



MessageVortex

Unlinkable, censorship-resistant
communication



TABLE OF CONTENT

- ▶ Why Anonymity
- ▶ 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 (and eIDs and workspaces)
 - ▶ The Routing Block
 - ▶ The Operations
 - ▶ Processing
 - ▶ Routing Strategies
 - ▶ What are its benefits
- ▶ Any Questions Left?



BEWARE: High pace ahead.





Everyone shall have the right to hold opinions without interference

Everyone shall have the right to freedom of expression; this right shall include freedom to seek, receive and impart information and ideas of all kinds, regardless of frontiers, either orally, in writing or print, in the form of art, or through any other media of his choice.

Article 19 of the International Covenant on Civil and Political Rights (ICCPR)

MESSAGEVORTEX

WHY DO WE NEED IT?

In other words: Wherever censorship exists, 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...)
- ▶ Censorship is done to...
 - ▶ fight „terrorism.“
 - ▶ fight „state decomposing elements.“
 - ▶ stopping “immoral or anti-religious works.”

Agreement:
Censorship exists,
its use may be questionable

MESSAGEVORTEX

WHY DO WE NEED IT?



Conclusion: Anonymity matters especially in **censored** environments.

- ▶ Achieving anonymity is hard from an academic point
- ▶ Achieving anonymity in a censored environment is far harder
 - ▶ All identifiable technologies are censored
- ▶ We lack systems designed for censoring environments
 - ▶ Some offer extensions which try to address this issue (e.g., Tor and its pluggable transports)

MESSAGEVORTEX

WHY DO WE NEED IT?



- ▶ Onion/Garlic Routing (SOR, Tor, I2P, ...)
- ▶ DHT (Salsa, Tarzan, ...)
- ▶ DC-Nets (Herbivore, Dissent, ...)
- ▶ Broadcast (Hordes, ...)
- ▶ Distributed Storages (Freenet, Guntella(2), ...)

- ▶ Remailers (0-III)
 - ▶ Nym Remailers
 - ▶ Cypherpunk
 - ▶ Mixmaster
 - ▶ Mixminion

PRE-EXISTING SYSTEMS

None of these systems were designed with a censor in mind!



- ▶ 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.

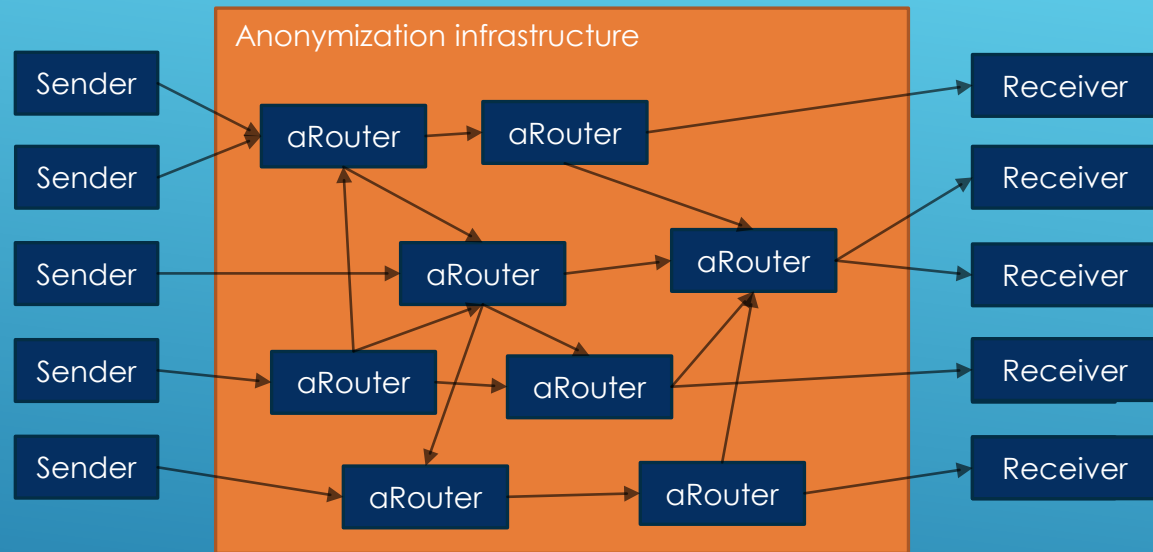
- ▶ Juristic

- ▶ Every jurisdiction applies their own laws
- ▶ Laws may disagree with each other

1010
1010



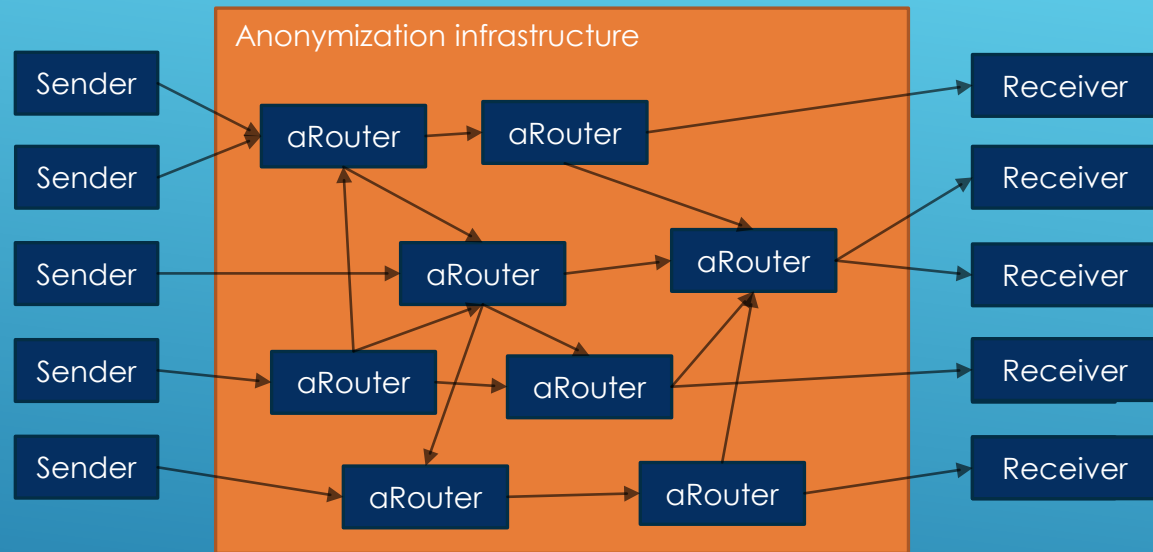
WHAT ARE THE PROBLEMS?



- Identifiable meta data
- Identifiable messages
- Replayable messages
- Bugable parts
- Tagable parts
- Bootstrapping problem
- Identification problem
- Huge traffic overhead

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

MESSAGEVORTEX

GENERAL WEAK SPOTS OF ANONYMIZATION SOLUTIONS



▶ Principles

- ▶ Unobservable from the outside
- ▶ 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
- ▶ Decoy traffic is not even identifiable for the node generating or routing.
- ▶ Resistant to common attacks which are hard to cover

HOW DOES IT WORK

THE GENERAL IDEA



- ▶ Messages are sent from node to node in a mix type like system
 - ▶ One VortexNode consists of...
 - ▶ An always connected device of an end-users
 - ▶ Common Transport account (e.g., Gmail account)
 - ▶ Mixer Applies operations controlled by the routing block builder (RBB)
- ▶ Outer Messages of the system
 - ▶ Have machine generated content (e.g., password recovery request)
 - ▶ Include at least one image
 - ▶ VortexMessage is steganographically hidden within the Image

HOW DOES IT WORK

THE GENERAL IDEA (2)



- ▶ 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:
- ▶ Any payload block may be decoy or true message content!
 - ▶ Even the routing node generating decoy cannot tell decoy from message traffic apart.
- ▶ Agile
 - ▶ Broken (crypto) algorithms must not break the protocol
 - ▶ Network does not depend on a monolithic software version
 - ▶ Any type of algorithm should have an alternative

HOW DOES IT WORK

THE GENERAL IDEA (2; FUNCTIONING)

- ▶ 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

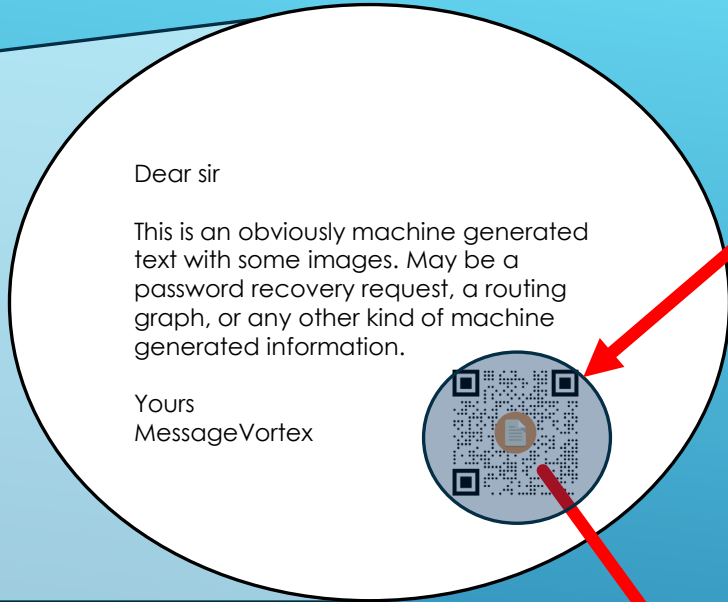
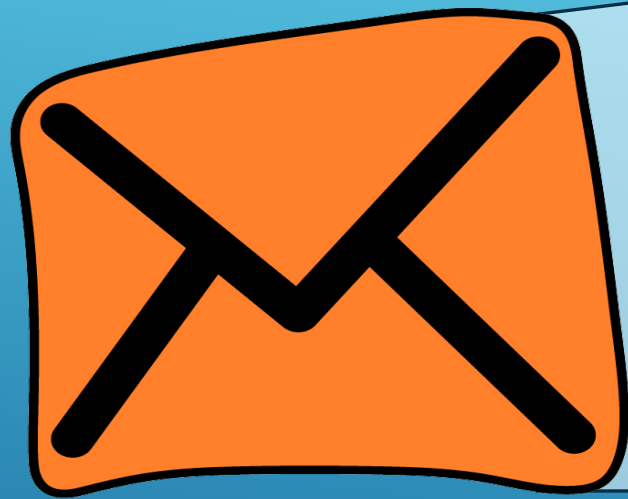


MESSAGEVORTEX

HOW DOES IT WORK?

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

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

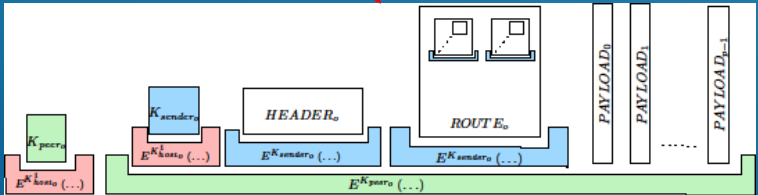


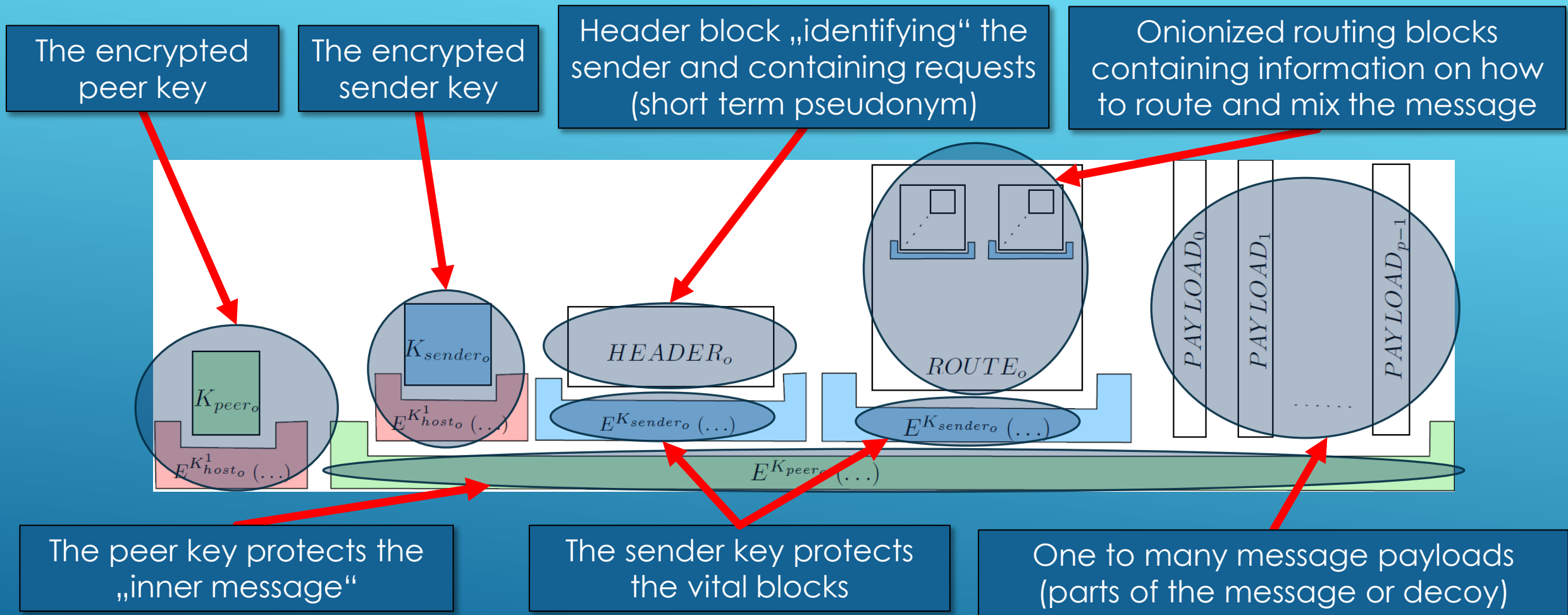
- Generated image. E.g.,
- QR-Code
 - Network graph
 - Identicon
 - Gravatar
 - Random-Avatr

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

MESSAGEVORTEX

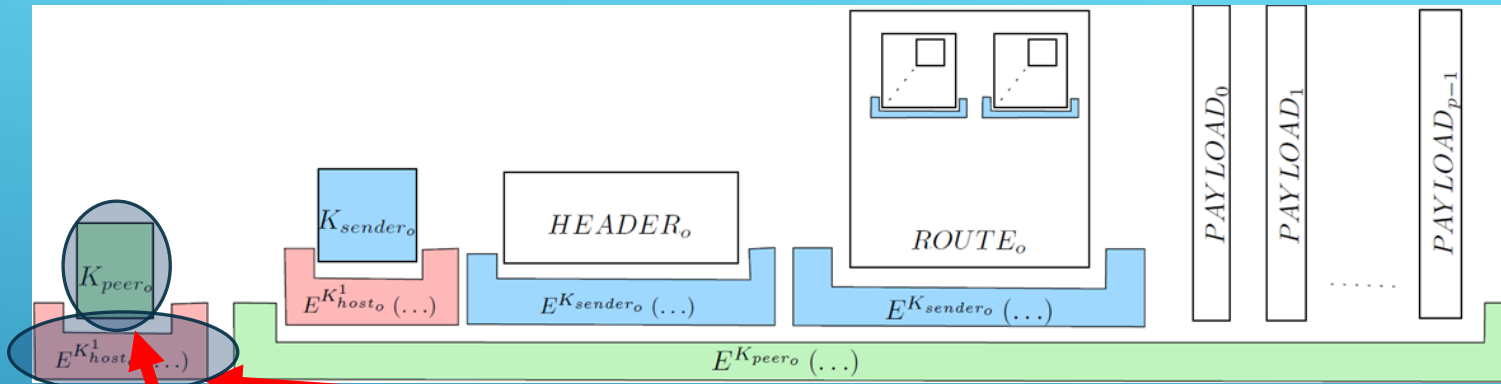
HOW DOES A MESSAGE LOOK LIKE (ON THE TRANSPORT LAYER)?





MESSAGEVORTEX

HOW DOES A MESSAGE LOOK LIKE?



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

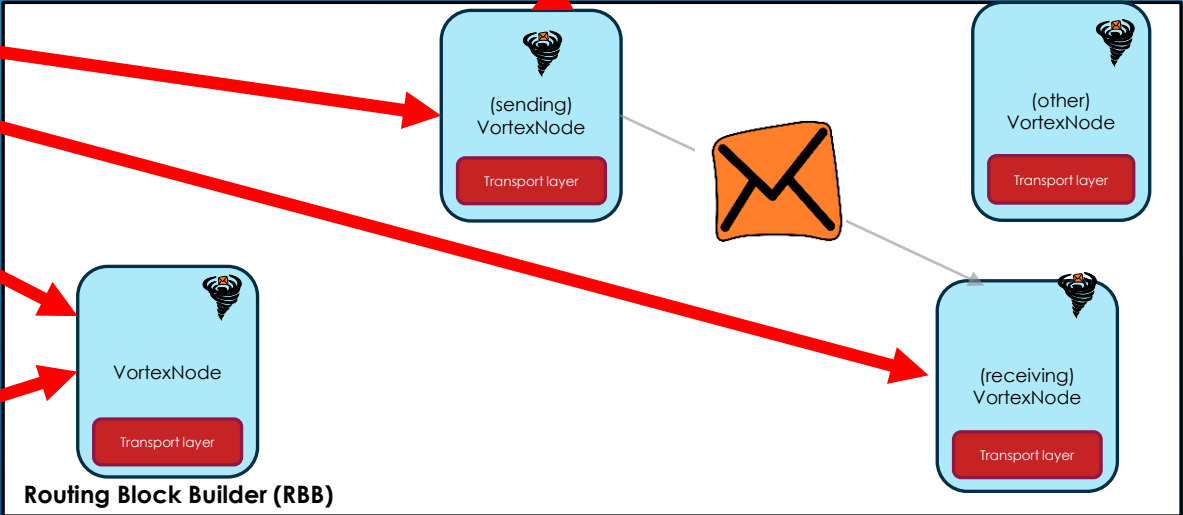
The peer key was also included in the message build instructions (in the routing block) of the sending node

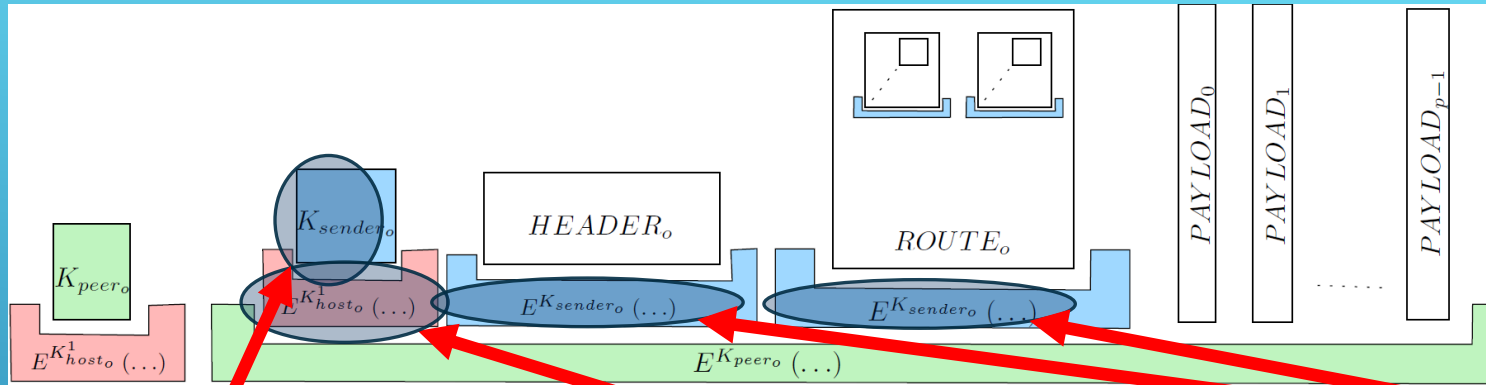
The peer key is only known to the two peering partners and the RBB

MESSAGEVORTEX

THE PEER KEY

The RBB is the creator of the routing block





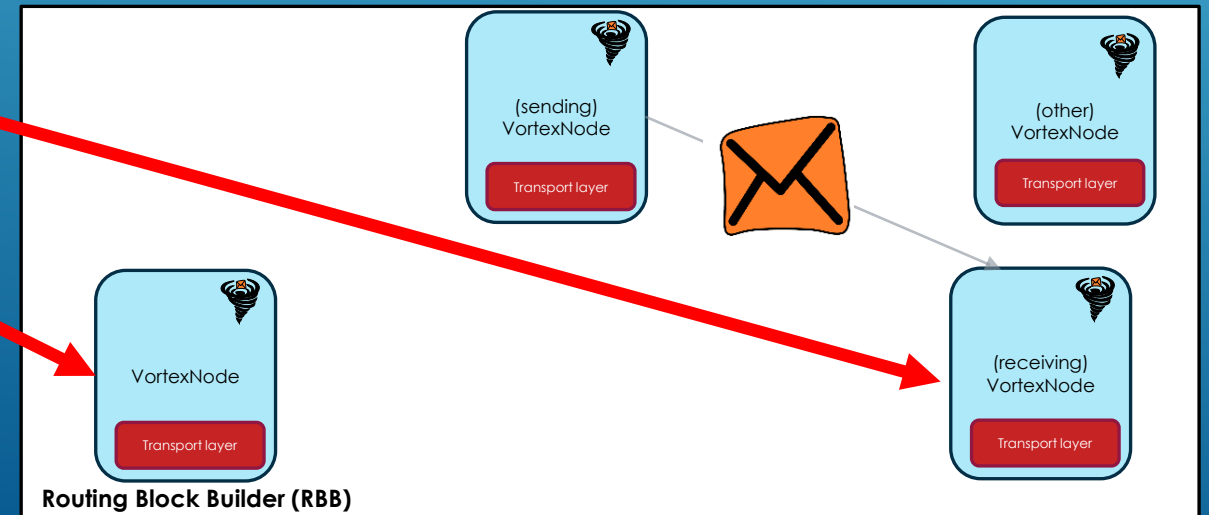
The sender key is only known to the receiving node and the RBB.

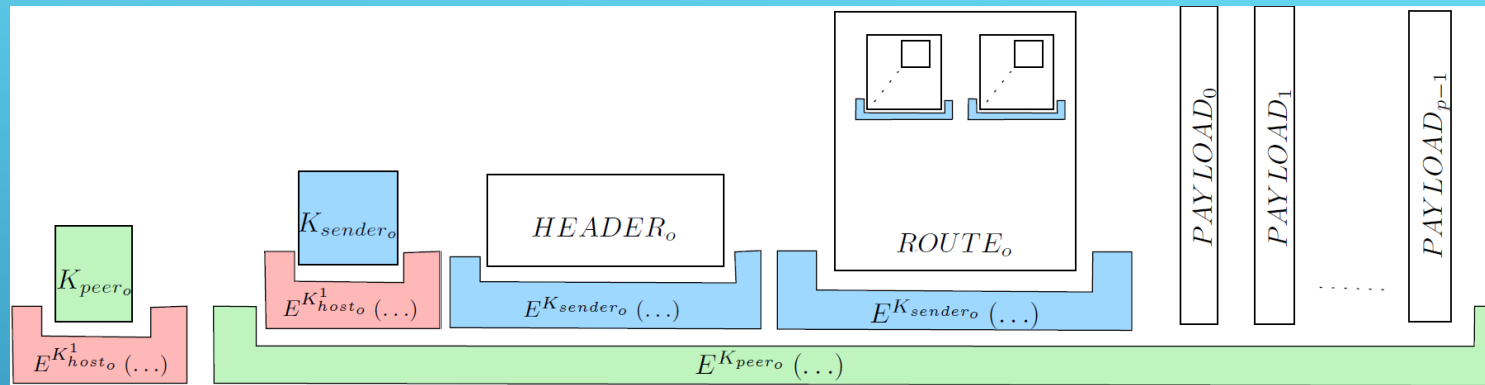
The sender key is encrypted with the receiving nodes' public key.

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

MESSAGEVORTEX

THE SENDER KEY



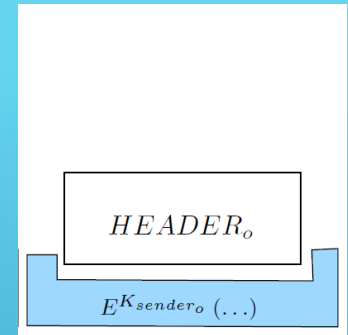


- Contains:
 - An „ephemeral ID“ (eID; public key; synonymous to the sender)
 - A serial (allows replay protection)
 - Validity interval
- May contain:
 - Requests (and the reply block)
 - Proof of work (if requesting something)
- Important: All eIDs sign their header blocks with their private key.

MESSAGEVORTEX

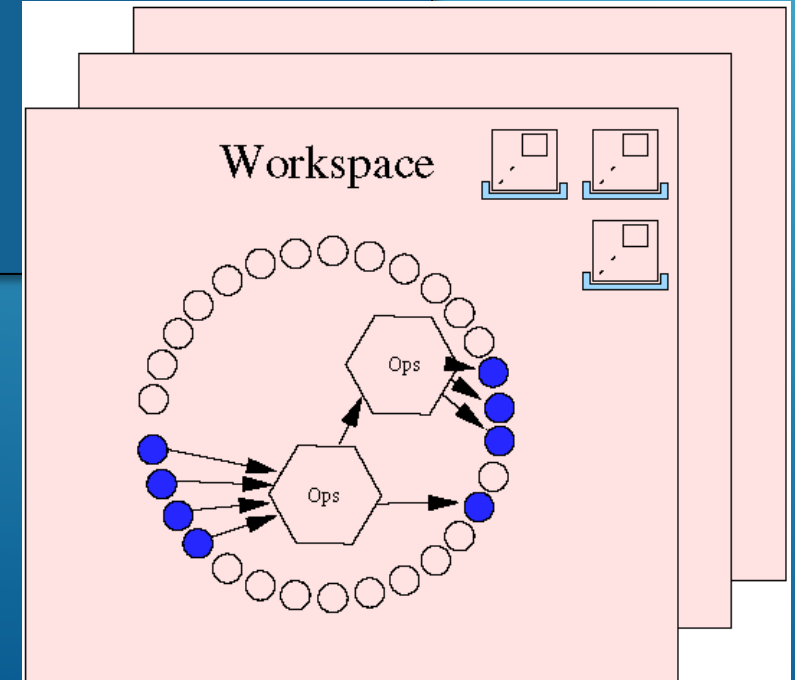
THE HEADER

- 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
 - Payload blocks (referenced by IDs)
 - Routing blocks/combos
 - Operations



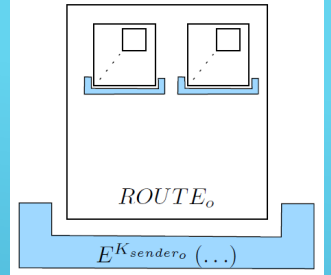
MESSAGEVORTEX

WHAT IS AN eID? WHAT IS A WORKSPACE?



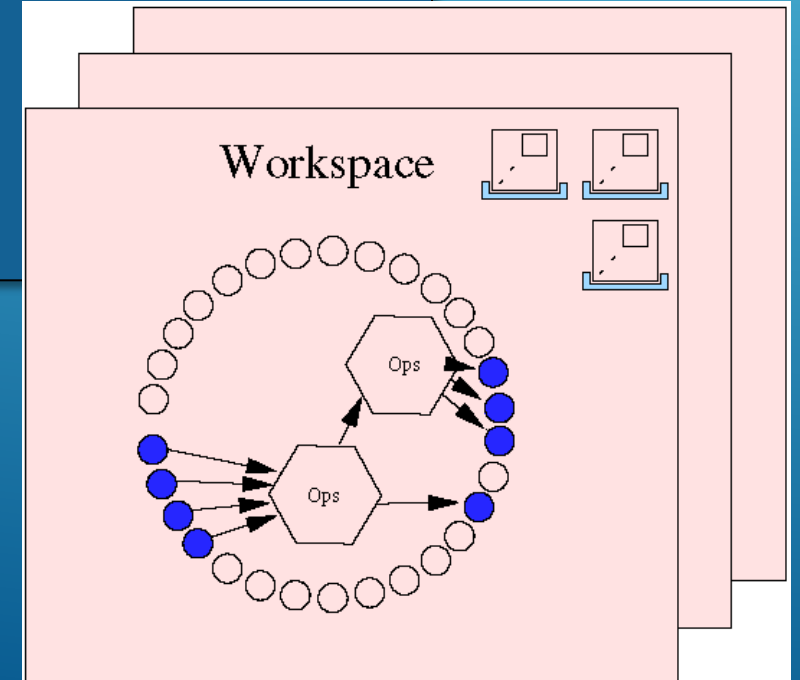
The routing block contains

- Handling instructions for the current node
Operations required for local handling which includes mapping of IDs from the message into the workspace.
- Onionized routing information required for building subsequent messages (Routing combos). They contain:
 - The pre-encrypted peer key for the next peer
 - The pre-encrypted header key for the next peer
 - The pre-encrypted header
 - The pre-Encrypted routing block
 - Mappings for workspace IDs into the message
 - The blending information for the generated message
 - Transport address
 - Blending method (e.g., F5 with key "test123")

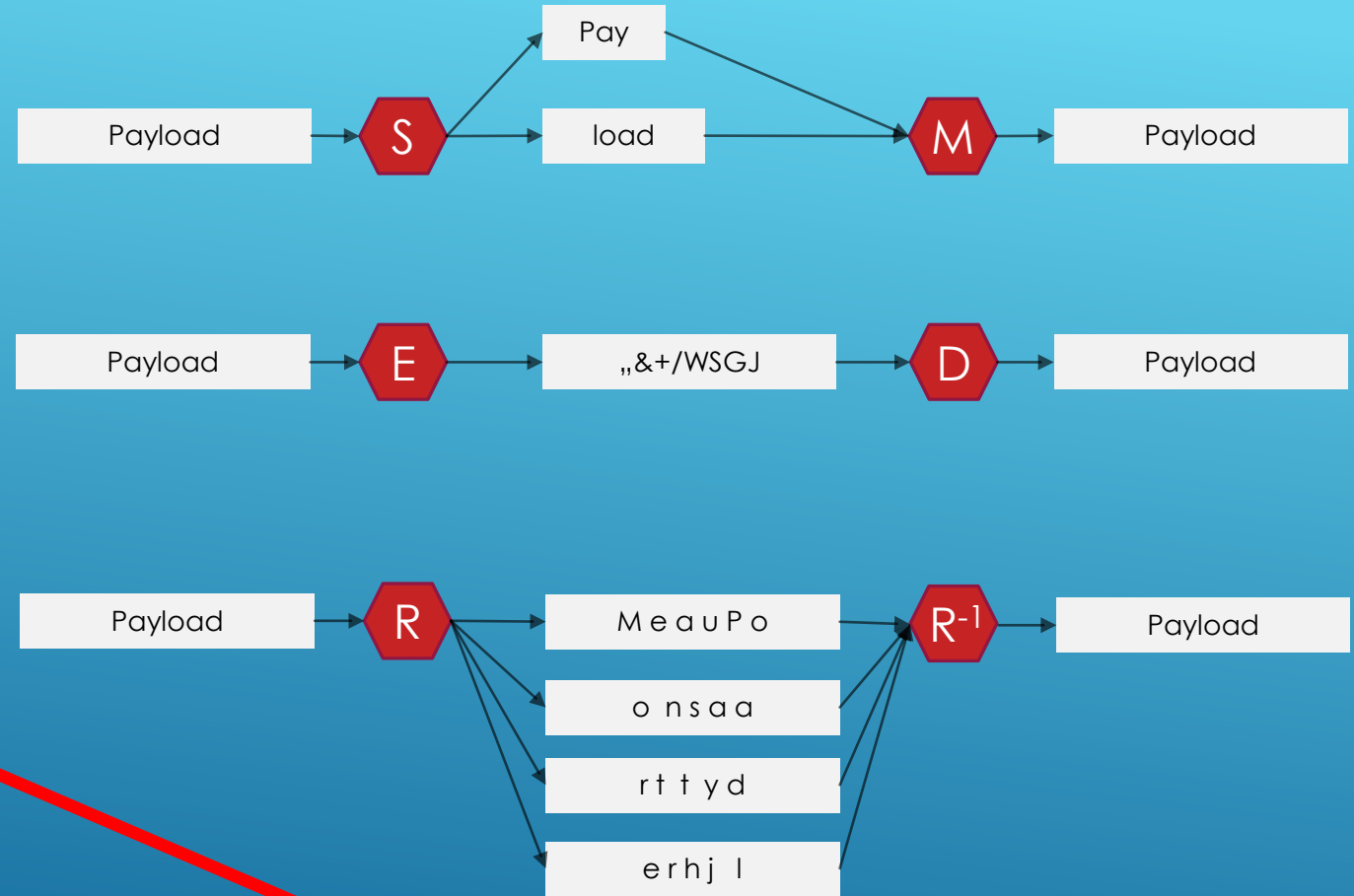


MESSAGEVORTEX

THE ROUTING BLOCK



- ▶ 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

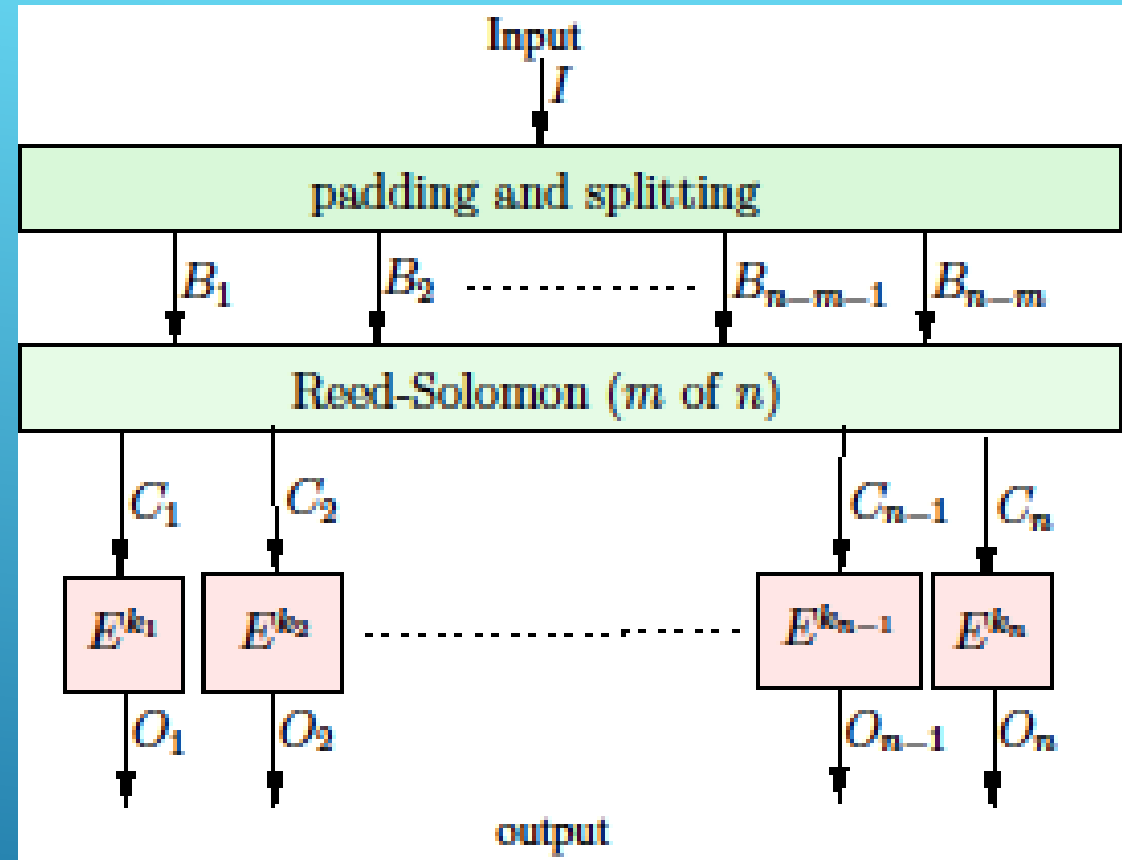


MESSAGEVORTEX

WHAT ARE THE AVAILABLE OPERATIONS?

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 $\rightarrow \langle B_1, B_2, \dots, B_{n-m} \rangle$
 - ▶ Apply Reed-Solomon to the data vectors (allowing up to m out of n vectors to fail) $\rightarrow \langle C_1, C_2, \dots, C_n \rangle$
 - ▶ Encrypt all resulting vectors $\rightarrow \langle O_1, O_2, \dots, O_n \rangle$
- ▶ This operation creates n blocks. $n-m$ blocks are required to reverse the operation.



MESSAGEVORTEX

THE addRedundancy OPERATION (1)

- ▶ 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 number of bytes specified by the RBB as a fixed stuffing $\rightarrow C2$
 - ▶ Plus $C1 \cdot (\text{len}(X) - 4)$
This will disappear when decoding p
 - ▶ Decoding:
 - ▶ Take first four bytes (p)
 - ▶ Calculate $i = \text{len}(M) = p \pmod{\text{len}(X) - 4}$
- ▶ The rest is filled with a seeded PRNG \rightarrow A known seed allows to verify padding.

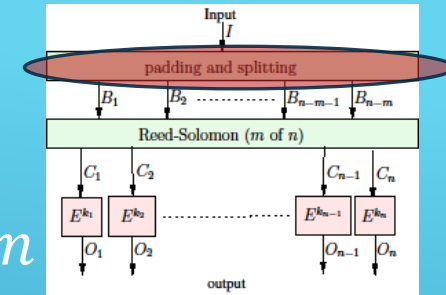
$$i = \text{len}(M)$$

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

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

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

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



The padded message

The padding value
(length; 4 Bytes uint)

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

The message

MESSAGEVORTEX

THE addRedundancy OPERATION (2)

$$i = \text{len}(\mathbf{M})$$

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

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

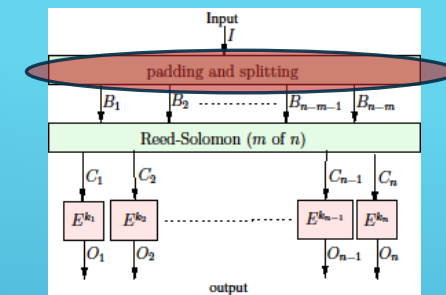
$$p = i + \left(C1 \cdot l \left(\text{mod} \left\lfloor \frac{2^{32} - 1 - i}{l} \right\rfloor \cdot l \right) \right)$$

$$\begin{aligned} X &= \langle p, \mathbf{M}, R(s, l - i) \rangle \\ &= \langle p, \mathbf{M}, R(s, l - (p \bmod (\text{len}(\mathbf{X}) - 4)) \rangle \end{aligned}$$

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
 $\text{len}(\langle \mathbf{M}, R(s, l - i) \rangle)$ without the padding itself



MESSAGEVORTEX

THE addRedundancy OPERATION (3)

$$i = \text{len}(\mathbf{M})$$

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

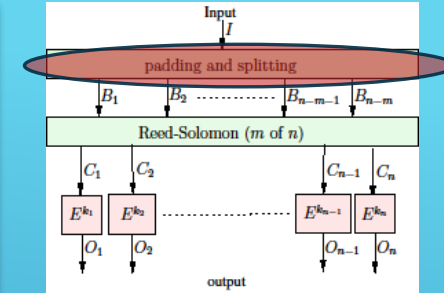
$$l = \left\lceil \frac{i + C2}{e} \right\rceil \cdot e$$

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

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

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

C2 is a fixed value of bytes to be added to the message. By adding C2, we make it possible, that value $p \leq \text{len}(\mathbf{X}) - 4$ is valid.



C1 is a generator for multiples of l and guarantees that any $p > \text{len}(\mathbf{X}) - 4$ is valid as well.

We do NOT require the values for successful decoding.

MESSAGEVORTEX

THE addRedundancy OPERATION (3)

- Up until now we have padded the message with the following properties:
 - The message has a perfect length to allow:
 - Apply the Reed-Solomon operation without further padding
 - Apply the encryption without further padding
 - Any value of p may reflect a valid padding

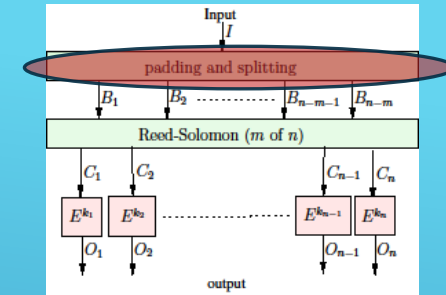
Interlude:

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 striped manner in RAID6 systems to allow multiple disks to fail without losing data.

→ Let's apply Reed-Solomon

MESSAGEVORTEX

THE addRedundancy OPERATION (4)



► $t = n - 1$

► $A = \text{vec2mat}\left(X, \frac{\text{len}(X)}{n-m}\right)$

We align the padded data in a matrix

► $V = \begin{pmatrix} 0^0 & \dots & 0^{n-m-1} \\ \vdots & \ddots & \vdots \\ t^0 & \dots & t^{n-m-1} \end{pmatrix}$

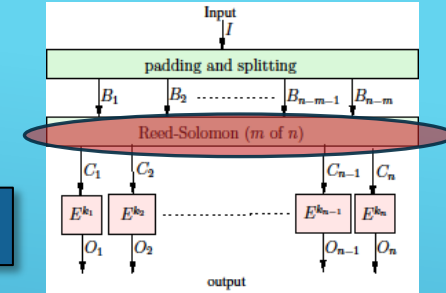
We prepare a suitable Vandermonde matrix (m rows and n columns)

► $P = V \cdot A \left(GF(2^\omega) \right)$

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

► $\langle C_1, C_2, \dots, C_n \rangle = \text{row2vec}(P)$

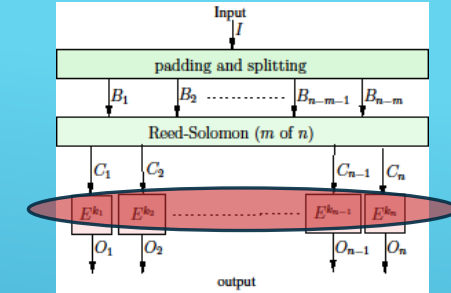
We extract the rows as vector



MESSAGEVORTEX

THE addRedundancy OPERATION (5)

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

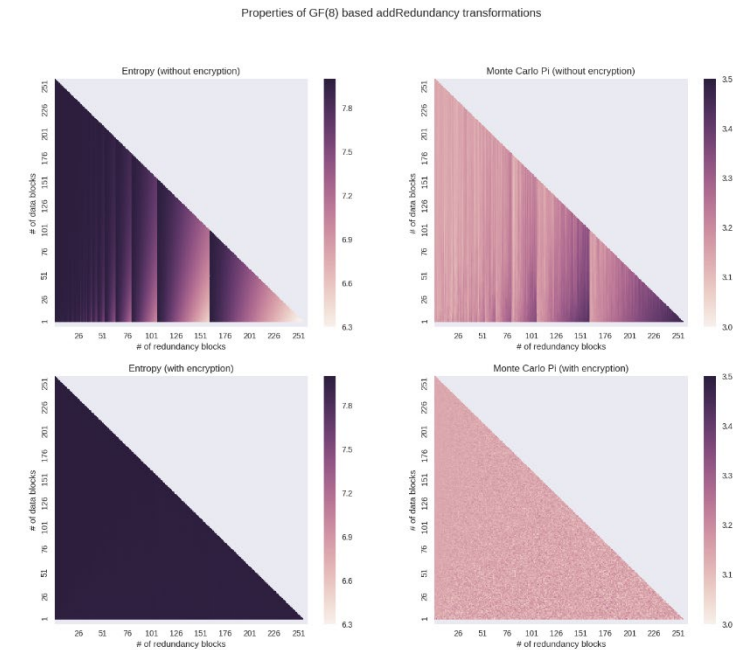


Entropy and Monte Carlo PI WITHOUT encryption with varying parameters on the same.

Entropy and Monte Carlo PI WITH encryption with varying parameters on the same block.

MESSAGEVORTEX

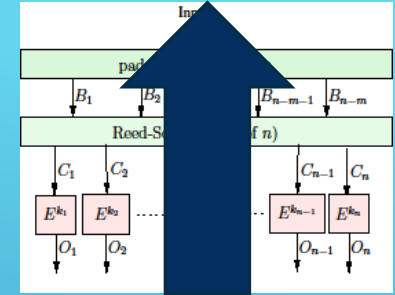
THE addRedundancy OPERATION (6)





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

- ▶ The reverse operation is applied by inverting 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 \mathbf{A} .
 - ▶ Eliminate in the Vandermonde matrix the discarded payload lines $\rightarrow \mathbf{V}'$
 - ▶ invert $\mathbf{V}' \rightarrow \mathbf{V}'^{-1}$
 - ▶ Calculate $\mathbf{D} = \mathbf{V}'^{-1}\mathbf{A}$ ($\mathbf{GF}(2^\omega)$) resulting in $\langle B_1, B_2, \dots, B_{n-m} \rangle = \text{row2vec}(\mathbf{D})$ and arrange \mathbf{M} as a byte stream.
 - ▶ Get the first four bytes p and cut the payload to its original size by calculating $i = p(\bmod(\text{len}(X) - 4))$.
- ▶ Verify the padding (if possible; requires s):
 - ▶ Verify that the padded space matches $R(s, \text{len}(X) - p - 4)$



MESSAGEVORTEX

THE addRedundancy OPERATION (7)

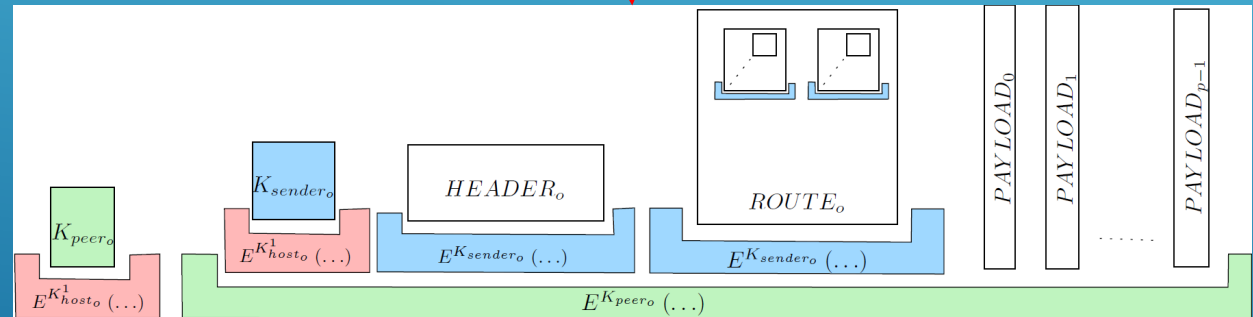
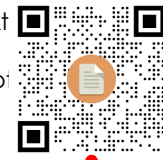
► Extract message by the blending layer

- 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

Dear sir

This is an obviously machine generated text with some images. May be a password recovery request, a routing graph, or any other kind of machine generated information.

Yours
MessageVortex

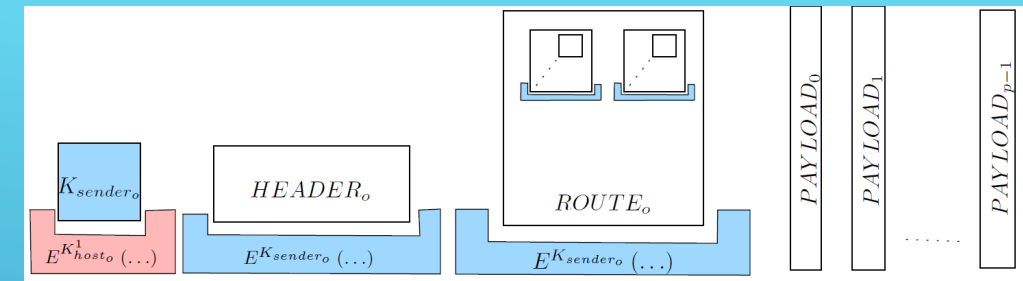


MESSAGEVORTEX

Processing an incoming message

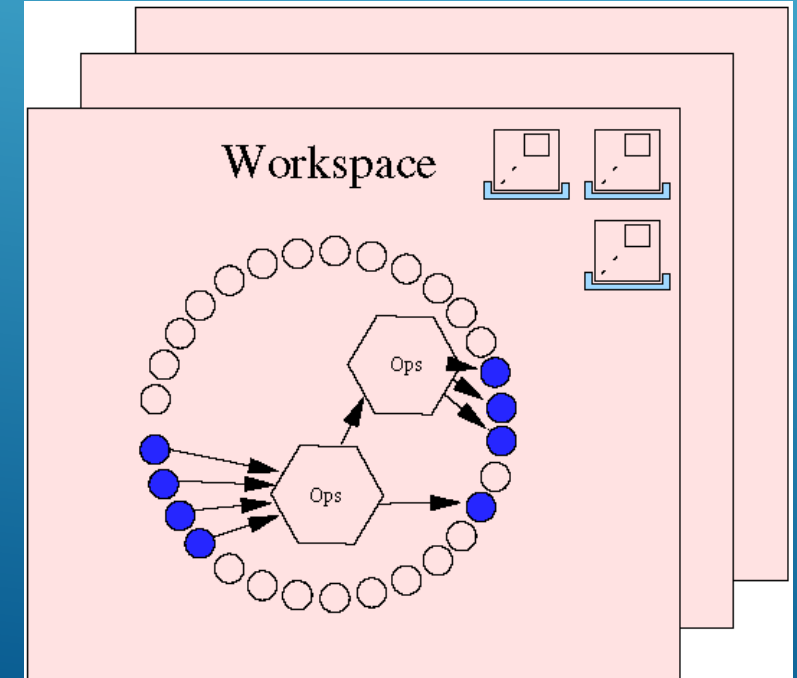


- ▶ Extract message by the blending layer
 - ▶ Extract message prefix block and decrypt it.
 - ▶ If a valid structure with a peer key is provided continue with the inner message by decrypting with the key provided in the prefix block.
- ▶ Get sender key and decrypt header, and routing block
- ▶ Verify 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
- ▶ If a message is written to payload ID 0 then this message is for the local user



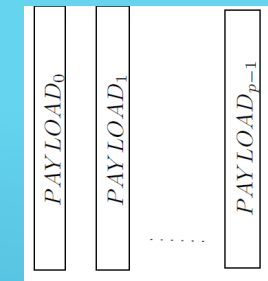
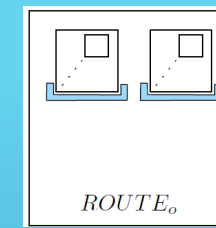
MESSAGEVORTEX

Processing an incoming message



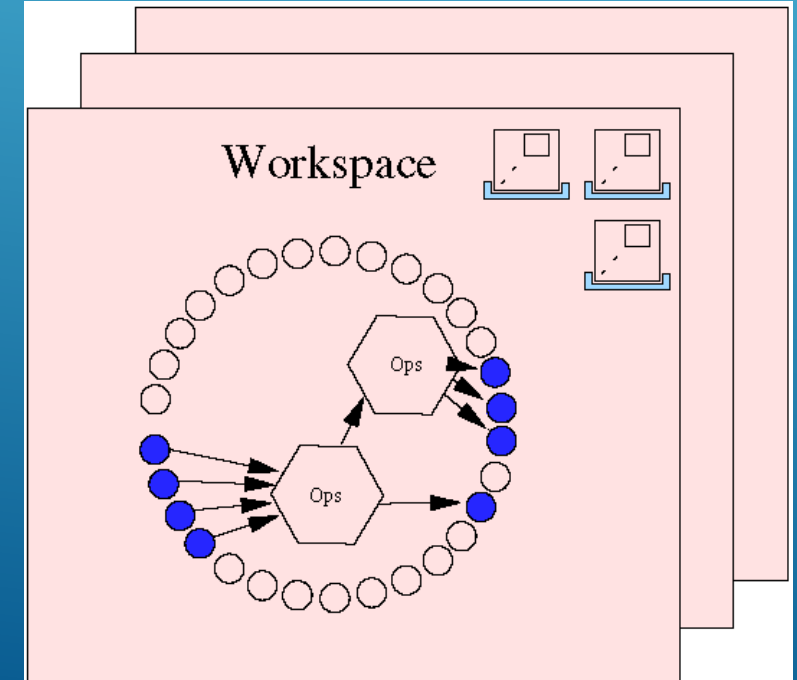


- ▶ Extract message by the blending layer
 - ▶ Extract message prefix block and decrypt it.
 - ▶ If a valid structure with a peer key is provided continue with the inner message by decrypting with the key provided in the prefix block.
- ▶ **Get sender key and decrypt header, and routing block**
- ▶ Verify 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

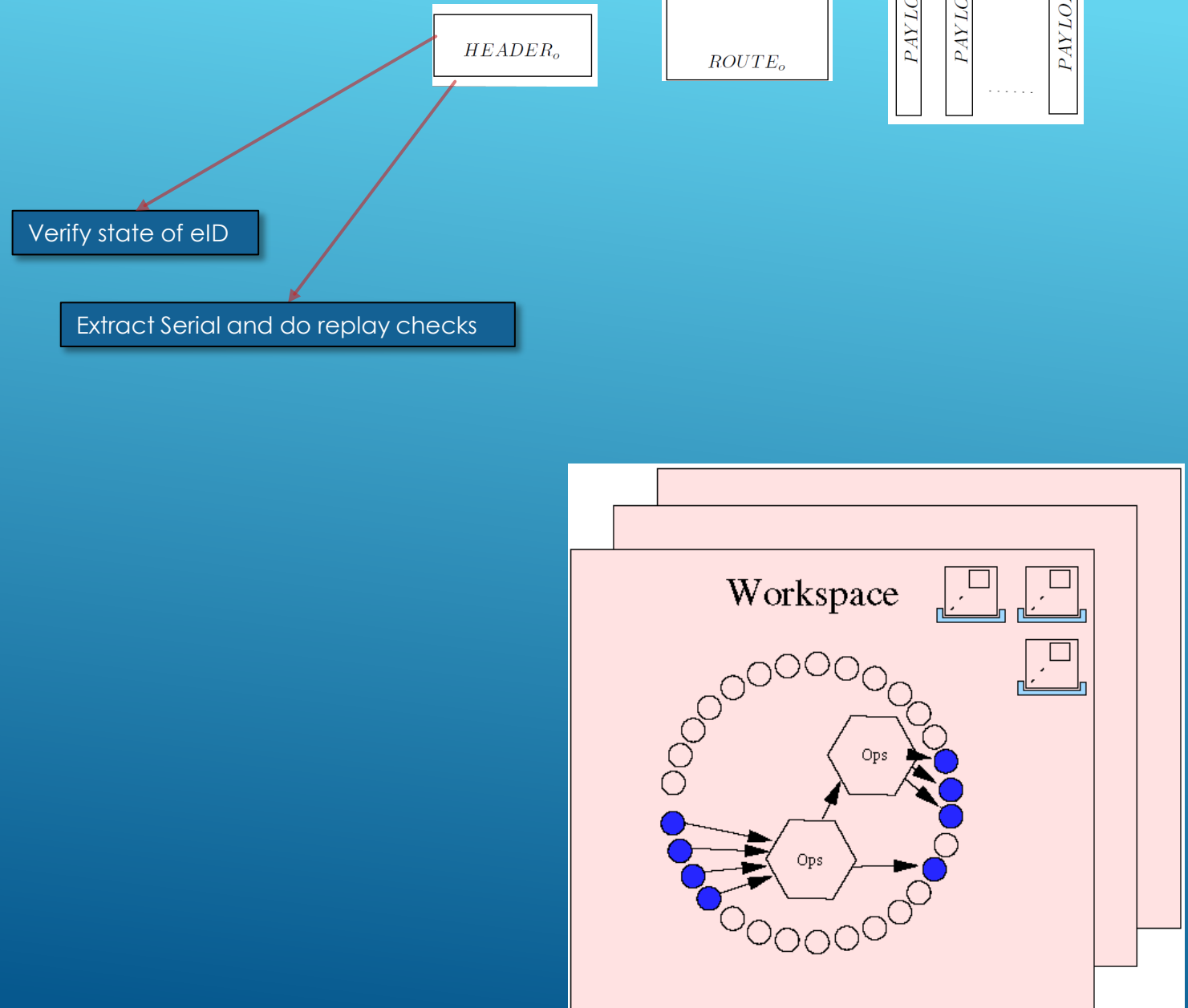




- ▶ Extract message by the blending layer
 - ▶ Extract message prefix block and decrypt it.
 - ▶ If a valid structure with a peer key is provided continue with the inner message by decrypting with the key provided in the prefix block.
- ▶ Get sender key and decrypt header, and routing block
- ▶ **Verify 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

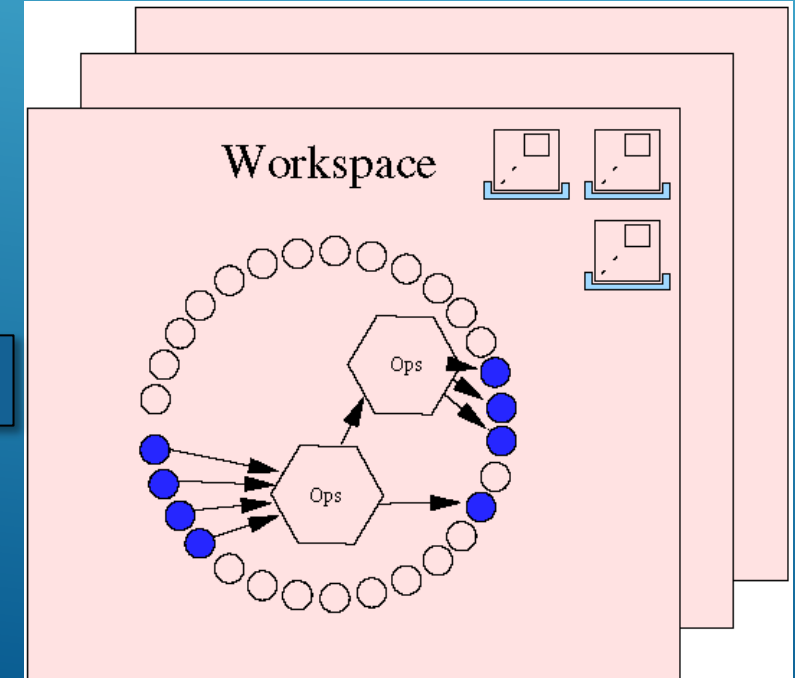
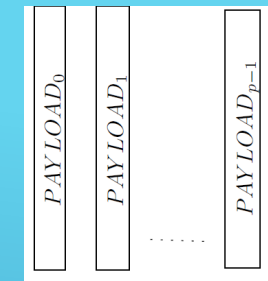
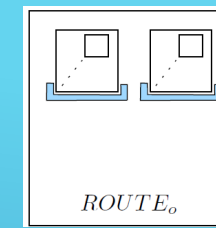
MESSAGEVORTEX

Processing an incoming message

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

PoW or anonymous payment may apply

$HEADER_o$

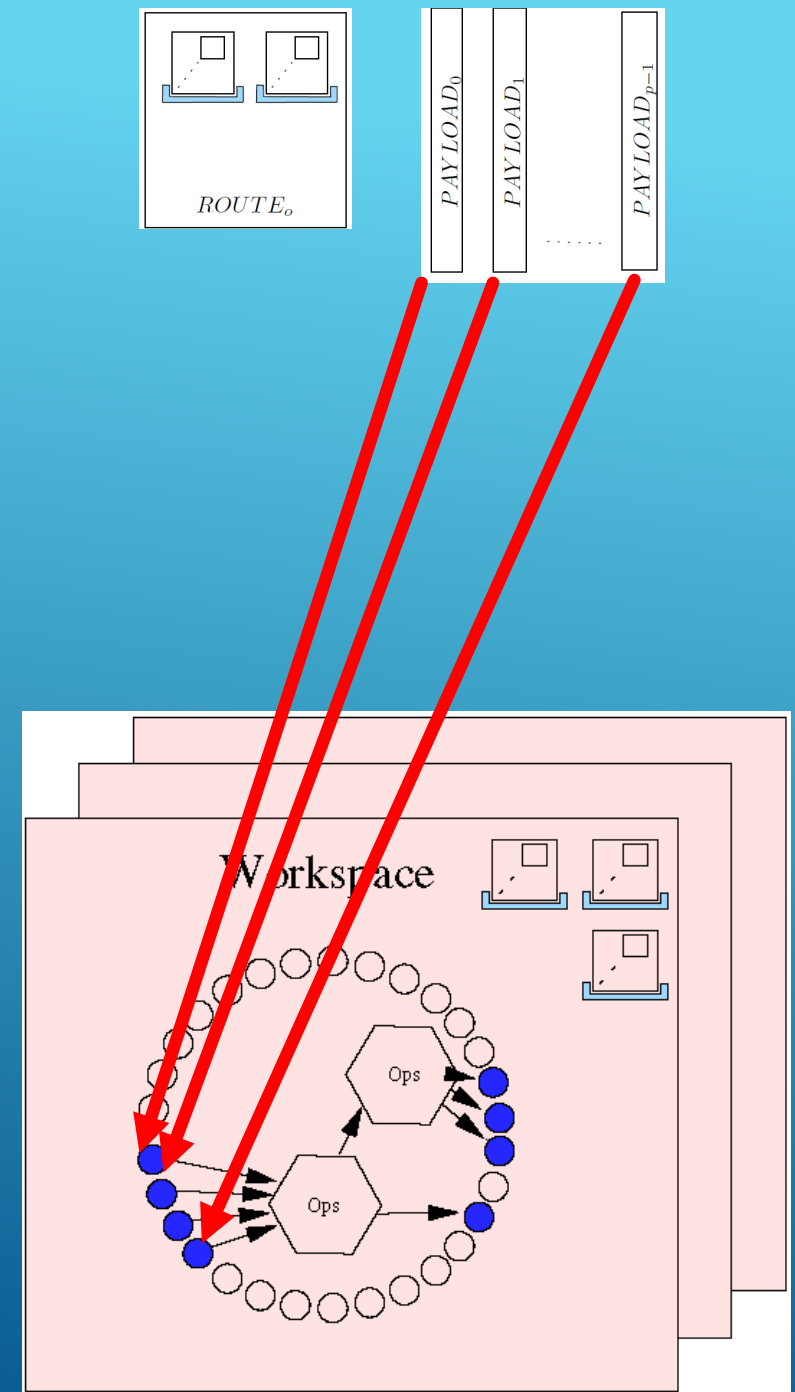




- ▶ Extract message by the blending layer
 - ▶ Extract message prefix block and decrypt it.
 - ▶ If a valid structure with a peer key is provided continue with the inner message by decrypting with the key provided in the prefix block.
- ▶ Get sender key and decrypt header, and routing block
- ▶ Verify 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

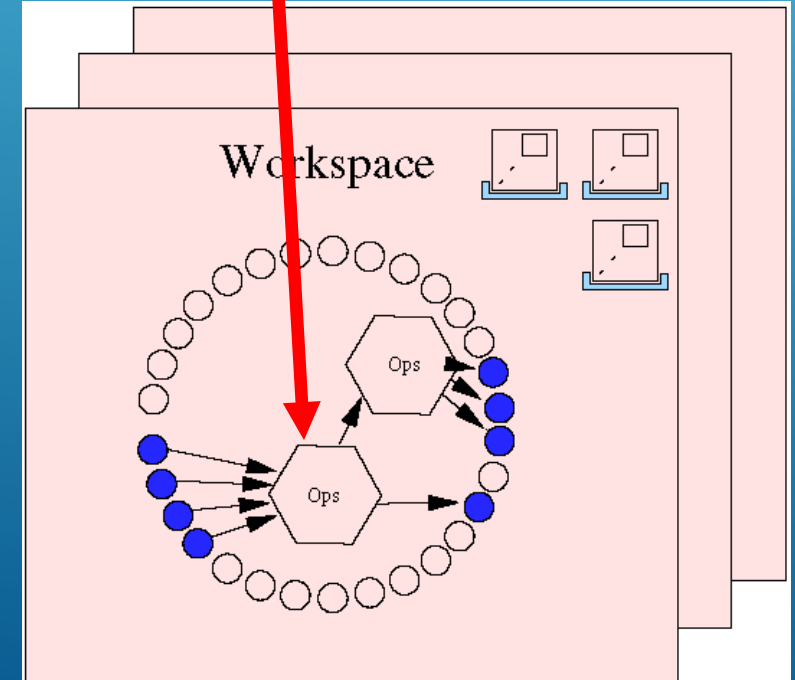
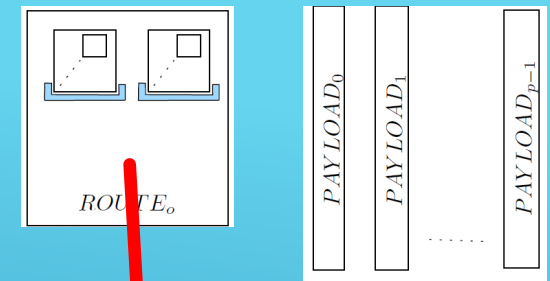
Processing an incoming message



- ▶ 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

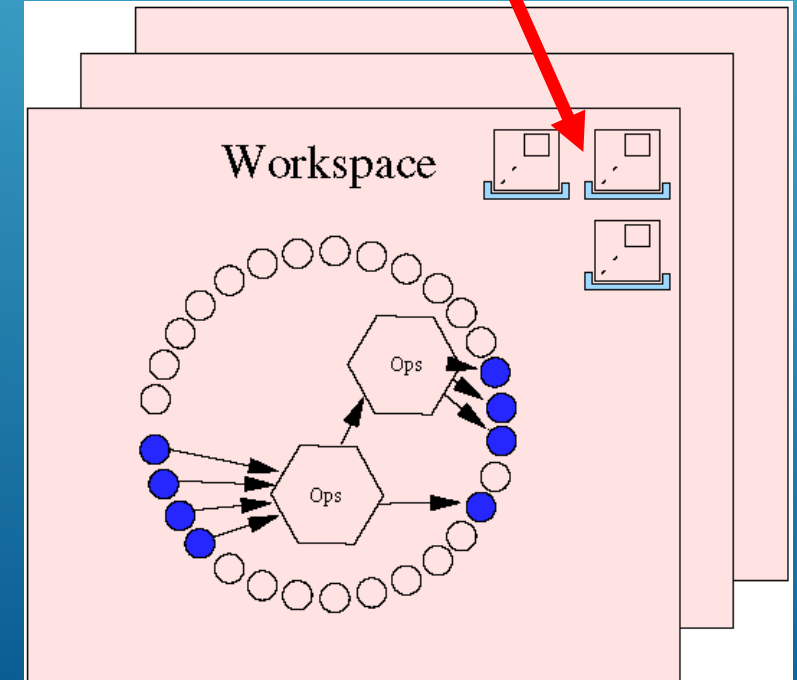
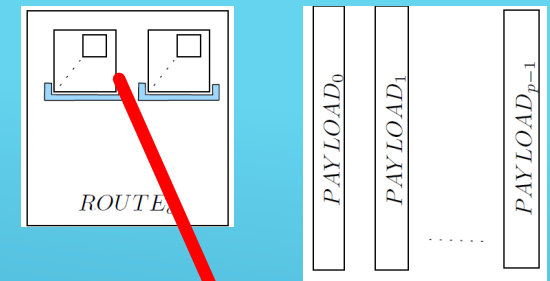




- ▶ Extract message by the blending layer
 - ▶ Extract message prefix block and decrypt it.
 - ▶ If a valid structure with a peer key is provided continue with the inner message by decrypting with the key provided in the prefix block.
- ▶ Get sender key and decrypt header, and routing block
- ▶ Verify 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





- ▶ 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)

Avoid repeating patterns!

MESSAGEVORTEX

ROUTING STRATEGIES (GENERAL)



- ▶ Choose VortexNodes and allocate eIDs if required
- ▶ Choose a directed suitable graph (Nodes=VortexNodes; Edges=Messages).
- ▶ Select timing information for the blocks.
- ▶ Select path(s) for message transferal to final recipient.
- ▶ Select operations
 - ▶ For the message
 - ▶ For the decoy traffic
- ▶ Assemble routing block (it is always one at the beginning).
- ▶ Apply routing block to message and hand it over to the router (local)
- ▶ **The algorithm does not matter! Just follow the rule:**
 - ▶ **All operations are valid for a data stream**
 - ▶ **No pattern is repeated on any node except for the sender and the recipient**

MESSAGEVORTEX

A SIMPLE STRATEGY FOR BUILDING A ROUTING BLOCK



- ▶ For brevity, the following algorithm has no split/merge operations and all nodes receive at least the entire message. This has some drawbacks:
 - ▶ Huge message overhead
 - ▶ Minimum size of message may be determined and traced
 - ▶ Route diagnosis not covered
- ▶ But...
 - ▶ It credibly anonymizes sender and recipient.

MESSAGEVORTEX

A SIMPLE STRATEGY FOR BUILDING A ROUTING BLOCK (A WORD OF CAUTION)



```
1: function GETGRAPH(startNode,endNode, numNodes, minEdges, listOfAllNodes)
2:   nodes  $\leftarrow$  GETNODES(numNodes, ListOfAllNodes, startNode, Endnode)
3:   edges  $\leftarrow$  GETEDGES(minEdges, nodes)
4:   return [nodes, edges]
5: end function

6: function GETNODES(numberOfNodes, ListOfKnownNodes, startNode, Endnode)
7:   nodeList  $\leftarrow$  [startNode, endNode]
8:   while len(nodeList) < numberOfNodes do
9:     randomNode  $\leftarrow$  PICKRANDOMNODEFROMSET(listOfKnownNodes)
10:    if  $\neg$ nodeList.contains(randomNode) then
11:      nodeList.append(randomNode)
12:    end if
13:  end while
14:  return nodeList
15: end function
```

MESSAGEVORTEX

A SIMPLE STRATEGY FOR BUILDING A ROUTING BLOCK (THE MAIN PROCEDURE)

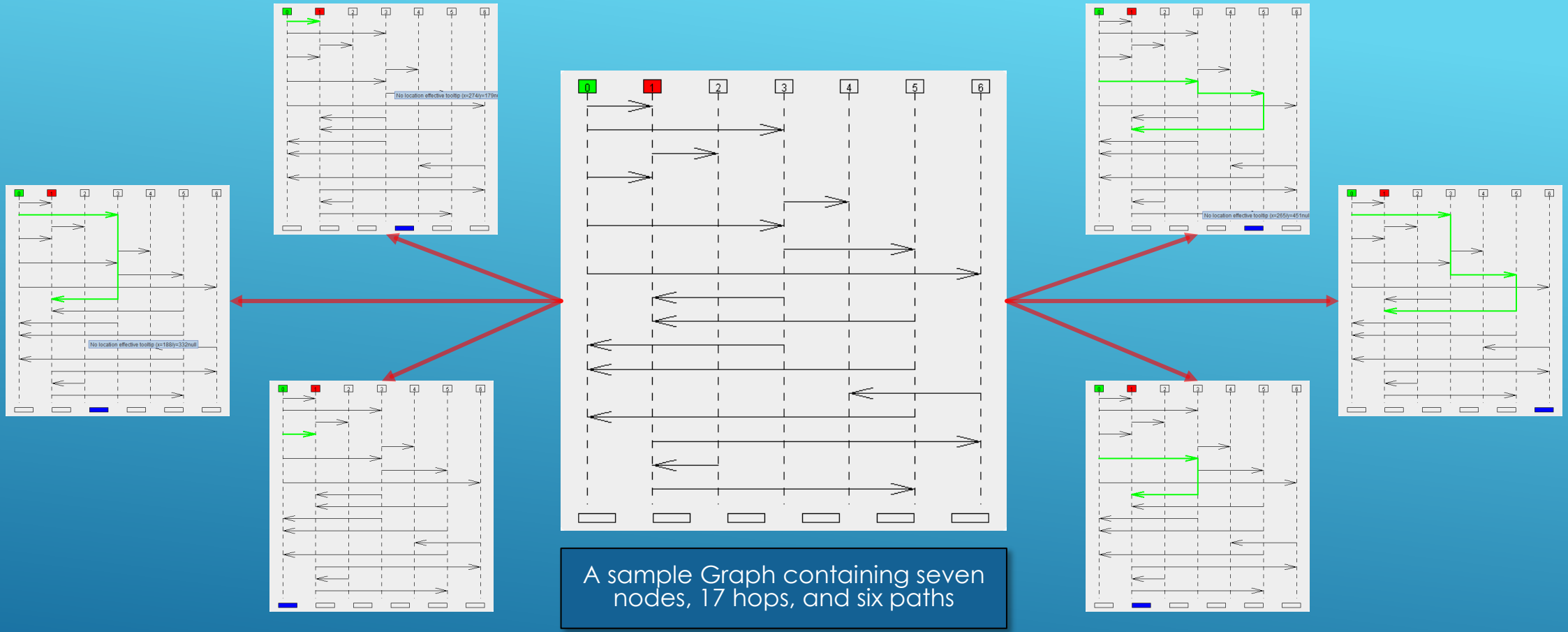


```
16: function NODESREACHED(edgeList,startNode)
17:   reachedNodeList  $\leftarrow$  [startNode]
18:   for all  $e \in$  edgeList do
19:     if  $\neg$ reachedNodeList.contains( $e[0]$ ) then
20:       reachedNodeList.append( $e[0]$ )
21:     end if
22:   end for
23: end function

24: function GETEDGES(minEdges, listOfAllNodes)
25:   while len(edgeList)<minEdges or
      nodesReached(edgeList,listOfAllNodes[0])<len(listOfAllNodes) do
26:     listOfReachedNodes  $\leftarrow$  GETREACHEDNODES(listOfAllNodes,edgeList)
27:     startNode  $\leftarrow$  RANDOMNODE(listOfReachedNodes)
28:     endNode  $\leftarrow$  RANDOMNODE(listOfAllNodes)
29:     edgeList.append([startNode, endNode])
30:   end while
31:   return edgeList
32: end function
```

MESSAGEVORTEX

A SIMPLE STRATEGY FOR BUILDING A ROUTING BLOCK (THE MAIN PROCEDURE 2)



MESSAGEVORTEX

A SIMPLE STRATEGY FOR BUILDING A ROUTING BLOCK (THE MAIN PROCEDURE 2)



```
1: function GETTIMING(edges, maxTime,minHopTime)
2:   if len(edges) × (minHopTime - 1) > maxTime then
3:     throw "maxTime too small for constraints"
4:   end if
5:   earliestTime ← 0
6:   maxRemainingTime ← maxTime – earliestTime
7:   remainingHops ← len(edges) – 1
8:   times ← []
9:   for all e ∈ edges do
10:    maxShare ← remainingTime – remainingHops × minHopTime
11:    share ←  $\frac{\text{maxShare}}{\text{remainingHops}}$ 
12:    minTime ← GETRANDOMTIME(earliestTime, earliestTime + share, earliestTime + maxShare)
13:    maxTime ← GETRANDOMTIME(minTime, minTime + share, earliestTime + maxShare)
14:    earliestTime ← maxTime + minHopTime
15:    remainingHops ← remainingHops – 1
16:    maxRemainingTime ← maxTime – earliestTime
17:    times.append(minTime, maxTime)
18:  end for
19:  return times
20: end function
```

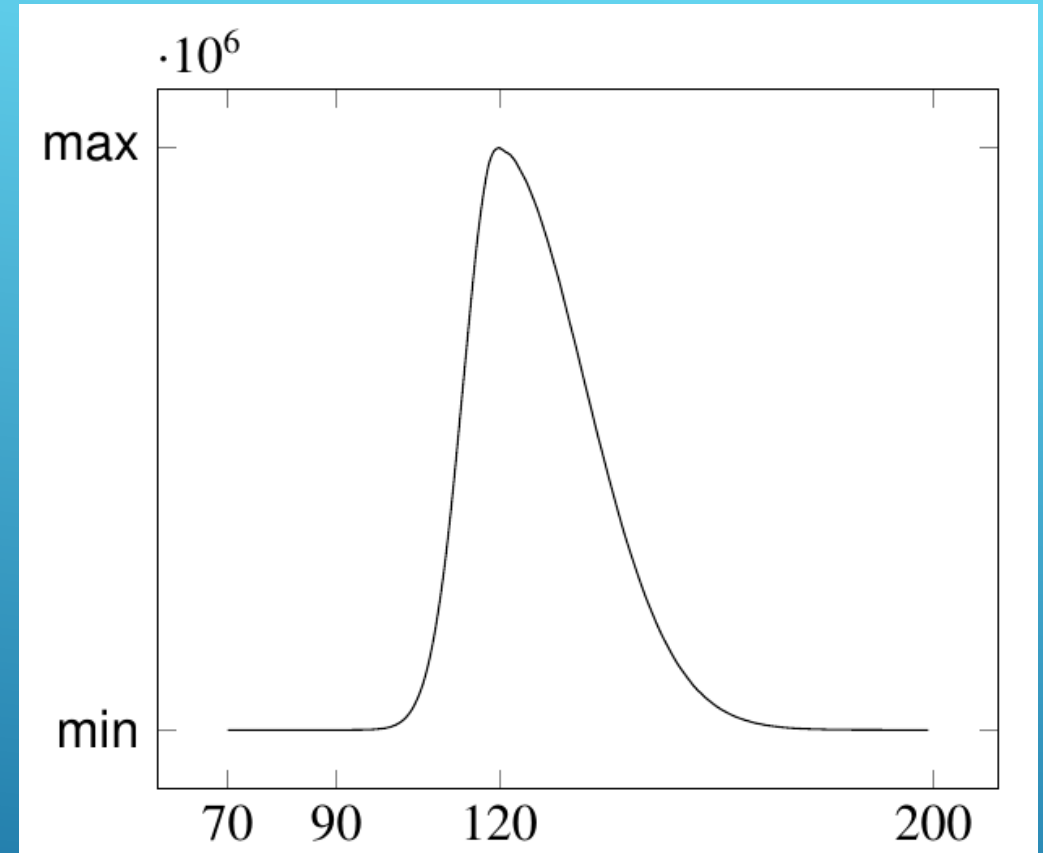
MESSAGEVORTEX

A SIMPLE STRATEGY FOR BUILDING A ROUTING BLOCK (ASSIGN TIMING)

```

21: function GETRANDOMTIME(min, peak, max)
22:   value  $\leftarrow$  -1
23:   while value < min or value > max do
24:     value  $\leftarrow$  NEXTRANDOMGAUSSIAN( )
25:     d  $\leftarrow$  NEXTDOUBLE( )
26:     if  $d < (peak - min) / (max - min)$  then
27:       value  $\leftarrow$  peak -  $\frac{abs(value) \times (peak - min)}{5}$ 
28:     else
29:       value  $\leftarrow$  peak +  $\frac{abs(value) \times (max - peak)}{5}$ 
30:     end if
31:   end while
32:   return value
33: end function

```



MESSAGEVORTEX


A SIMPLE STRATEGY FOR BUILDING A ROUTING BLOCK (ASSIGN TIMING)



```
1: function GETROUTING(edges, redundantRoutes, messaageld)
2:   if redundantRoutes < 1 then
3:     throw "At least one route is required"
4:   end if
5:   routes ← getRoutes(edges)
6:   if len(routes) < redundantRoutes then
7:     throw "Graph has not enough redundant routes"
8:   end if
9:   numRoute ← 0
10:  while redundantRoutes < numRoute do
11:    currentRoute ← routes[numRoute]
12:    ASSIGNROUTE(currentRoute, payloadId, currentRoute[LAST], 0)
13:    numRoute ← numRoute + 1
14:  end while
15:  ▷ Add sensible operations to decoy routes
16:  for all  $r \in \text{getUnsuedRoutes}(\text{edges})$  do
17:    ASSIGNROUTE(r, r.getRandopOperation().getUnusedIds(1), r[LAST], r[LAST].getFreeId(1))
18:  end for
19:  ADDMESSAGE Mapping(edges)
20: end function
```

MESSAGEVORTEX


A SIMPLE STRATEGY FOR BUILDING A ROUTING BLOCK (ASSIGN OPERATIONS)



```
21: function ASSIGNROUTE(route, payloadIds, lastNode, targetIds)
22:   source ← route.getSourceNode()
23:   if payloadIds.isEmpty() then
24:     PayloadIds ← source.getRandopOperation().getUnusedIds(1)
25:     payloadSet ← ASSIGNROUTE(route[2-], targetIds.forward(), lastNode, targetIds.reverse())
26:   else
27:     targetIds ← ASSIGNOPERATION(route.getSourceNode(), payloadIds, lastNode, targetIds)
28:     payloadSet ← ASSIGNROUTE(route[2-], targetIds.forward(), lastNode, targetIds.reverse())
29:   end if
30: end function
```

MESSAGEVORTEX

A SIMPLE STRATEGY FOR BUILDING A ROUTING BLOCK (ASSIGN OPERATIONS)



```
31: procedure ASSIGNOPERATION(node, transportIds, reverseNode, targetIds)
32:   out ← node.outEdges()
33:   in ← node.inEdges()
34:   if out > 1 or extRandomInt(3) = 1 then
35:     ▶ assign addRedundancy
36:     numBlocks ← max(out+1, nextRandomInt(out+4))
37:     seed ← nextRandomInt( $2^{256} - 1$ )
38:     op ← node.addRedundancy(transportIds, numBlocks - 1, numBlocks, seed)
39:     reverseOp ← reverseNode.removeRedundancy(targetIds, op)
40:     newId ← op.getUnusedIds(1)
41:     newId.addReverseIds(reverseOp)
42:   else
43:     ▶ assign encrypt
44:     keySize ← (nextRandomInt(3) + 2) * 64
45:     key ← nextRandomInt( $2^{\text{keySize}}$ )
46:     op ← node.encrypt(transportIds, "AES", keySize, key)
47:     reverseOp ← reverseNode.decrypt(targetIds, op)
48:     newId ← op.getUnusedIds(1)
49:     newId.addReverseIds(reverseOp)
50:   end if
51:   return newIds
52: end procedure
```

MESSAGEVORTEX

A SIMPLE STRATEGY FOR BUILDING A ROUTING BLOCK (ASSIGN OPERATIONS)



- ▶ In a censoring environment
 - ▶ You need to collect them yourself (manually)
 - ▶ You may route messages outside of the censors reach. Once outside traditional mixing nodes may be used
- ▶ 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?



- ▶ The first holistic approach targeting the needs of a partially/fully censoring environment
- ▶ Bases on proven technology
 - ▶ Reed-Solomon operations are proven and well known
 - ▶ Mixing (and its limitations) are well known
- ▶ Is Cryptoagile
- ▶ Allows diagnosis
- ▶ Allows localized 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 routed data know.
 - ▶ A working system with a reference implementation

MESSAGEVORTEX

A WHAT ARE ITS BENEFITS?

- 
- 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
 - Identifiable messages
 - Replayable messages
 - Bugable parts
 - Tagable parts
 - Identification problem
 - Bootstrapping problem
 - Huge traffic overhead

MESSAGEVORTEX

GENERAL WEAK SPOTS OF ANONYMIZATION SOLUTIONS



Remains unsolved for censored environments



QUESTIONS?

