MessageVortex

Unlinkable, censorship-resistant communication

ANONYMITY MATTERS IN CENSORED ENVIRONMENTS

- Achieving anonymity is hard
  - Many attempts such as remailers, mix nets, DC-nets, distributed hash tables (DHT), or broadcast based networks exist
- Achieving anonymity in a censored environment is far harder
  - Few attempts exist (e.g., ToR, Riffle, or SCION)
  - All actively used technologies are censored

CONTENT

- Predecessor Systems
- Anonymity Technologies
- What are the problems?
- MessageVortex
  - Why do we need it
  - What are the problems
  - How does it work
    - The General Idea
    - The Message
    - The Peer Key
    - The Sender Key
    - The Header
    - eID
    - Workspace
    - The Operations
    - Processing
    - Routing Strategies
  - What are its benefits
- Any Questions Left?

PREDECESSOR SYSTEMS

- Remailers (0-III)
  - Nym Remailers
  - Cypherpunk
  - Mixmaster
  - Mixminion
- Onion Routing (SOR, Tor, I2P…)
- DHT (Salsa, Tarzan,…)
- DC-Nets (Herbivore, Dissent)
- Broadcast (Hordes)
- Distributed Storages (Freenet,Guntella(2))

TECHNOLOGIES FOR ANONYMITY

- Distribute data
  - Across multiple nodes
- Hide data
  - Steganography
  - Mimicking of traffic (content and size wise)
  - Side channel transmission
- Outcurve censorship measurements
  - Use modified TCP stacks
  - Use UDP

WHAT ARE THE PROBLEMS?

- Technology
  - A (network) packet is always visible…
    - everyone may see it.
    - everyone may analyze it.
    - everyone may forward(modified or unmodified) or keep it.
  - If we distribute data, we need to recover enough parts.
  - It is relatively easy to build something for the lab but most of the approaches (e.g., broadcast based) do not scale.
- Juristic
  - Every jurisdiction applies their own laws
  - Laws may disagree with each other

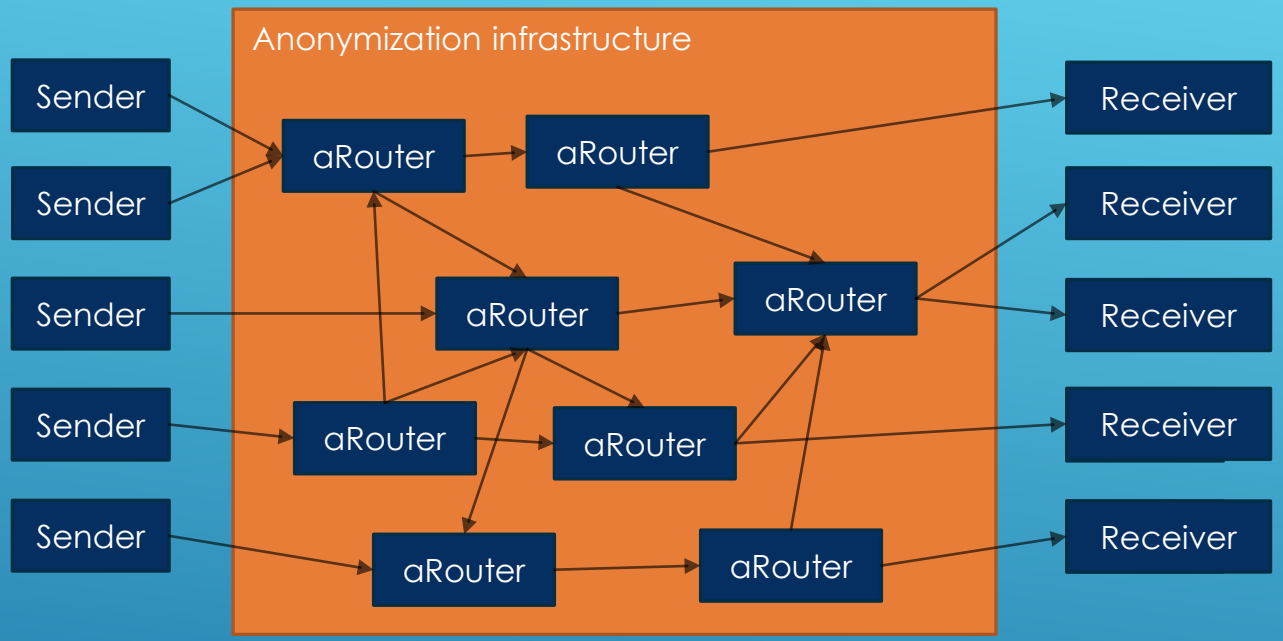
- Anonymity is essential
  - Without anonymity there is no freedom of speech
- Censoring (in general and of Anonymity technology) happens today
  - China (blocks Tor, calculate citizen score)
  - Turkey (VPNs banned, Websites censored)
  - United Arab Emirates (VPN is illegal for illegal activities ;-))
  - Iran (Only Government approved VPNs are legal)
  - Egypt (Website censorship, working on banning VPNs)
  - And more … (Bahrain, Ethiopia…)
  - This is not necessarily (officially) used for censorship. It is to…
    - fight „terrorism.“
    - fight „state decomposing elements.“
    - stopping “immoral or anti-religious works.”

MESSAGEVORTEX

WHY DO WE NEED IT?

Agreement: Censorship exists‘

it‘s use may be questionable



MESSAGEVORTEX

GENERAL WEAK SPOTS OF ANONYMIZATION SOLUTIONS

- Identifiable sets for senders and receivers
- Identifiable infrastructure
  - Nodes in general
  - Entry nodes
  - Exit nodes
- Central elements in infrastructure
  - Directory servers
- Not prone to active adversaries
- Identifiable (censorable) protocols
- (Limited) trust in infrastructure
- Identifiable meta data (e.g., message sizes, throughput, long living pseudonymity, or shrinking member sets)
- Identifiable messages
- Replayable routing information
- Bugable protocols/Message content
- Tagable protocols/Message content
- Bootstrapping problem (e.g., key trust or key distribution problem)
- Identification problem
- Huge traffic overhead

... and many more ...

- Send anonymous messages across the internet
- Unobservable from the outside
- Holistic approach piggybacking common protocols (e.g., SMTP, XMPP)
- Distrust everyone (…)
  - Routing nodes learn as little as possible from routing a message
  - The routing process is controlled by the builder of the routing block in the message
  - The routing path is diagnoseable by the builder of the routing block
  - Our only trust: initial sender of the message and final recipient of the message
- Messages are sent from node to node in a mix type like system
- Decoy traffic is not even identifiable for the node generating or routing.
- Resistent to common attacks which are hard to cover

HOW DOES IT WORK

THE GENERAL IDEA

- Each node gets a couple of payload blocks
- It recombines available payload blocks
  - Payload block of any message from the same sender (short term pseudonym)
- Outcome may be bigger or smaller depending on the applied operation:
  - Split or merge block
  - Encrypt or decrypt block
  - Add or remove redundancy information
- Any payload block may be decoy or true message content!
  - Even the routing node generating decoy cannot tell decoy from message traffic apart.
- Broken (crypto) algorithms must not break the protocol
  - Any type of operation should have an alternative

HOW DOES IT WORK

THE GENERAL IDEA (2)

MESSAGEVORTEX

HOW DOES IT WORK?

- Nodes send and mix messages
  - All equal
    - All may be endpoint of a message
    - All do routing
  - Hide their traffic in common transport protocols
- Messages
  - Purely controlled by an onionized routing block
  - Not identifiable unless you possess a node specific key

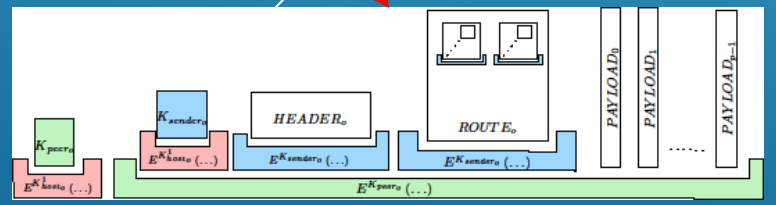
Always connected end-user device
(e.g., mobile phone)

Any Transport layer infrastructure
(e.g., Gmail account)

Extract
(e.g., „F5“ or similar steganographic algorithm)

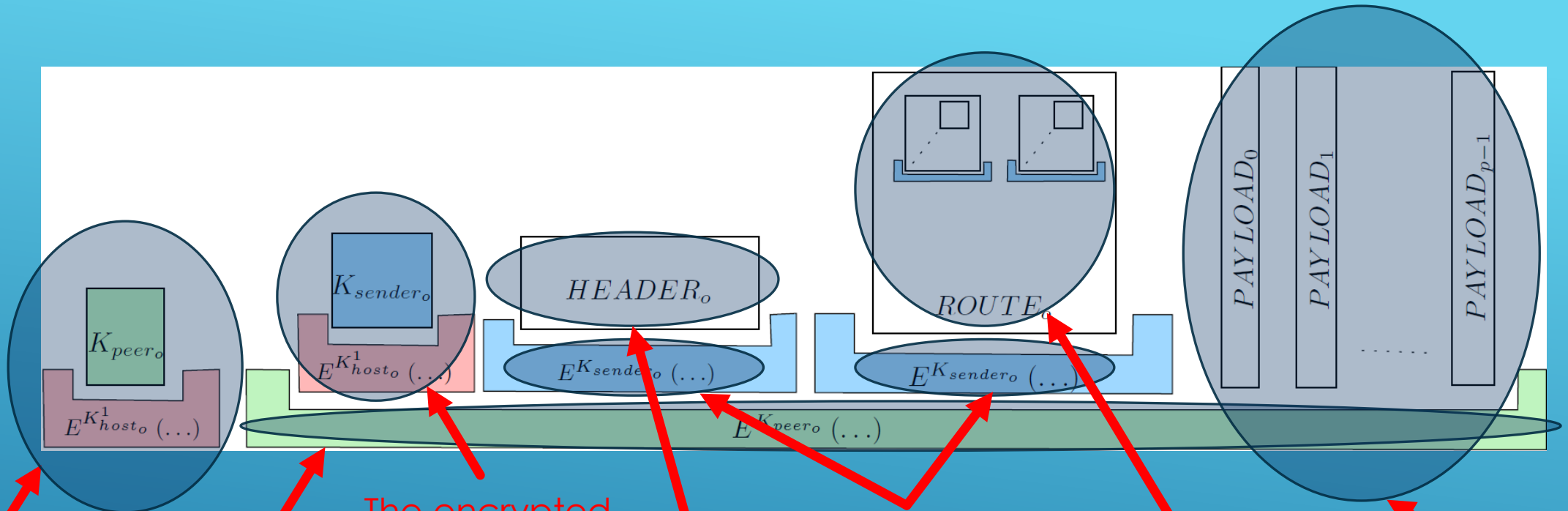
MESSAGEVORTEX

HOW DOES A MESSAGE LOOK LIKE (ON THE TRANSPORT LAYER)?



MESSAGEVORTEX

HOW DOES A MESSAGE LOOK LIKE?



The encrypted peer key

The peer key protects the „inner message“

The encrypted sender key

The sender key protects the vital blocks.

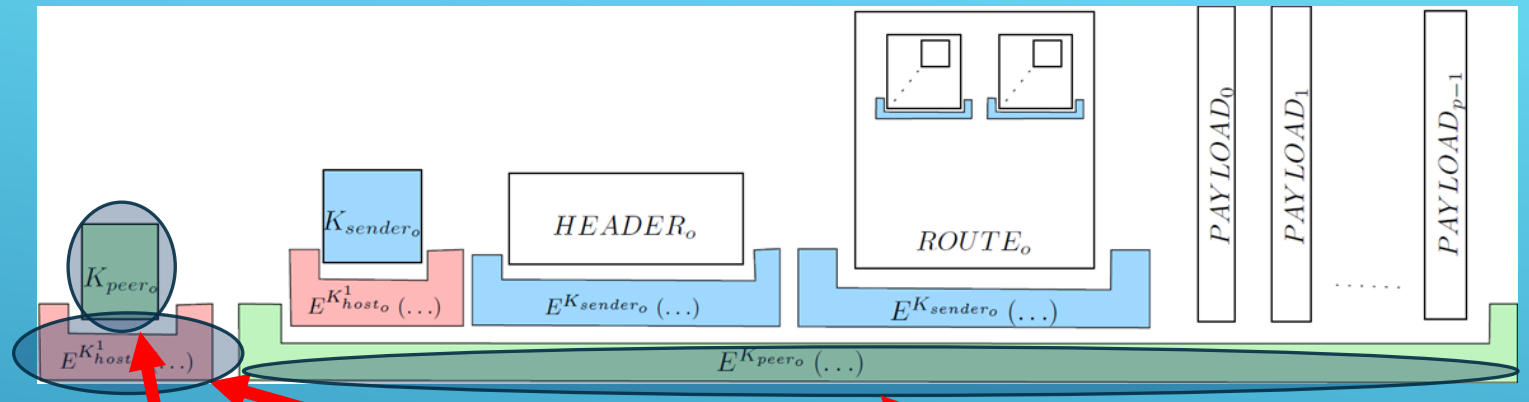
Header block „identifying“ the sender and containing requests (short term pseudonym).

Onionized routing blocks containing information on how to route and mix the message

One to many message payloads (parts of the message or decoy)

MESSAGEVORTEX

THE PEER KEY



Important side note: The sending node does not(need to) have the receiving nodes public key

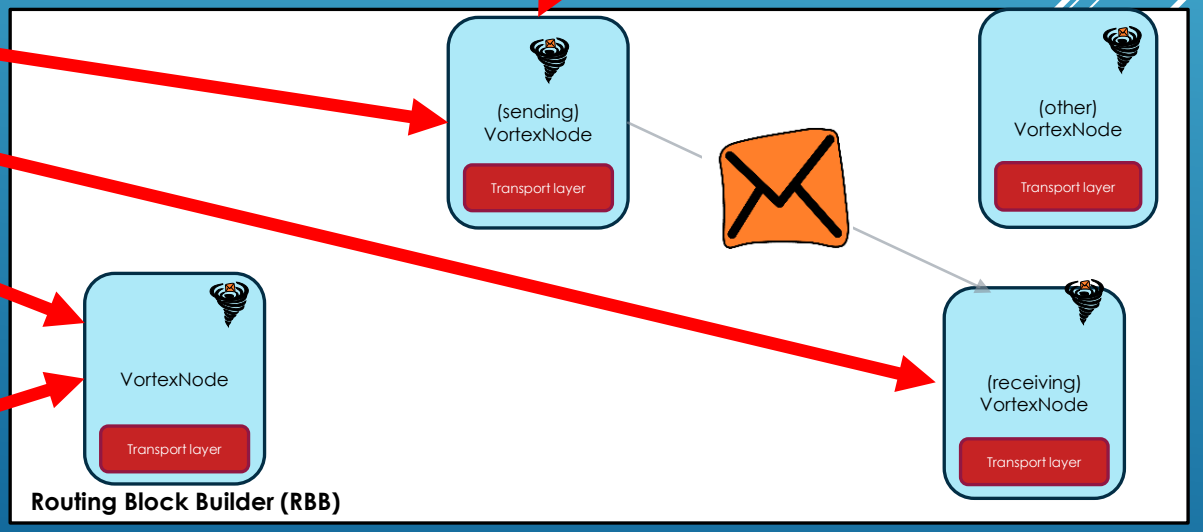
The peer key is encrypted with the receiving nodes public key

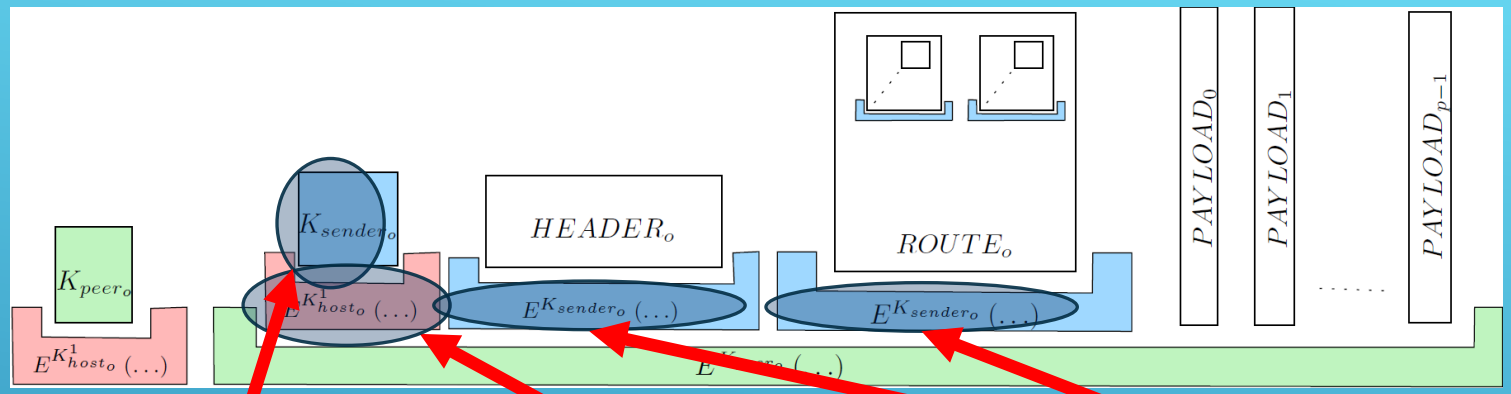
Reminder: The peer key protects what we call the inner message

The peer key was also included in the message build instructions (in the routing block) of the sending node

The peer key is only known to the two peering partners and the RBB

The RBB is the creator of the routing block





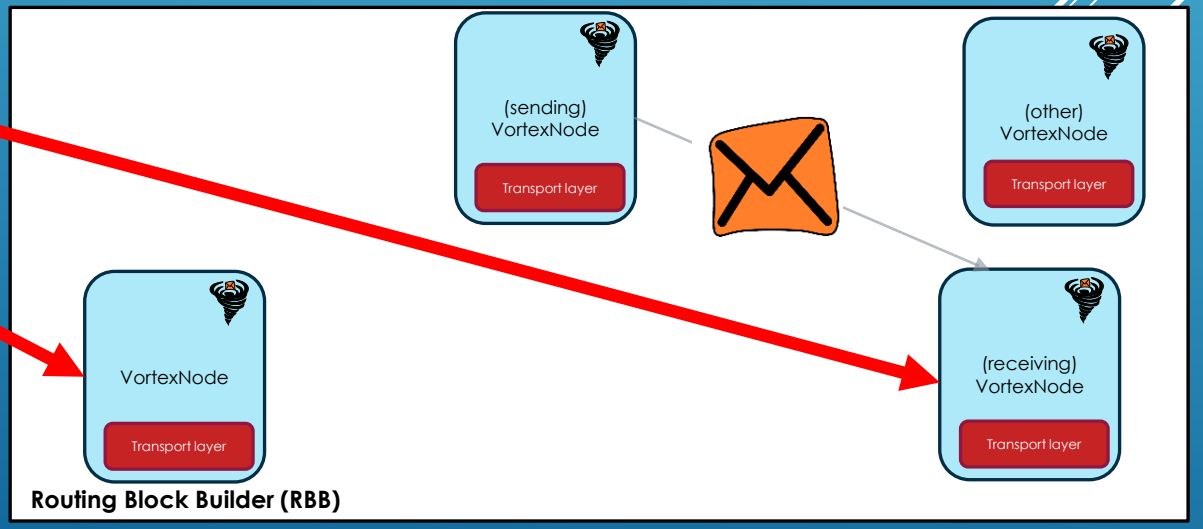
The sender key is only known to the receiving node and the RBB.

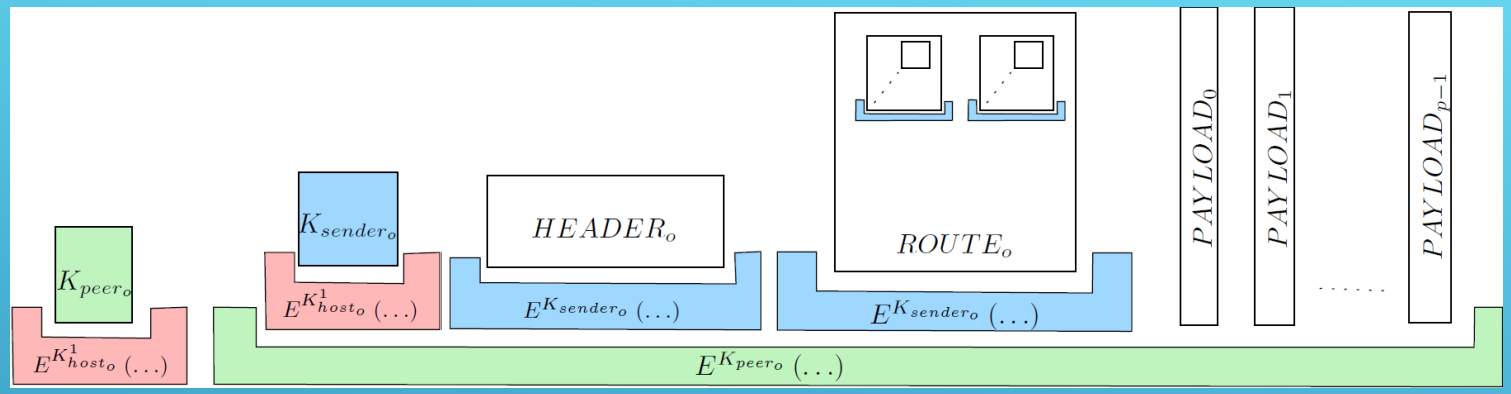
The sender key is encrypted with the receiving nodes‘ public key.

Reminder: The sender key protects the header and the routing block additionally. Otherwise, the sending peer would know its content.

MESSAGEVORTEX

THE SENDER KEY



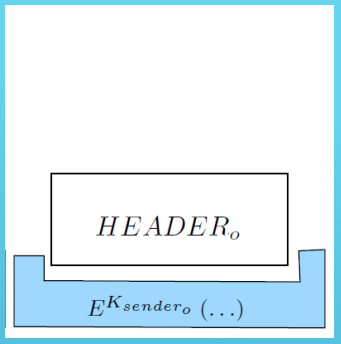


- Contains:
  - An „ephemeral ID“ (eID; public key; synonymous to the sender)
  - A serial (allows replay protection)
  - Validity interval
- May contain:
  - Requests (and the reply block)
  - Proof of work (if requesting something)
- Important: All eIDs sign their header blocks with their private key.

MESSAGEVORTEX

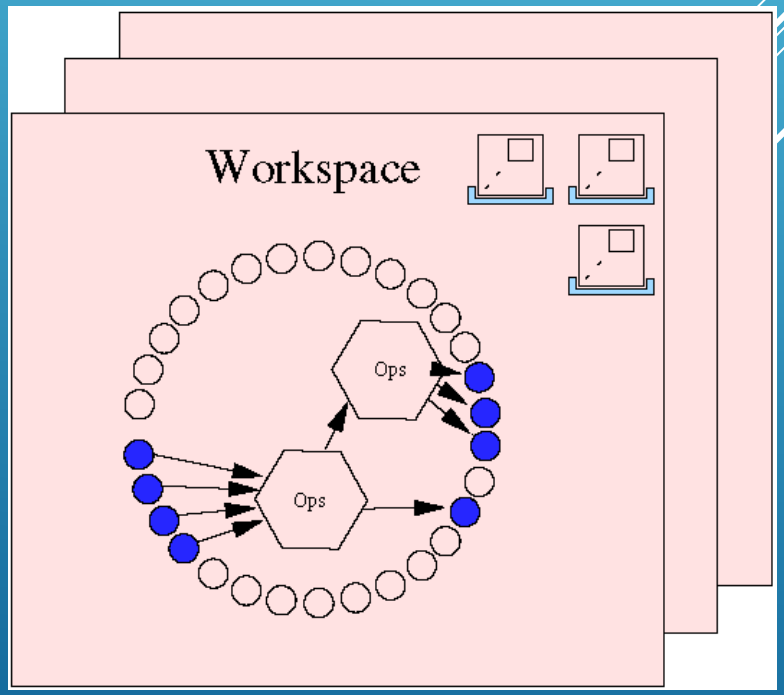
THE HEADER

- An eID is a short term synonym for the sender.
- A sender must request eIDs on each node (with a request) before sending messages with a payload.
- A sender may require solving a crypto puzzle to get an eID.
- An eID enables a sender to have a temporary, exclusively assigned storage on the node called „workspace“.
- Each eID has a quota assigned.
  - The maximum number of inbound messages with a payload.
  - The maximum number of outbound bytes of messages with a payload.
- A workspace is…
  - For one eID
  - Contains
    - Routing blocks/combos
    - Operations
    - Payload blocks (referenced by IDs)



MESSAGEVORTEX

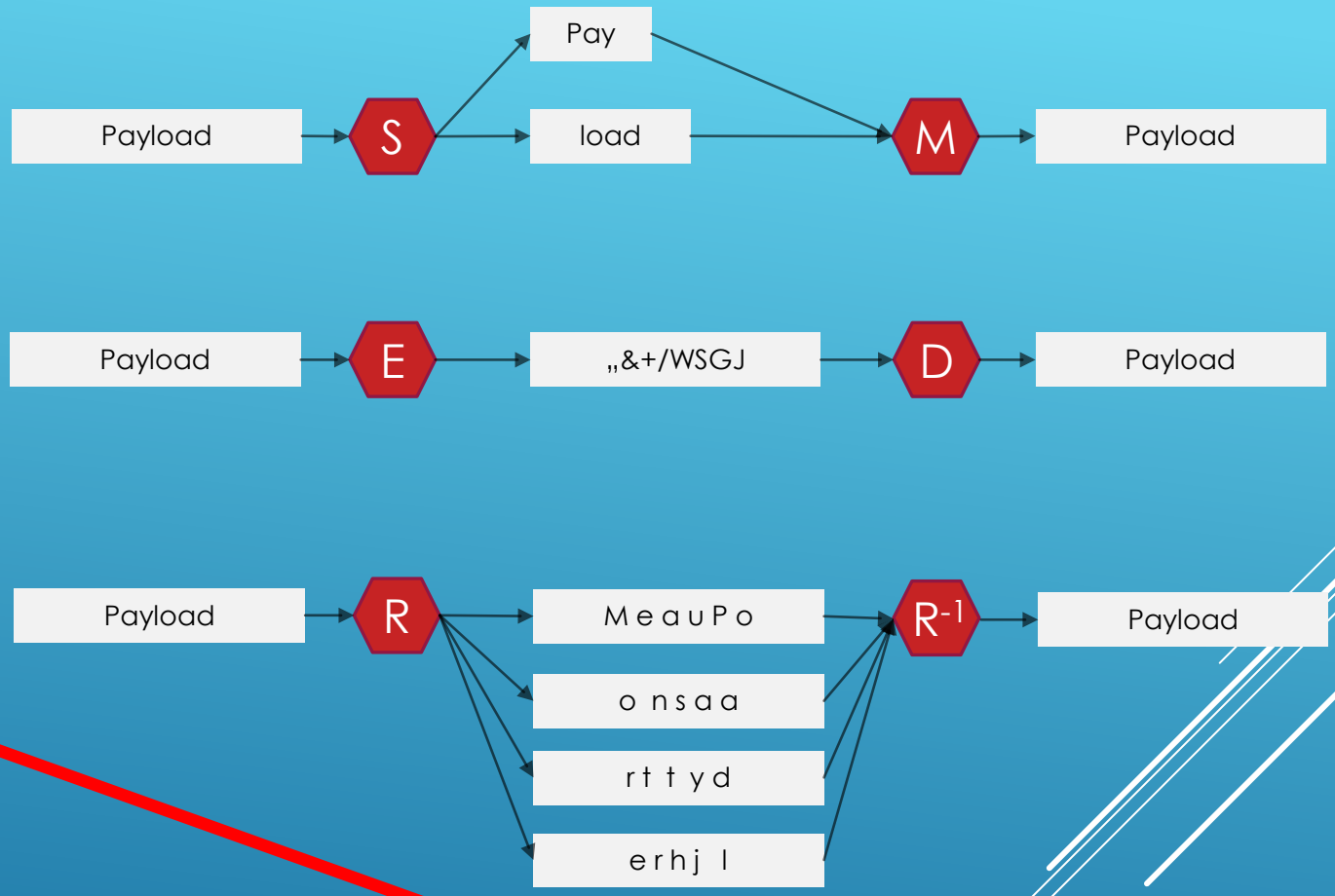
WHAT IS AN eID? WHAT IS A WORKSPACE?



MESSAGEVORTEX

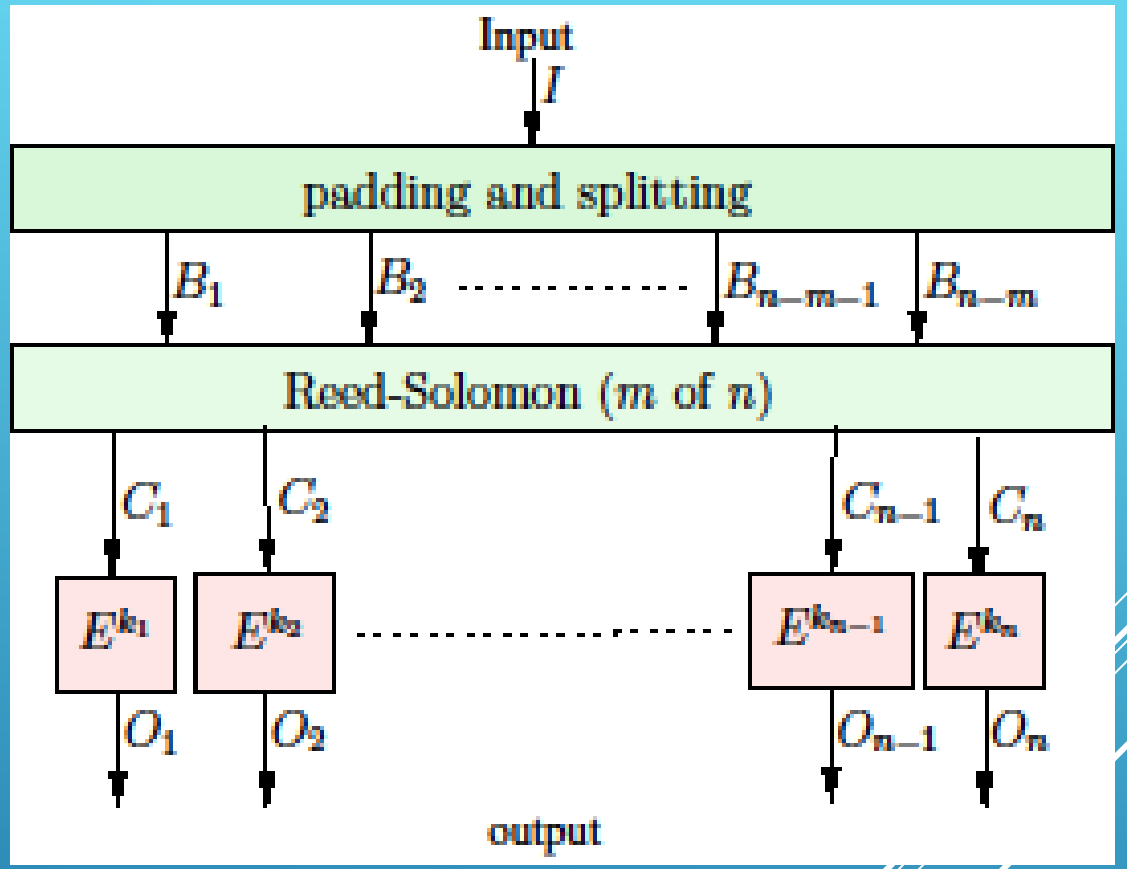
WHAT ARE THE AVAILABLE OPERATIONS?

- Each operation is defined by input Ids and output Ids.
- We have three types of operations (always a tuple):
  - Split/Merge a payload
  - Encrypt/Decrypt a payload
  - addRedundancy and removeRedundancy



This is where the magic happens

- The primary purpose of this operation is to add size to the data without leaking who is getting decoy and who is getting message traffic!
- As a side effect, we may realize redundant paths only containing partial message content.
- We carry addRedundancy out in three steps:
  - Pad data and split it into vectors → $\langle B_1, B_2, \cdots, B_{n-m} \rangle$
  - Apply Reed-Solomon to the data vectors (allowing up to *m* out of *n* vectors to fail) →$\langle C_1, C_2, \cdots, C_n \rangle$
  - Encrypt all resulting vectors →$\langle O_1, O_2, \cdots, O_n \rangle$
- This operation creates *n* blocks. *n-m* blocks are required to reverse the operation.



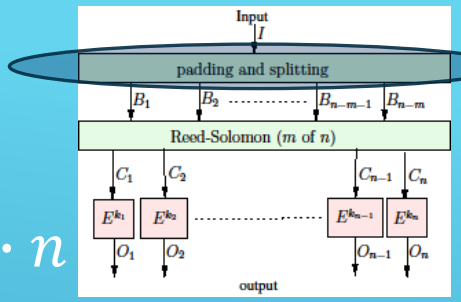
MESSAGEVORTEX

THE addRedundancy OPERATION (1)

MESSAGEVORTEX

THE addRedundancy OPERATION (2)

- We need a special padding! With a normal padding we would leak successful recovery of the original operation and may leak an illegal operation.
- We need a padding where all results are plausible
- Our padding is a specialized length prefix with a defined padding filler (seeded PRNG)
  Calculate
- The idea:
  - Take message size
  - Plus a specified (by RBB) number of bytes (fixed stuffing)
    Reflected by C2
  - Plus $C1 \cdot (len(\boldsymbol{X}) - 4)$
    This will disappear when decoding p
  - *Decoding:*
    - *Take first four bytes (p)*
    - *Calculate* $i = len(\boldsymbol{M}) = p \ (mod \ len(\boldsymbol{X}) - 4)$
- *The rest is filled with a seeded PRNG* → A known seed allows to verify padding.



$$i = len(\boldsymbol{M})$$

$$e = blocksize(E^K) \cdot n$$

$$l = \left\lceil \frac{i + C2}{e} \right\rceil \cdot e$$

$$p = i + \left(C1 \cdot l \left(mod \left\lfloor \frac{2^{32} - 1 - i}{l} \right\rfloor \cdot l \right) \right)$$

$$X = \langle p, \boldsymbol{M}, R(s, l - i) \rangle$$

The padded message

The padding value (length: 4 Bytes unint)

The message

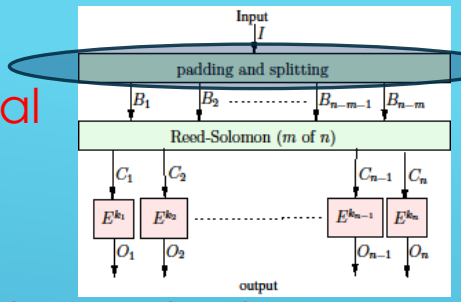
The padding data (PRNG seeded with s and length $l - i$)

$$i = len(\boldsymbol{M})$$
$$e = blocksize(E^K) \cdot n$$
$$l = \left\lceil \frac{i + C2}{e} \right\rceil \cdot e$$
$$p = i + \left(C1 \cdot l \left(mod \left\lfloor \frac{2^{32} - 1 - i}{l} \right\rfloor \cdot l \right) \right)$$
$$X = \langle p, \boldsymbol{M}, R(s, l - i) \rangle$$
$$= \langle p, \boldsymbol{M}, R(s, l - (p \bmod (len(\boldsymbol{X}) - 4)) \rangle$$

The length of the original message

Block size of the operation (dependent on the number of stripes and the cipher block size)

Length of padded stream len($\langle \boldsymbol{M}, R(s, l - i) \rangle$) without the padding itself

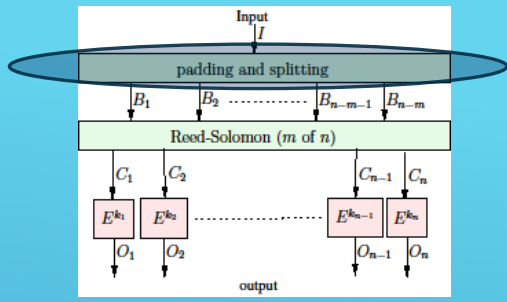


MESSAGEVORTEX

THE addRedundancy OPERATION (3)

$$i = len(\boldsymbol{M})$$
$$e = blocksize(E^K) \cdot n$$
$$l = \left\lceil \frac{i + C2}{e} \right\rceil \cdot e$$
$$p = i + \left(C1 \cdot l \left(mod \left\lfloor \frac{2^{32} - 1 - i}{l} \right\rfloor \cdot l \right) \right)$$
$$X = \langle p, \boldsymbol{M}, R(s, l - i) \rangle$$
$$= \langle p, \boldsymbol{M}, R(s, l - (p \bmod (len(\boldsymbol{X}) - 4))) \rangle$$

C2 is a fixed value of bytes to be added to the message.
By adding C2, we make it possible, that value *p<=len(X)-4* is valid.



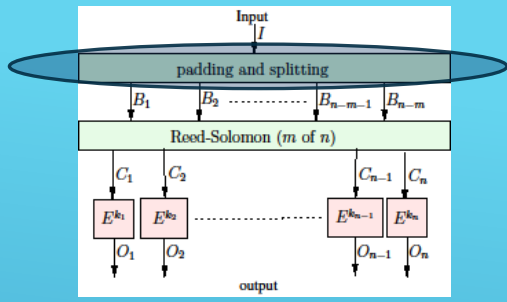
C1 is a generator for multiples of *l* and guarantees that any p>len(X)-4 is valid as well.

MESSAGEVORTEX

THE addRedundancy OPERATION (3)

We require C1 and C2 when decoding only for verifying the padding value. We do NOT require the values for successful decoding.

- Up until now we have padded the message with the following properties:
  - The message has a perfect length to allow:
    - Apply the Read-Solomon operation without further padding
    - Apply the encryption without further padding



Interlude:
Reed Solomon (RS) is a code used typically for error correction. It adds a constant but selectable error correction information to the data, allowing m out of n data chunks to fail.
In the real world, RS is typically used in a stripped manner in RAID6 systems to allow multiple disks to fail without losing data.

→ Let‘s apply Reed-Solomon

MESSAGEVORTEX

THE addRedundancy OPERATION (4)

MESSAGEVORTEX

THE addRedundancy OPERATION (5)

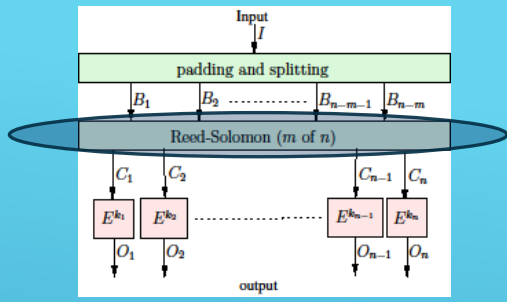
- $t = n - 1$
- $\boldsymbol{A} = vec2mat\left(X, \frac{len(\boldsymbol{X})}{m}\right)$
- $\boldsymbol{V} = \begin{pmatrix} 0^0 & \cdots & 0^{m-1} \\ \vdots & \ddots & \vdots \\ t^0 & \cdots & t^{m-1} \end{pmatrix}$
- $\boldsymbol{P} = \boldsymbol{V} \cdot \boldsymbol{A}\left(GF(2^{\omega})\right)$
- $\langle C_1, C_2, \cdots, C_n \rangle = \text{row2vec}(\boldsymbol{P})$

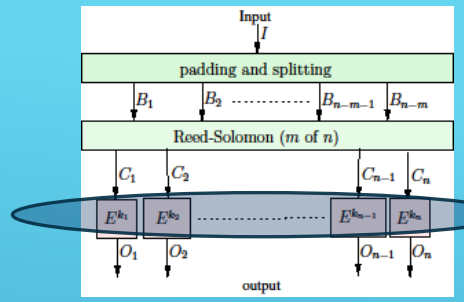
We allign the padded data in a matrix

We prepare a suitable Vandermonde matrix (*m* rows and *n* colums)

We apply RS
The result is a matrix with the message and added redundancy information.

We extract the rows as vector

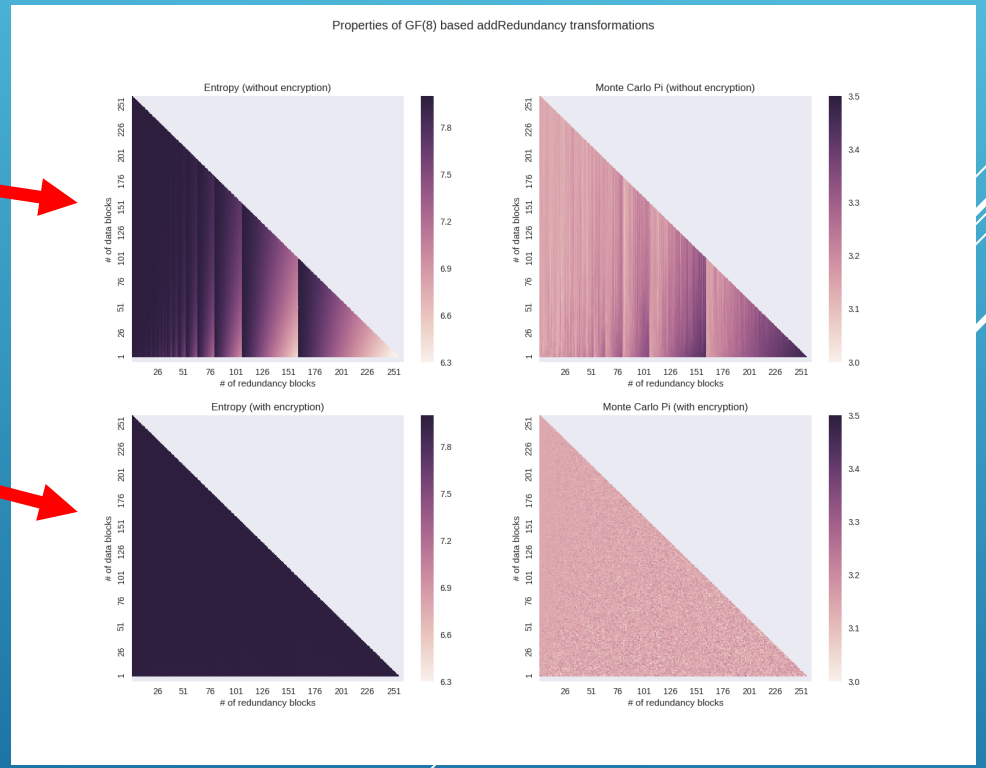




- Now we do the last step (the encryption)
- $O_i = E^{K_i}(C_i)$
- Encryption is necessary as the chunks $(O_0, \cdots, O_n)$ leak parameters of the RS operation

Entropy and Monte Carlo PI without encryption with varying parameters on the same block WITHOUT encryption

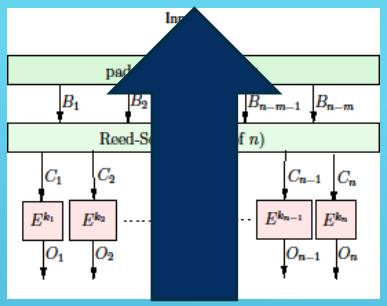
Entropy and Monte Carlo PI without encryption with varying parameters on the same block WITH encryption



MESSAGEVORTEX

THE addRedundancy OPERATION (5)

$$\boldsymbol{X} = \langle p, \boldsymbol{M}, R(s, p \bmod (len(\boldsymbol{X}) - 4)) \rangle$$



- The reverse operation is applied by inversoing all operations
  - Pick any available $n - m$ blocks of payload (they all must have the same length).
  - Decrypt the blocks by applying $C_j = D^{K_j}(I_j)$ and arrange them in rows of a matrix $\boldsymbol{A}$.
  - Eliminate in the Vandermonde matrix the discarded payload lines → $\boldsymbol{V'}$
  - invert $\boldsymbol{V'} \rightarrow \boldsymbol{V'^{-1}}$
  - Calculate $\boldsymbol{M} = \boldsymbol{V'^{-1}A}$ $(\boldsymbol{GF}(2^{\omega}))$ resulting in $\langle B_1, B_2, \dots, B_{n-m} \rangle = row2vec(\boldsymbol{M})$and arrange **M** as a byte stream.
  - Get the first four bytes p and cut the payload to its original size by calculating $\mathrm{i} = p \bmod (len(\boldsymbol{X}) - 4)$.
- Verify the padding (if possible; requires s):
  - Verify that the padded space matches $R(s, len(\boldsymbol{X}) - p - 4)$

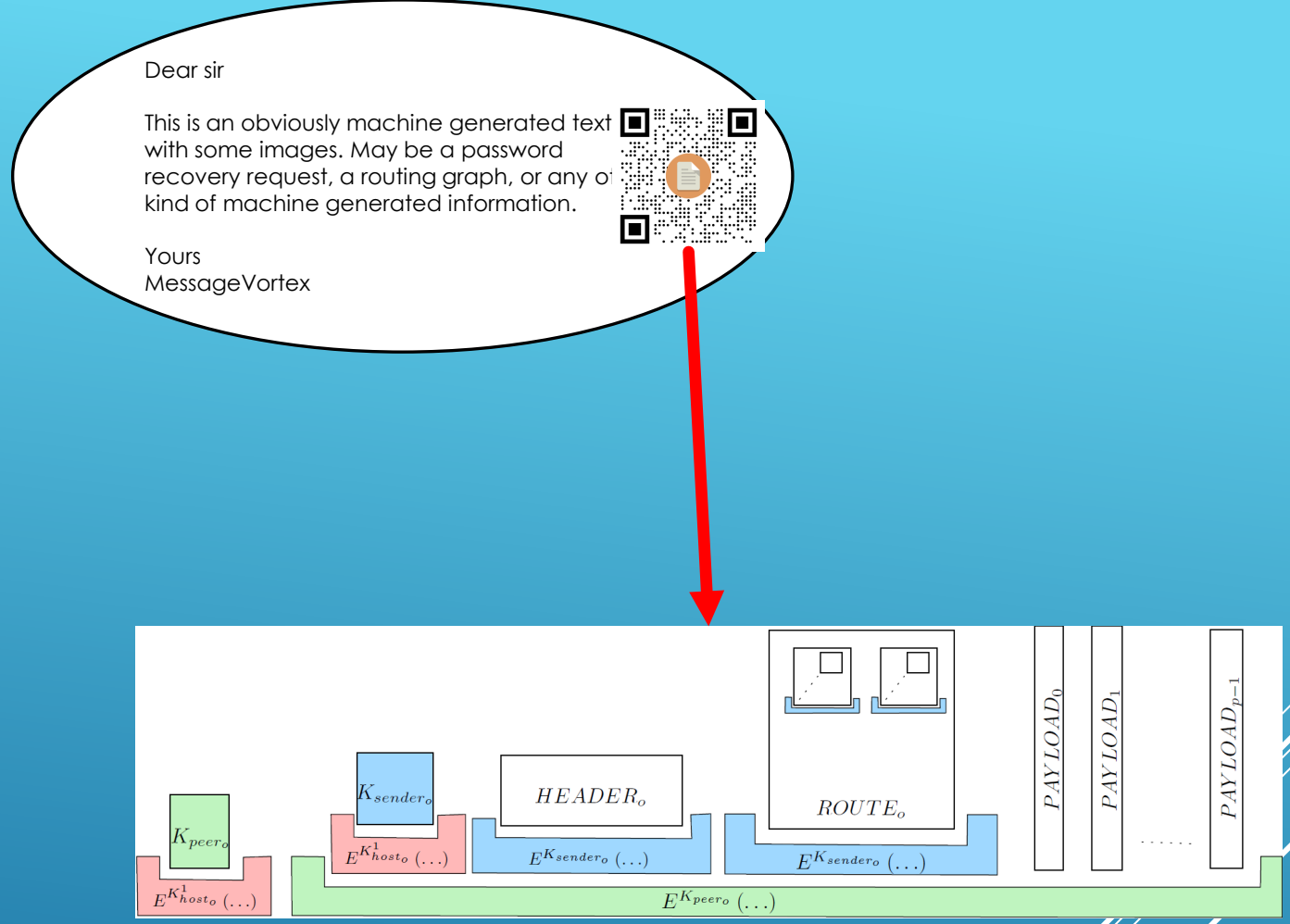
MESSAGEVORTEX

THE addRedundancy OPERATION (6)

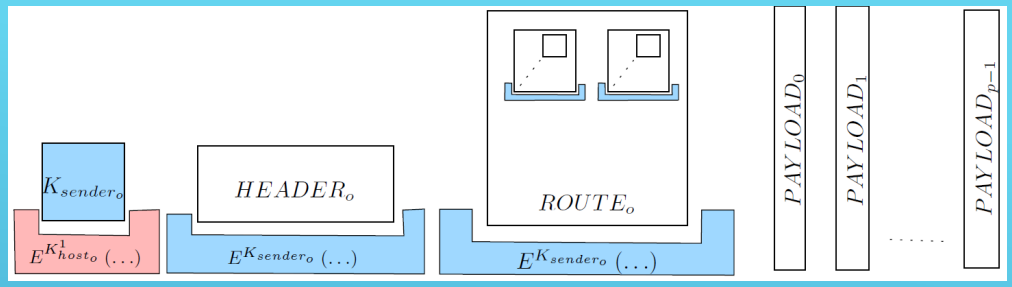
MESSAGEVORTEX

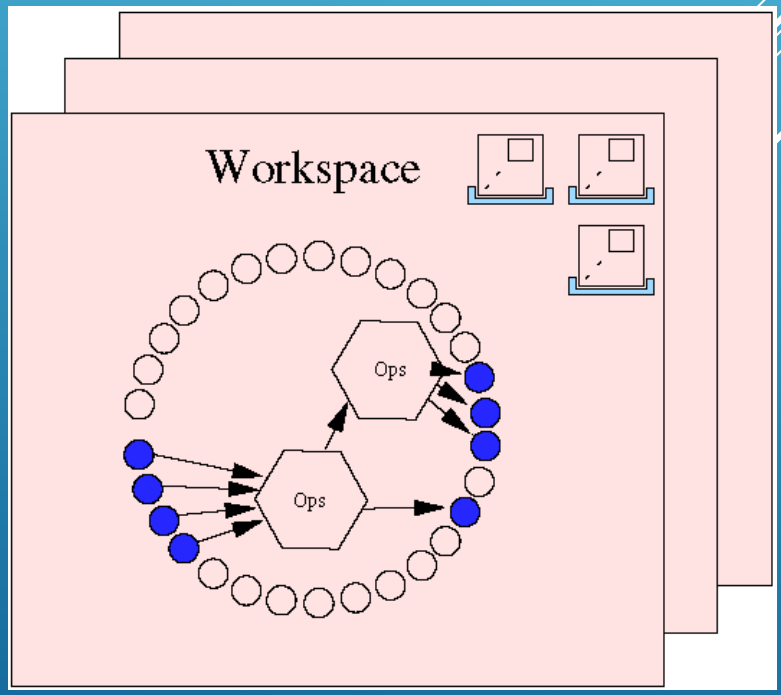
Processing an incoming message

- **Extract message by the blending layer**
  - Extract message prefix block and decrypt it.
  - If a valid structure with a peer key is provided continue with the inner message by decrypting with the key provided in the prefix block.
- Get sender key and decrypt header, and routing block
- Verify signature of header and rules for processing
- Process header requests (if applicable)
- Apply mapping operations for IDs 1 up to 127 into workspace (mapping of payload blocks into the workspace)
- Add operations to the workspace
- Add routing combos to the workspace
- If a message is written to payload ID 0 then this message is for the local user



- Extract message by the blending layer
  - **Extract message prefix block and decrypt it.**
  - **If a valid structure with a peer key is provided continue with the inner message by decrypting with the key provided in the prefix block.**
- Get sender key and decrypt header, and routing block
- Verify signature of header and rules for processing
- Process header requests (if applicable)
- Apply mapping operations for IDs 1 up to 127 into workspace (mapping of payload blocks into the workspace)
- Add operations to the workspace
  - If a message is written to payload ID 0 then this message is for the local user
- Add routing combos to the workspace
- If a message is written to payload ID 0 then this message is for the local user

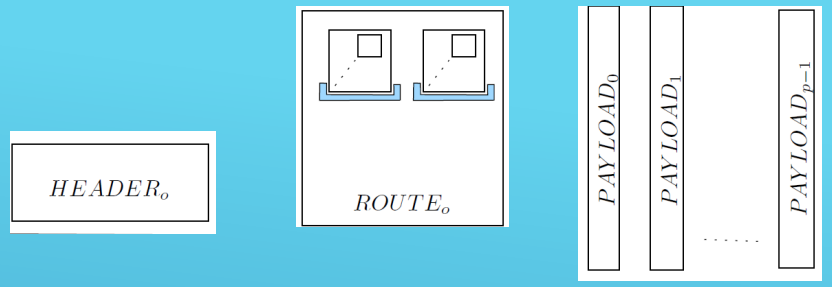




MESSAGEVORTEX

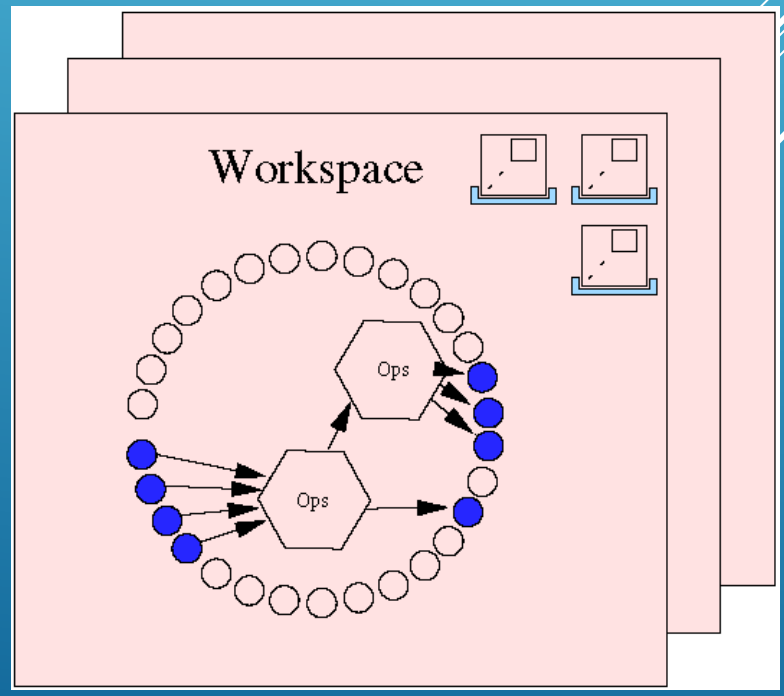
Processing an incoming message

- Extract message by the blending layer
  - Extract message prefix block and decrypt it.
  - If a valid structure with a peer key is provided continue with the inner message by decrypting with the key provided in the prefix block.
- **Get sender key and decrypt header, and routing block**
- Verify signature of header and rules for processing
- Process header requests (if applicable)
- Apply mapping operations for IDs 1 up to 127 into workspace (mapping of payload blocks into the workspace)
- Add operations to the workspace
  - If a message is written to payload ID 0 then this message is for the local user
- Add routing combos to the workspace



MESSAGEVORTEX

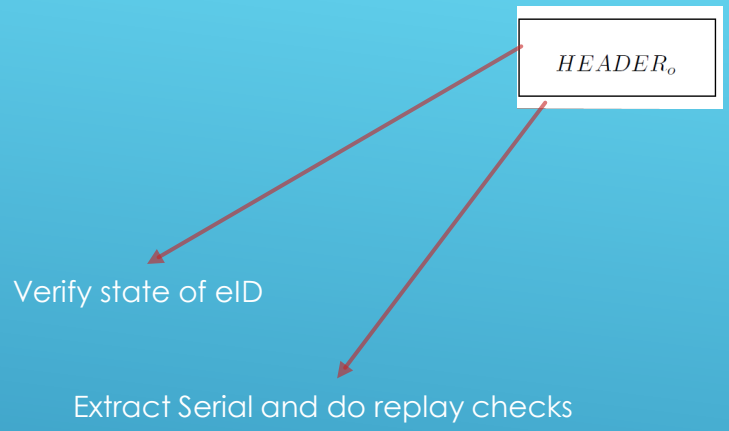
Processing an incoming message

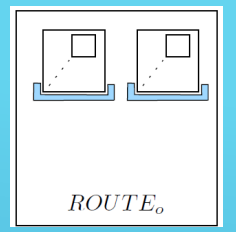


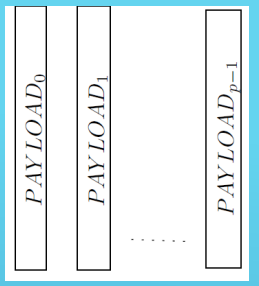
MESSAGEVORTEX

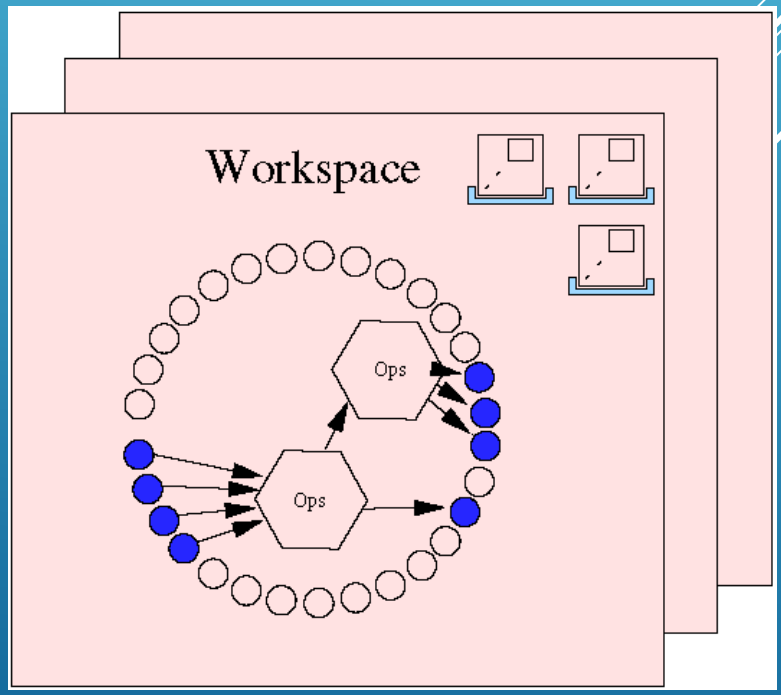
Processing an incoming message

- Extract message by the blending layer
  - Extract message prefix block and decrypt it.
  - If a valid structure with a peer key is provided continue with the inner message by decrypting with the key provided in the prefix block.
- Get sender key and decrypt header, and routing block
- **Verify signature of header and rules for processing**
- Process header requests (if applicable)
- Apply mapping operations for IDs 1 up to 127 into workspace (mapping of payload blocks into the workspace)
- Add operations to the workspace
  - If a message is written to payload ID 0 then this message is for the local user
- Add routing combos to the workspace





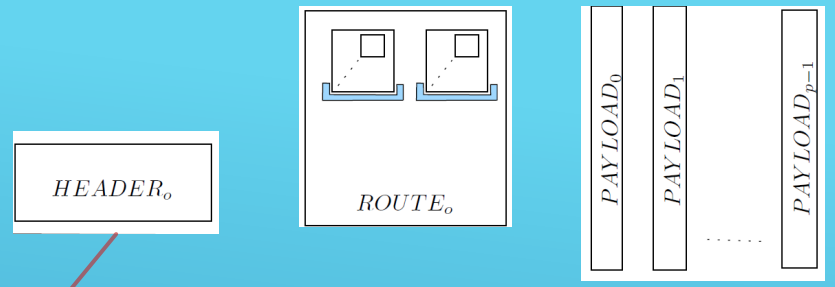




- Extract message by the blending layer
  - Extract message prefix block and decrypt it.
  - If a valid structure with a peer key is provided continue with the inner message by decrypting with the key provided in the prefix block.
- Get sender key and decrypt header, and routing block
- Verify signature of header and rules for processing
- **Process header requests (if applicable)**
- Apply mapping operations for IDs 1 up to 127 into workspace (mapping of payload blocks into the workspace)
- Add operations to the workspace
  - If a message is written to payload ID 0 then this message is for the local user
- Add routing combos to the workspace

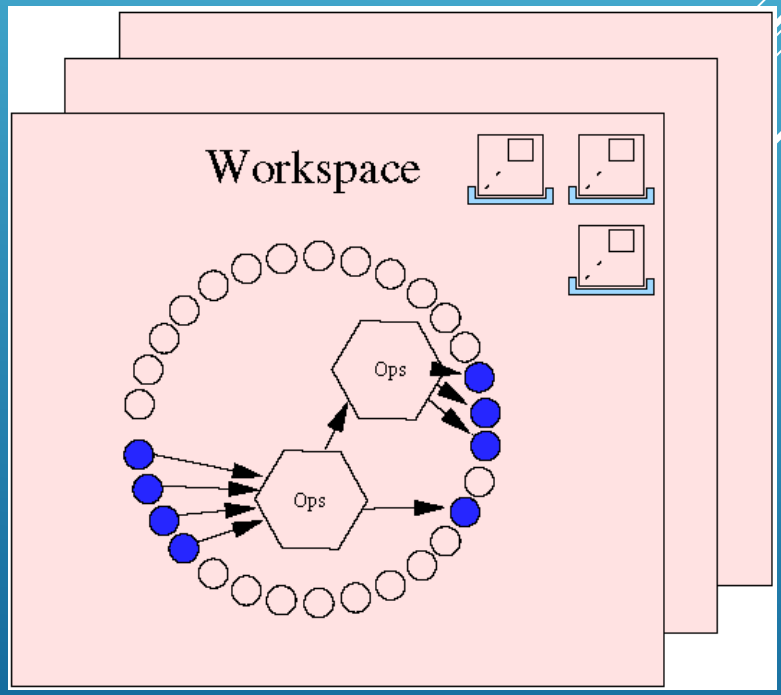
MESSAGEVORTEX

Processing an incoming message



Process header requests:
- Create new eID
- Get node capabilities
- Get quotas
- Increase quotas
- Request routing nodes
- Replace identity

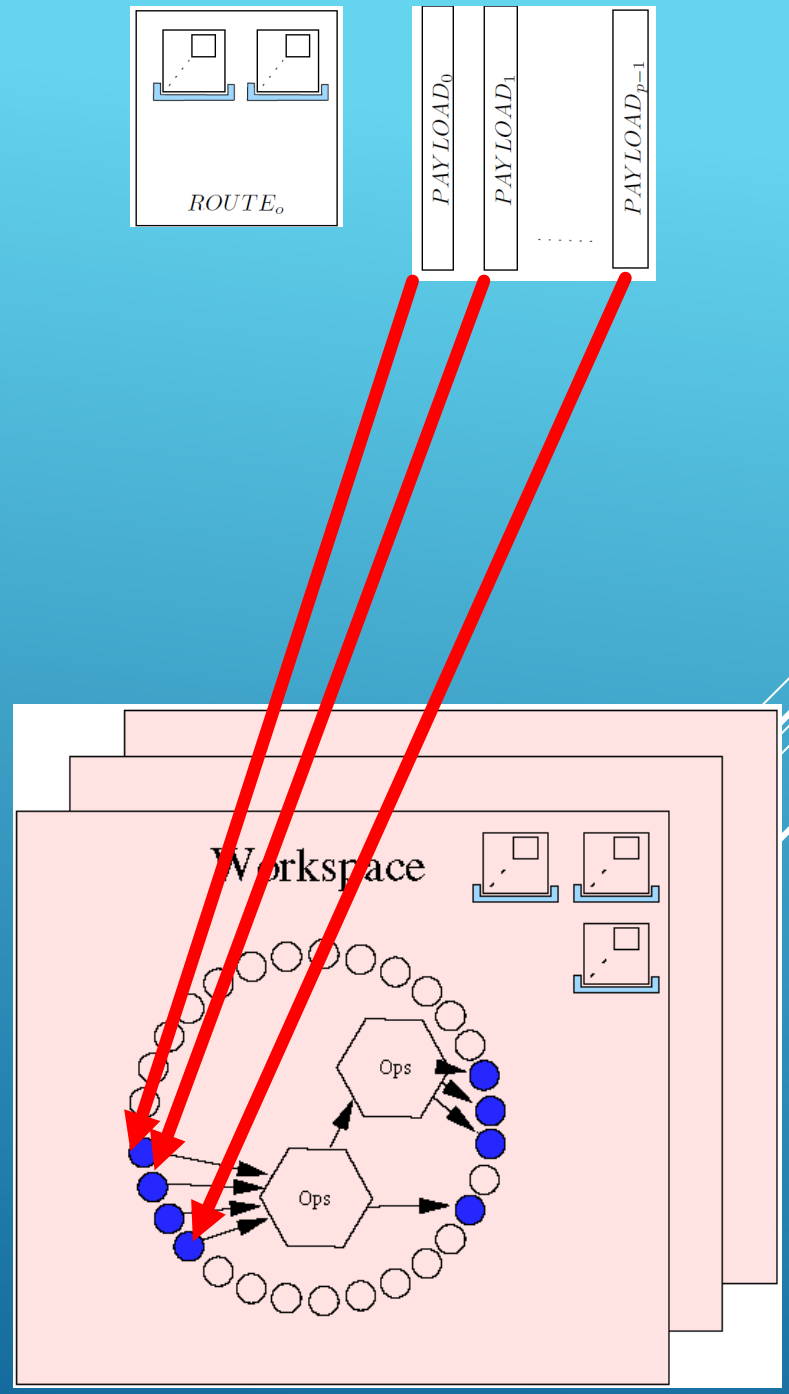
PoW or anonymous payment may apply



- Extract message by the blending layer
  - Extract message prefix block and decrypt it.
  - If a valid structure with a peer key is provided continue with the inner message by decrypting with the key provided in the prefix block.
- Get sender key and decrypt header, and routing block
- Verify signature of header and rules for processing
- Process header requests (if applicable)
- **Apply mapping operations for IDs 1 up to 127 into workspace (mapping of payload blocks into the workspace)**
- Add operations to the workspace
  - If a message is written to payload ID 0 then this message is for the local user
- Add routing combos to the workspace

MESSAGEVORTEX

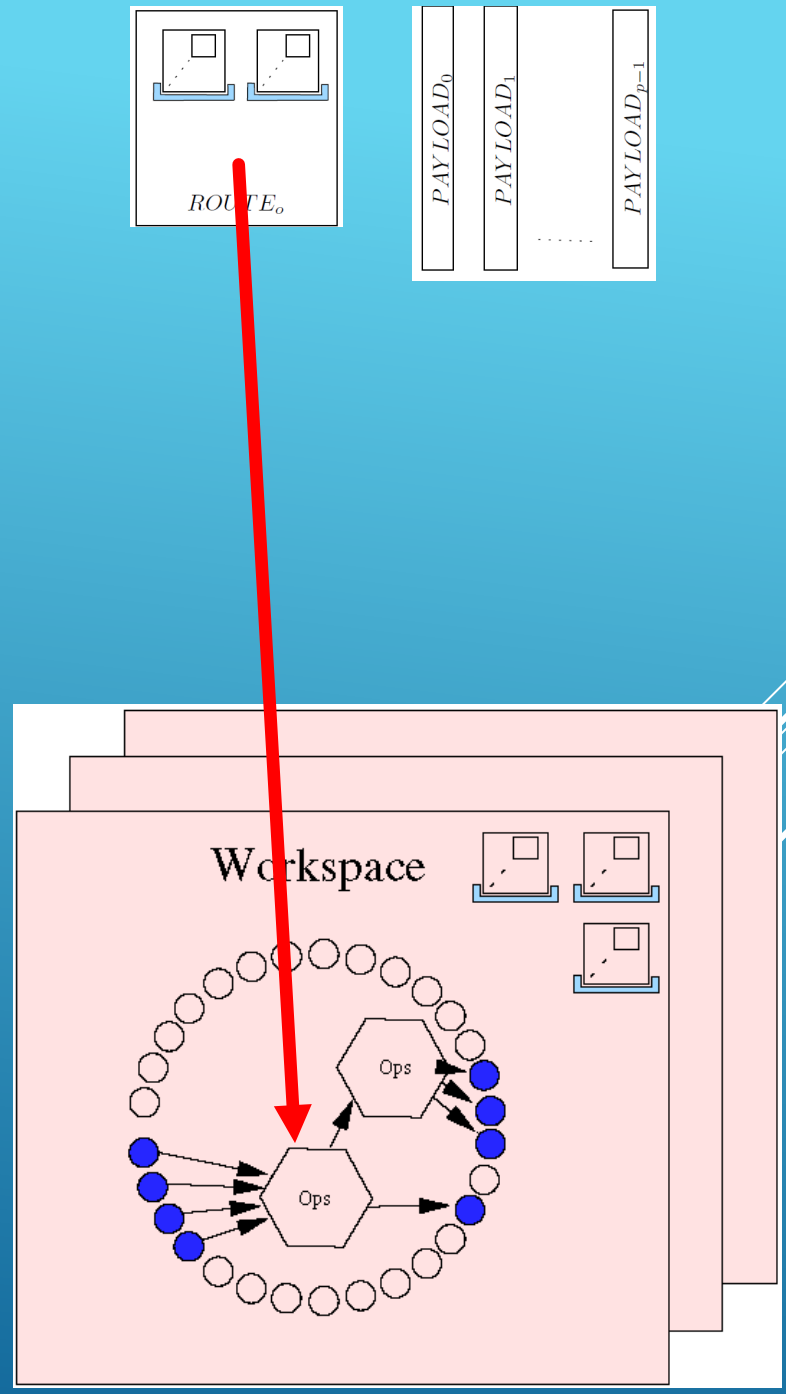
Processing an incoming message



- Extract message by the blending layer
  - Extract message prefix block and decrypt it.
  - If a valid structure with a peer key is provided continue with the inner message by decrypting with the key provided in the prefix block.
- Get sender key and decrypt header, and routing block
- Verify signature of header and rules for processing
- Process header requests (if applicable)
- Apply mapping operations for IDs 1 up to 127 into workspace (mapping of payload blocks into the workspace)
- **Add operations to the workspace**
  - **If a message is written to payload ID 0 then this message is for the local user**
- Add routing combos to the workspace

MESSAGEVORTEX

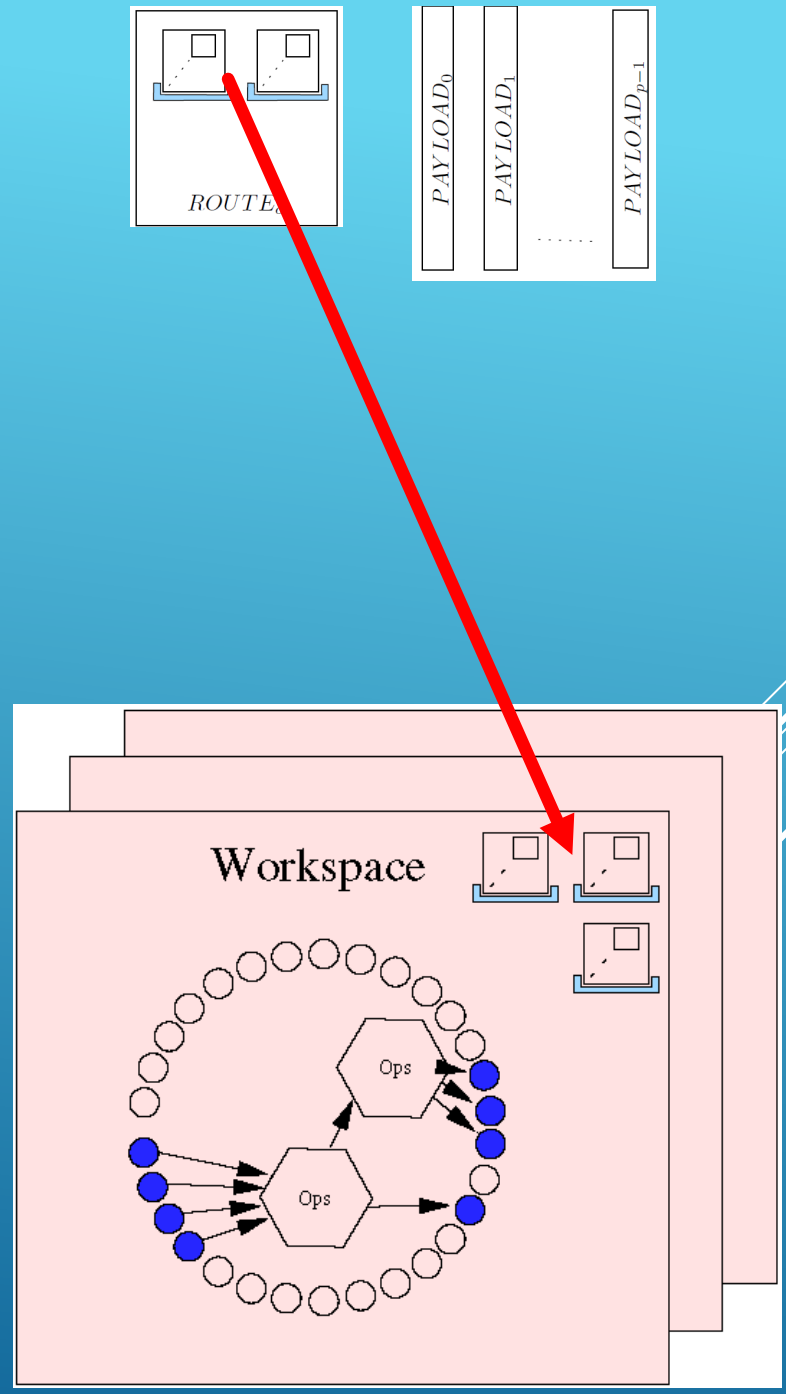
Processing an incoming message



- Extract message by the blending layer
  - Extract message prefix block and decrypt it.
  - If a valid structure with a peer key is provided continue with the inner message by decrypting with the key provided in the prefix block.
- Get sender key and decrypt header, and routing block
- Verify signature of header and rules for processing
- Process header requests (if applicable)
- Apply mapping operations for IDs 1 up to 127 into workspace (mapping of payload blocks into the workspace)
- Add operations to the workspace
  - If a message is written to payload ID 0 then this message is for the local user
- **Add routing combos to the workspace**

MESSAGEVORTEX

Processing an incoming message



- Triggered by the timing information in the routing combo
- Assemble the payload blocks by applying the operations provided. If done execute the mapping to the IDs 1-127
- Add routing block, prefix blocks (containing the sender and peer keys), and the header block to the message (all blocks are provided as preencrypted binary blobs)
- Encrypt inner message
- Apply blending according to spec in the routing combo.
- Send message by using the transport layer.

MESSAGEVORTEX

Processing an outgoing message

- Possible by routing parts of the message back to the originator.
  - As all operations are known the resulting payload blocks are foreseeable.
- If traffic is illicit, additional messages picking up more parts from other workspaces may be injected to narrow down diagnosis of misbehaving nodes
- Successful message delivery may be tracked by routing parts of the message from the recipients node back to the sender (directly or indirectly).

MESSAGEVORTEX

ROUTING DIAGNOSIS

- In General:
  - Use addRedundancy to add decoy
  - Use either encrypt or decrypt to (de)onionize
  - Use Split and merge to join/distribute traffic (requires onionizing or addRedundancy step)

MESSAGEVORTEX

ROUTING STRATEGIES (GENERAL)

MESSAGEVORTEX

ROUTING STRATEGIES (SPECIFIC)

- Creating decoy traffic
  - Use addRedundancy to add decoy
    - Same restrictions apply as if sending a message part
- Removing decoy traffic
  - Do not do anything with a payload block
    - For the node this looks as:
      - Decoy
      - Redundant traffic not picked up
      - Diagnosis traffic not handled
      - Malfunctioning path
- Sending received decoy traffic
  - Same as sending a message part

- Sending a message part
  - Split and encrypt
  - Join and encrypt
  - Encrypt only
    (may be decrypt only as well; Use with caution)
  - addRedundancy
    - May include additional split or join
  - **NO** reocurrence of data
    - No forwarding of data without at least one encryption step involved
    - No reverse operations anywhere on untrusted nodes

- Choose VortexNodes and allocate eIDs if required
- Choose a directed suitable graph (Nodes=VortexNodes; Edges=Messages).
- Select path(s) for message transferal to final recipient.
- Select operations
  - For the message
  - For the decoy traffic
- Select timing information for the blocks.
- Assemble routing block (it is always one at the beginning).
- Apply routing block to message and hand it over to the router (local)

MESSAGEVORTEX

A SIMPLE STRATEGY FOR BUILDING A ROUTING BLOCK

- In a censoring environment
  - You need to collect them yourself (manually)
- In a non censoring environment (where usage of MessageVortex is no problem)
  - Query the nodes directly with a capability request (HeaderRequestCapability)
  - Ask a node to recover public routing nodes (`HeaderRequestNodes`)

MESSAGEVORTEX

WHERE DO I GET MY PEER PARTNERS FROM?

MESSAGEVORTEX

A WHAT ARE ITS BENEFITS?

- The first holistic approach targeting the needs of a partially/fully censoring environment
- Bases on proven technology
  - Reed-Solomon operations are well known
  - Mixing (and its limitations) are well known
- Is Cryptoagile
- Allows diagnosis
- Allows local trust
- Gives full control of all privacy related parameters to the builder of the routing block.
- Contributes
  - A new padding type not leaking any information about successful decryption.
  - An addRedundancy operation allowing to add redundancy and or decoy traffic without letting the routing node the type of data know.

QUESTIONS?