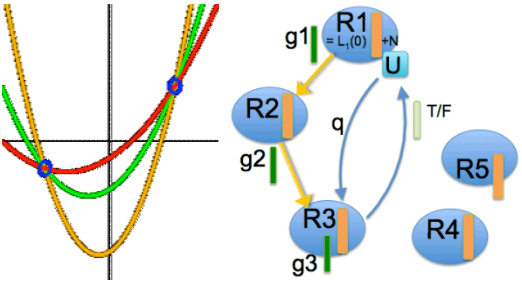
SAND2017-8294C



Secure Distributed Membership Tests via Secret Sharing

David Zage, david.zage@intel.com[1]

Helen Xu, hxu@sandia.gov

Thomas Kroeger, tmkroeg@sandia.gov

Bridger Hahn, bhahn@sandia.gov

Nolan Donoghue, npdonog@sandia.gov

Thomas Benson, thomas.benson@tufts.edu

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

1 Work performed while at Sandia National Laboratories

Providing Data Security and Availability?

Sandia National Laboratories

Data security and availability for operational use are frequently seen as fundamentally opposing forces.

Encryption is fragile:
Tools like homomorphic encryption are a start but most encryption algorithms are at best assumed to be secure, and often in reality just delayed release.

Secret Sharing Provides Provably Secure Systems
Archives that distribute data with secret sharing can provide information theoretic data protections and a resilience to:

1. malicious insiders,
2. compromised systems, and
3. untrusted components.

We are developing ways to functionally use secret shares without reassembly.

Background: Shamir Secret Sharing

Sandia National Laboratories

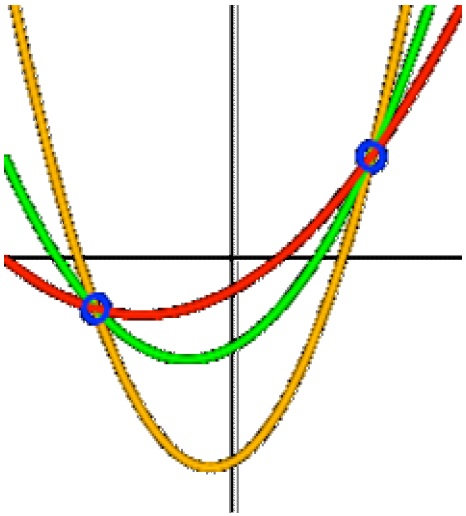
Shamir's Secret Sharing (SSS) provides a data protection that goes far beyond just splitting the data into parts. It is **information theoretically secure** and uses points on a polynomial curve to securely encode sensitive data.

Example:
Any 3 of 5 Secret $S = 1234$

1. Create a polynomial of degree 2 by generating 2 random coefficients
   $$f(x) = S + 166x + 94x^2$$
2. Generate 5 points along that curve:
   $$f(1)=1494;\ f(2)=1942;\ f(3)=2578;$$
   $$f(4)=3402;\ f(5)=4414$$

Any three points enables a user to solve for S. With one or two points you know nothing more than when you had none; S is one of infinite possible Y-intercepts.

NOTE: S = *f(0)*

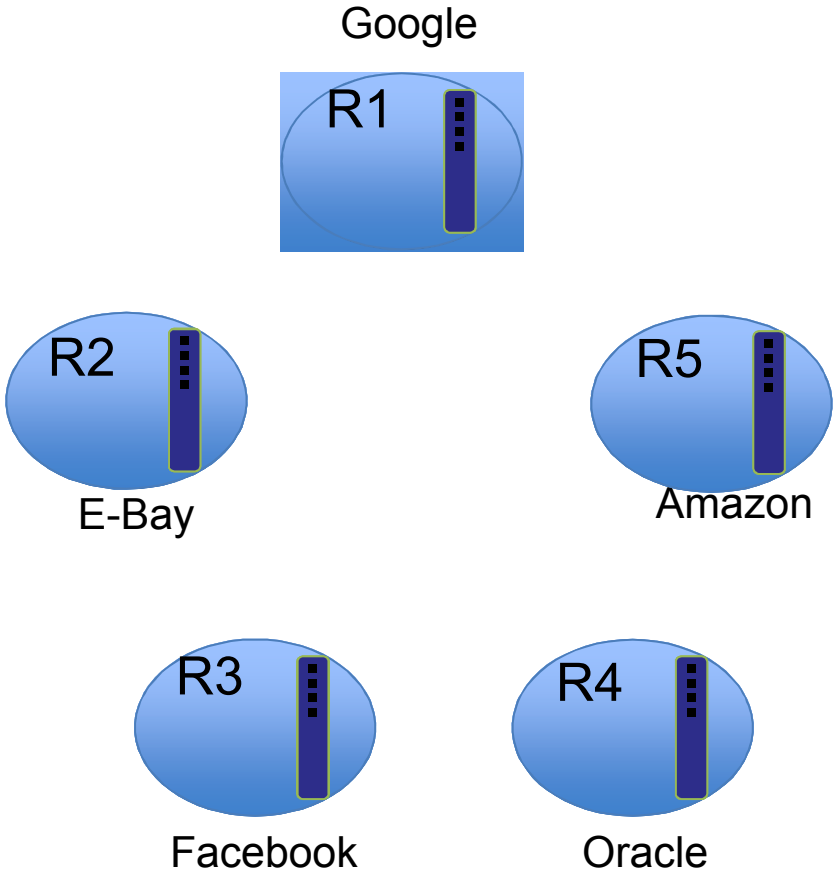


An infinite number of polynomials of degree 2 exist through 2 points.

Distributed Archive Using Secret Sharing

5 Companies want to share a list of bad IP address securely so that if any one has a breach the list isn't exposed.

Each company hosts a repository for points on the curves.



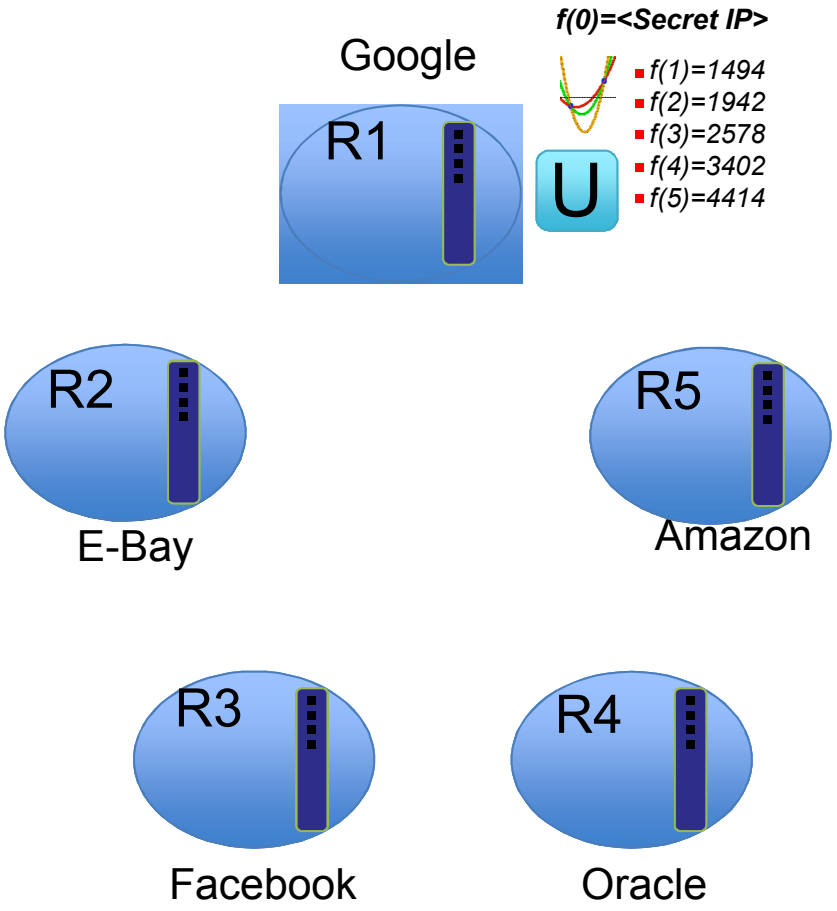
Distributed Archive Using Secret Sharing

5 Companies want to share a list of bad IP address securely so that if any one has a breach the list isn't exposed.

Each company hosts a repository for points on the curves.

User (**U**) at Google wants to insert:

1) U encodes its new IP address as 5 points on a polynomial curve.



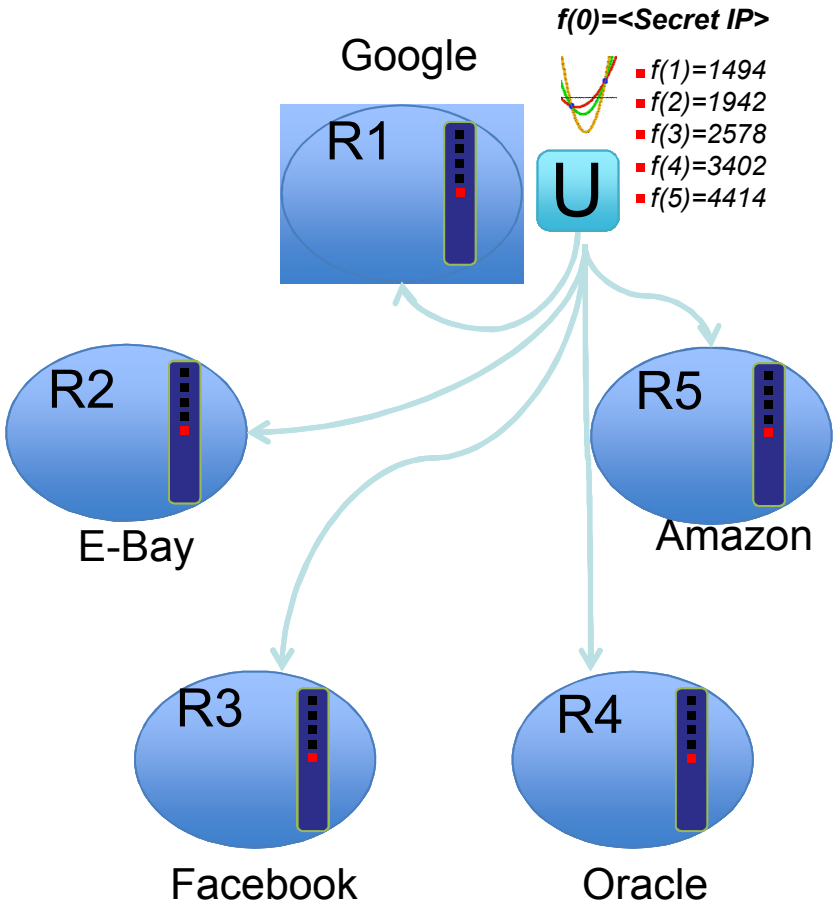
Distributed Archive Using Secret Sharing

5 Companies want to share a list of bad IP address securely so that if any one has a breach the list isn't exposed.

Each company hosts a repository for points on the curves.

User (**U**) at Google wants to insert:

1) U encodes its new IP address as 5 points on a polynomial curve.
2) U sends a point to each repository, e.g. R2 (2, 1942) R3 (3, 2578)…



Distributed Archive Using Secret Sharing

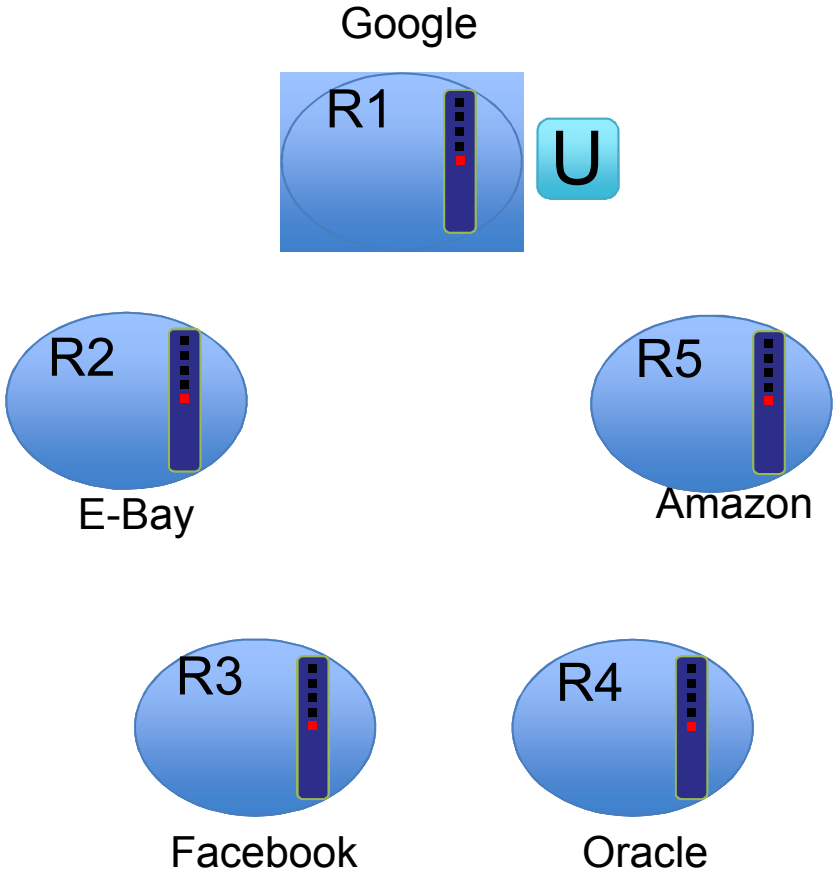
5 Companies want to share a list of bad IP address securely so that if any one has a breach the list isn't exposed.

Each company hosts a repository for points on the curves.

User (**U**) at Google wants to insert:

1) U encodes its new IP address as 5 points on a polynomial curve.
2) U sends a point to each repository, e.g. R2 (2, 1942) R3 (3, 2578)…

At this point if U erases that data it exists **NOWHERE**.



Distributed Archive Using Secret Sharing

Sandia National Laboratories

5 Companies want to share a list of bad IP address securely so that if any one has a breach the list isn't exposed.

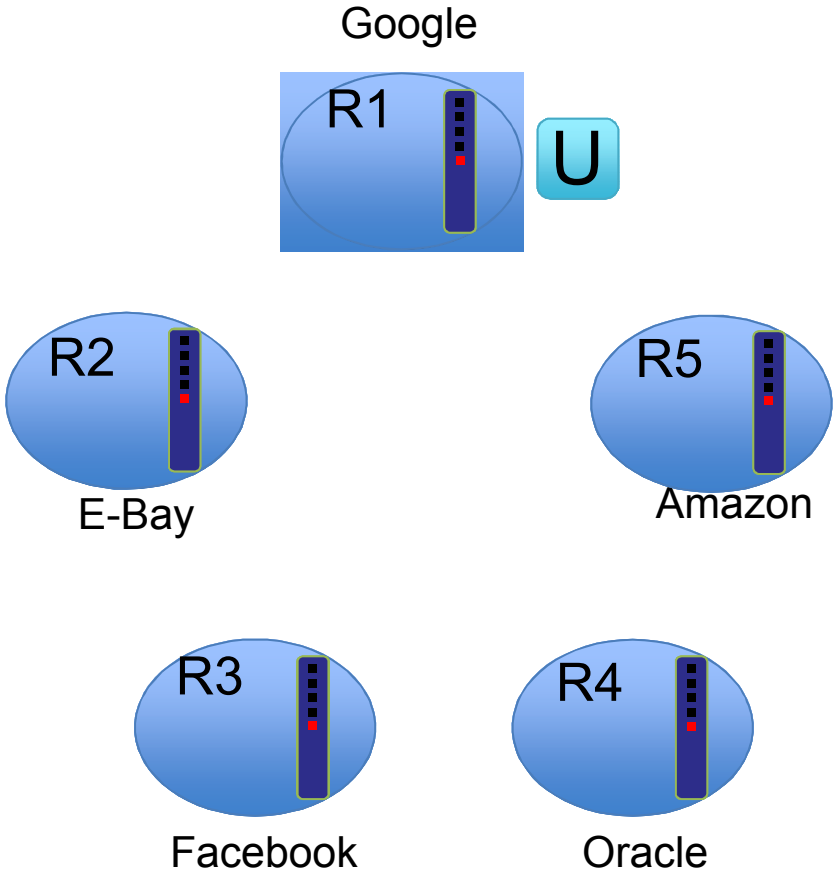
Each company hosts a repository for points on the curves.

User (**U**) at Google wants to insert:

1) U encodes its new IP address as 5 points on a polynomial curve.
2) U sends a point to each repository, e.g. R2 (2, 1942) R3 (3, 2578)…

At this point if U erases that data it exists **NOWHERE**.

BUT : To use this data, you would query 3 repositories and reassemble the actual list.



Lagrange Interpolation

Lagrange Interpolation allows for the recovery of the original function using a set of known points.
Given the set of points ($y_1 ... y_n$) we determine *f(j)* as:

$$f(j) = \sum_{i=1}^{n} L_i(j) \cdot y_i \qquad = \qquad L_1(0)y_1 + L_2(0)y_2 ... L_n(0)y_n$$

Where $L_i(j)$ is defined as follows:

$$L_i(j) = \prod_{\substack{1 \le m \le n \\ m \neq i}} \frac{(x - x_m)}{(x_i - x_m)} \qquad = \frac{(j - x_1)}{(x_i - x_1)} \cdots \frac{(j - x_{i-1})}{(x_i - x_{i-1})} \cdot \frac{(j - x_{i+1})}{(x_i - x_{i+1})} \cdots \frac{(j - x_T)}{(x_i - x_T)}$$

Everything to compute $L_i()$ is publicly know e.g. $X_1=1$

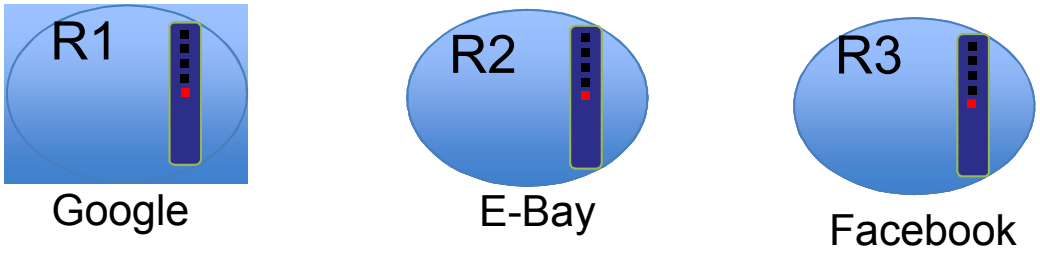
Our secrets are in the values of $y_1 \ldots . y_n$ which is what we must protect.

Serial Interpolation

Using Lagrange Interpolation serially across the archive, we can use our data while preserving information theoretic data protections.

For example we could calculate S serially as follows:

$$S = f(0) = \quad L_1(0) \cdot y_1 \quad + \quad L_2(0) \cdot y_2 \quad + \quad L_3(0) \cdot y_3$$



Since our goal is to never see S ever recreated in one spot again we need to modify our calculations to protect the points ($y_1 \ldots y_n$) from being disclosed and protect S from being the result.

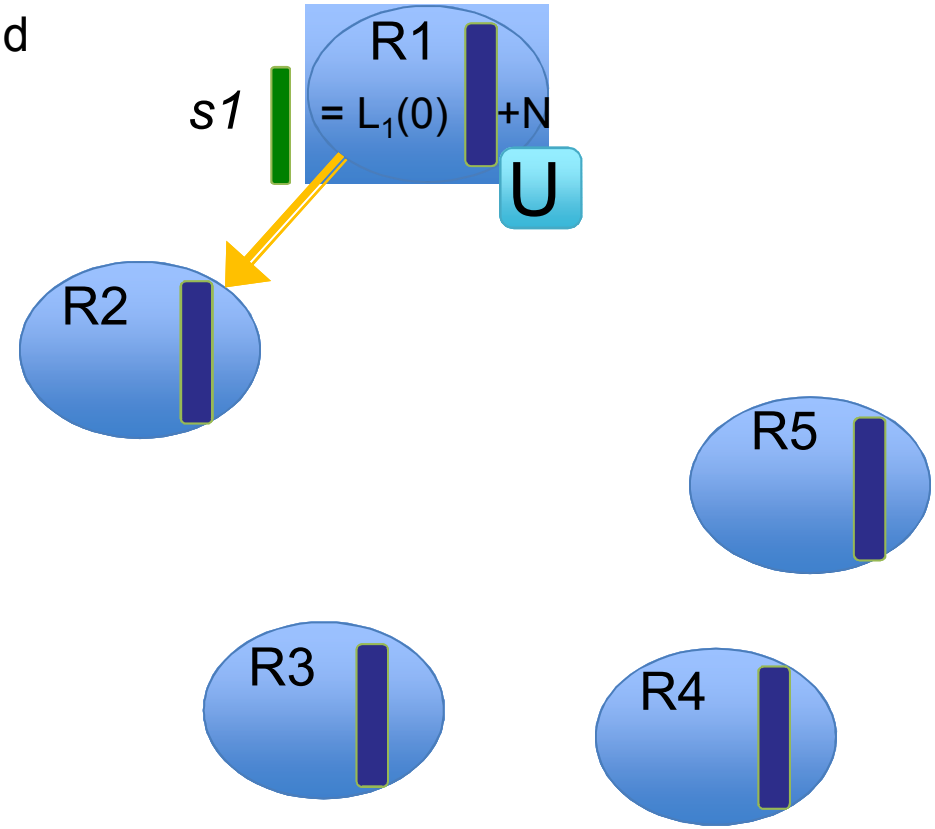
To do this we mask our calculations with a Nonce.

Example SIF use

User U, wants to test if address Z is on our list of addresses

1) U creates a nonce N, a privately held random number.

2) U & R1 calculate $s1 = L_1(x)f(1) + N$ and send *s1* to R2



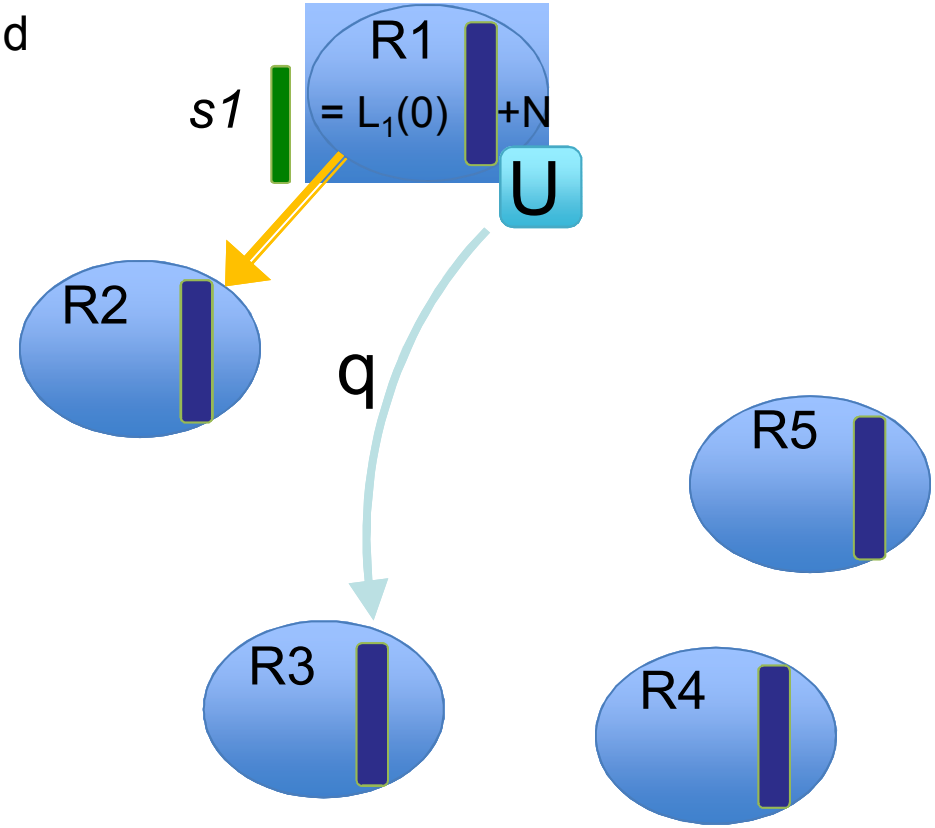
Example SIF use

User U, wants to test if address Z is on our list of addresses

1) U creates a nonce N, a privately held random number.

2) U & R1 calculate $s1 = L_1(x)f(1) + N$ and send *s1* to R2

3) U sends *q=Z+N to R3*



Example SIF use

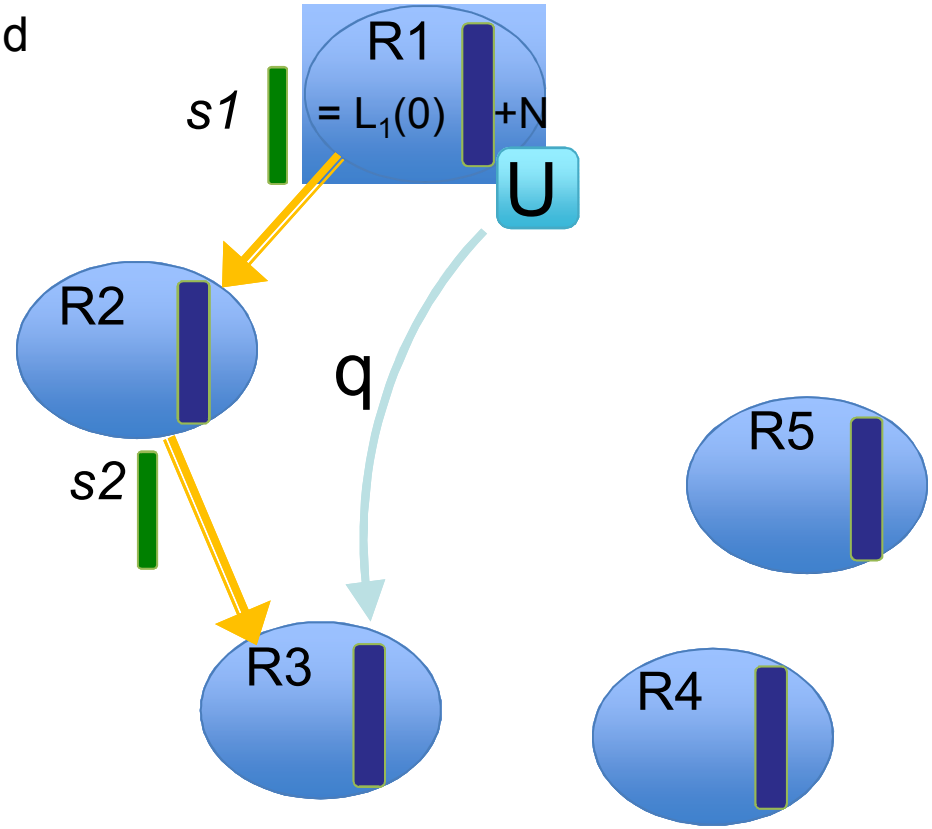
User U, wants to test if address Z is on our list of addresses

1) U creates a nonce N, a privately held random number.

2) U & R1 calculate $g1 = L_1(x)f(1) + N$ and send $g1$ to R2

3) U sends $q=Z+N$ *to R3*

4) R2 calculates $s2 = L_2(x)f(2) + s1$ sends $s2$ to R3



Example SIF use

User U, wants to test if address Z is on our list of addresses

1) U creates a nonce N, a privately held random number.

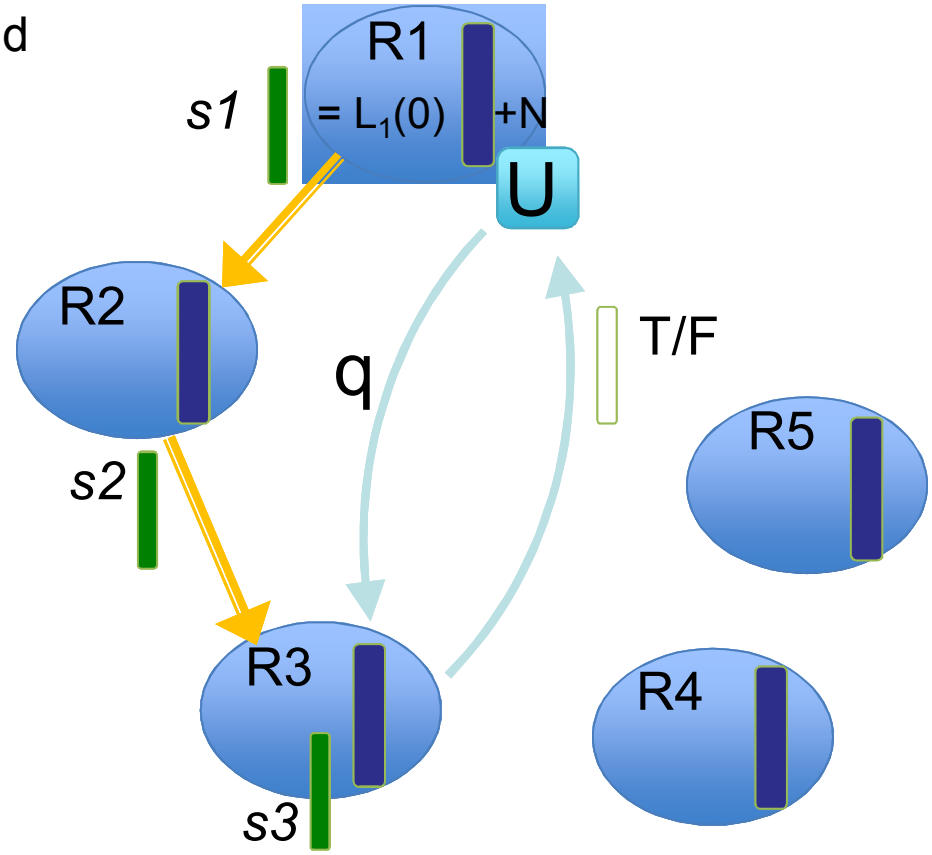
2) U & R1 calculate $s1 = L_1(0)f(1) + N$ and send $s1$ to R2

3) U sends $q=Z+N$ to R3

4) R2 calculates $s2 = L_2(0)f(2) + s1$ sends $s2$ to R3

5) R3 calculates $s3 = L_3(0)f(3) + s2$

6) R3 For all entries where $s3==q$
send TRUE to U
else
send FALSE to U



Security Analysis

Sandia National Laboratories

Model: Honest-but-curious participants

- Participants send correct protocol responses
- Perform extra calculations
- Not allowed to aggregate information not normally seen (no collusion)

By SSS, each repository is unable to calculate other shares

- Not enough points on the curve

By nonces, each partial share is protected

- In essence, a one-time pad

However, collusion during reconstruction can allow for data exposure.

Byzantine Adversaries

Sandia National Laboratories

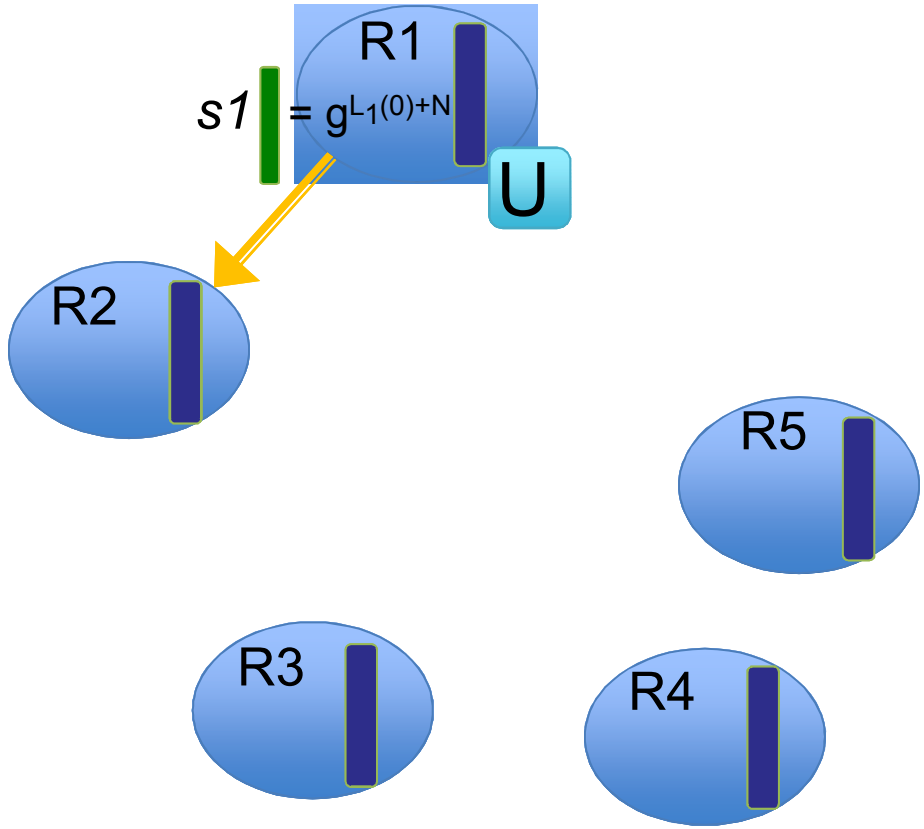
- Assumes less than k adversarial nodes
  - Behave arbitrarily and are only limited by cryptographic constraints
- We consider a (multiplicative) cyclic group C of order p
  - Assume discrete logarithm problem is hard
- We propose cSIF, which is secure given at most k-1 adversaries

Example cSIF use

User U, wants to test if address Z is on our list of addresses

1) U creates a nonce N, a privately held random number.

2) U & R1 calculate $s1 = g^{L_1(x)f(1) + N}$ and send *s1* to R2



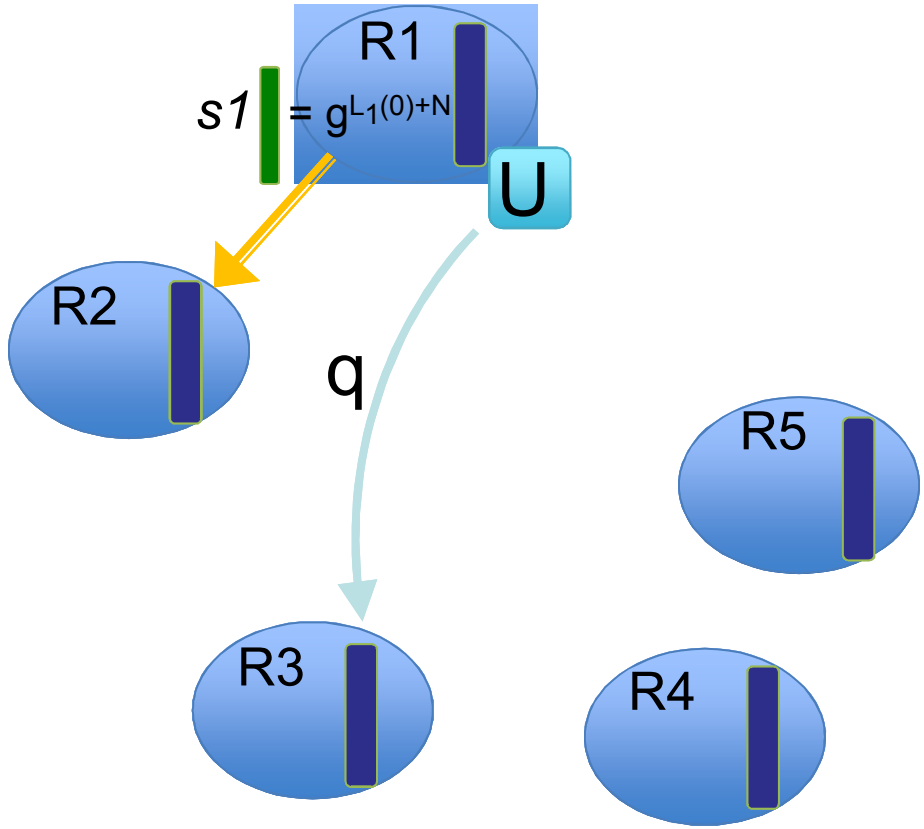
Example cSIF use

User U, wants to test if address Z is on our list of addresses

1) U creates a nonce N, a privately held random number.

2) U & R1 calculate $s1 = g^{L_1(x)f(1) + N}$ and send *s1* to R2

3) U sends $q=g^{Z+N}$ *to R3*



Example cSIF use

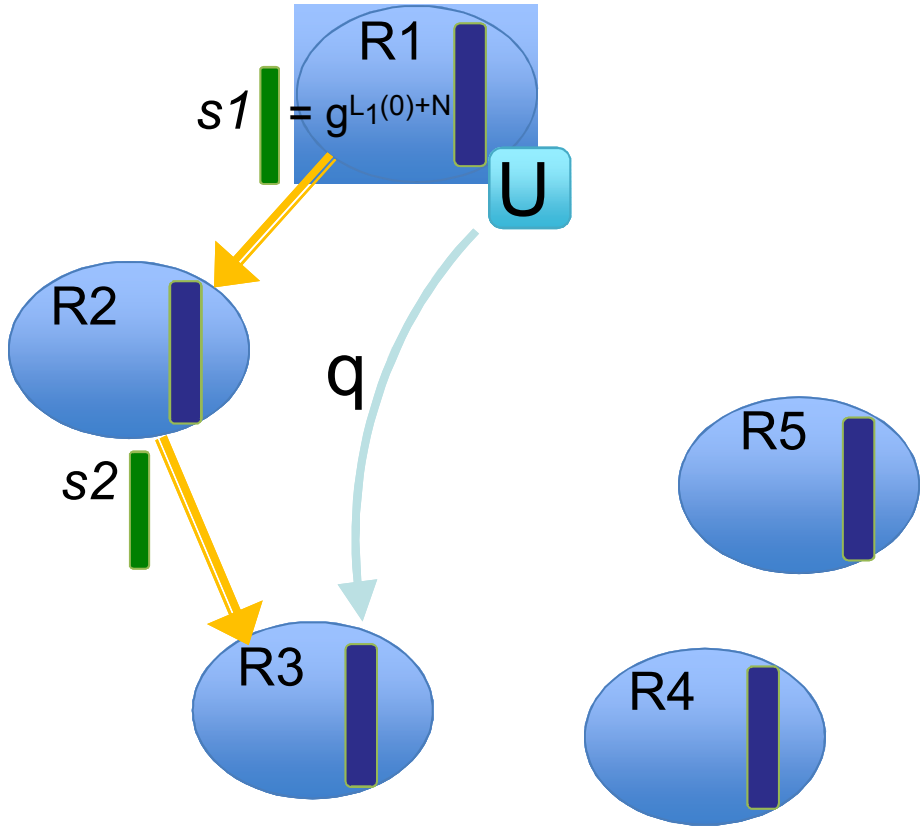
User U, wants to test if address Z is on our list of addresses

1) U creates a nonce N, a privately held random number.

2) U & R1 calculate $s1 = g^{L_1(x)f(1) + N}$ and send *s1* to R2

3) U sends $q=g^{Z+N}$ *to R3*

4) R2 calculates $s2 = g^{L_2(x)f(2)} \circ s1$ sends *s2* to R3



Example cSIF use

User U, wants to test if address Z is on our list of addresses

1) U creates a nonce N, a privately held random number.

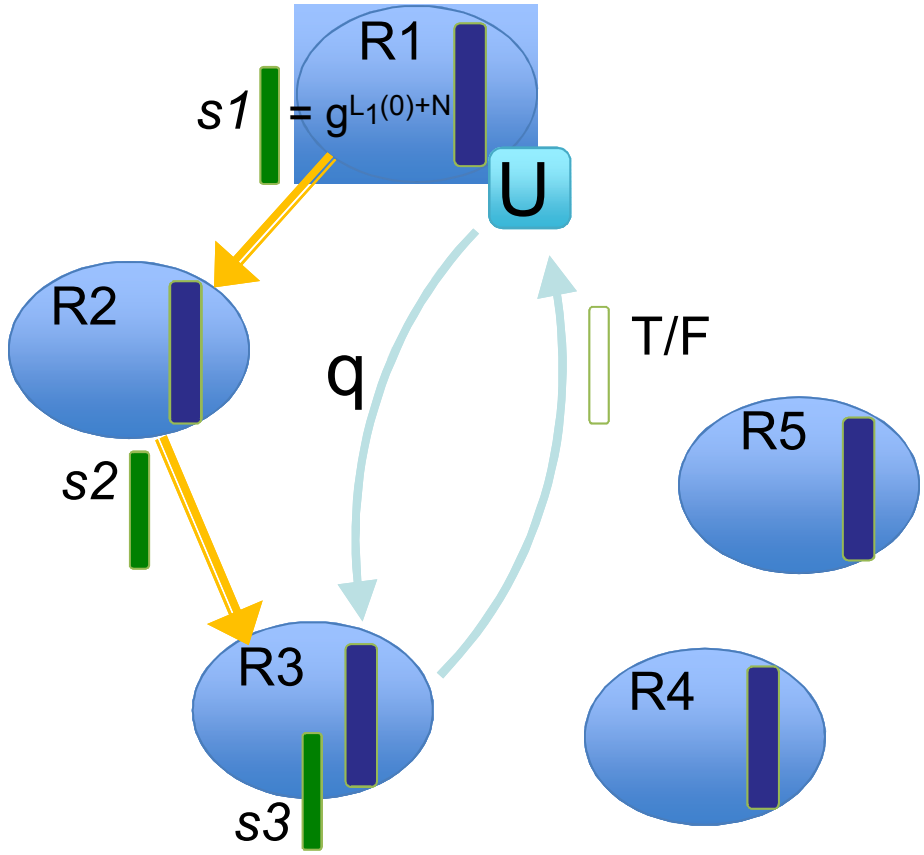
2) U & R1 calculate $s1 = g^{L_1(x)f(1) + N}$ and send *s1* to R2

3) U sends $q=g^{Z+N}$ *to R3*

4) R2 calculates $s2 = g^{L_2(x)f(2)} \circ s1$ sends *s2* to R3

5) R3 calculates $s3 = g^{L_3(x)f(3)} \circ + s2$

6) R3 For all entries where *s3==q*
send TRUE to U
else
send FALSE to U



Byzantine Security Analysis

Sandia National Laboratories

- Assuming the discrete log is hard, colluding repositories cannot recover the original share values.
- Other security guarantees are covered by SSS

Conclusions

- Usable security is challenging
  - Data security and availability seen as opposing forces
- SIF and cSIF provide mechanisms for set membership using secured and distribute data without revealing the original
  - SIF defends against Honest-but-curios adversaries
  - cSIF defends against Byzantine adversaries
- Looking to extend SIF to provide information theoretic protection against Byzantine adversaries