

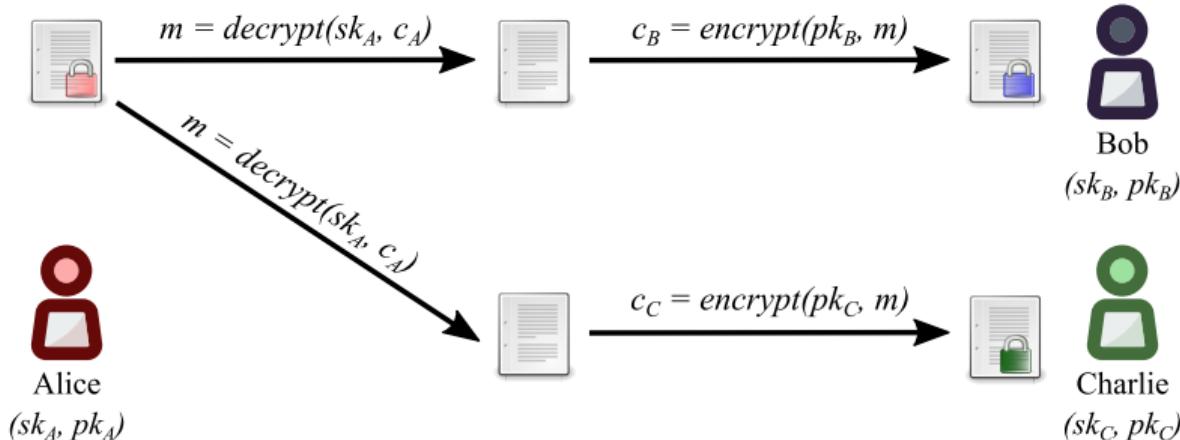


# NuCypher

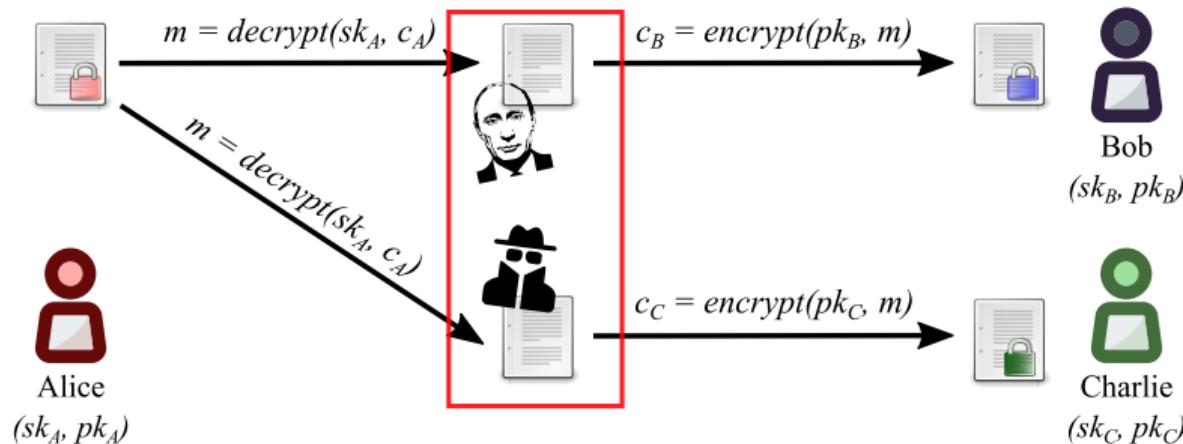
<presenter name(s), role(s)>

<event name>, <dd MMM yyyy>

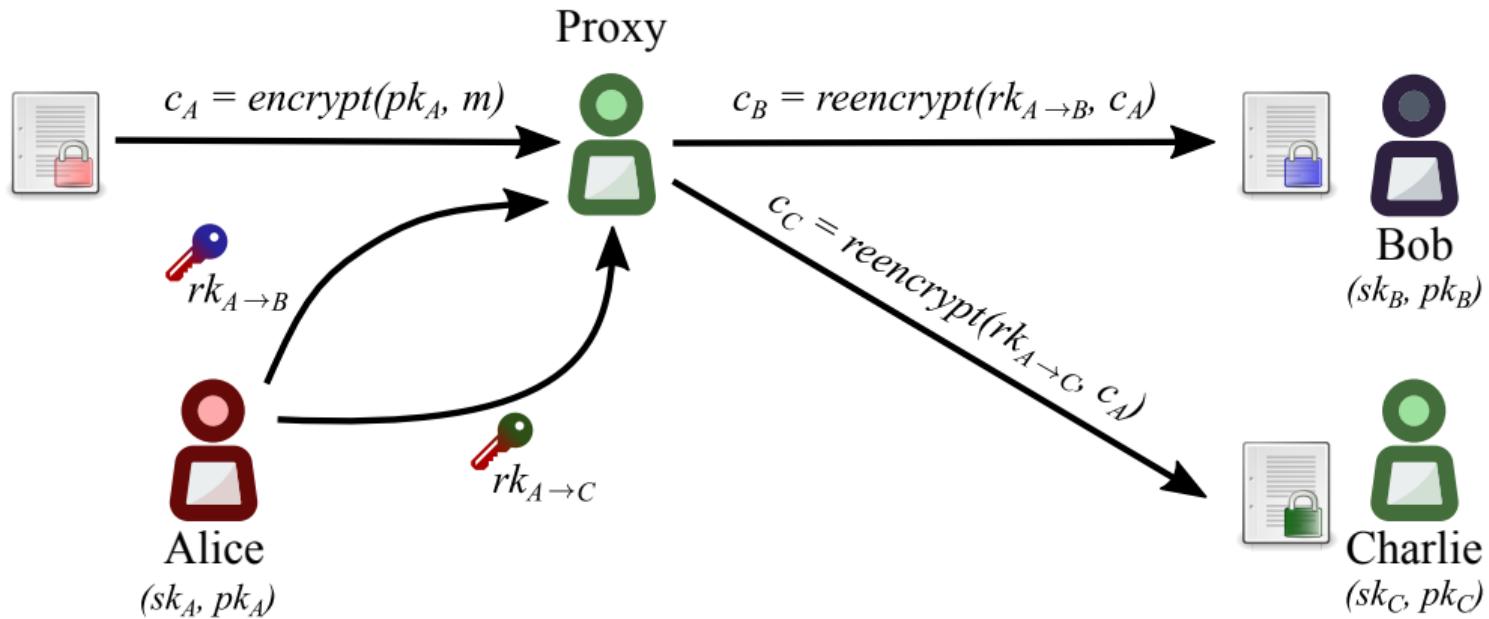
# Public Key Encryption (PKE)



# Public Key Encryption (PKE)

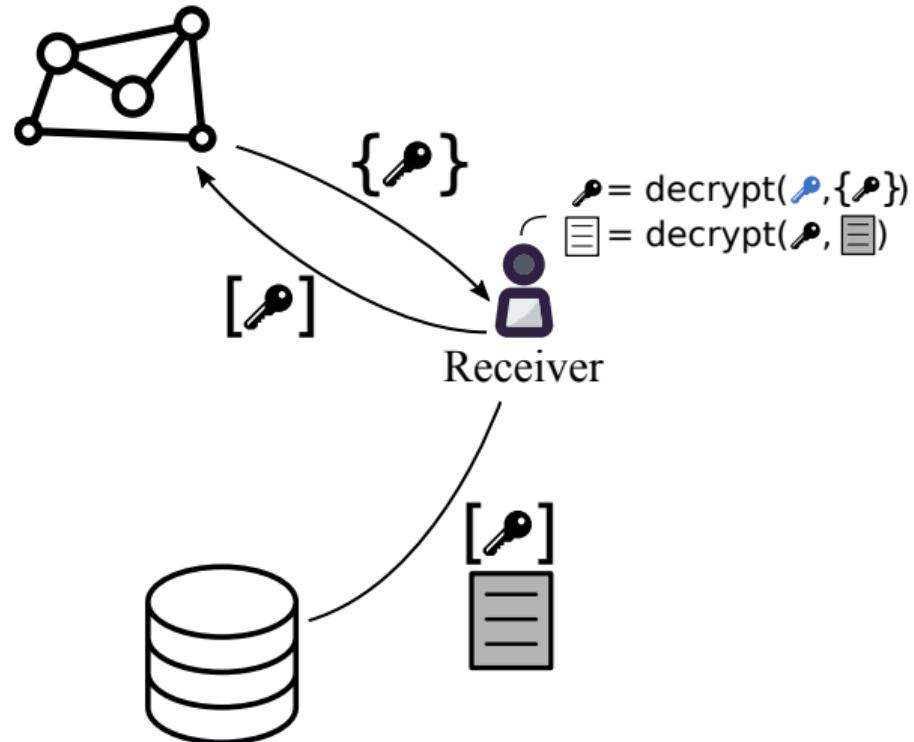


# What is proxy re-encryption (PRE)



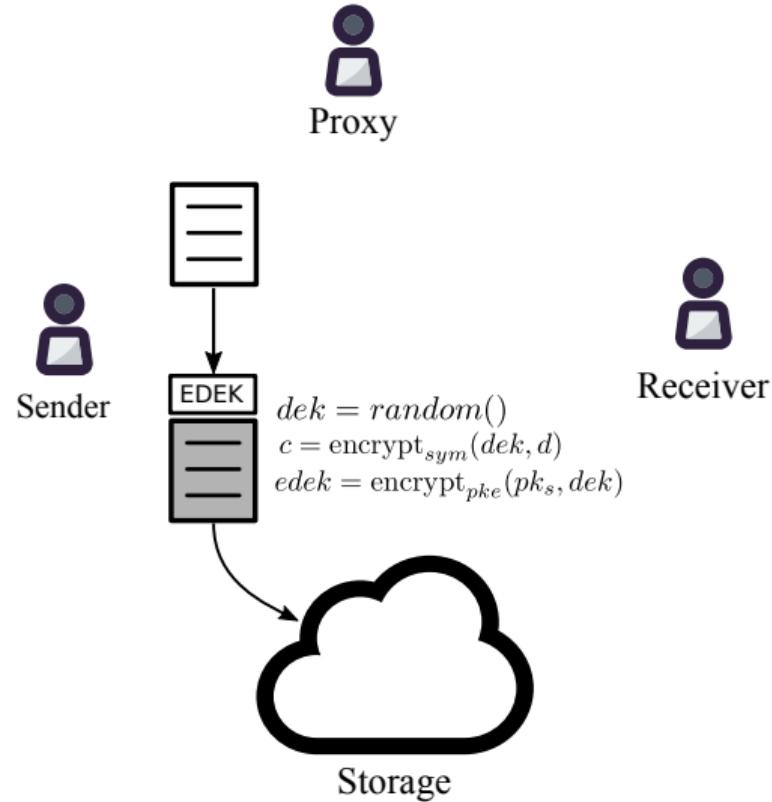
# Solution

Proxy re-encryption + Key Management



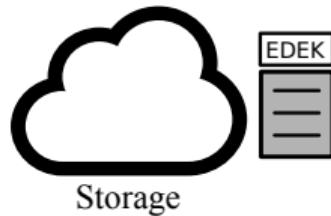
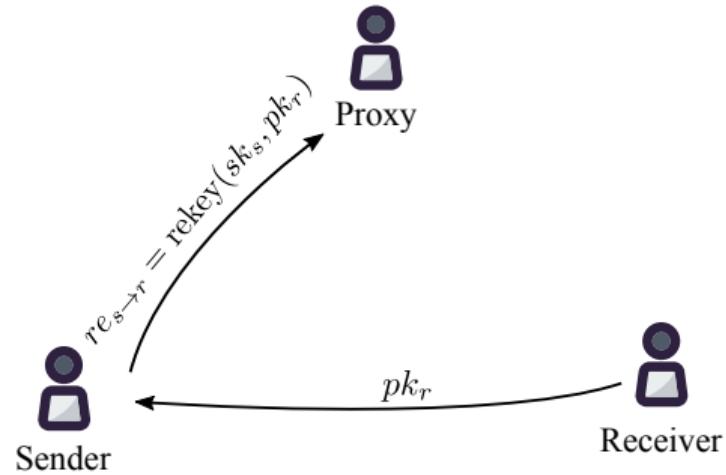
# Centralized KMS using PRE

## Encryption



# Centralized KMS using PRE

## Access delegation



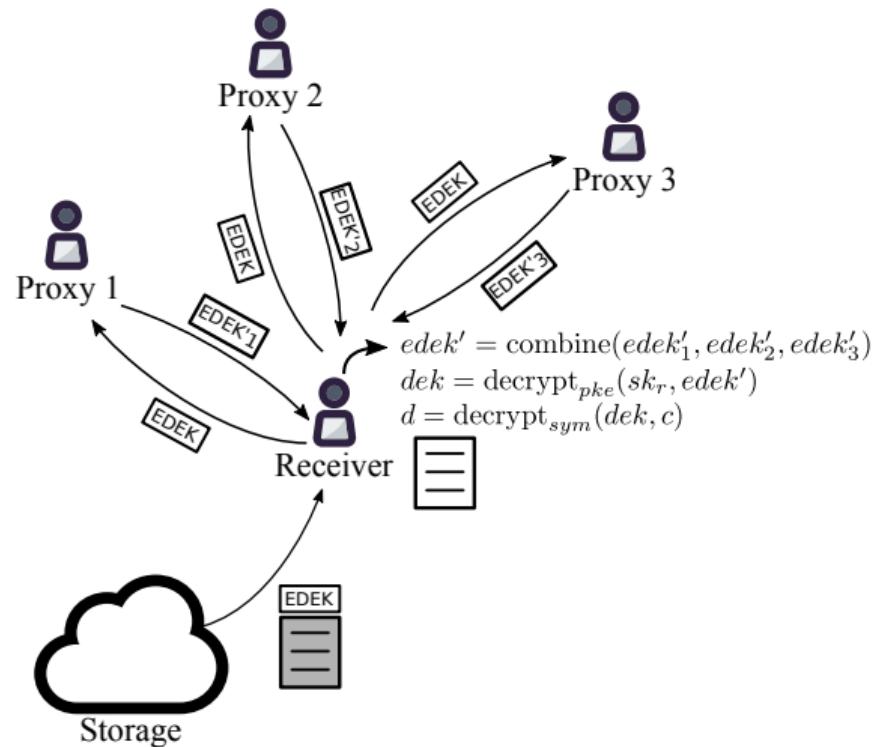
# Centralized KMS using PRE

## Decryption



# Decentralized Key Management

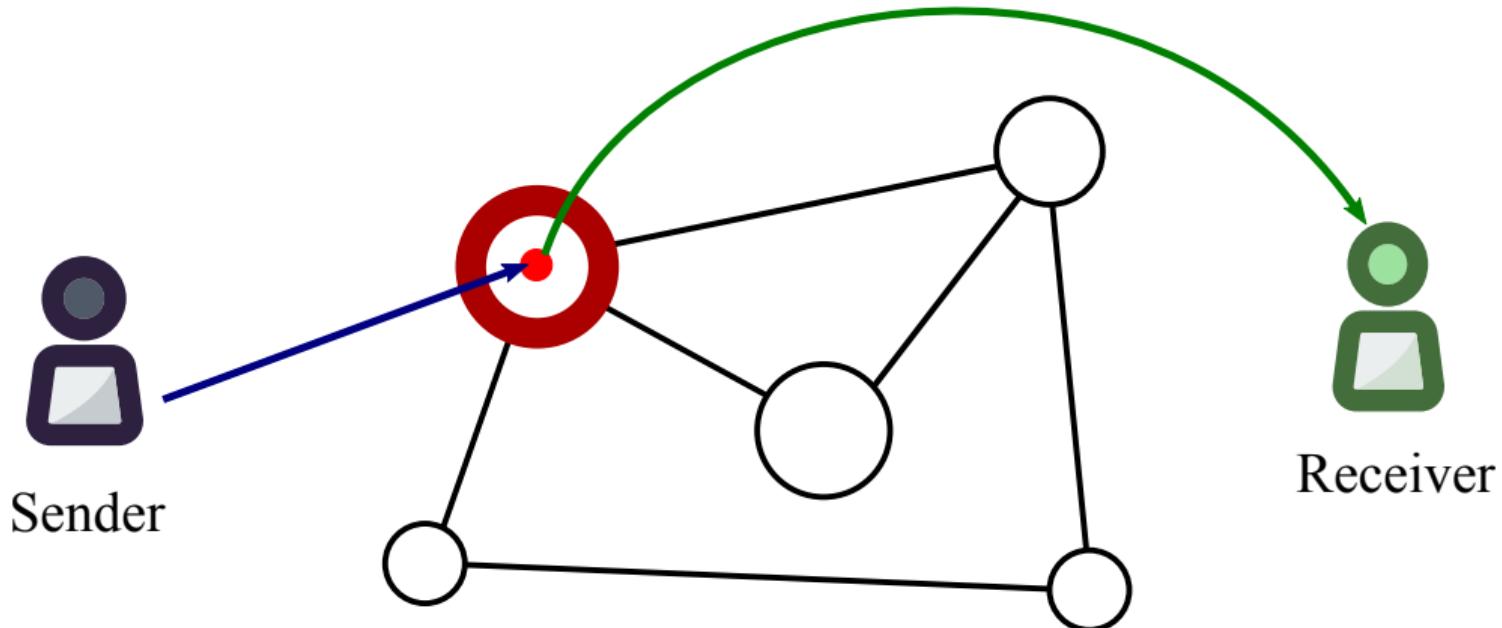
Using threshold split-key re-encryption (Umbral)



# Umbral: Threshold Proxy Re-encryption

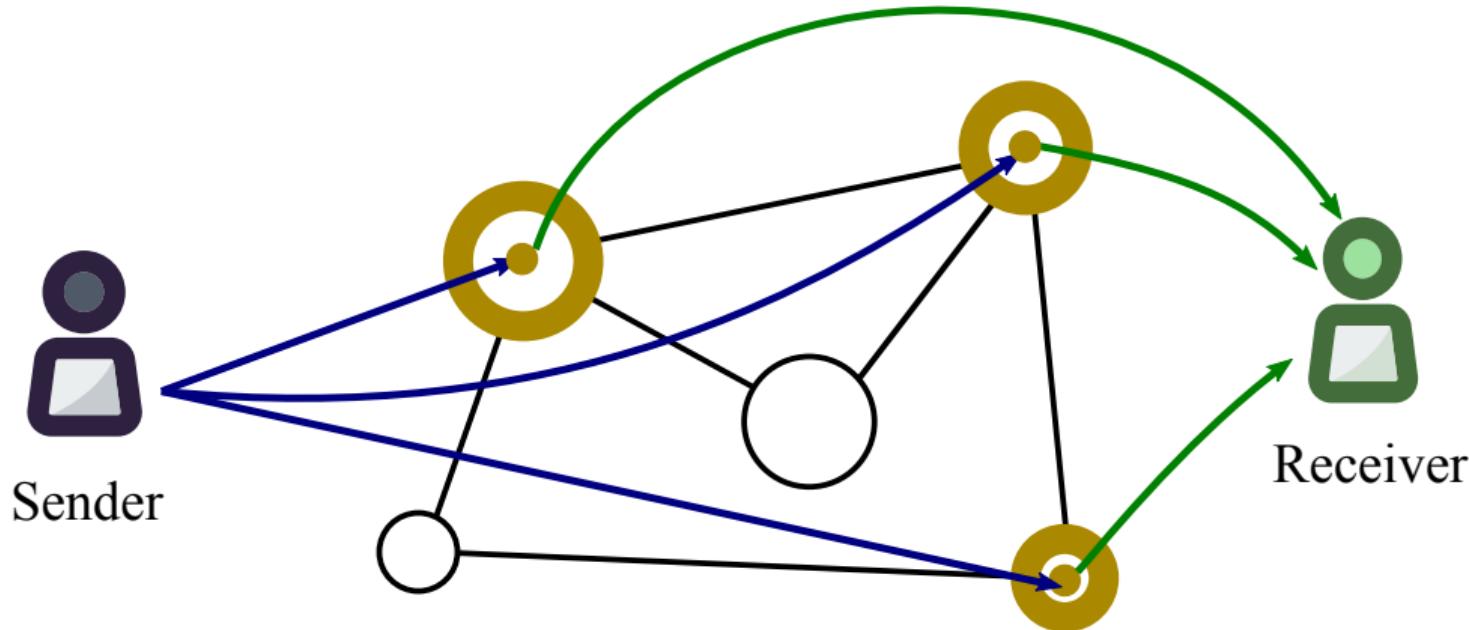
- “Umbral” is Spanish for “threshold”
- PRE properties: Unidirectional, single-hop, non-interactive
- Follows a KEM/DEM approach:
  - ▶ UmbralKEM provides the threshold re-encryption capability
  - ▶ Uses ECIES for key encapsulation with ZK proofs of correctness for verifiability on prime order curves (such as secp256k1)
  - ▶ DEM can be any authenticated encryption (currently ChaCha20-Poly1305)
- IND-PRE-CCA security
- Key splitting is analogous to Shamir Secret Sharing
- Verification of re-encryption correctness through Non-Interactive ZK Proofs
- Reference implementation: <https://github.com/nucypher/pyUmbral>
- Documentation: <https://github.com/nucypher/umbral-doc>

## KMS Network: Data Sharing + PKE



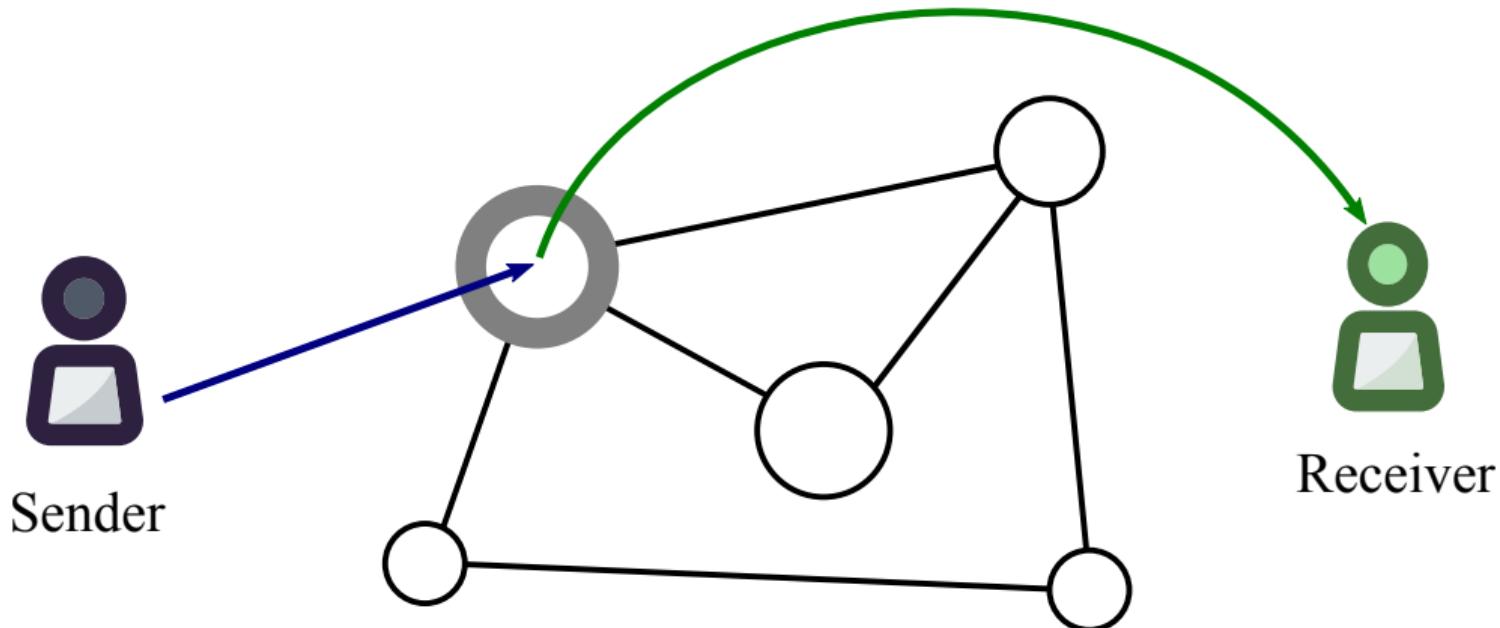
- Single node has access to data
- Single node can deny to do work

# KMS Network: Data Sharing + PKE + Shamir Secret Sharing



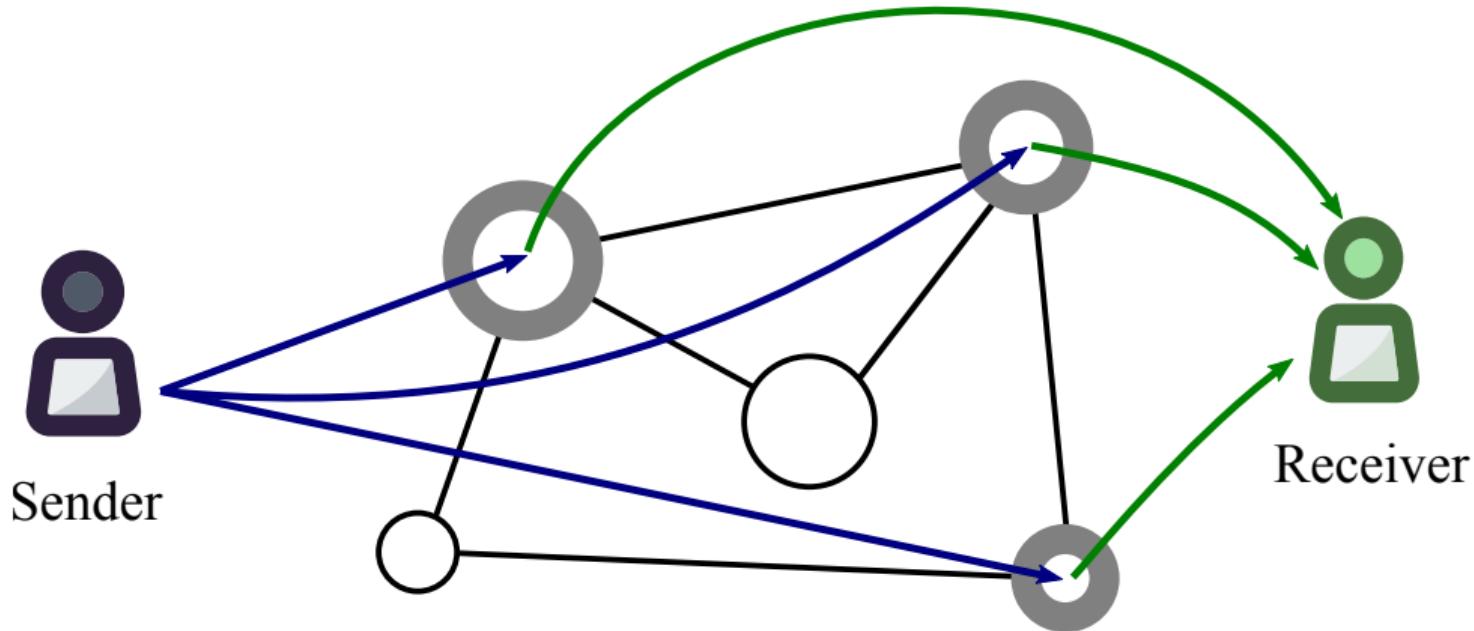
- Nodes can collude to gain access to data

## KMS Network: Data Sharing + PRE



- Single node collusion with receiver possible
- Single node can deny to do work

# KMS Network: Data Sharing + Threshold PRE (Umbral)



- Collusion now requires  $m$  nodes + receiver

# NU Token

## Purpose

- Splitting trust across re-encryption nodes
  - ▶ More tokens = more trust, more work, and more compensation
- Proof of Stake for minting new coins according to the mining schedule
- Security deposit at stake against malicious behavior of nodes

# Data Sharing Policies

- Time-based
- Conditional on payment
  - ▶ “Grant access once paid, continue granting while paying”
- Smart contract (public) method

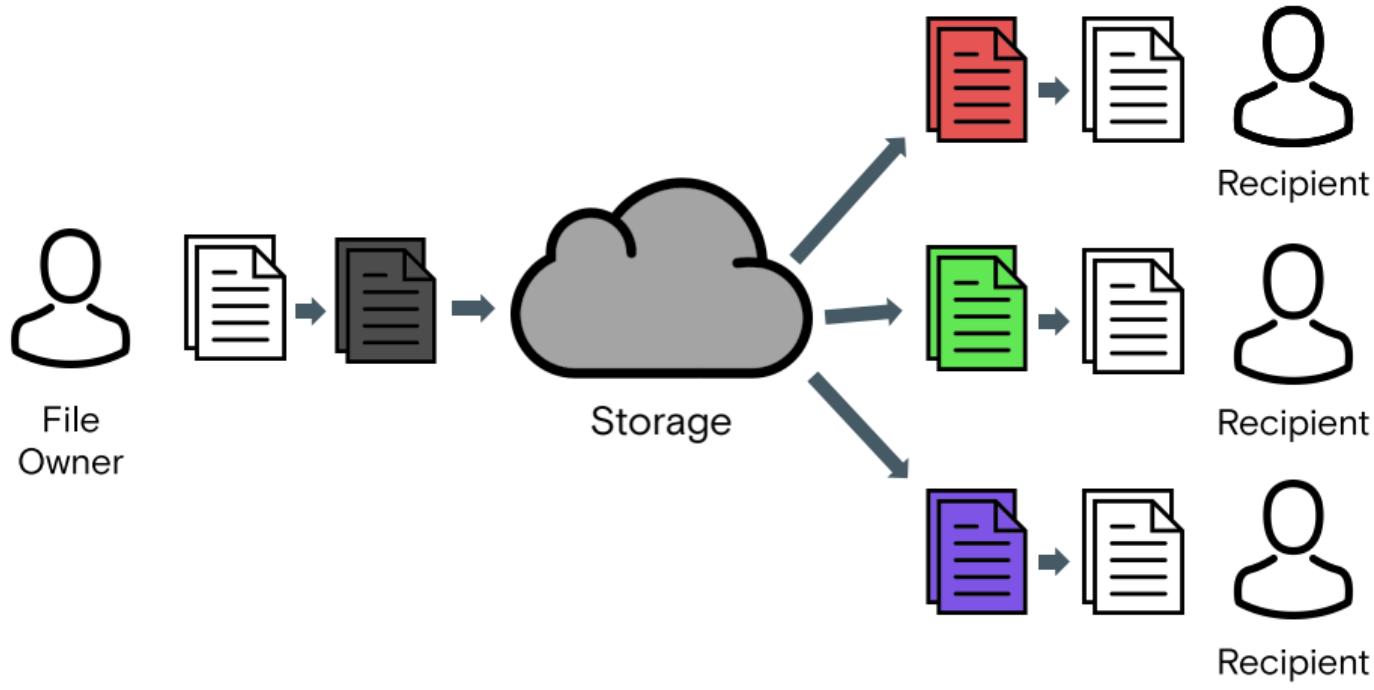
Decentralized re-encryption nodes (Ursulas) relied on to apply conditions without having the ability to decrypt data

# Demo



# Use Cases

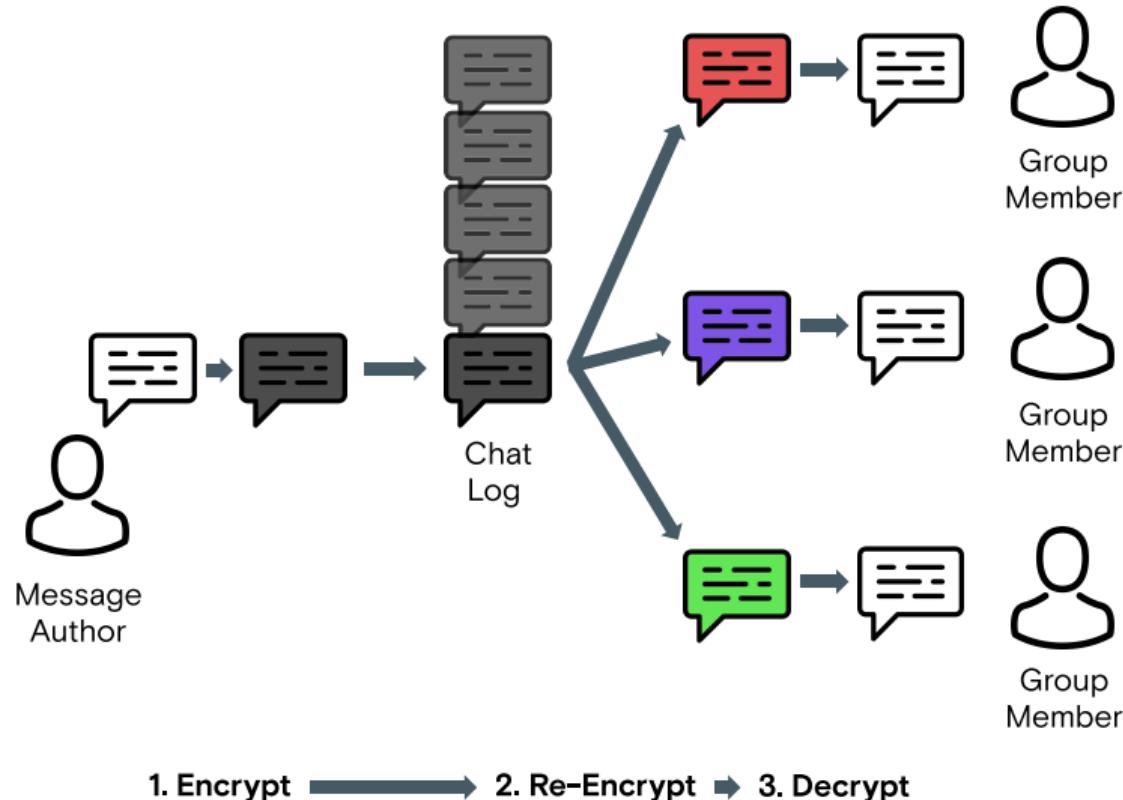
## Encrypted file sharing



1. Encrypt → 2. Re-Encrypt → 3. Decrypt

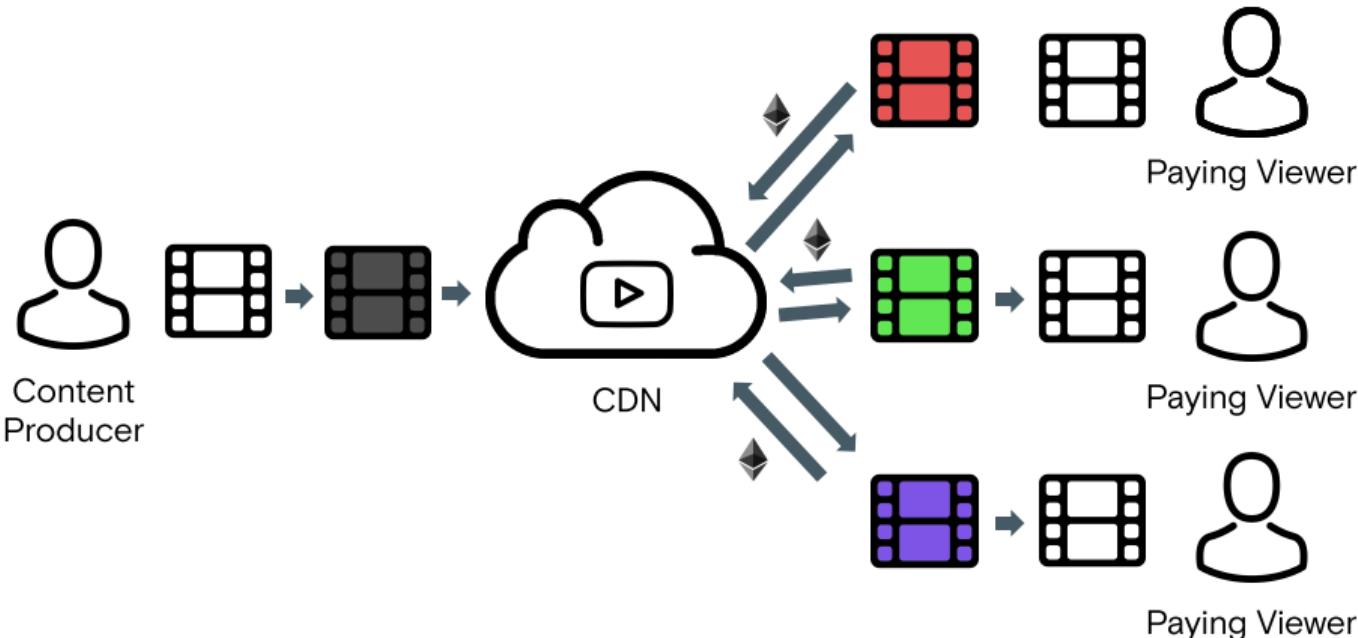
# Use Cases

## Encrypted multi-user chats



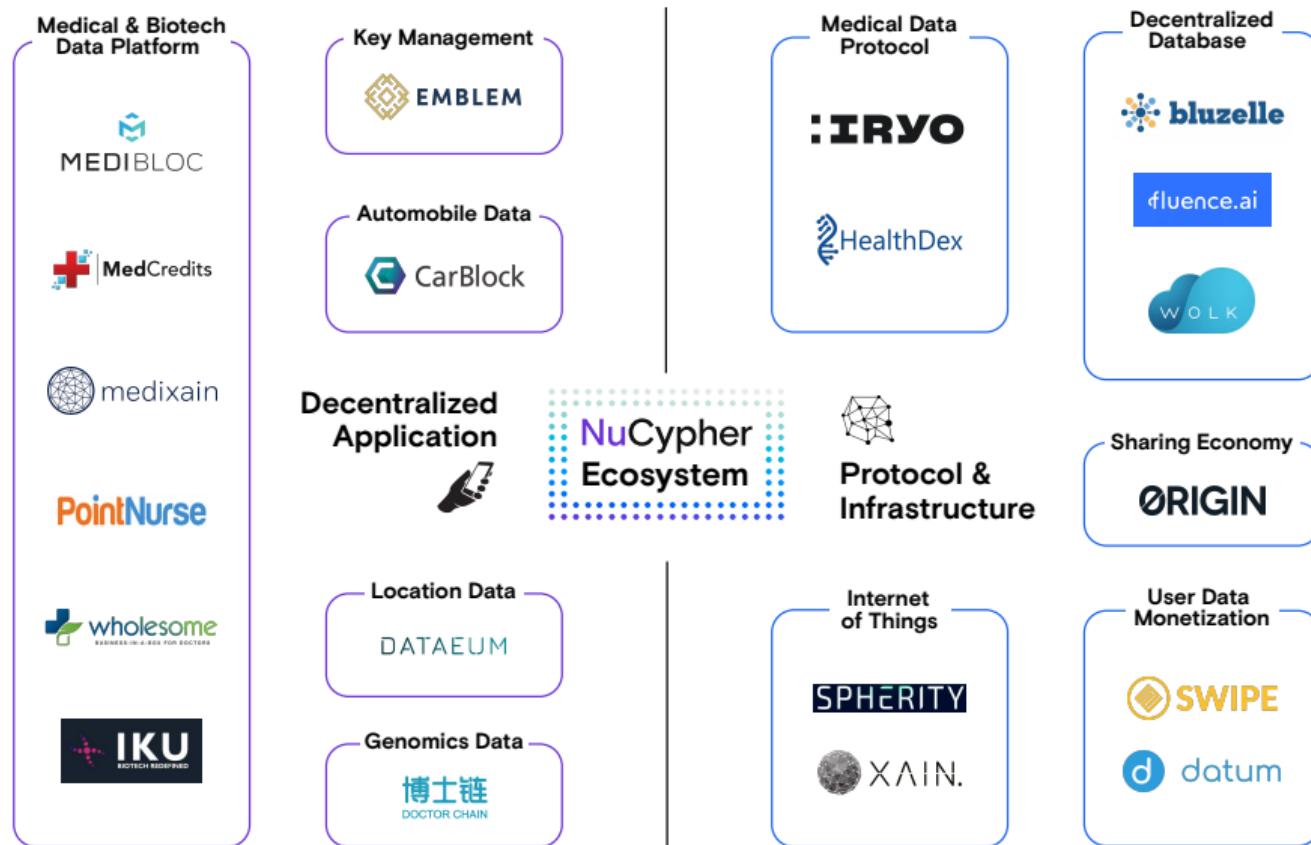
# Use Cases

Decentralized access-controlled content



1. Encrypt
2. Conditional Re-Encrypt
3. Decrypt

# Early Users



# Competing Technology

## Data Masking and Tokenization

- Less secure for data with underlying patterns
- Reduce the value of data by obfuscating it

## Public Key Encryption

- Data must be decrypted before it is shared
- Not Scalable

## Multi-Party Computation

- Interactive protocol
- Slow Performance

## Fully Homomorphic Encryption

- Slow Performance
  - ▶ NuCypher has developed a GPU-accelerated FHE library: nuFHE

# Fully Homomorphic Encryption

## nuFHE library

- GitHub: <https://nucypher.dev/nufhe>
- GPU implementation of fully homomorphic encryption
- Uses either FFT or integer NTT
- Achieved 100x performance over TFHE benchmarks

Platform	Library	Performance (ms/bit)	
		Binary Gate	MUX Gate
Single Core/Single GPU - FFT	TFHE (CPU)	13	26
	nuFHE	0.13	0.22
	Speedup	<b>100.9</b>	<b>117.7</b>
Single Core/Single GPU - NTT	cuFHE	0.35	N/A
	nuFHE	0.35	0.67
	Speedup	<b>1.0</b>	-

# FHE Proof of Concept

## Sputnik

- GitHub: <https://nucypher.dev/sputnik>
- Assembly language and interpreter for FHE that uses nuFHE
- Commits a merkle root of computation to the blockchain for proof of logic flow
- Used to execute first homomorphic smart contract at ETHBerlin 2018



ETHBerlin  
@ETHBerlin

Follow



PLEASE give a round of applause to  
Sputnik!!! They are the first winners of our  
open track!! They designed A byte code  
assembly type language! YAAAAAASSSSS  
GUYS #ETHBerlin

## More Information



**NuCypher**

**Website:** <https://www.nucypher.com>

**Whitepaper:** <https://www.nucypher.com/whitepapers/english.pdf>

**Proxy Re-encryption Network:** <https://nucypher.dev/nucypher>

**Umbral Reference Implementation:** <https://nucypher.dev/pyUmbral>

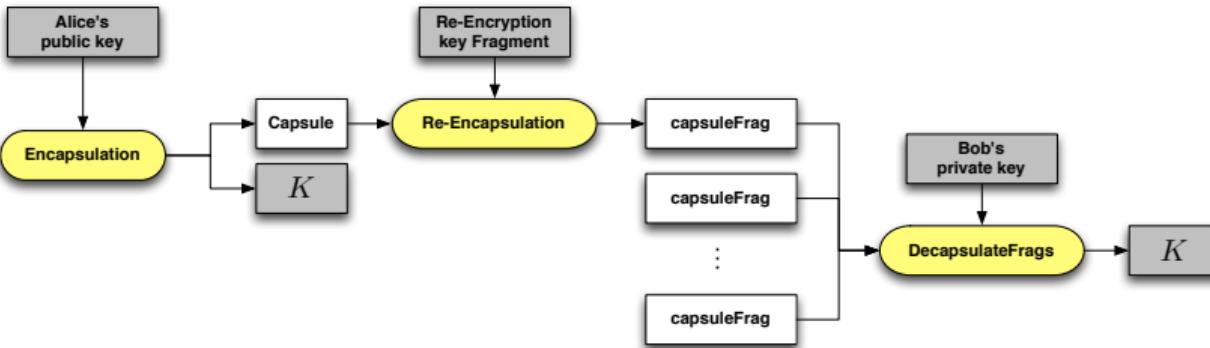
**nuFHE:** <https://nucypher.dev/nufhe>

**Discord:** <http://discord.nucypher.com>

**E-mail:** <email\_prefix>@nucypher.com

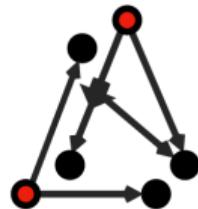
**E-mail:** hello@nucypher.com

## Appendix: Umbral Flow Diagram



- Reference implementation: <https://nucypher.dev/pyUmbra>
  - Documentation: <https://nucypher.dev/umbra-doc>

# Appendix: Security Audits



**Least Authority**  
Freedom Matters

# Appendix: NU Token Metrics

## Mining

Mining & Staking Economics: <https://nucypher.dev/mining-paper>

Mining reward:

$$\kappa = \left( 0.5 + 0.5 \frac{\min(T_i, T_1)}{T_1} \right)$$

$$T_{i,\text{initial}} \geq T_{\min}$$

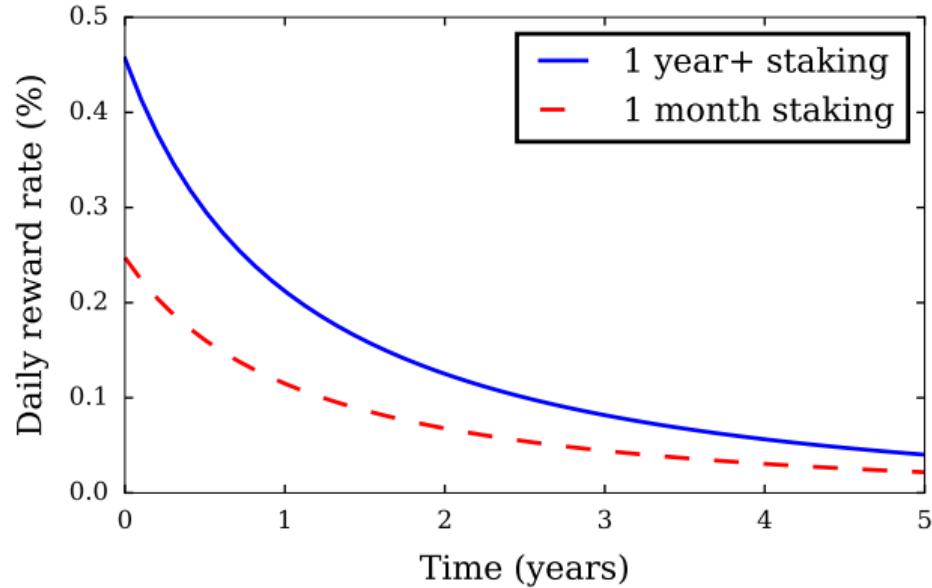
$$\delta S_{i,t} = \kappa \frac{l_i}{\sum l_j} \frac{\ln 2}{T_{1/2}} (S_{\max} - S_{t-1})$$

Results into:

$$\text{reward} \propto 2^{\frac{t}{T_{1/2}}}$$

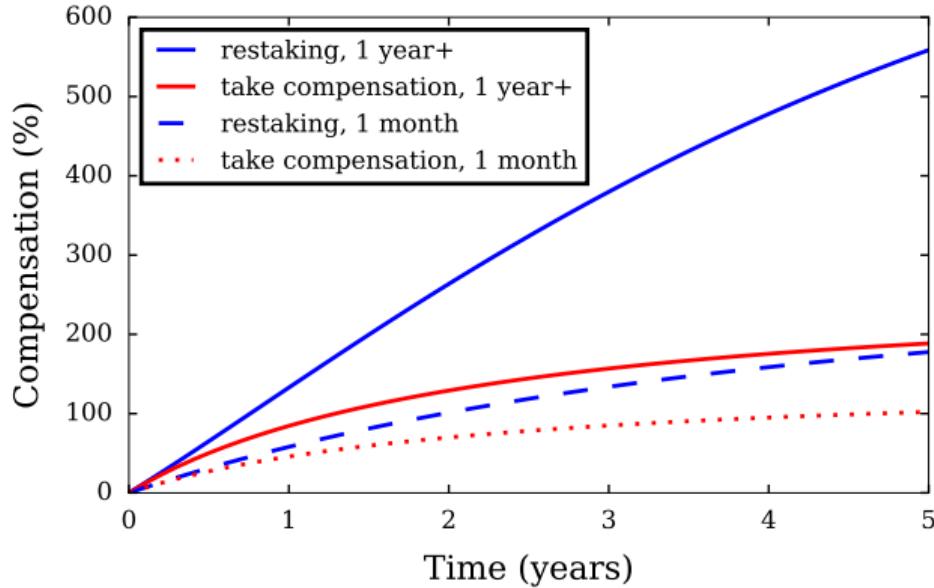
# Appendix: NU Token Metrics

## Graph of daily mining compensation



# Appendix: NU Token Metrics

## Relocking mining rewards



# Appendix: Team

## Founders



MacLane Wilkison  
Co-founder and CEO



Michael Egorov, PhD  
Co-founder and CTO

## Advisors



Prof. Dave Evans



Prof. Giuseppe Ateniese  
Stevens Inst. of Technology



John Bantleman  
Rainstor



Tony Bishop  
Equinix

## Employees



David Nuñez, PhD  
Cryptographer



John Pacific (tux)  
Engineer



Justin Myles Holmes  
Engineer



Sergey Zотов  
Engineer



Kieran Prasch  
Engineer



Bogdan Opanchuk, PhD  
Engineer



Ryan Caruso  
Community



Derek Pierre  
Business Development



Arjun Hassard  
Product & Partnerships