RSAConference2018
San Francisco | April 16 – 20 | Moscone Center

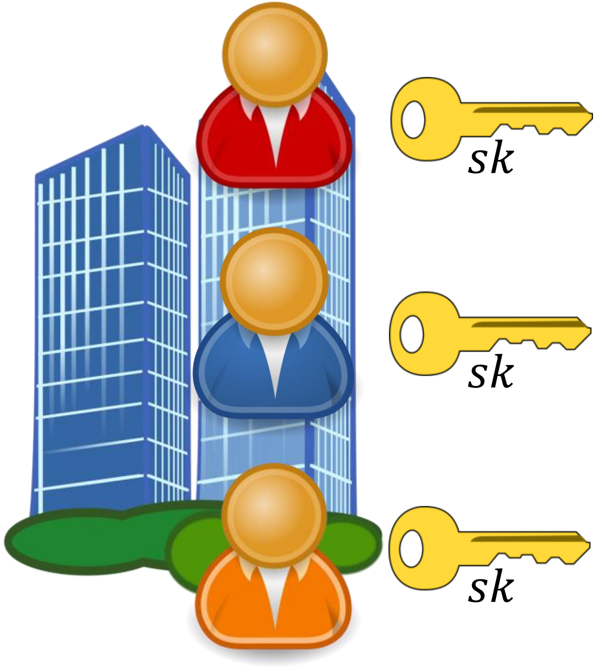
SESSION ID: CRYP-W04

CRYPTOGRAPHIC PROTOCOLS: PRACTICAL REVOCATION AND KEY ROTATION

Adam Shull

Recent Ph.D. Graduate
Indiana University

Access revocation on the cloud

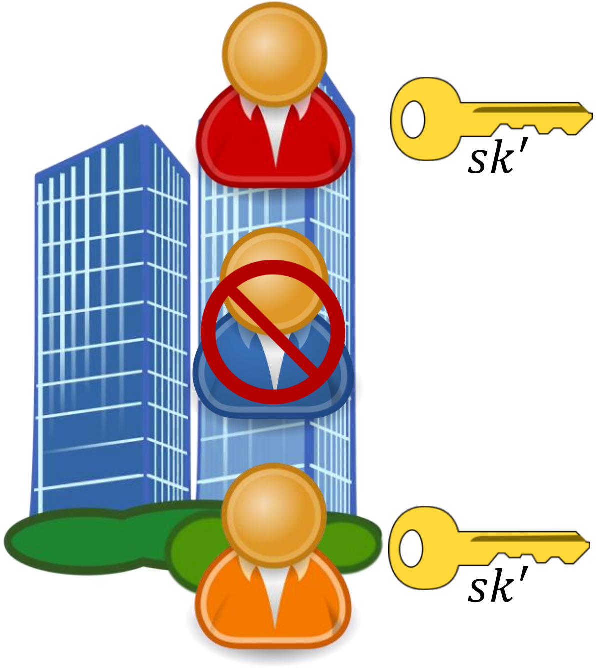


$$\left(\mathbf{Enc}_{pk}^{\mathrm{Pub}}(k_1), \mathbf{Enc}_{k_1}^{\mathrm{Sym}}(f_1)\right)$$

$$\left(\mathbf{Enc}_{pk}^{\mathrm{Pub}}(k_2), \mathbf{Enc}_{k_2}^{\mathrm{Sym}}(f_2)\right)$$

$$\left(\mathbf{Enc}_{pk}^{\mathrm{Pub}}(k_3), \mathbf{Enc}_{k_3}^{\mathrm{Sym}}(f_3)\right)$$

Access revocation on the cloud



$$\left(\mathbf{Enc}_{pk}^{\mathrm{Pub}}(k_1), \mathbf{Enc}_{k_1}^{\mathrm{Sym}}(f_1)\right)$$

$$\left(\mathbf{Enc}_{pk}^{\mathrm{Pub}}(k_2), \mathbf{Enc}_{k_2}^{\mathrm{Sym}}(f_2)\right)$$

$$\left(\mathbf{Enc}_{pk}^{\mathrm{Pub}}(k_3), \mathbf{Enc}_{k_3}^{\mathrm{Sym}}(f_3)\right)$$

Revocation using proxy re-encryption

$$\left(\boxed{\mathbf{Enc}^{\mathrm{Pub}}_{pk}(k_1)}, \mathbf{Enc}^{\mathrm{Sym}}_{k_1}(f_1)\right)$$

$$\left(\boxed{\mathbf{Enc}^{\mathrm{Pub}}_{pk}(k_2)}, \mathbf{Enc}^{\mathrm{Sym}}_{k_2}(f_2)\right)$$

$$\left(\boxed{\mathbf{Enc}^{\mathrm{Pub}}_{pk}(k_3)}, \mathbf{Enc}^{\mathrm{Sym}}_{k_3}(f_3)\right)$$

Proxy re-encrypt from pk to pk'

$$\left(\boxed{\mathbf{Enc}^{\mathrm{Pub}}_{pk'}(k_1)}, \mathbf{Enc}^{\mathrm{Sym}}_{k_1}(f_1)\right)$$

$$\left(\boxed{\mathbf{Enc}^{\mathrm{Pub}}_{pk'}(k_2)}, \mathbf{Enc}^{\mathrm{Sym}}_{k_2}(f_2)\right)$$

$$\left(\boxed{\mathbf{Enc}^{\mathrm{Pub}}_{pk'}(k_3)}, \mathbf{Enc}^{\mathrm{Sym}}_{k_3}(f_3)\right)$$

Key-scraping attack

$$\left(\mathbf{Enc}_{pk}^{\mathrm{Pub}}(k_1), \mathbf{Enc}_{\boxed{k_1}}^{\mathrm{Sym}}(f_1)\right) \qquad \left(\mathbf{Enc}_{pk'}^{\mathrm{Pub}}(k_1), \mathbf{Enc}_{\boxed{k_1}}^{\mathrm{Sym}}(f_1)\right)$$

The symmetric keys for the files are not changed!

$$\left(\mathbf{Enc}_{pk}^{\mathrm{Pub}}(k_2), \mathbf{Enc}_{\boxed{k_2}}^{\mathrm{Sym}}(f_2)\right) \qquad \left(\mathbf{Enc}_{pk'}^{\mathrm{Pub}}(k_2), \mathbf{Enc}_{\boxed{k_2}}^{\mathrm{Sym}}(f_2)\right)$$

A revoked user may have stored k_1, k_2, and k_3

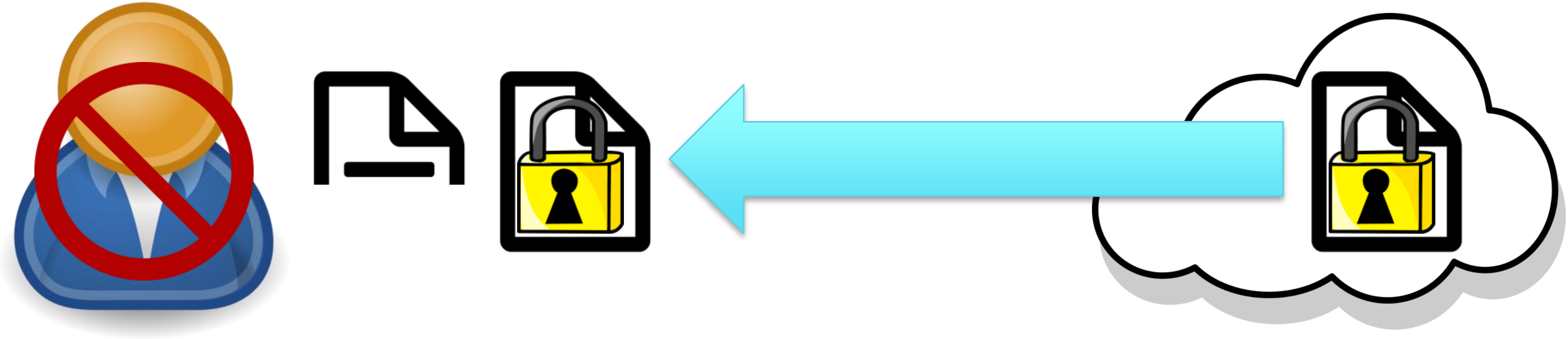
$$\left(\mathbf{Enc}_{pk}^{\mathrm{Pub}}(k_3), \mathbf{Enc}_{\boxed{k_3}}^{\mathrm{Sym}}(f_3)\right) \qquad \left(\mathbf{Enc}_{pk'}^{\mathrm{Pub}}(k_3), \mathbf{Enc}_{\boxed{k_3}}^{\mathrm{Sym}}(f_3)\right)$$

The symmetric key must be changed!

- Decrypt with old key, encrypt with new key
  - Requires trusted re-encryptor and takes two full passes for re-encryption
- Encrypt existing ciphertext with new key
  - Decryption takes one full pass for each previous re-encryption
- Key-homomorphic pseudorandom functions
  - Allow untrusted party to re-encrypt to fresh key
  - Existing key-homomorphic pseudorandom functions are extremely slow

Security model

Security model



All-or-nothing transform (AONT)

0111101100101111010010011100100011001110001111010100010010010111

T

1010110000111111010010101011101001110110001010100010101001110000

T^{-1}

0111101100101111010010011100100011001110001111010100010010010111

All-or-nothing transform (AONT)

0111101100101111010010011100100011001110001111010100010010010111

$\Downarrow$

T

$\Downarrow$

1010110 001111 110100101010 11 10011101100010 01 0010101 0 11000

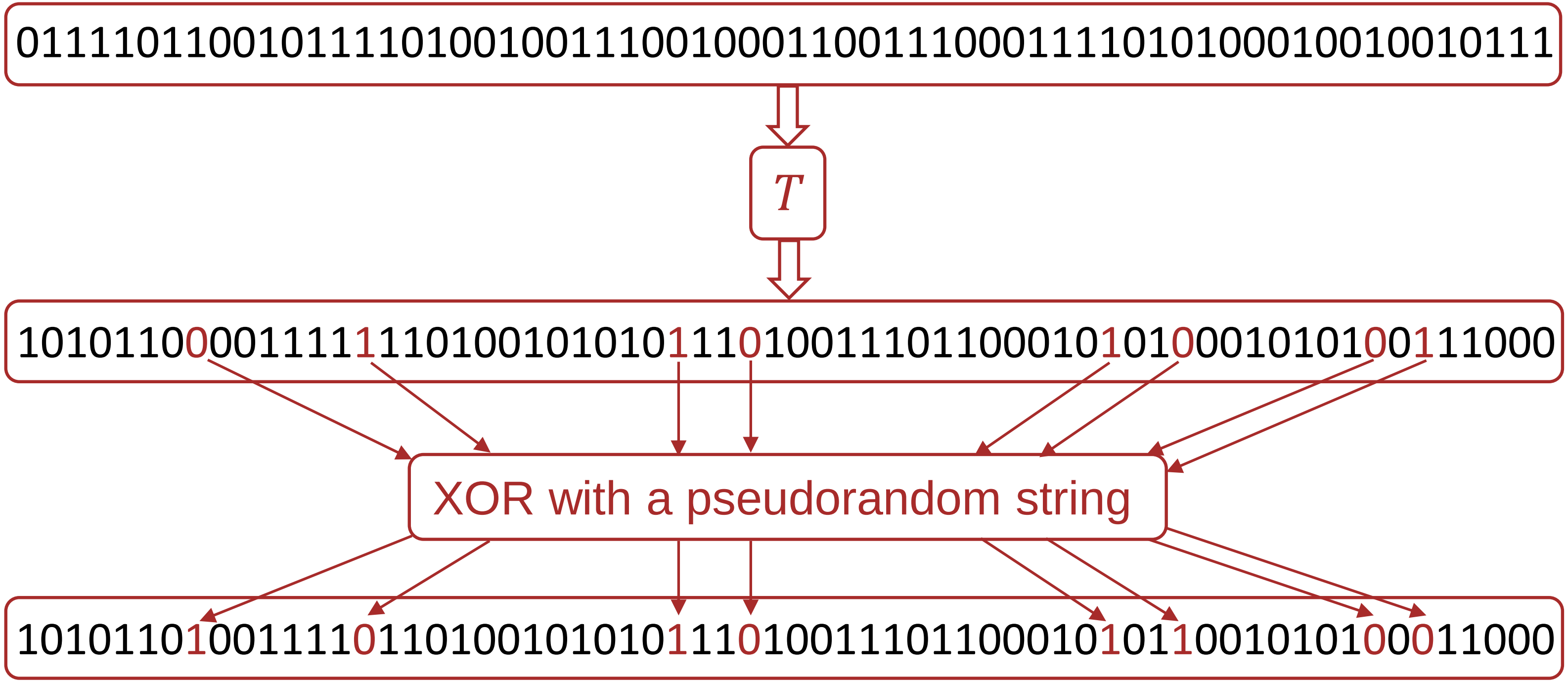
$\Downarrow$

T^{-1}

$\Downarrow$

?

Our approach using an AONT



Security intuition

1010110000111111101001010101110100111011000101010001010100111000

10101100001111111010010101011101

10101100001111111010010101011101

+

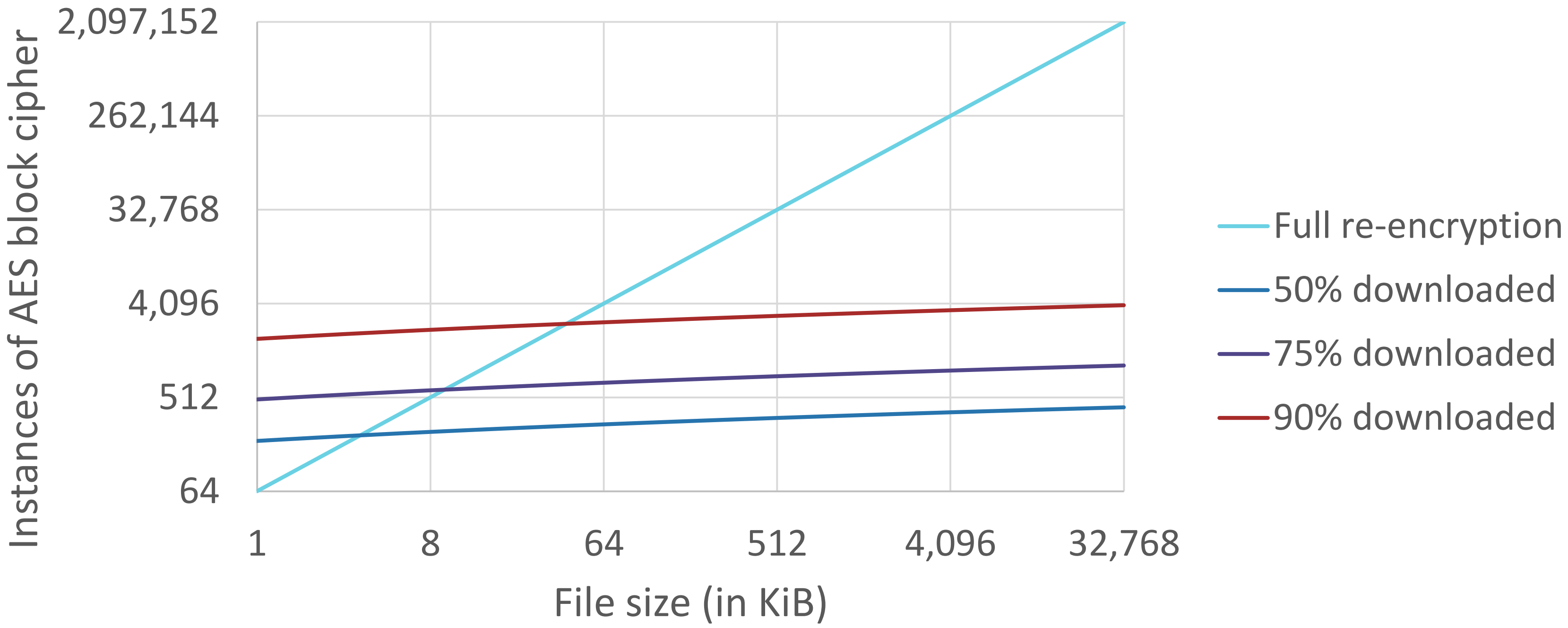
1010110100111101101001010101110100111011000101011001010100011000

1010110000111111101001010101110100111011000101011001010100011000

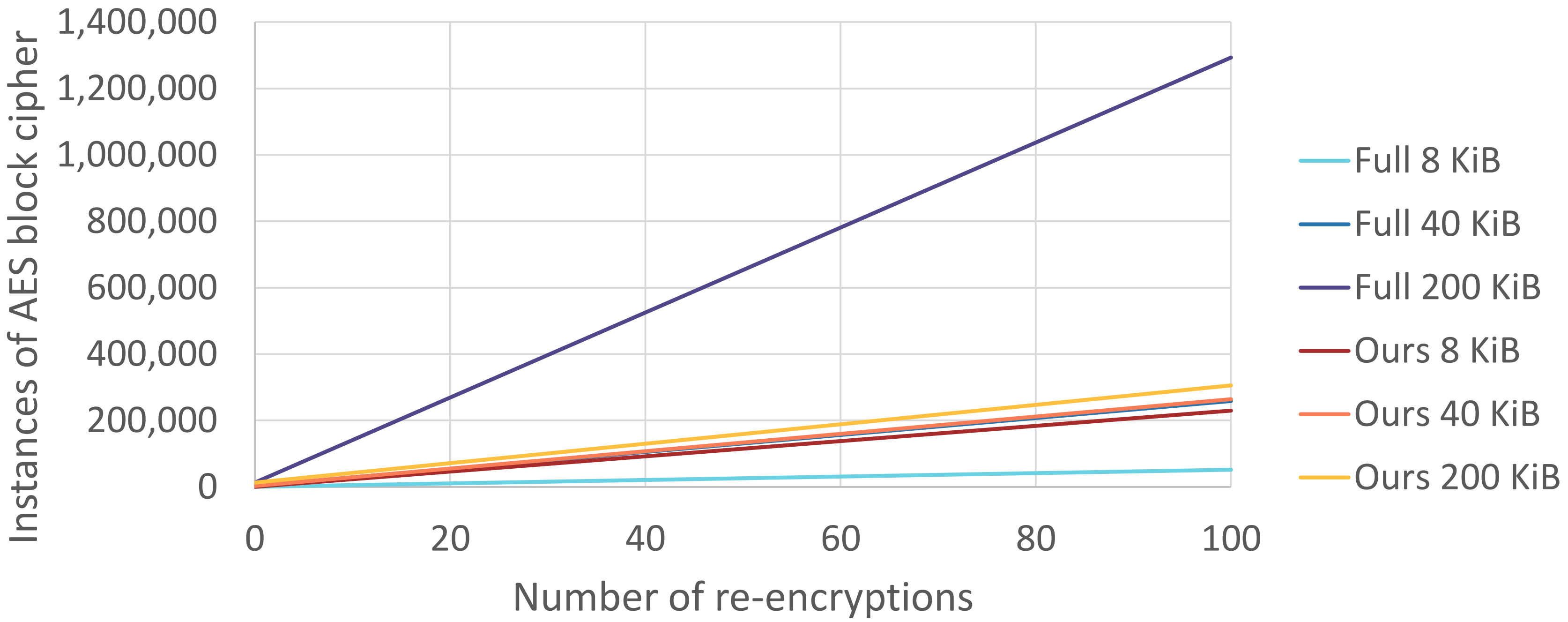
Proxy re-encryption construction

- Initial ciphertext: $\left(\mathbf{Enc}_{pk}(k_0), T\left(\mathbf{Enc}_{k_0}^{\mathrm{Sym}}(f)\right)\right)$
- Once re-encrypted ciphertext:
$$\left(\mathbf{Enc}_{pk'}(k_0), \left[\mathbf{Enc}_{pk'}(s_1, k_1)\right], \left[T\left(\mathbf{Enc}_{k_0}^{\mathrm{Sym}}(f)\right)\right]_{\mathrm{Ind}(s_1),\mathrm{Ctr}(k_1)}\right)$$
- Twice re-encrypted ciphertext:
$$\left(\begin{array}{l}\mathbf{Enc}_{pk''}(k_0), \left[\mathbf{Enc}_{pk''}(s_1, k_1), \mathbf{Enc}_{pk''}(s_2, k_2)\right], \\ \left[\left[T\left(\mathbf{Enc}_{k_0}^{\mathrm{Sym}}(M)\right)\right]_{\mathrm{Ind}(s_1),\mathrm{Ctr}(k_1)}\right]_{\mathrm{Ind}(s_2),\mathrm{Ctr}(k_2)}\end{array}\right)$$

Result: Much faster re-encryption



Result: Much faster decryption



Summary

- In scenarios such as access revocation and key rotation, symmetric-key ciphertexts may need to be re-encrypted
- Existing solutions are either insecure or too slow to be used in practice
- Using an all-or-nothing transform, we can re-encrypt efficiently while achieving provable security under a reasonable model
- We provide constructions for updatable symmetric-key encryption, public-key and identity-based proxy re-encryption, and revocable-storage attribute-based encryption

Apply What You Have Learned

- Realize the need to update the symmetric key
  - Many papers on public-key revocation don’t consider hybrid encryption
  - Realistic security models must address key-scraping attacks
- Possible future work:
  - Produce a general theorem encompassing all uses of symmetric-key encryption
  - Assess tradeoffs between streaming efficiency and security
  - Provide a full implementation of the construction

RSAConference2018
San Francisco | April 16 – 20 | Moscone Center

MATTERS NOW

#RSAC

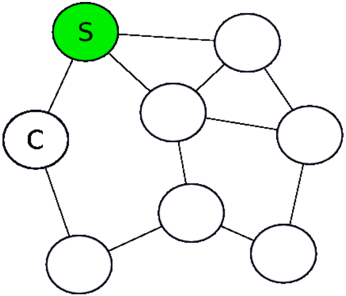
SESSION ID: CRYP-W04

ASYNCHRONOUS PROVABLY-SECURE HIDDEN SERVICES

Fernando Krell   Philippe Camacho

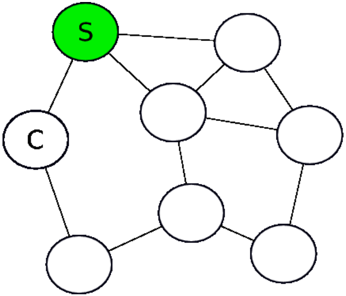
DREAMLAB TECHNOLOGIES

Problem: how to hide the location of a server?

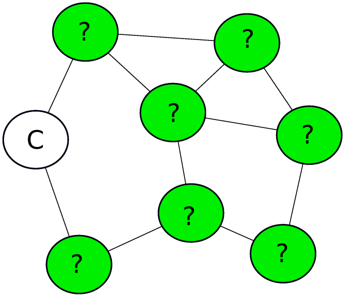


- Arbitrary network topology
- One node acts as a server
- Other nodes can be clients

Problem: how to hide the location of a server?

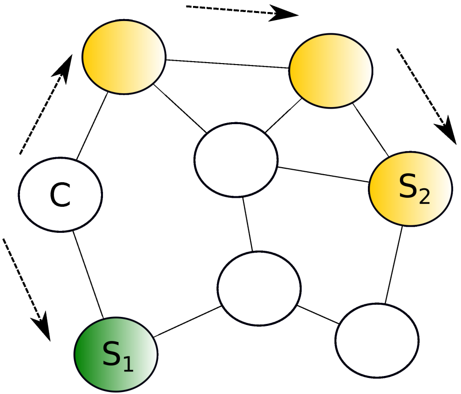


- Arbitrary network topology
- One node acts as a server
- Other nodes can be clients



- Avoid DoS
- Reduce attack surface
- Censorship resistance
- Traffic analysis

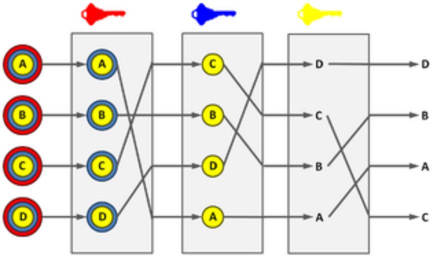
Naive Solution: Recursive Multicast



- If C contacts S_1, the response will arrive after $\approx 2T$
- If C contacts S_2, the response will arrive after $\approx 6T$

Anonymity: Synchronous Solutions

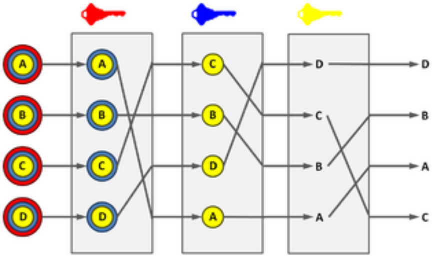
Mix-nets [Chaum '81]



- Provably secure

Anonymity: Synchronous Solutions

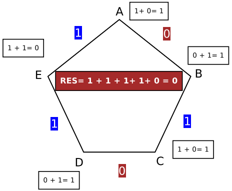
Mix-nets [Chaum '81]



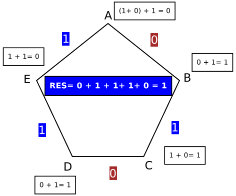
- Provably secure

DC-nets [Chaum '88]

A paid



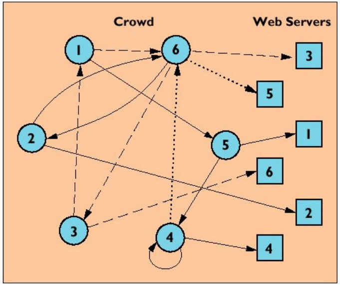
A did not pay



- Provably secure

Anonymity: Asynchronous Alternatives

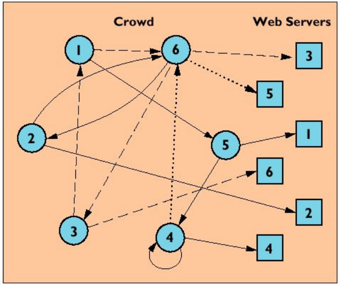
Crowds [Reiter & Rubin '98]



- Asynchronous
- Several attacks

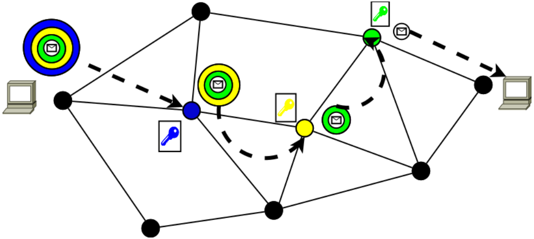
Anonymity: Asynchronous Alternatives

Crowds [Reiter & Rubin ’98]



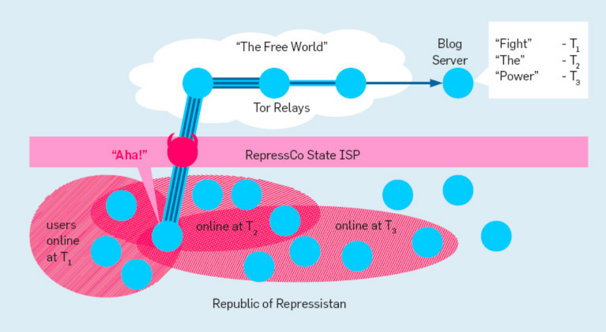
- Asynchronous
- Several attacks

Tor [Dingledine & Mathewson & Syverson ’04]



- Asynchronous
- Several attacks
- Most popular

Intersection attack
⇒ lower bound on communication



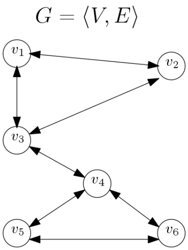
Thus all the nodes must participate in order to hide the server's location.

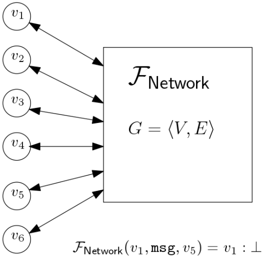
Can we get the best of both worlds (Provably Secure and Asynchronous)?

	Asynchronous	**Synchronous**
Provably Secure	This work	DC-nets/ mix-nets, DO'00
Heuristic Security	Tor,Crowds	Herbivore [GRPS '03]

Model

- Simulation based security definition.
- Communication restricted to use $\mathcal{F}_{\text{Network}}$
  - P_i is allowed to send message to P_j, if they are directly connected.





$\mathcal{F}_{\text{Network}}(v_1, \mathtt{msg}, v_5) = v_1 : \perp$

$\mathcal{F}_{\text{Network}}(v_1, \mathtt{msg}, v_2) = v_2 : \langle \mathsf{msg}, v_1 \rangle$

Overview of our solution

Participant's behavior is indistinguishable from server's.

1. **Client**. Broadcast the request
2. **Player** P_i. Upon seeing a request message, send a random value s_i to the server (broadcast)
3. **Player** P_i. Upon seeing everybody's values $\{s_j\}$:
   - **If** P_i **= Server**. Secret share response r using $\{s_j\}$.
     Send share $r - \sum s_j$ to client.
   - **Else**. Submit s_i to client.
4. **Client**. Upon receiving all shares, reconstruct the server's response r.

Overview of our solution

Participant's behavior is indistinguishable from server's.

1. **Client**. Broadcast the request
2. **Player** P_i. Upon seeing a request message, send a random value s_i to the server (broadcast)
3. **Player** P_i. Upon seeing everybody's values $\{s_j\}$:
   - **If** P_i **= Server**. Secret share response r using $\{s_j\}$.
     Send share $r - \sum s_j$ to client.
   - **Else**. Submit s_i to client.
4. **Client**. Upon receiving all shares, reconstruct the server's response r.

Naive implementation has $O(n^2)$ communication complexity.

Efficient Implementation

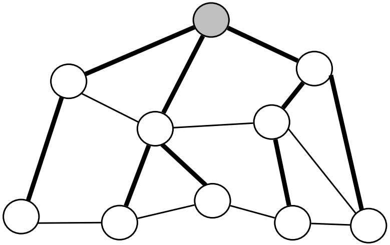
- Avoid recursive multicast on every message.
- Combine encrypted shares on intermediate nodes.

Efficient Implementation

- Avoid recursive multicast on every message.
- Combine encrypted shares on intermediate nodes.

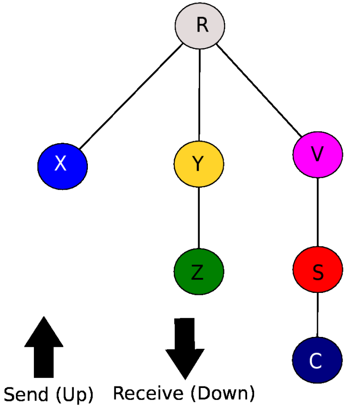
Extra Tools:

- **Homomorphic Encryption**
  - $\mathsf{Enc}_{\mathsf{pk}}(m_1) + \mathsf{Enc}_{\mathsf{pk}}(m_2) = \mathsf{Enc}_{\mathsf{pk}}(m_1 + m_2)$
- **Spanning Tree**

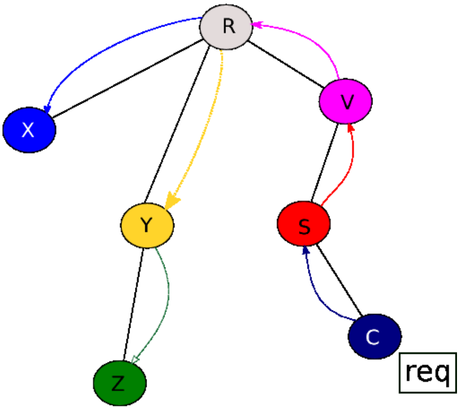


Communication Pattern

Avoiding quadratic complexity:

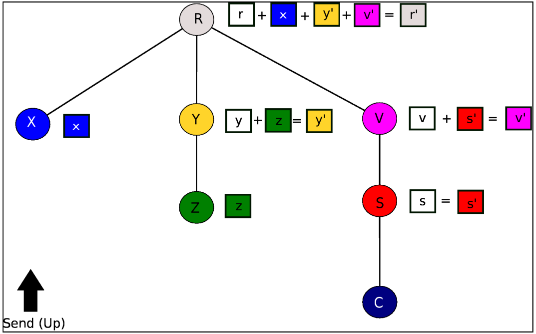


Phase 1: Broadcast the Request



Phase 2.a): Shares UP to root

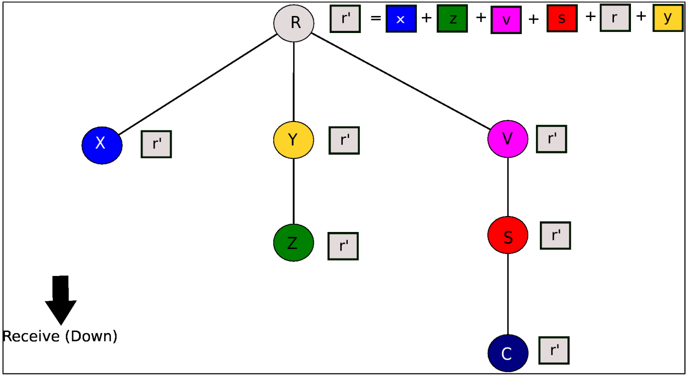
Shares are encrypted for the server, and sent up the tree.



All shares are added using homomorphic encryption:

$$\mathrm{Enc}_{\mathrm{pk}^{\mathcal{S}}}(\mathbf{y}) \cdot \mathrm{Enc}_{\mathrm{pk}^{\mathcal{S}}}(\mathbf{z}) = \mathrm{Enc}_{\mathrm{pk}^{\mathcal{S}}}(\mathbf{y}+\mathbf{z}) = \mathrm{Enc}_{\mathrm{pk}^{\mathcal{S}}}(\mathbf{y}')$$

Phase 2.b): Shares' sum DOWN to server



The encrypted sum $\sum_{N_i \neq C} \text{share}_{N_i}$ is sent down the tree so that the server S can decrypt it.

Phase 3: Server change its share

- The response to $\texttt{req}$ is computed by the server S:

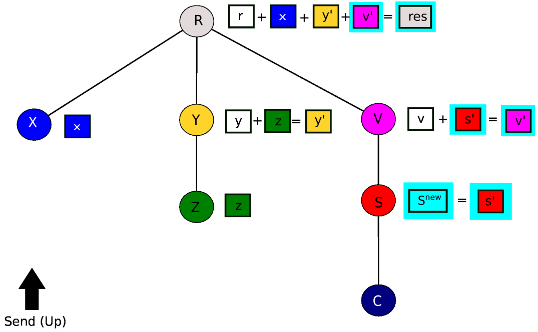
$$\texttt{res} := F(\texttt{req})$$

- The server recomputes its own share:

$$\text{share}_S^{new} := \texttt{res} - \Big(\sum_{N_i \neq C} \text{share}_{N_i} - \text{share}_S^{old}\Big)$$

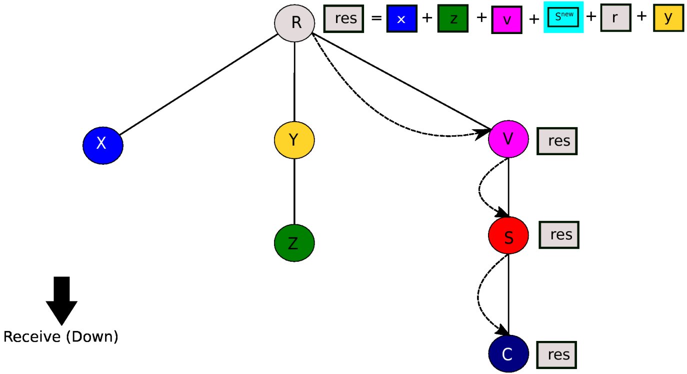
- The new share of the server and the share of the other nodes add up to $\texttt{res}$:

Phase 4.a): Response shares sent to root



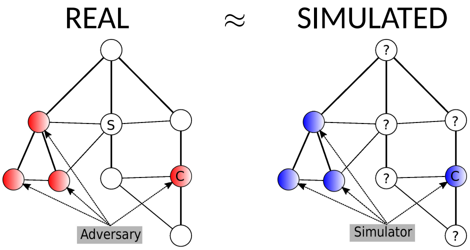
- All $N_i \neq C$ (including S) send their share share_i to **C**.
- S will send share_S^{new} instead of share_S^{old}.
- All shares are added using homomorphic encryption (**using** C**'s public key**): $\text{Enc}_{\text{pk}^C}(\mathbf{y}) \cdot \text{Enc}_{\text{pk}^C}(\mathbf{z}) = \text{Enc}_{\text{pk}^C}(\mathbf{y} + \mathbf{z}) = \text{Enc}_{\text{pk}^C}(\mathbf{y}')$

Phase 4.b) Encrypted response sent to client



The encrypted response $\texttt{res} = \sum_{N_i \neq C} \text{share}_{N_i}$ is sent down the tree so that the client C can decrypt it.

Security based on simulation



- When client is not corrupted: just simulate protocol under fake messages.
- When client is corrupted:
  - Simulator $\mathcal{S}$ gets response from ideal functionality.
  - $\mathcal{S}$ changes honest parties shares so that they reconstruct the correct response.

Linear Complexity

- $O(1)$ messages per Spanning Tree Edge.
- $O(1)$ homomorphic encryption operations.

Although a node can have $O(n)$ worst case complexity.

Malicious Adversaries (Overview)

Adversary's strategies:

- Drop messages. DoS.
- Change shares. DoS.

New Protocol:

- Messages are signed.
- Use recursive multicast for all messages ($O(n^2)$ Comm. complexity).
- Append zero-knowledge proof that ciphertexts encrypt same share. Allow identification of malicious players.

Zero-knowledge proof

- Prove that two ciphertexts encrypt same message, except…

Zero-knowledge proof

- Prove that two ciphertexts encrypt same message, except…
- Server actually **changes** the share.
- Proof needs to convince that
  - **[1] The two ciphertexts encrypt same message OR**
  - **[2] The issuer is the server**

Zero-knowledge proof

- Prove that two ciphertexts encrypt same message, except…
- Server actually **changes** the share.
- Proof needs to convince that
  - **[1] The two ciphertexts encrypt same message OR**
  - **[2] The issuer is the server**
- Do not reveal whether which of **[1]** , **[2]** is true.
- Reduces to simple Σ-protocol for relation

$$R_{g_1,g_2} = \underbrace{\{(A, B; r) : A = g_1^r \wedge B = g_2^r\}}_{[1]} \bigcup \underbrace{\{(D; s) : D = g^s\}}_{[2]}$$

Future work

- Resilience. Protocol needs to succeed even if some players disappears
- Improve communication complexity of second protocol
- Empirical Study
- Find trade-offs to scale current solution
- Server anonymity

Questions?