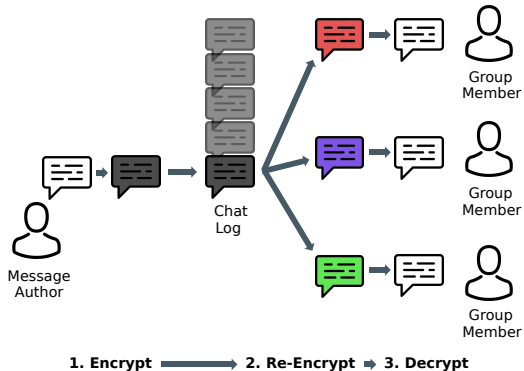


Michael Egorov

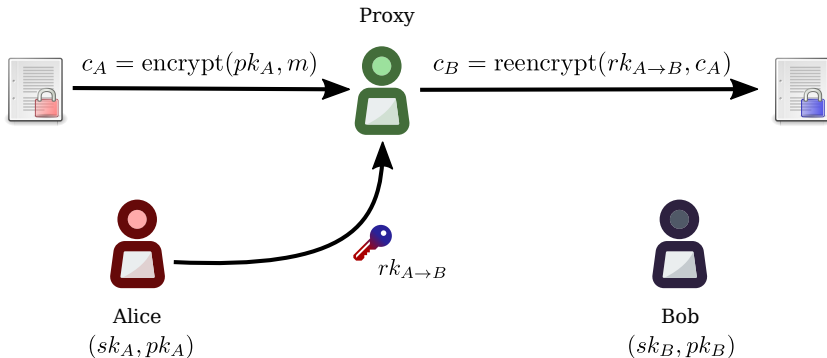
ETH Security UnConference, 6 Sep 2018

# Why

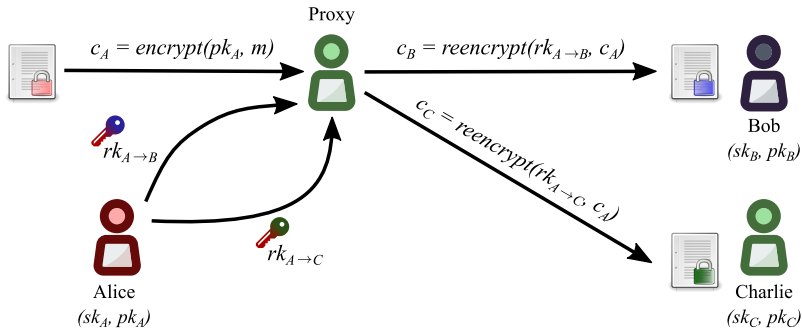
## Sharing data while encrypted



# What is proxy re-encryption (PRE)

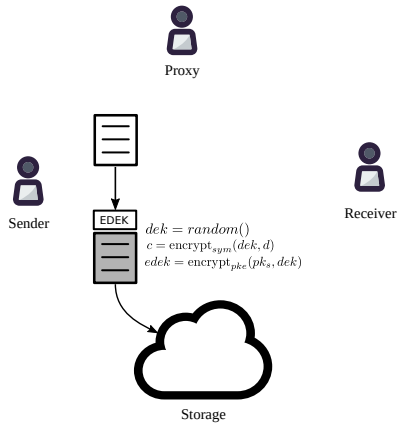


# PRE and multiple receivers



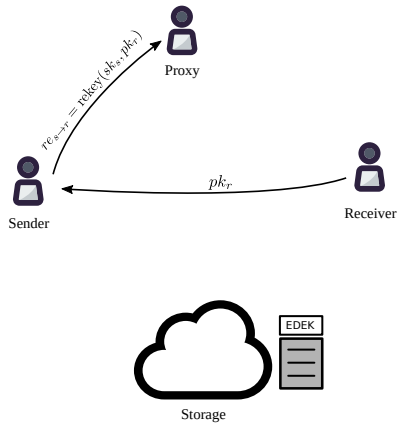
# Key management using PRE

## Encryption



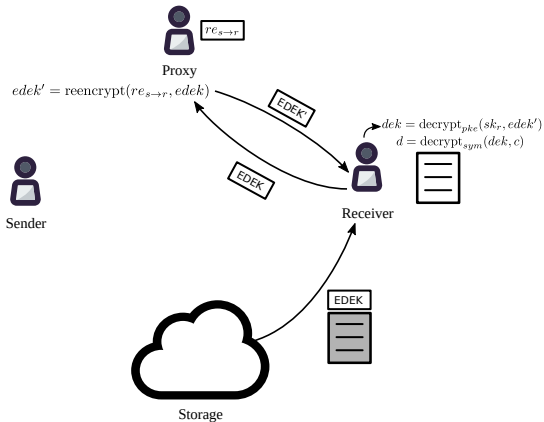
# Key management using PRE

## Access delegation



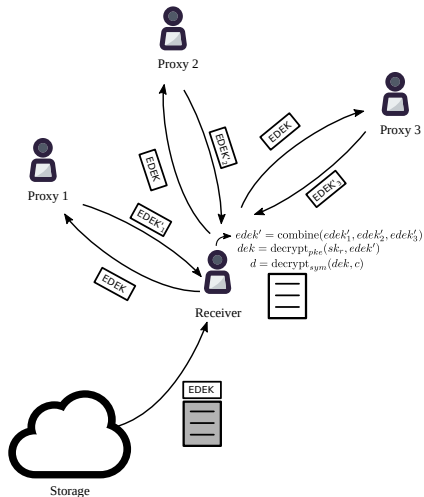
# Key management using PRE

## Decryption



# Decentralized key management

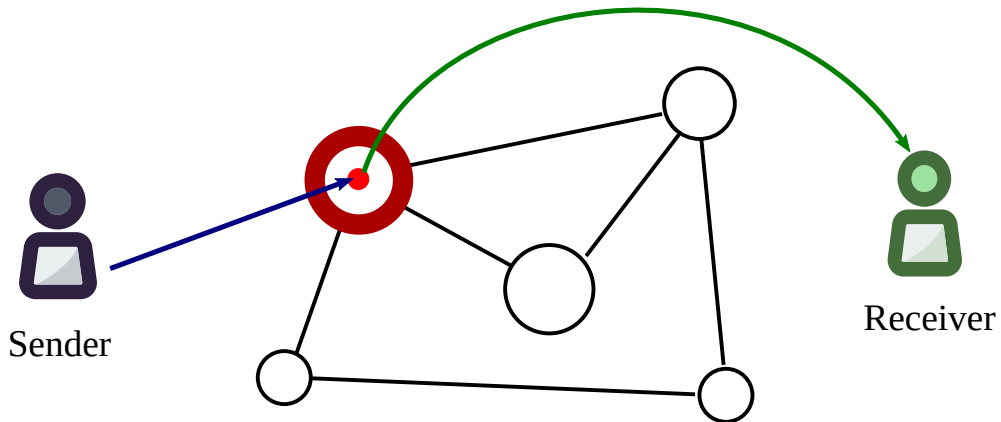
Using threshold split-key re-encryption (Umbral)



<https://github.com/nucypher/nucypher-kms/>

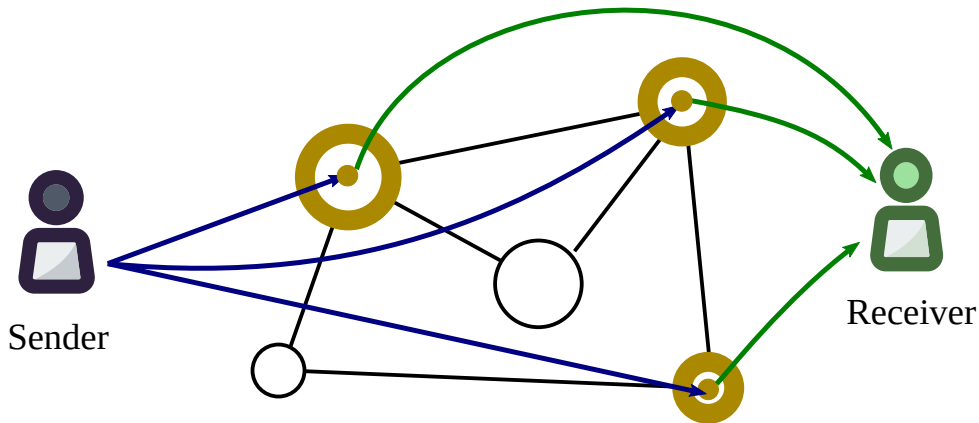


## Sharing in permissioned network



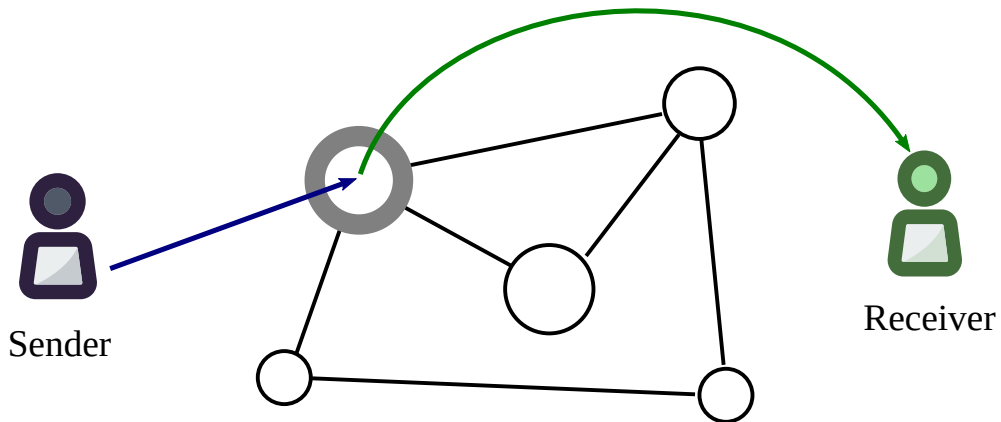
- Node sees everything;
- Node can deny to work.

## Permissioned network + SSS



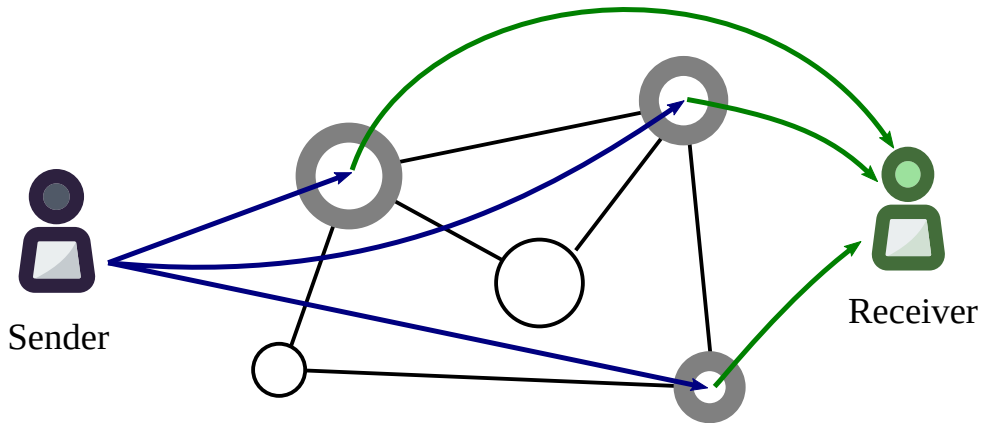
- Nodes can collude to see everything.

## Sharing with PRE



- Collusion with receiver possible,
- Node can deny to work.

## Sharing with threshold PRE

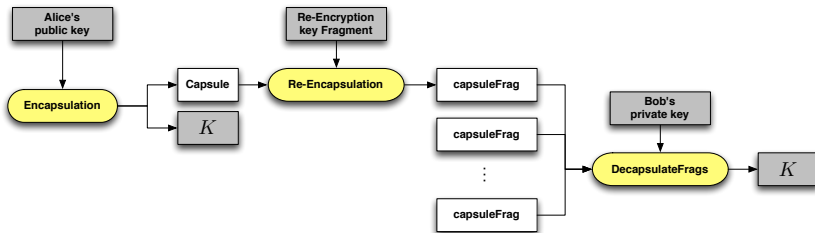


- Collusion with receiver:  $m$  nodes + receiver.

# Umbral: threshold proxy re-encryption

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

# Umbral: threshold proxy re-encryption



- Reference implementation: <https://github.com/nucypher/pyUmbral/>
- Documentation: <https://github.com/nucypher/umbral-doc>

# Wrapping OpenSSL for EC/BN maths

```
</michwill/Projects/pyUmbra/ | 160      # 1 is not-equal, 0 is equal, -1 is error
> docs/ | 161      return not bool(is_equal)
> tests/ | 162
> umbral.egg-info/ | 163      def __mul__(self, other) -> 'Point':
v umbral/ | 164          """
    __about__.py | 165      Performs an EC_POINT_mul on an EC_POINT and a BIGNUM.
    __init__.py | 166          """
    _pre.py | 167      # TODO: Check that both points use the same curve.
    config.py | 168      prod = openssl._get_new_EC_POINT(self.curve)
    curve.py | 169      with backend._tmp_bn_ctx() as bn_ctx:
    curvebn.py | 170          res = backend._lib.EC_POINT_mul(
    dem.py | 171          self.curve.ec_group, prod, backend._ffi.NULL,
    fragments.py | 172          self.ec_point, other.bignum, bn_ctx
    keys.py | 173          )
    openssl.py | 174          backend.openssl_assert(res == 1)
    params.py | 175
    point.py | 176      return Point(prod, self.curve)
    pre.py | 177
    signing.py | 178      __rmul__ = __mul__
    utils.py | 179
> vectors/ | 180      def __add__(self, other) -> 'Point':
    LICENSE.md | 181          """
    mypy.ini | 182      Performs an EC_POINT_add on two EC_POINTS.
    Pipfile | 183          """

<ome/michwill/Projects/pyUmbra/ point.py | 163,40 65% [Name] point.py
```

# Wrapping OpenSSL for EC/BN maths

```
</michwill/Projects/pyUmbra/ | 378
> docs/ | 379
> tests/ | 380
> umbral.egg-info/ | 381
v umbral/ | 382
  __about__.py | 383
  __init__.py | 384
  _pre.py | 385
  config.py | 386
  curve.py | 387
  curvebn.py | 388
  dem.py | 389
  fragments.py | 390
  keys.py | 391
  openssl.py | 392
  params.py | 393
  point.py | 394
  pre.py | 395
  signing.py | 396
  utils.py | 397
> vectors/ | 398
  LICENSE.md | 399
  mypy.ini | 400
  Pipfile | 401
<ome/michwill/Projects/pyUmbra/ | 381,63
pre.py [+] | 72% [Name] pre.py
```

```
378
379
380 def reencrypt(kfrag: KFrag, capsule: Capsule, provide_proof: bool = True,
381               metadata: Optional[bytes] = None) -> CapsuleFrag:
382
383     if not capsule.verify():
384         raise Capsule.Invalid
385
386     if not kfrag.verify_for_capsule(capsule):
387         raise KFrag.Invalid
388
389     rk = kfrag._bn_key
390     e1 = rk * capsule._point_e
391     v1 = rk * capsule._point_v
392
393     cfrag = CapsuleFrag(point_e1=e1, point_v1=v1, kfrag_id=kfrag._id,
394                        point_noninteractive=kfrag._point_noninteractive,
395                        point_xcoord=kfrag._point_xcoord)
396
397     if provide_proof:
398         prove_cfrag_correctness(cfrag, kfrag, capsule, metadata)
399
400     return cfrag
401
```

```
GenericUmbraError : class
v UmbraCorrectnessError : class
  +__init__ : function
  +decapsulate_original : function
  +decapsulate_reencrypted : function
  +encapsulate : function
  +open_capsule : function
  +decrypt : function
  +encrypt : function
  +reencrypt : function
  +split_rekey : function
  -
  -
  -
  -
```



## Useful links



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

**Github:** <https://github.com/nucypher/>

**PyUmbral:** <https://github.com/nucypher/pyUmbral/>

**GoUmbral:** <https://github.com/nucypher/goUmbral/>

**Mocknet:** <https://github.com/nucypher/mock-net/>

**Discord:** <https://discord.gg/7rmXa3S>

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

**E-mail:** [michael@nucypher.com](mailto:michael@nucypher.com)

**E-mail:** [hello@nucypher.com](mailto:hello@nucypher.com)