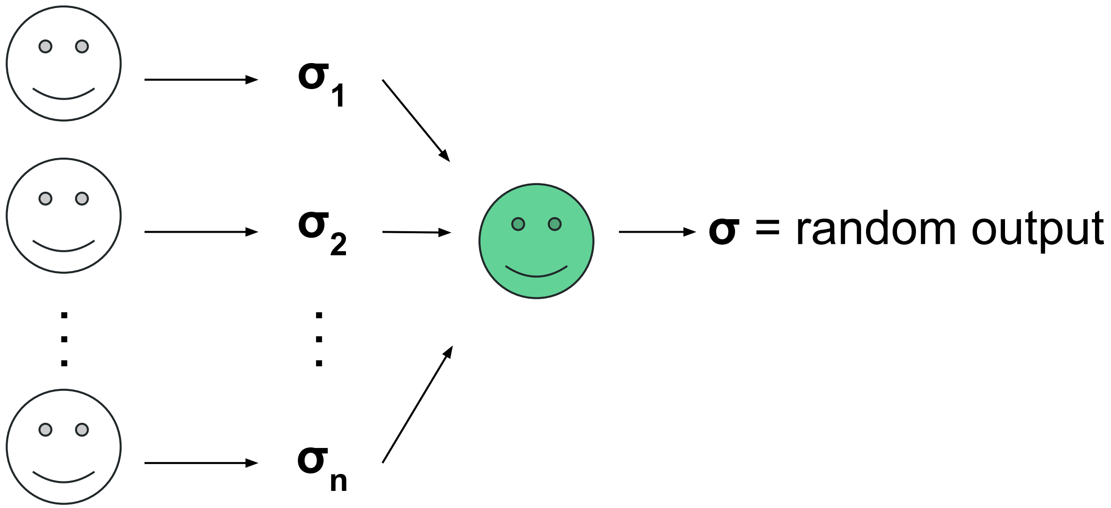


Naive Random Beacon: Commit-then-reveal

Approach: Commit-then-reveal random inputs
Problem: Dishonest participants refuse to reveal

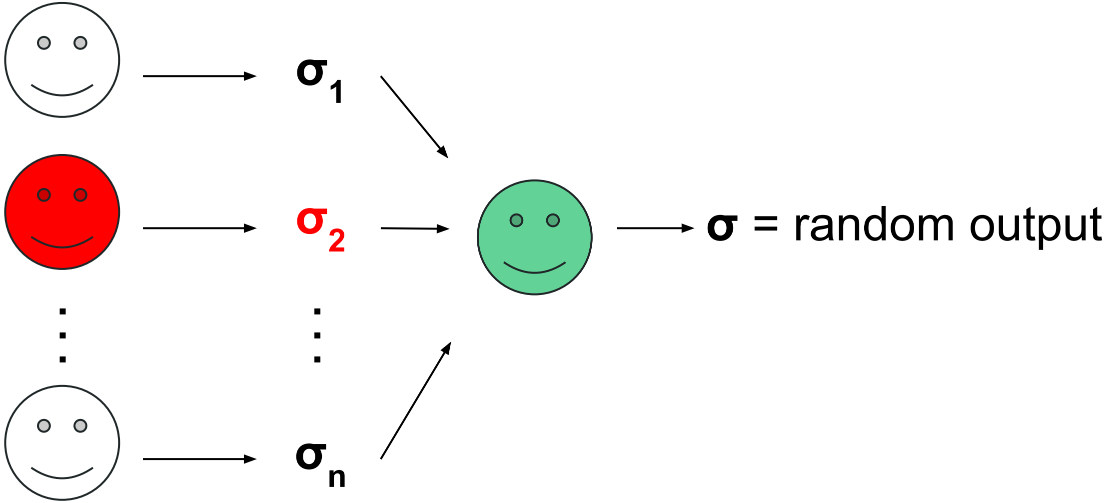


Solution: Use a threshold signature scheme



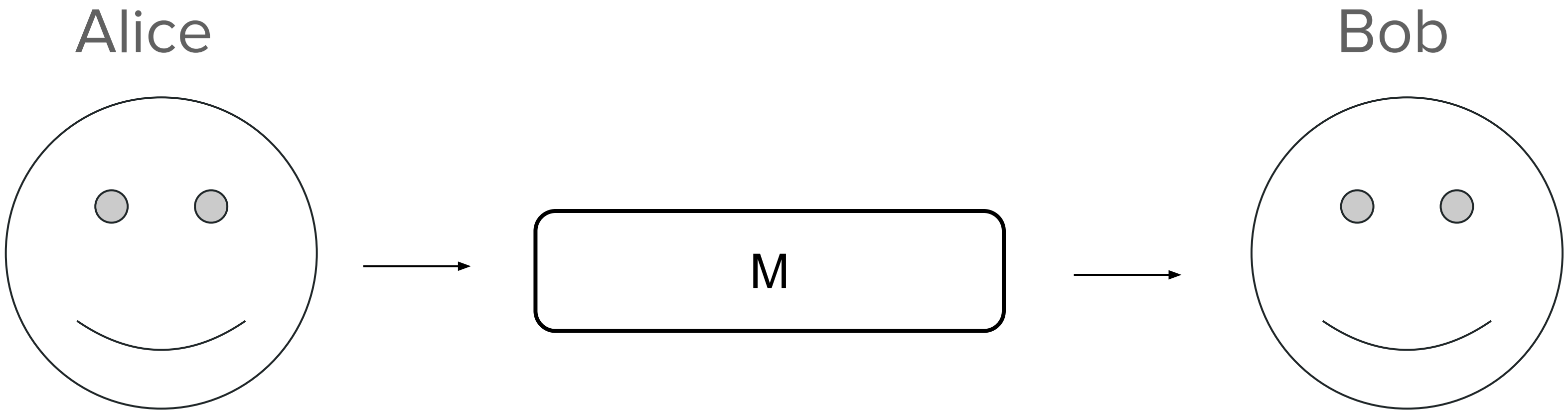
(e.g., DFINITY blockchain)

Solution: Use a threshold signature scheme



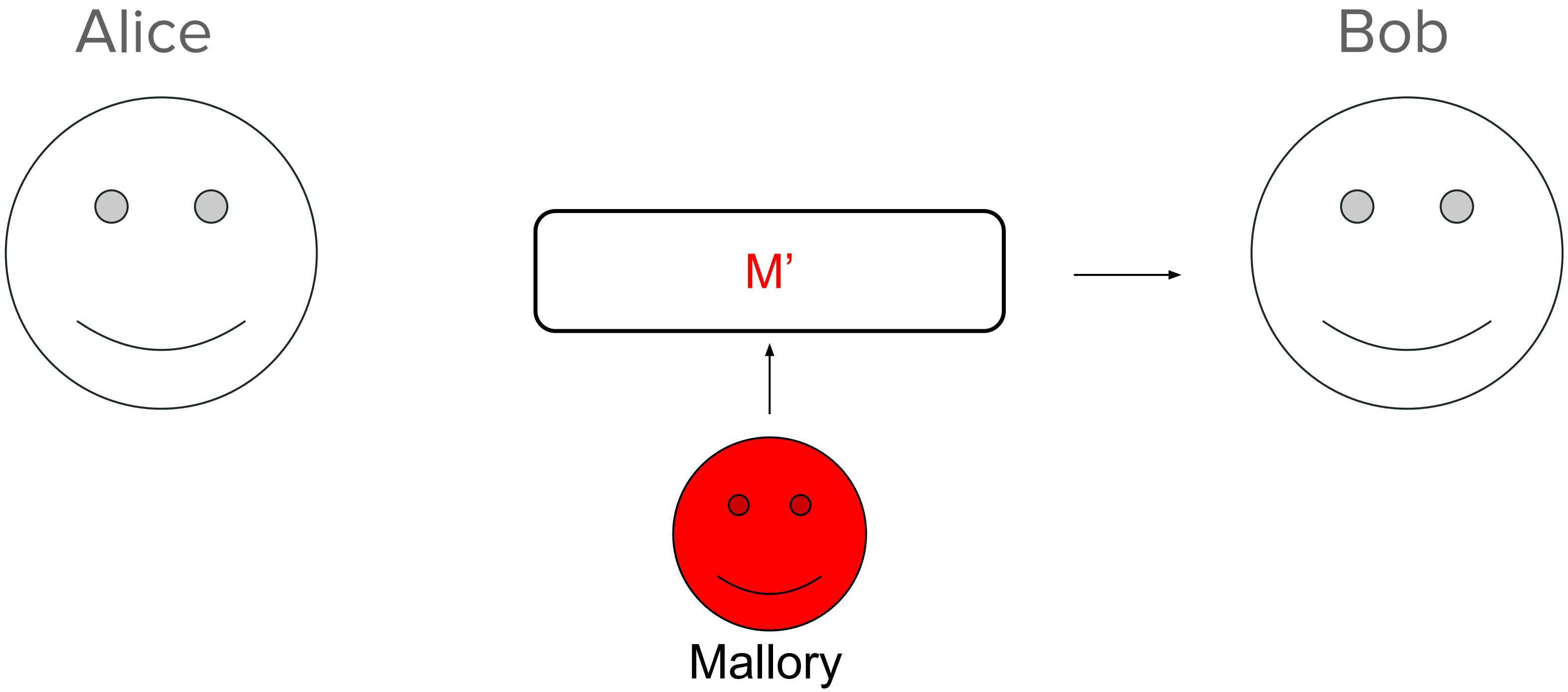
(e.g., DFINITY blockchain)

Digital Signatures: Motivation



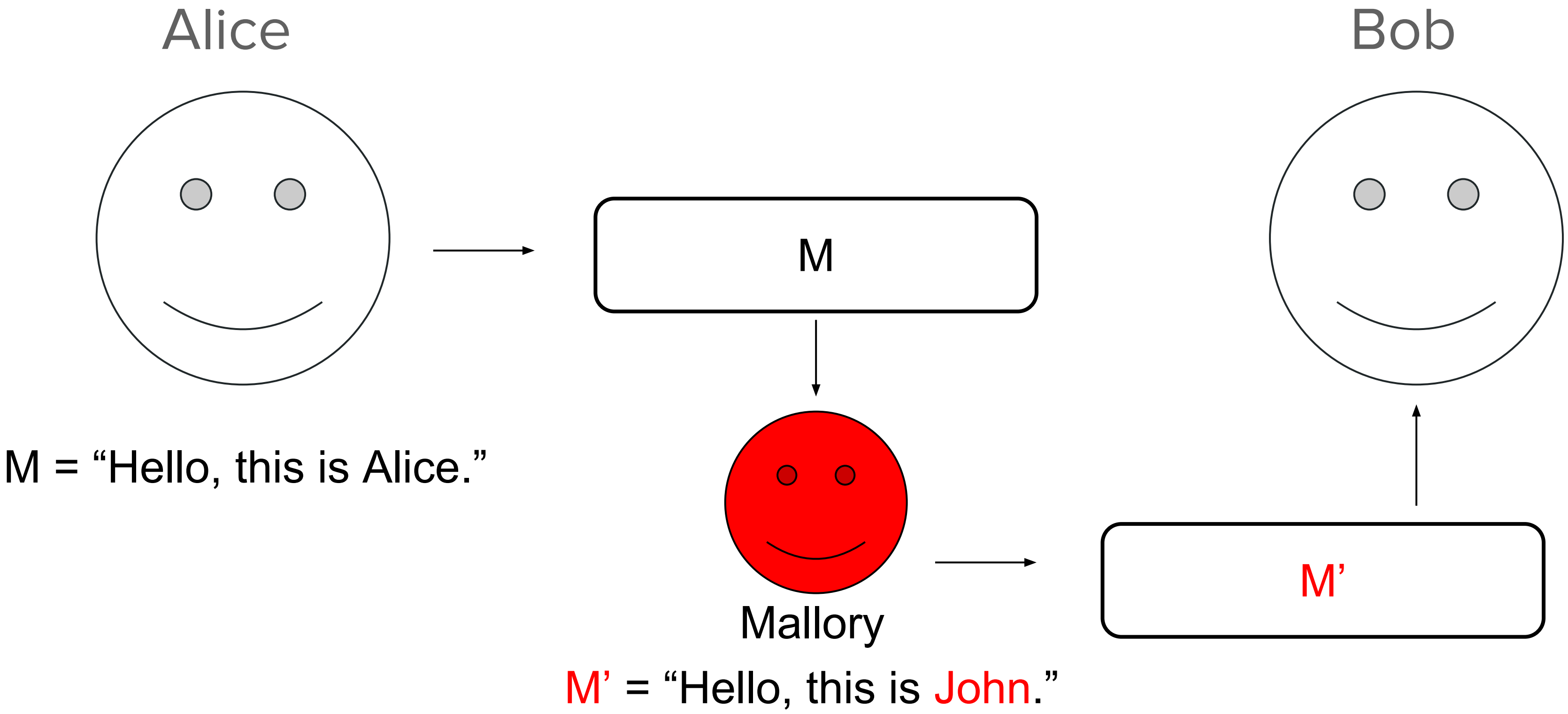
M = “Hello, this is Alice.”

Problem: Mallory can pretend to be Alice to Bob



M’ = “Hello, this is Alice.”

Problem: Mallory can tamper with Alice's messages



Solution: Digital Signatures

(Diffie-Hellman '76, RSA '78)

Alice

Bob

Alice has her own **secret key**

Bob has Alice's **public key**

Solution: Digital Signatures

(Diffie-Hellman '76, RSA '78)

Alice

M = “Hello, this is Alice.”

$\sigma = \text{Sign}(M, SK_{\text{Alice}})$

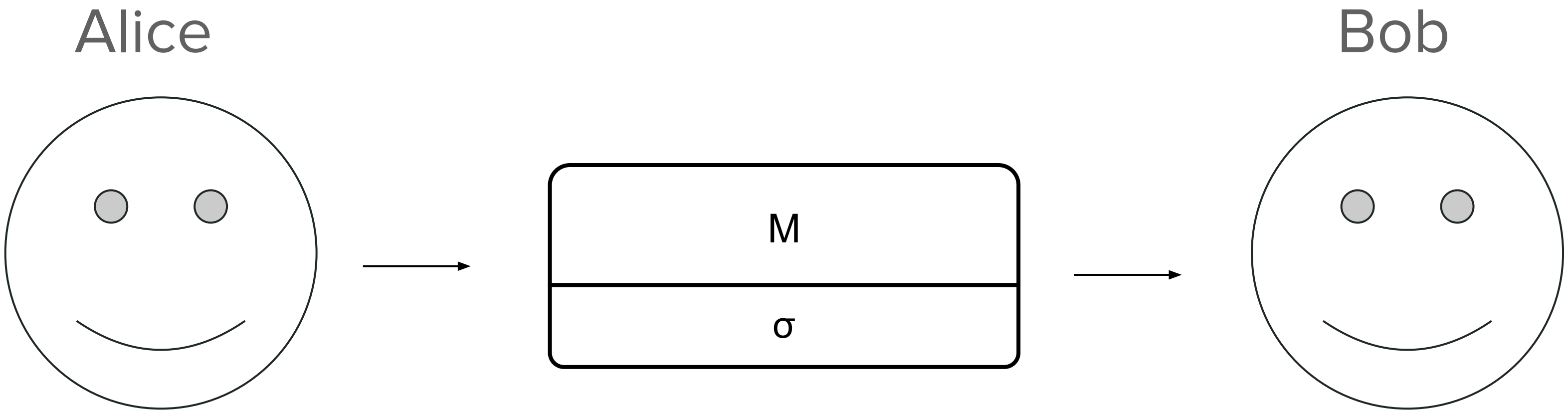
Alice has her own **secret key**

Bob

Bob has Alice’s **public key**

Solution: Digital Signatures

(Diffie-Hellman '76, RSA '78)



M = “Hello, this is Alice.”

σ = Sign(M, SK_{Alice})

Alice has her own **secret key**

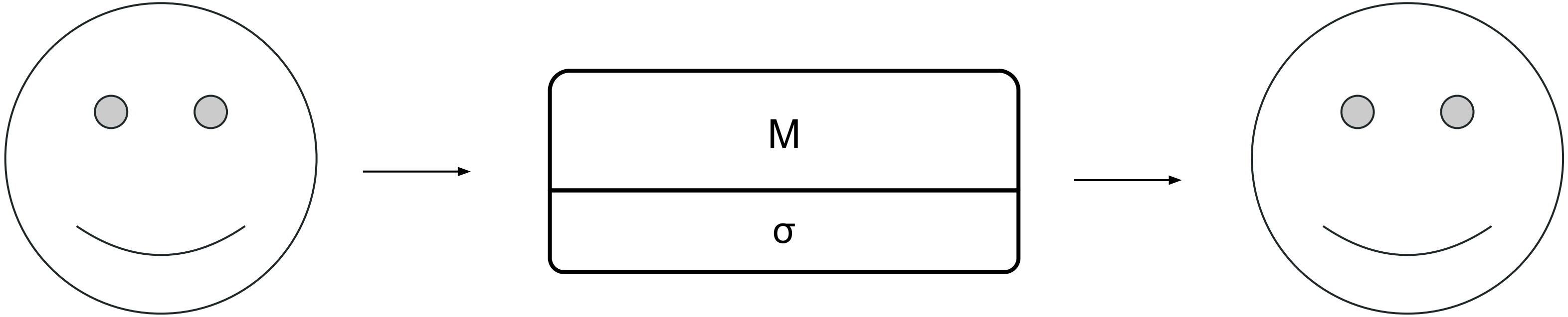
Bob has Alice’s **public key**

Solution: Digital Signatures

(Diffie-Hellman '76, RSA '78)

Alice

Bob



M = "Hello, this is Alice."

$\sigma = \text{Sign}(M, SK_{Alice})$

Alice has her own **secret key**

$\text{Verify}(\sigma, M, PK_{Alice}) = \text{true}$ ✓

Bob has Alice's **public key**

Naive Threshold Signatures

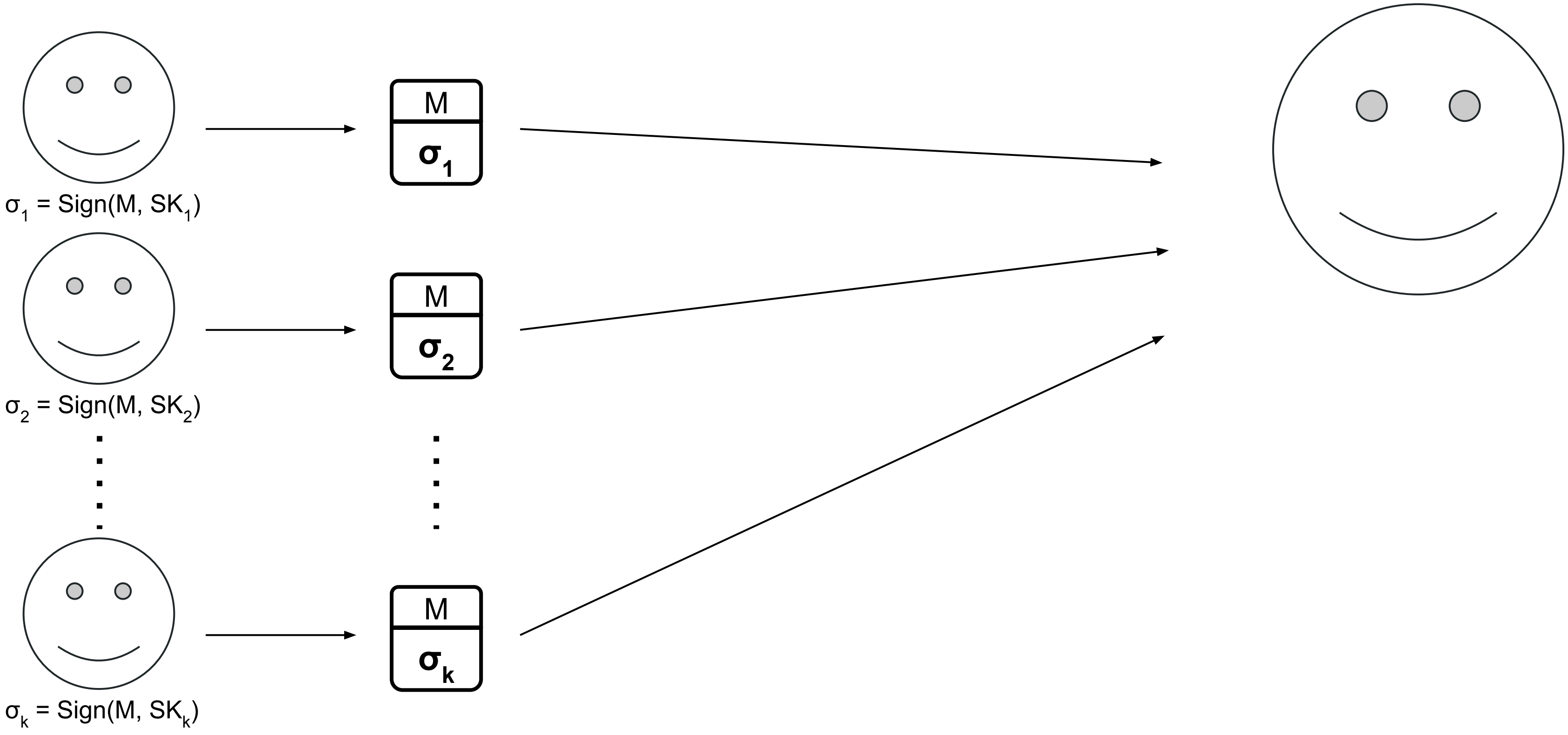
σ_1 = Sign(M, SK_1)

σ_2 = Sign(M, SK_2)

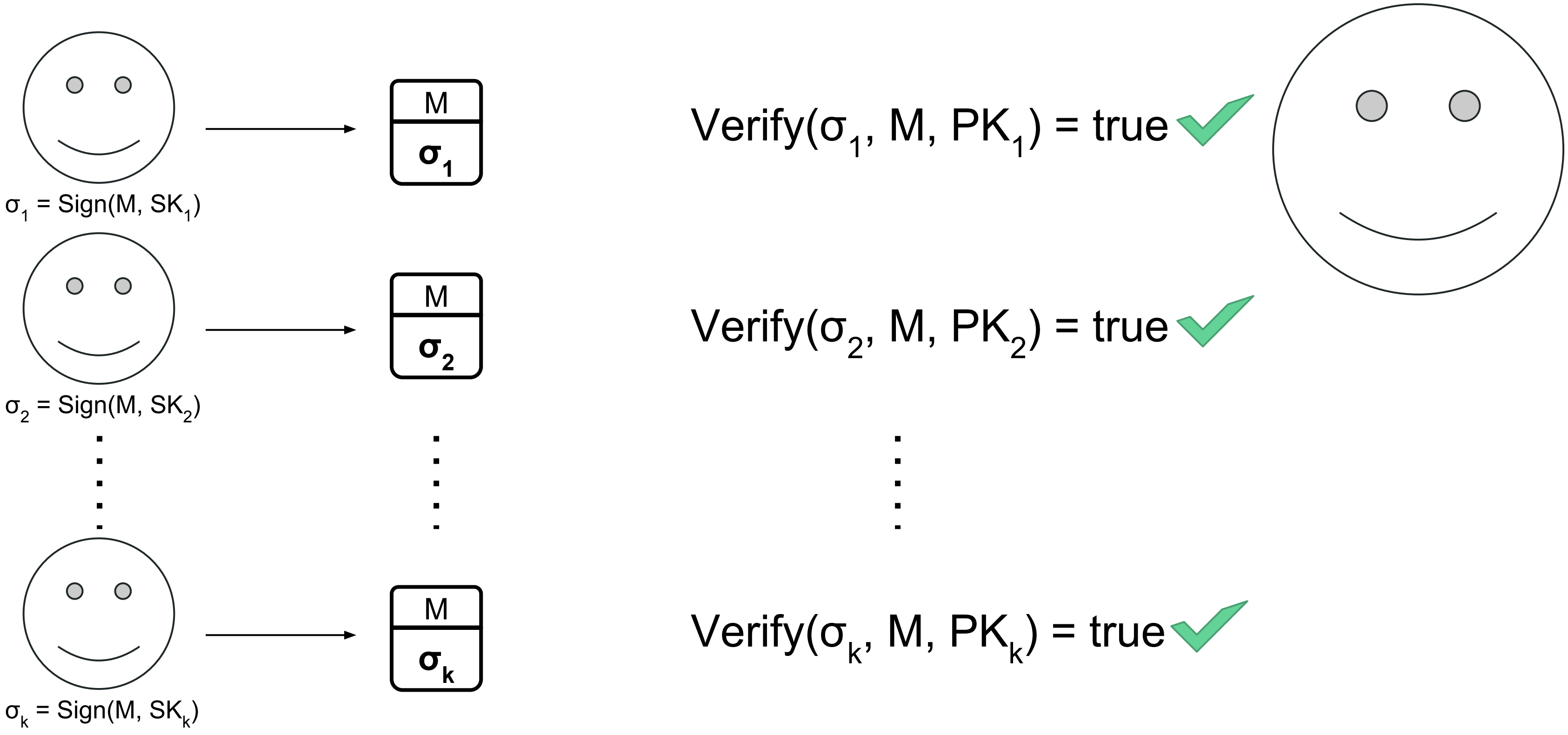
⋮

σ_k = Sign(M, SK_k)

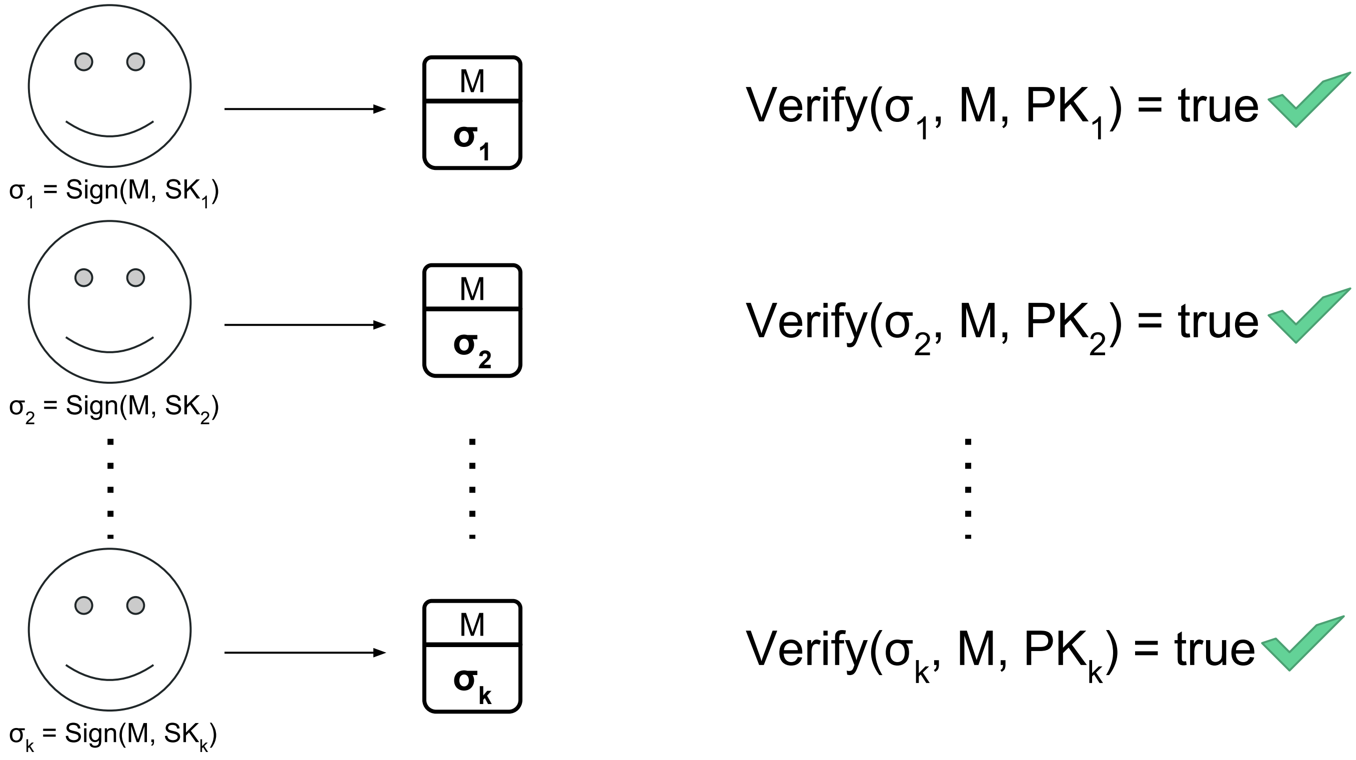
Naive Threshold Signatures



Naive Threshold Signatures



Naive Threshold Signatures

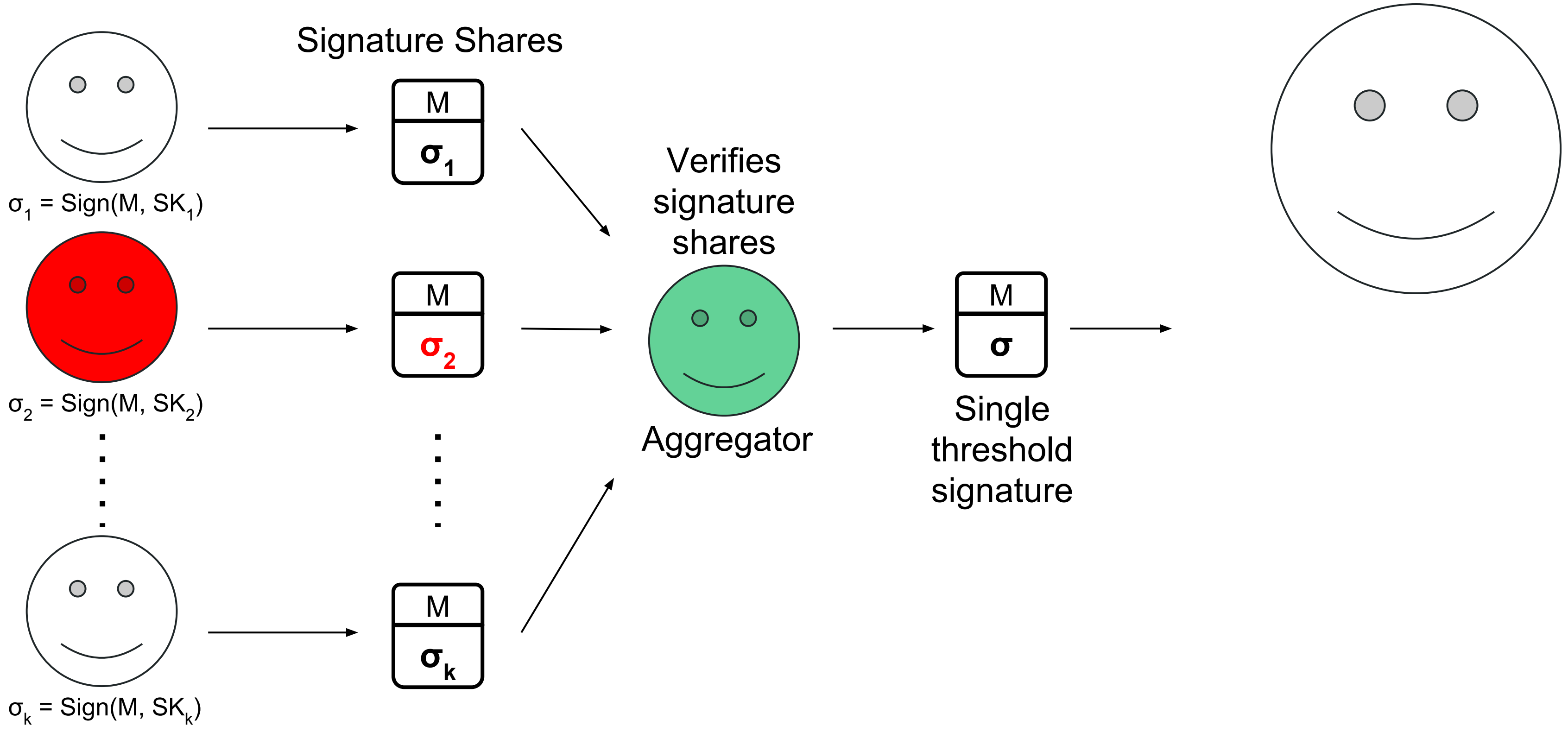


Too large
- k signatures

Too much time
- k verifications

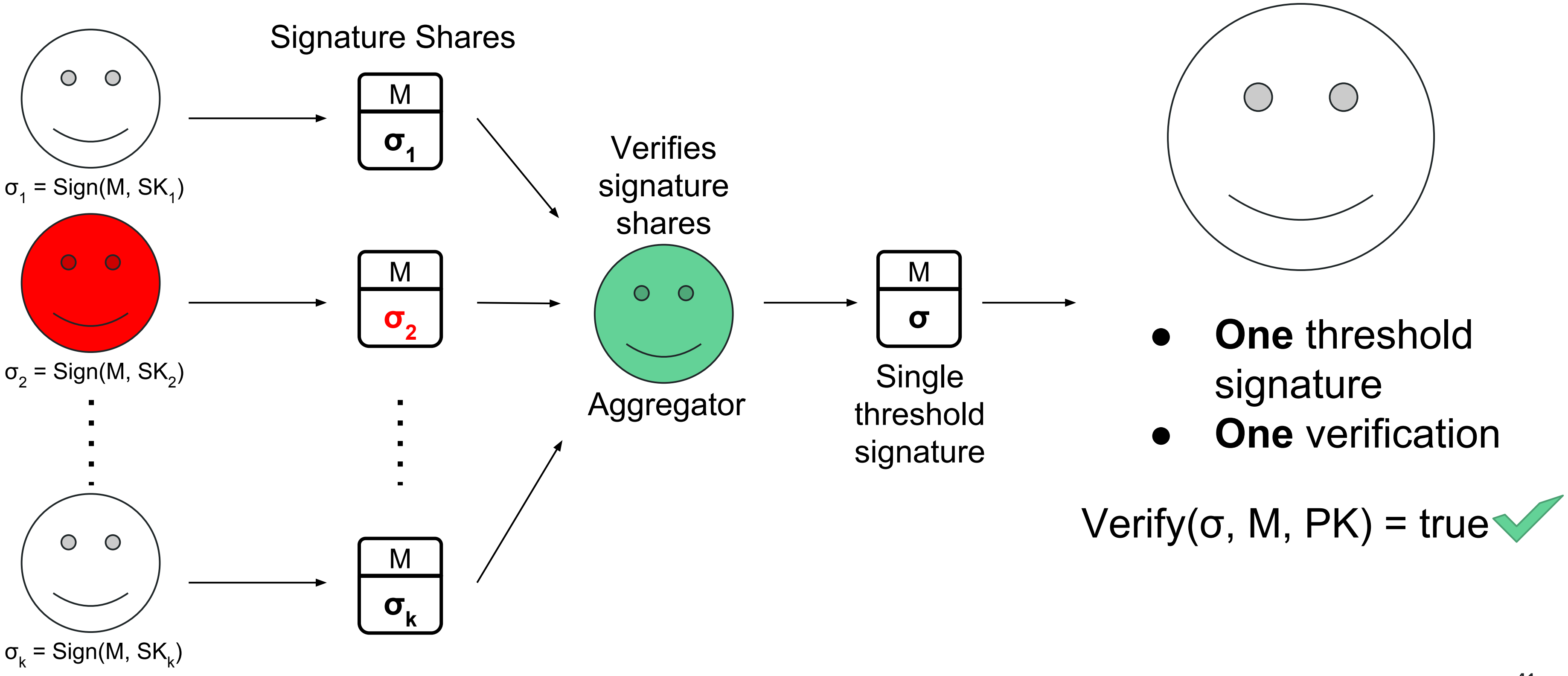
Threshold Signatures

(Desmedt, CRYPTO 1987)



Threshold Signatures

(Desmedt, CRYPTO 1987)

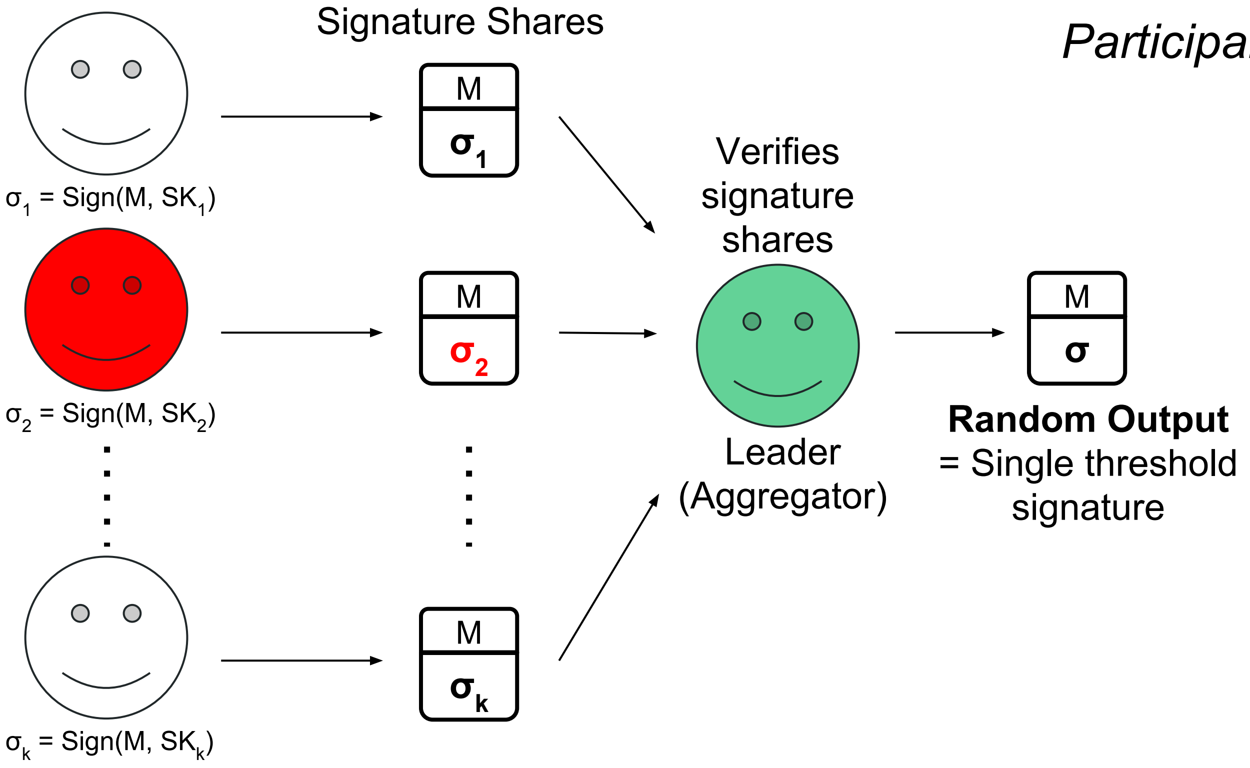


- **One** threshold signature
- **One** verification

Verify(σ, M, PK) = true

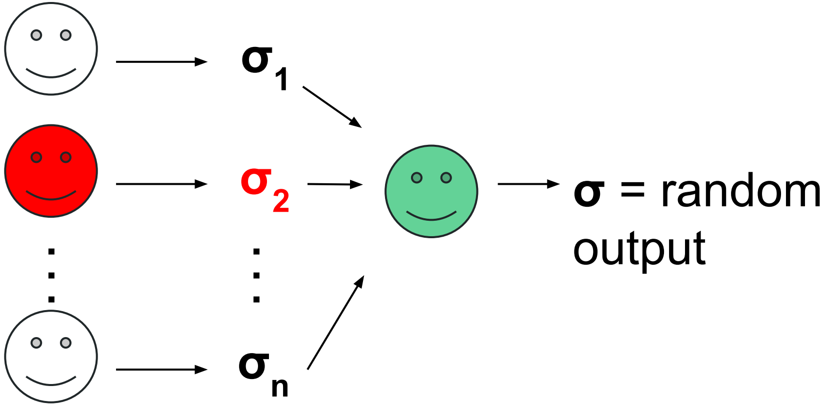
Random Beacon via Threshold Signatures

Participants sign M = current time.



Random Beacon Throughput

- Random beacon throughput = signature scheme throughput (assuming good network)
- High traffic at leader
- Multiple leaders ⇒ more throughput ⇒ more traffic :(



Random Beacon: Benefits of Threshold Signatures

Original Problems

- Last participant controls random output
- Dishonest participants refuse to reveal

Addressed using Threshold Signature Scheme

- Guaranteed to produce a signature, as long as k of the total n servers are honest
- Each message has a *unique* threshold signature

But... We Want a Scalable Random Beacon!

- Servers can be compromised
- Crucial to have a very large set of servers
- Can we get a **scalable** threshold signature scheme?

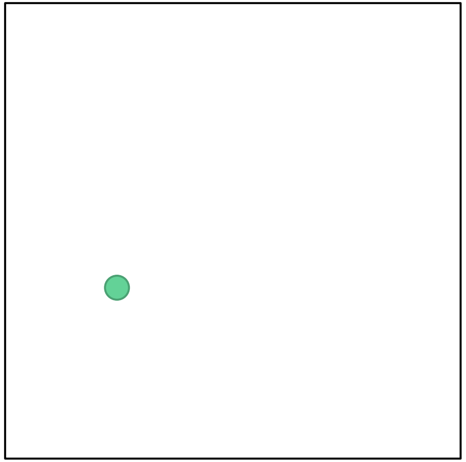
Shamir's Secret Sharing

- Recover secret given k shares

Shamir's Secret Sharing

● Recover secret given k shares

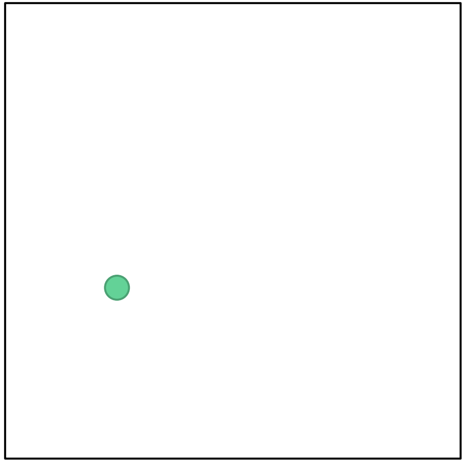
1 Point - Point



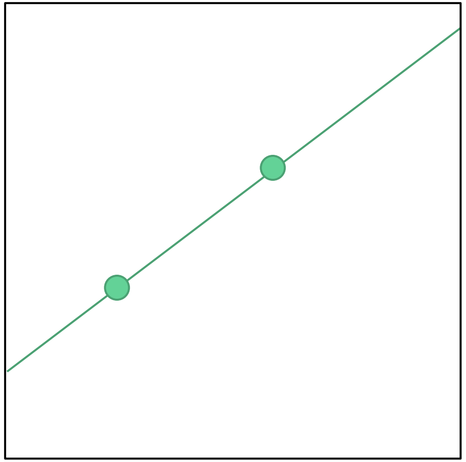
Shamir's Secret Sharing

- Recover secret given k shares

1 Point - Point



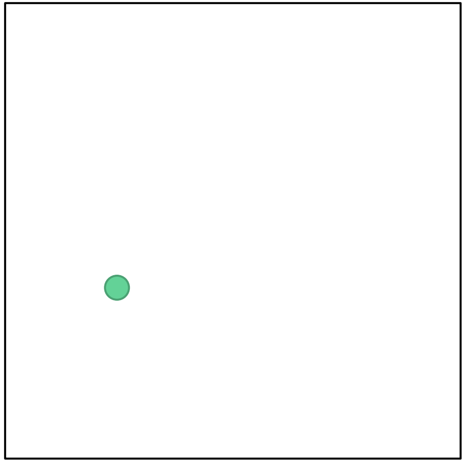
2 Points - Line



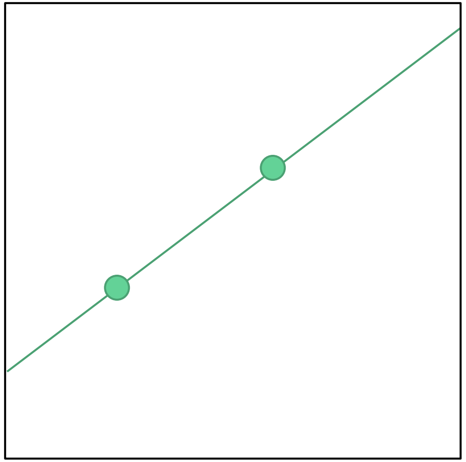
Shamir’s Secret Sharing

- Recover secret given k shares

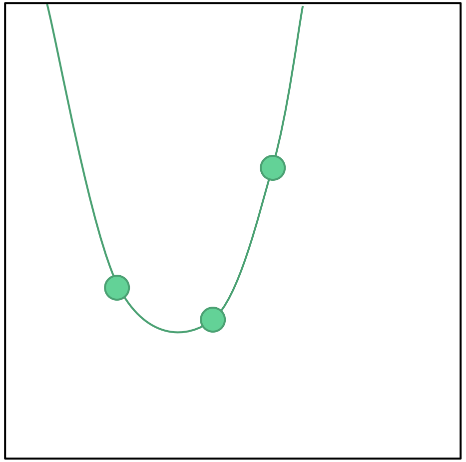
1 Point - Point



2 Points - Line



3 Points - Quadratic



Lagrange Interpolation for Secret Sharing

Current implementations are **inefficient**

- Given k points, takes **$O(k^2)$ time** to recover secret

We use some known mathematical tricks to speed this up to **$O(k \log^2 k)$ time**

Net result: We can aggregate a threshold signature from 100,000 participants in **20 seconds** rather than **13 minutes**.

Our Results: Scalable Threshold Signatures

Implementation Details:
Implemented in C++
Used libff and libntl

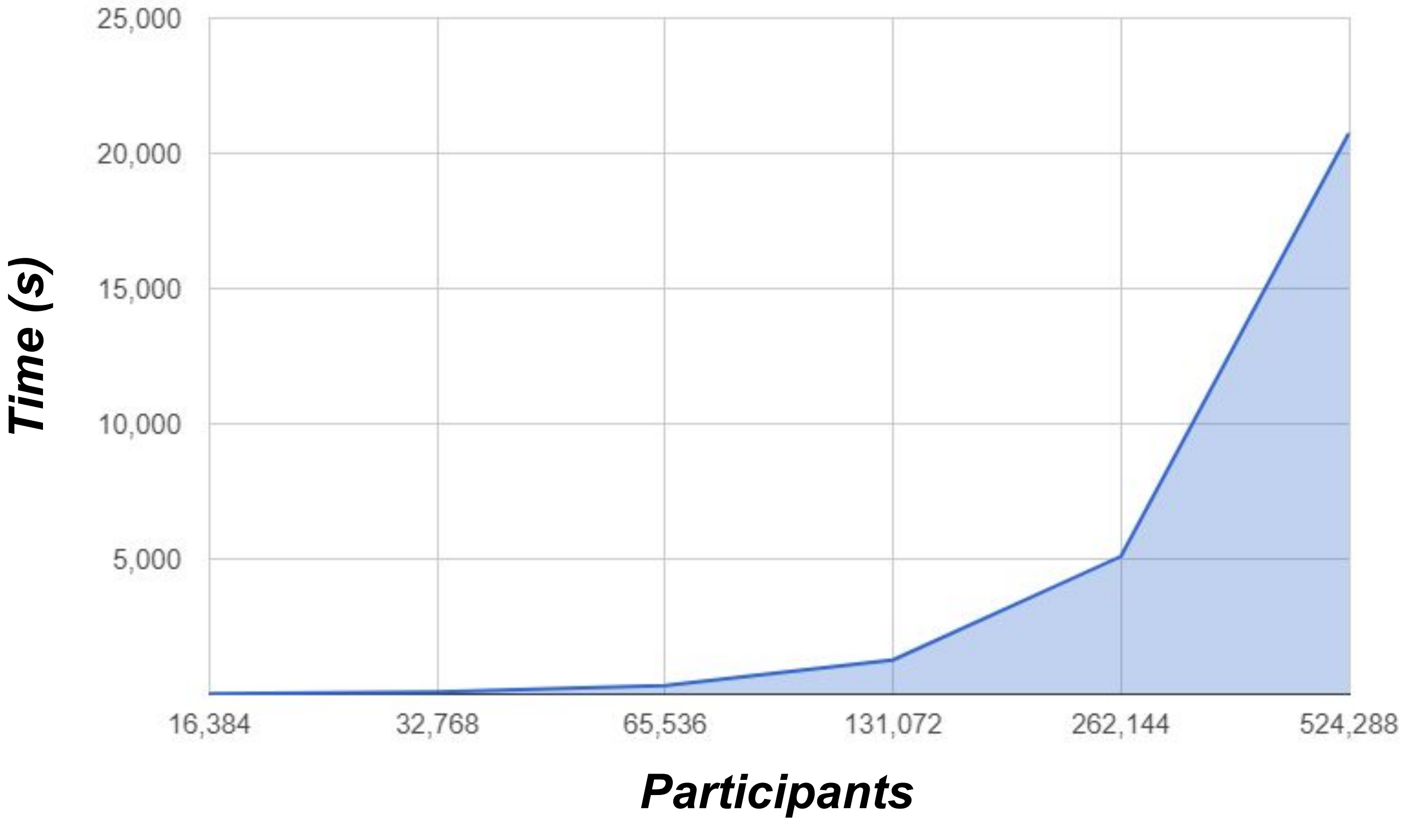
Our Results: Scalable Threshold Signatures

Implementation Details:
Implemented in C++
Used libff and libntl

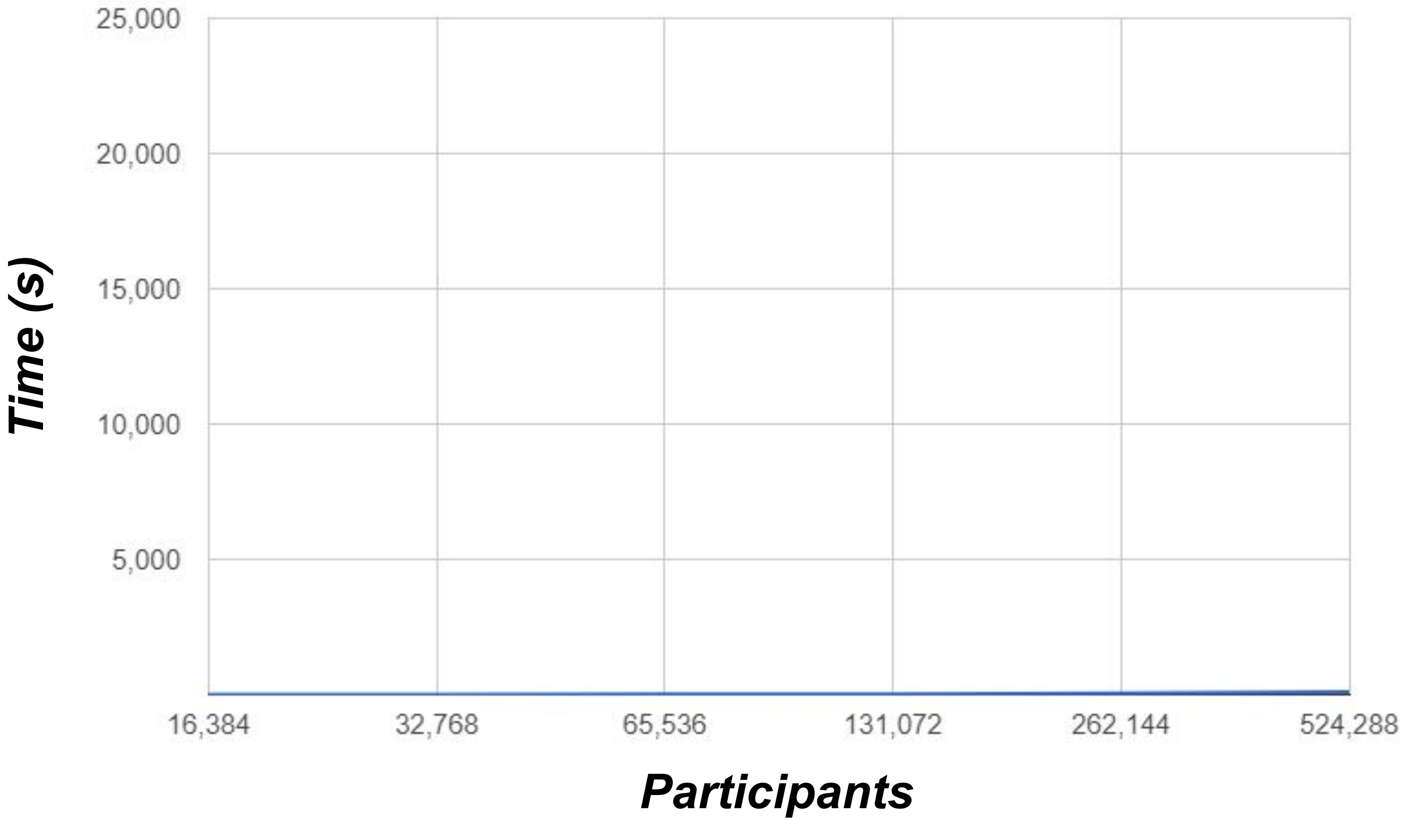
Machine Details:
ASUS ZenBook
Core i7-8550U CPU @ 1.80Ghz
16 GB of RAM

Ubuntu 16.04.5 LTS running inside VirtualBox 5.2.18 r124319

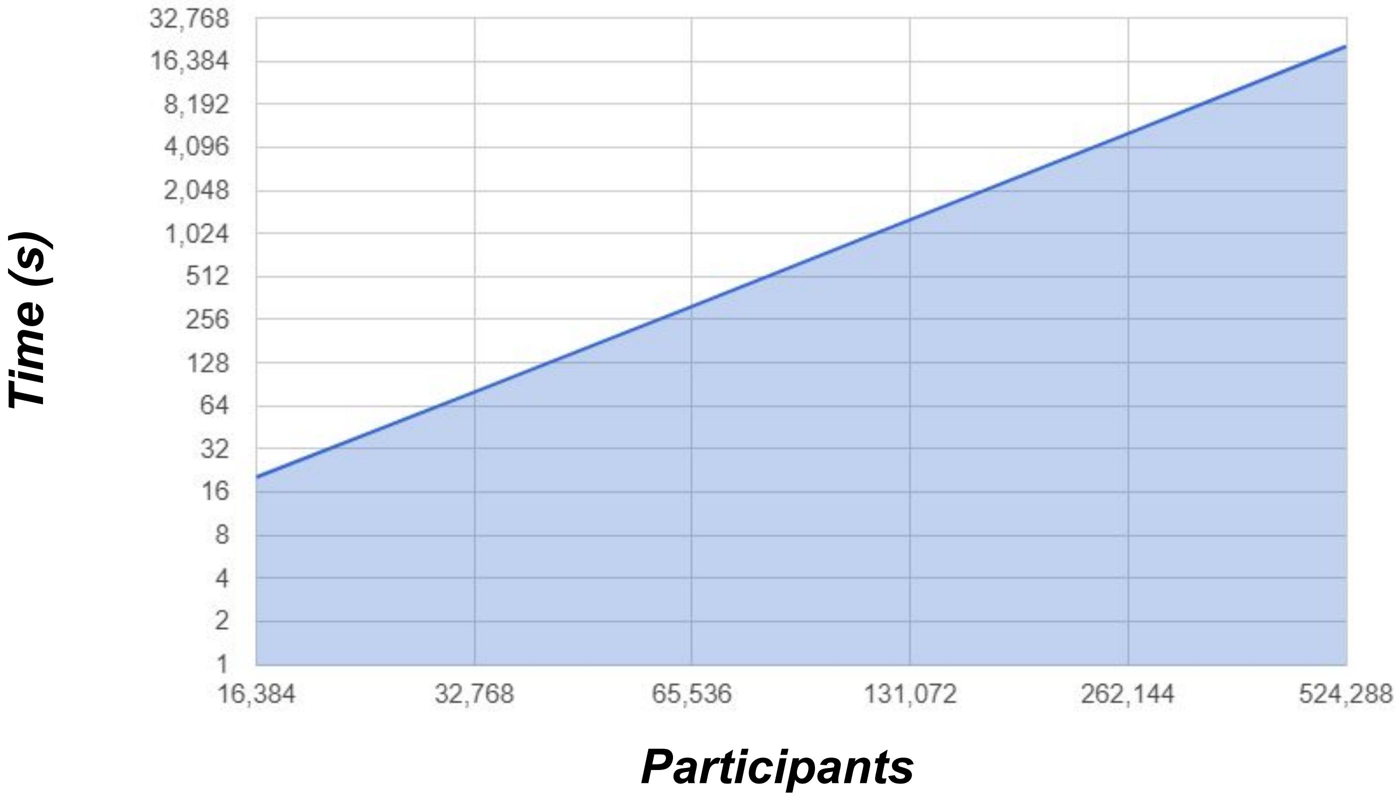
$O(k^2)$ Naive Aggregation Time



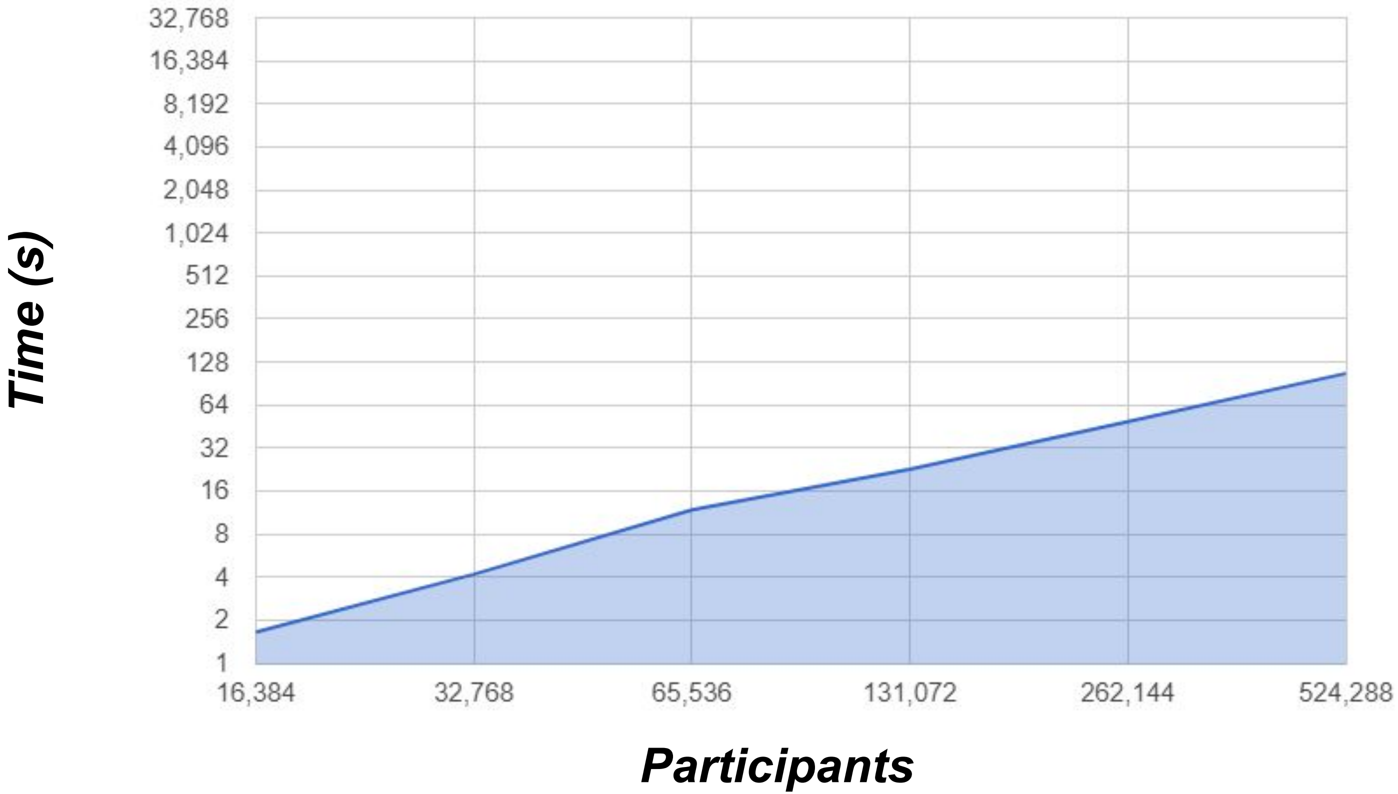
O(k log^2 k) Efficient Aggregation Time



O(k^2) Naive Aggregation Time



O(k log^2 k) Efficient Aggregation Time



Threshold Signatures: Not just for Random Beacons

Applications to:

- Consensus algorithms (such as the one used by Bitcoin)
- Securing HTTPs (every time you access a webpage)

Future Work

Implement random beacon protocol

- Threshold signature implementation works

Verifying signature shares is computationally expensive

- We speed it up using batch verification
- Fast when almost all shares are valid, slow when many are not

More parallelization by decreasing traffic

- Optimistically guess subset of k honest servers

Acknowledgements

I would like to thank:

- My mentor, Alin Tomescu, for his support and guidance
- Srini Devadas, for coordinating CS-PRIMES
- My parents and family
- MIT-PRIMES program

Thank you!

Questions?