Reducing Metadata Leakage from Encrypted Files and Communication with PURBs

Kirill Nikitin*, Ludovic Barman*, Wouter Lueks, Matthew Underwood, Jean-Pierre Hubaux, and Bryan Ford

École polytechnique fédérale de Lausanne (EPFL)

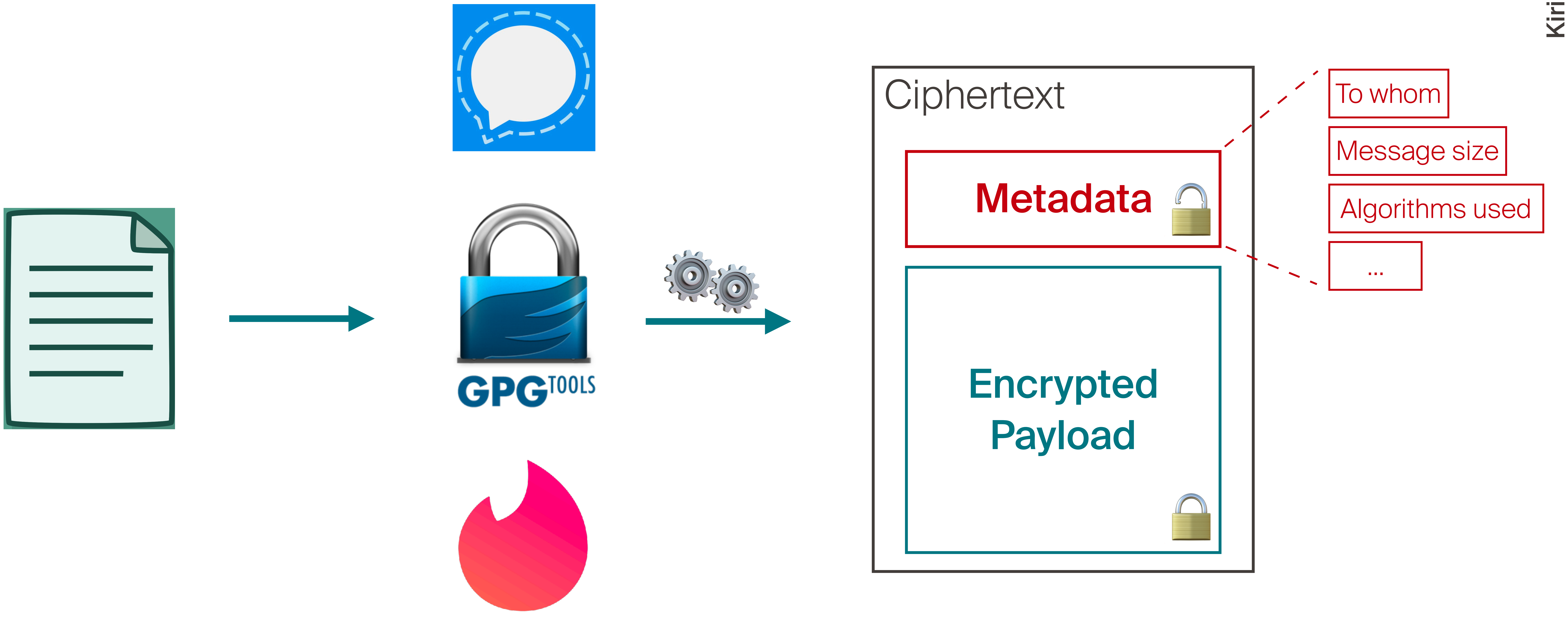
*Shared first authorship

@ni_kirill

@lbarman_ch

[Dog video]

Ciphertexts Expose Metadata in Clear



OpenPGP Packet Format

Packet Type

Format version

Recipient Key ID

Public-Key Algorithm

```
8502 0c01 9497 608d d051 8f79 010f ff46
bd7f 1821 27a9 42c4 01b4 7ecd 433e 7f90
. . . . . 74b8 139c a802 6678 ba0d 1abd
---------------------------------------
d264 014b 6a5a f586 e3fa b98e 92d1 6759
7186 2ccc ac50 3db7 fa03 4f31 dcd7 fa40
. . . . . 4b09 d9b4 1654 972d 5c22 47db
```

Session Key Part

Data Part

Encrypted Data

Credit for the picture of the attacker here and graphics afterwards is to Vecteezy.com

Kirill Nikitin

OpenPGP Packet Format

Packet Type | Format version | Recipient Key ID | Public-Key Algorithm

```
8502 0c01 9497 608d d051 8f79 010f ff46
bd7f 1821 27a9 42c4 01b4 7ecd 433e 7f90
. . . . . 74b8 139c a802 6678 ba0d 1abd
----------------------------------------
d264 014b 6a5a f586 e3fa b98e 92d1 6759
7186 2ccc ac50 3db7 fa03 4f31 dcd7 fa40
. . . . . 4b09 d9b4 1654 972d 5c22 47db
```

Session Key Part

Data Part

Encrypted Data

A message to the King of Sweden encrypted with RSA-512 using an outdated OpenPGP format??

Small key? Outdated format? I might crack it!

Credit for the picture of the attacker here and graphics afterwards is to Vecteezy.com

OpenPGP Packet Format

Packet Type | Format version | Recipient Key ID | Public-Key Algorithm

```
8502 0c01 9497 608d d051 8f79 010f ff46
bd7f 1821 27a9 42c4 01b4 7ecd 433e 7f90
. . . . . 74b8 139c a802 6678 ba0d 1abd
---------------------------------------
d264 014b 6a5a f586 e3fa b98e 92d1 6759
7186 2ccc ac50 3db7 fa03 4f31 dcd7 fa40
. . . . . 4b09 d9b4 1654 972d 5c22 47db
```

Session Key Part

Data Part

Encrypted Data

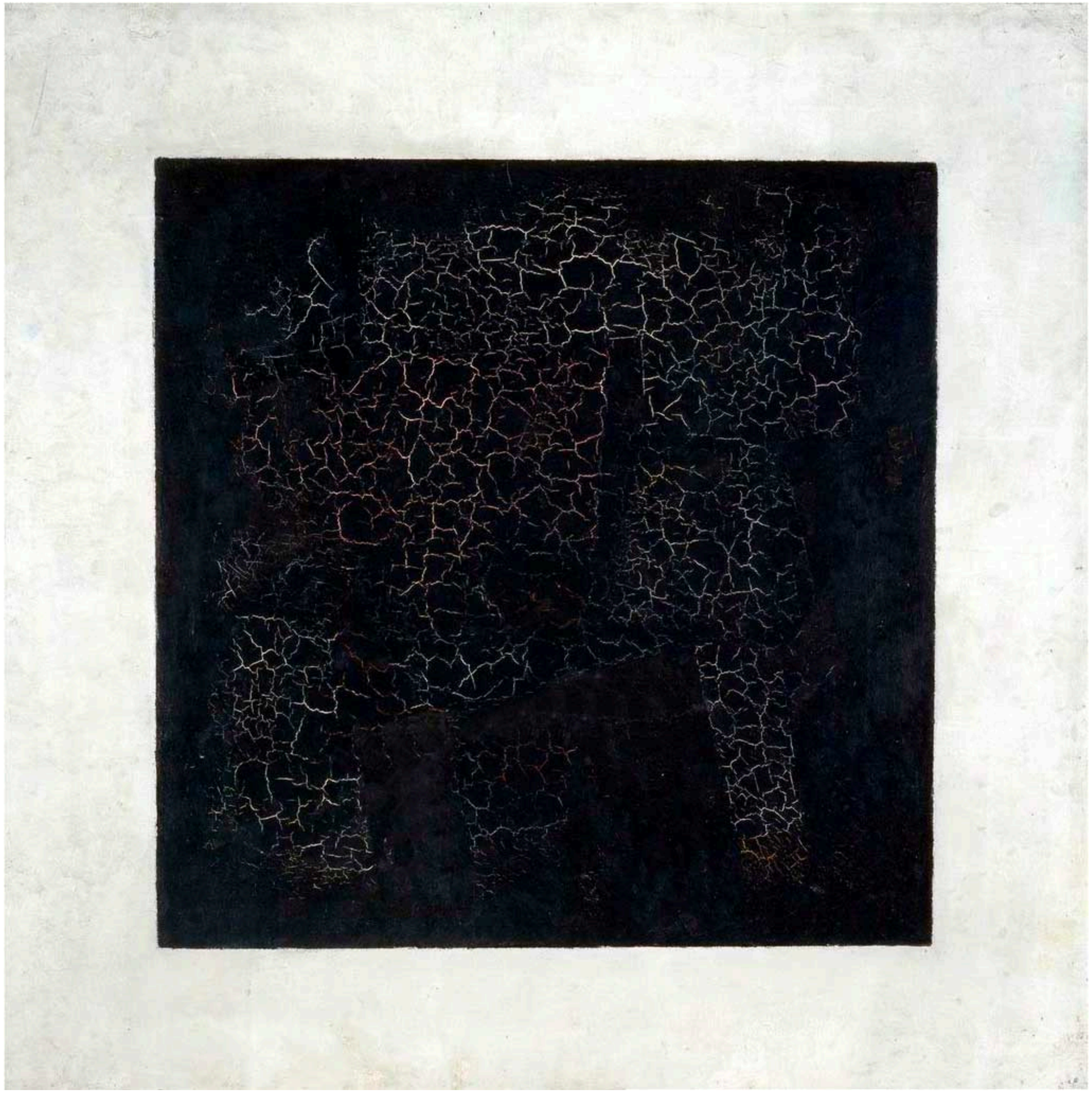
Is exposing metadata necessary?

A message to the King of Sweden encrypted with RSA-512 using an outdated OpenPGP format??

Small key? Outdated format? I might crack it!

Credit for the picture of the attacker here and graphics afterwards is to Vecteezy.com

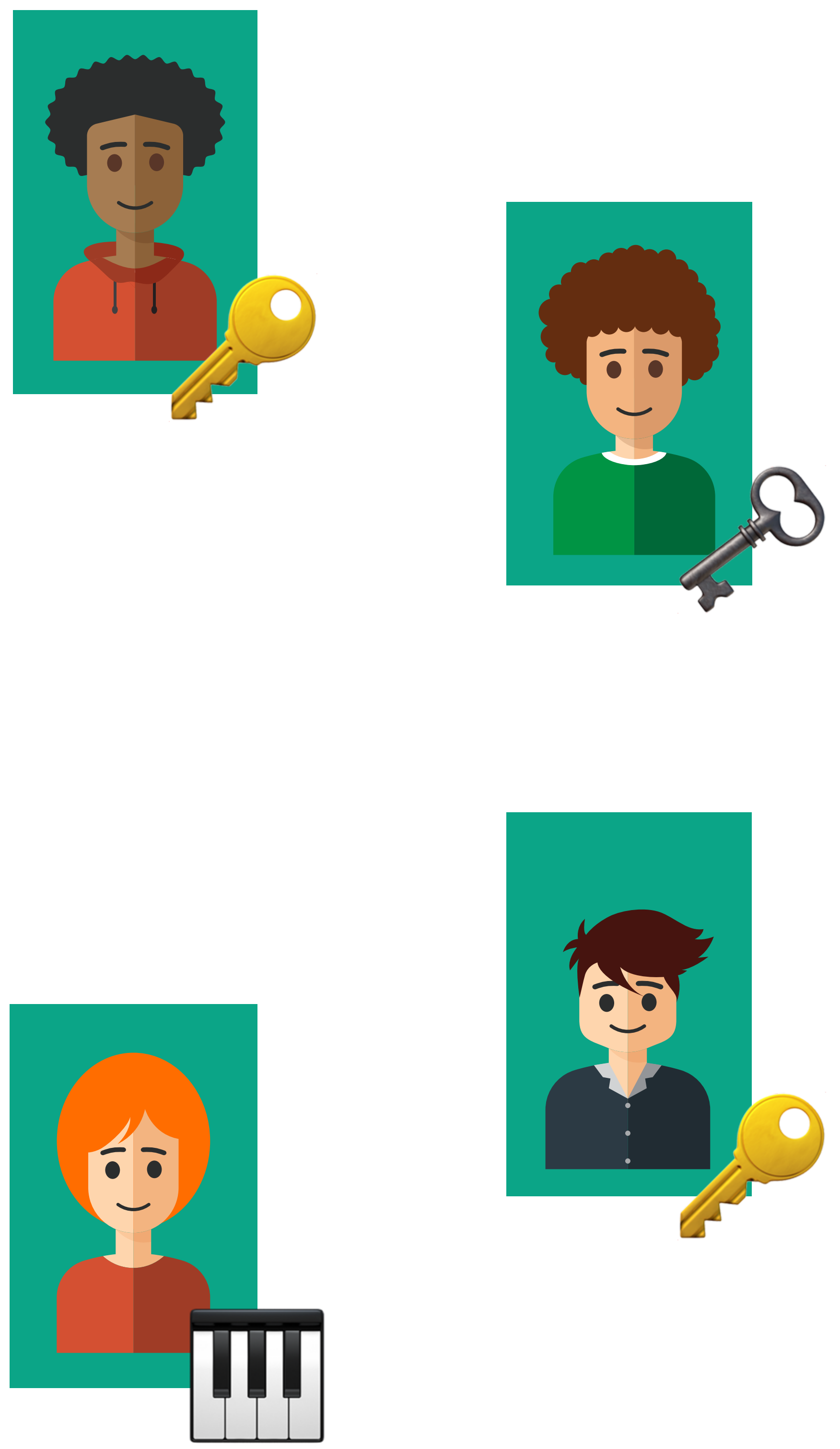
What If We Stripped Off All the Metadata?



“Black Square”, 1915,
by Kazimir Malevich

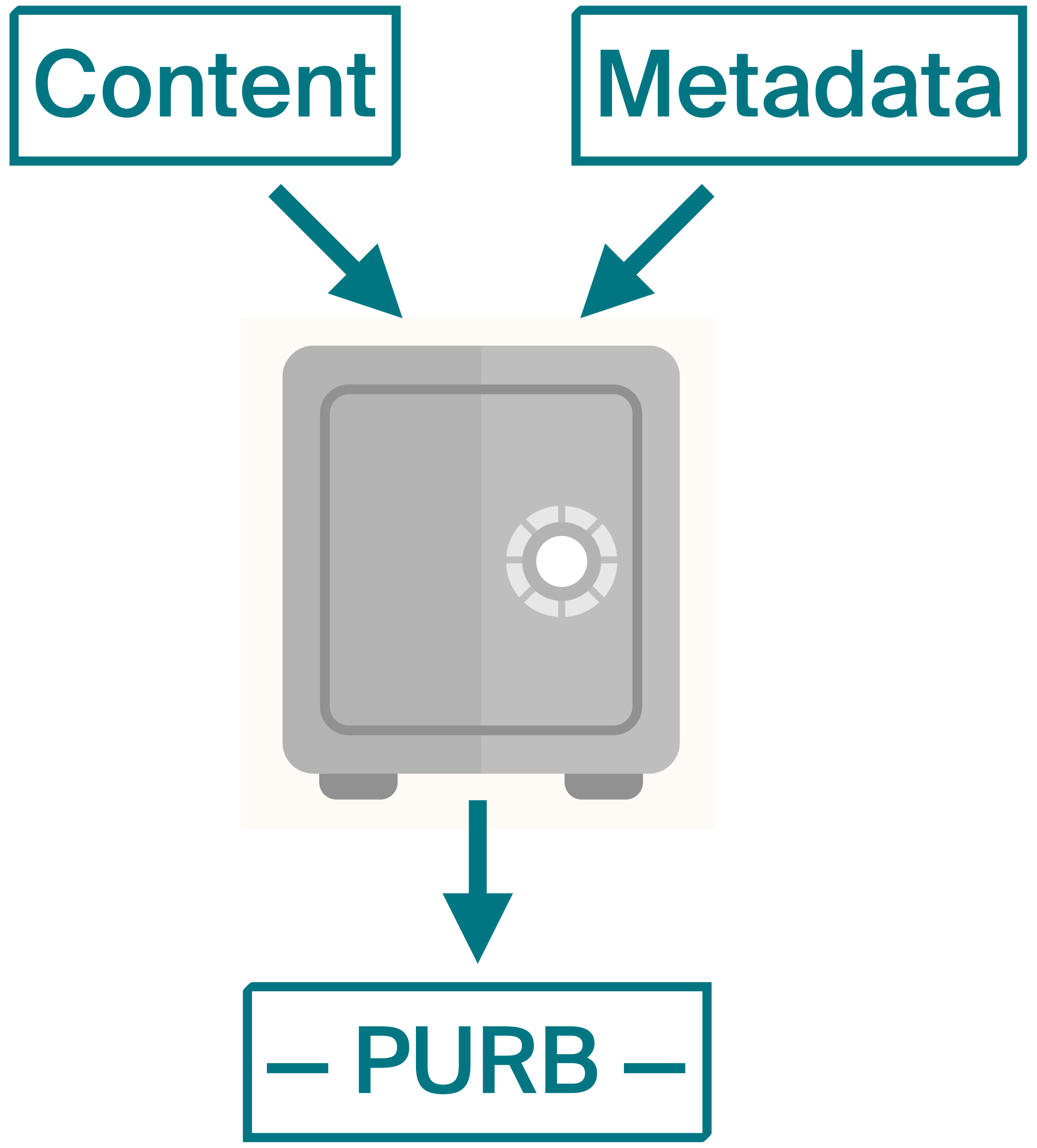
It Is Possible But Challenging

1. Efficient decoding
2. When addressing multiple recipients
3. Using different cryptographic algorithms

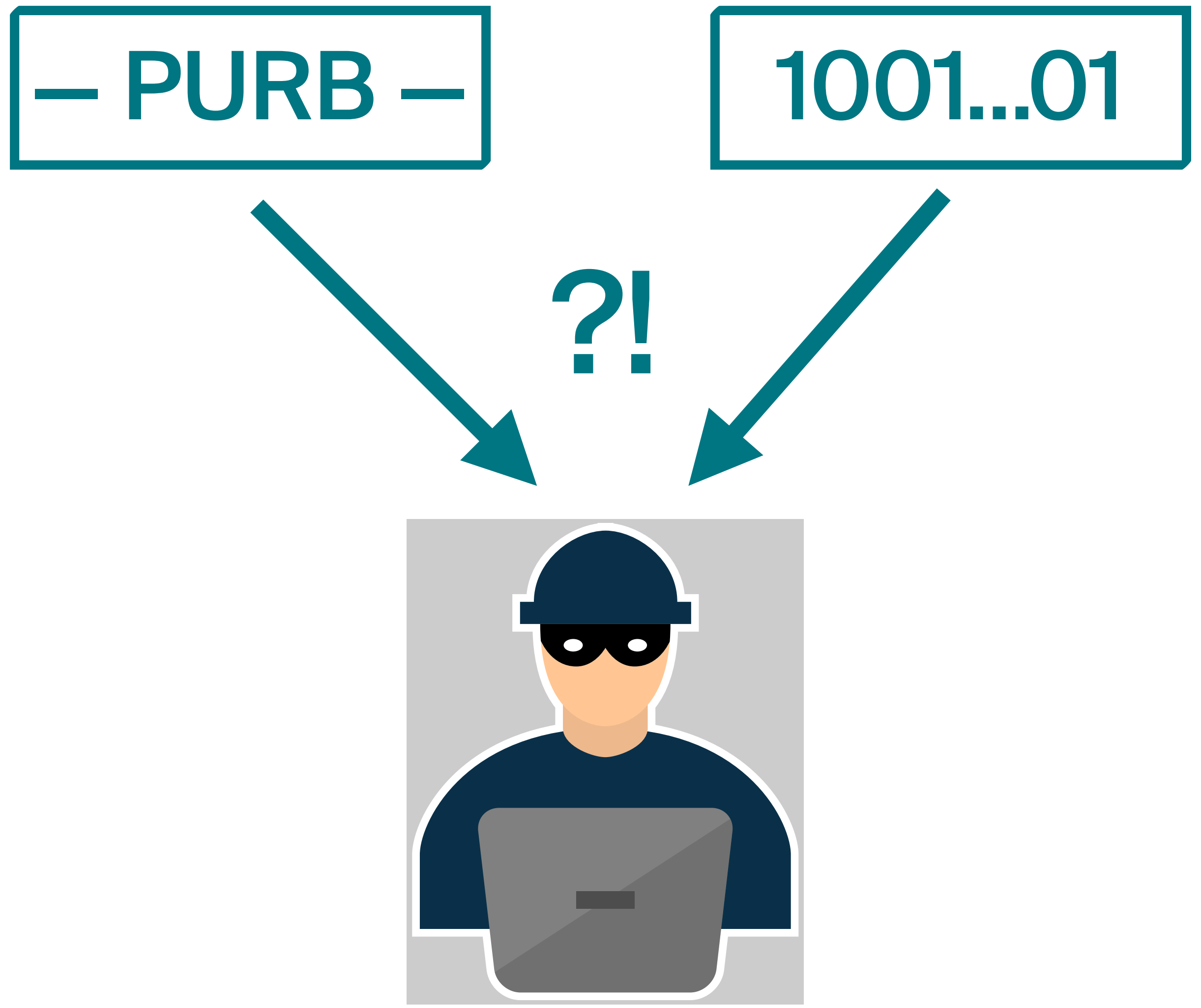


Padded Uniform Random Blobs (PURBs)

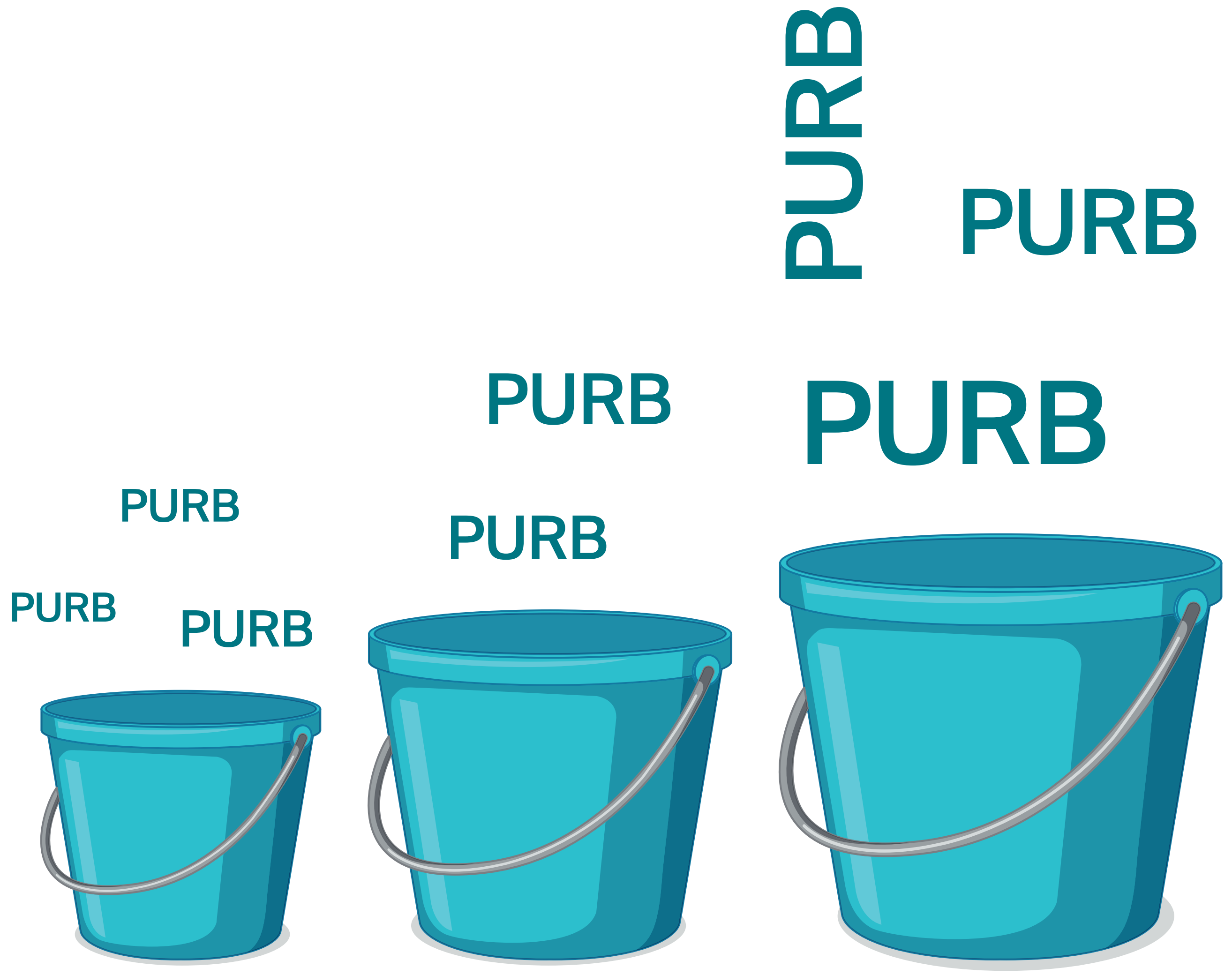
- A novel format for encrypted data without *any* metadata in clear.
- The properties (informally):



Content and metadata protection



Indistinguishability from random bits



Minimized length leakage

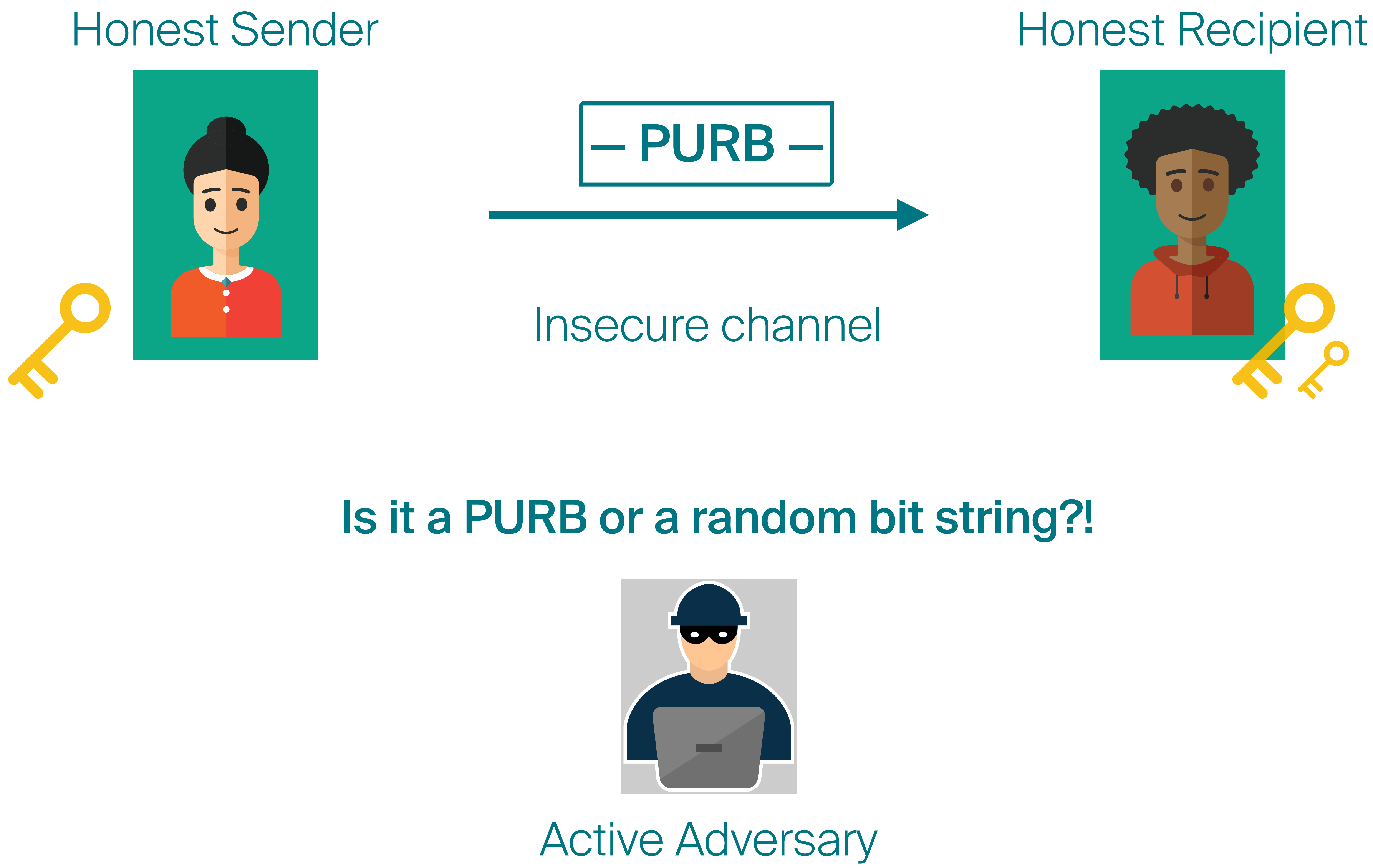
Padded Uniform Random Blobs (PURBs)

- Two core components
  - Encoding scheme (*Multi-Suite* PURB or MsPURB)
  - Padding scheme (Padmé)

Encoding scheme (MsPURB)

Roadmap to MsPURB

Single Recipient: Model



Single Recipient

Recipient – public key G^y

Similar to the Integrated Encryption Scheme (IES) [ABR01]

Single Recipient

Recipient – public key G^y

Similar to the Integrated Encryption Scheme (IES) [ABR01]

Sender:

1. Generates an ephemeral key pair x, G^x;

Single Recipient

Recipient – public key G^y

Similar to the Integrated Encryption Scheme (IES) [ABR01]

Sender:

1. Generates an ephemeral key pair x, G^x;
2. Computes a shared secret G^{yx};

Single Recipient

Recipient – public key G^y

Similar to the Integrated Encryption Scheme (IES) [ABR01]

$\boxed{Enc_K\,(data)}$

Payload

Sender:

1. Generates an ephemeral key pair x, G^x;
2. Computes a shared secret G^{yx};
3. Encrypts the data with one-time session key K;

Single Recipient

Recipient – public key G^y

Similar to the Integrated Encryption Scheme (IES) [ABR01]

Entry point	Payload
$AE_{G^{yx}}(K \mid\mid meta)$	$Enc_K (data)$

Sender:

1. Generates an ephemeral key pair x, G^x;
2. Computes a shared secret G^{yx};
3. Encrypts the data with one-time session key K;
4. Creates an entry point with K and other metadata, encrypted with G^{yx};

Single Recipient

Recipient – public key G^y

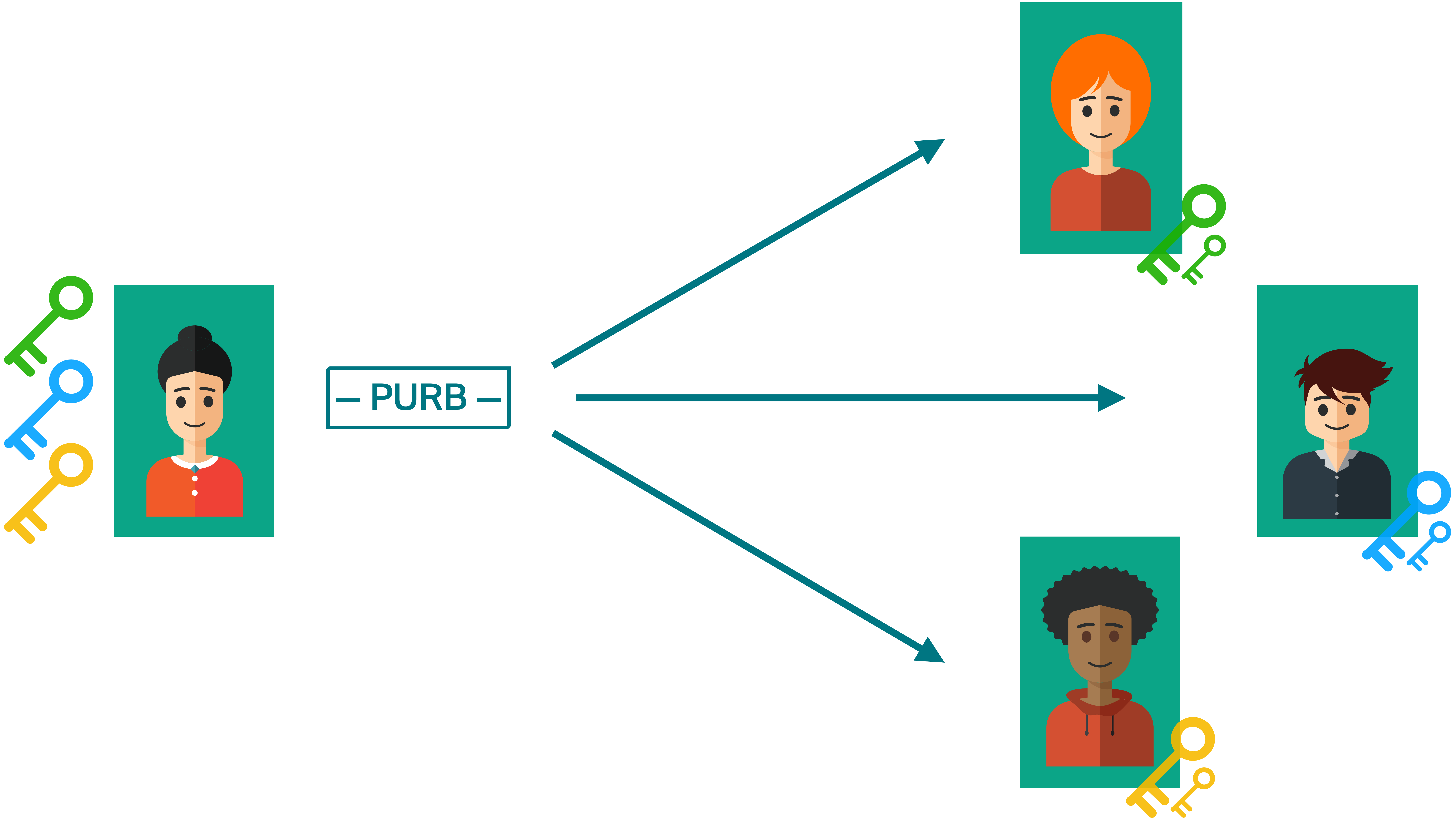
Similar to the Integrated Encryption Scheme (IES) [ABR01]

Hide(G^x)	$AE_{G^{yx}}$(K || meta)	Enc_K(data)
Encoded public key	**Entry point**	**Payload**

Sender:

1. Generates an ephemeral key pair x, G^x;
2. Computes a shared secret G^{yx};
3. Encrypts the data with one-time session key K;
4. Creates an entry point with K and other metadata, encrypted with G^{yx};
5. Encodes G^x as a uniform bit string, *e.g.*, with Elligator [BHKL13].

Multiple Recipients



Multiple Recipients

Recipients – public keys G^{y1}, G^{y2}, G^{y3}.

We create an entry point per recipient, each with K and metadata but encrypted with G^{y1x}, G^{y2x}, G^{y3x} respectively.

$AE_{G^{y1x}}(K\|\|meta)$	$AE_{G^{y2x}}(K\|\|meta)$	$AE_{G^{y3x}}(K\|\|meta)$

Multiple Recipients

Recipients – public keys G^{y1}, G^{y2}, G^{y3}.

We create an entry point per recipient, each with K and metadata but encrypted with G^{y1x}, G^{y2x}, G^{y3x} respectively.

$AE_{G^{y1x}}(K\|\|meta)$	$AE_{G^{y2x}}(K\|\|meta)$	$AE_{G^{y3x}}(K\|\|meta)$

But how do we organize these entry points in the PURB?

Linear Approach Strawman

Recipients – public keys G^{y1}, G^{y2}, G^{y3}.

Hide(G^x)

Enc_K (data)

We create an entry point per recipient, each with K and metadata but encrypted with G^{y1x}, G^{y2x}, G^{y3x} respectively.

Single Recipient | Multiple Recipients | Multiple Suites | Non-malleability

Linear Approach Strawman

Recipients – public keys G^{y1}, G^{y2}, G^{y3}.

Hide(G^x)	$AE_{G^{y1x}}$(K||meta)	$AE_{G^{y2x}}$(K||meta)	$AE_{G^{y3x}}$(K||meta)	Enc_K (data)

We create an entry point per recipient, each with K and metadata but encrypted with G^{y1x}, G^{y2x}, G^{y3x} respectively.

Linear Approach Strawman

Recipients – public keys G^{y1}, G^{y2}, G^{y3}.

Inefficient to decode

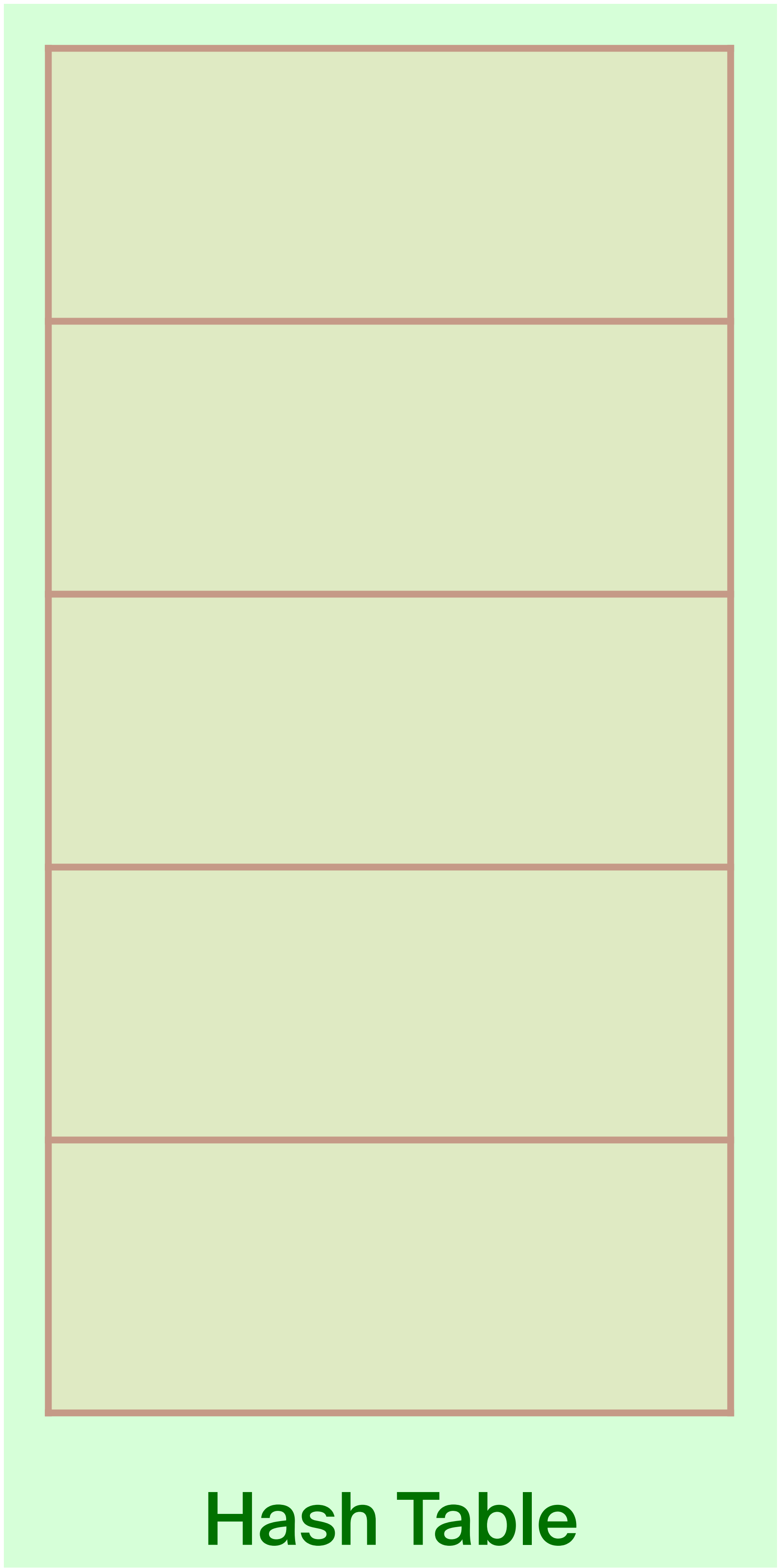
Hide(G^x)	$AE_{G^{y1x}}$(K||meta)	$AE_{G^{y2x}}$(K||meta)	$AE_{G^{y3x}}$(K||meta)	Enc_K (data)

We create an entry point per recipient, each with K and metadata but encrypted with G^{y1x}, G^{y2x}, G^{y3x} respectively.

Single Hash-Table Strawman

Recipients – public keys G^{y_1}, G^{y_2}, G^{y_3}.

Hide(G^x)

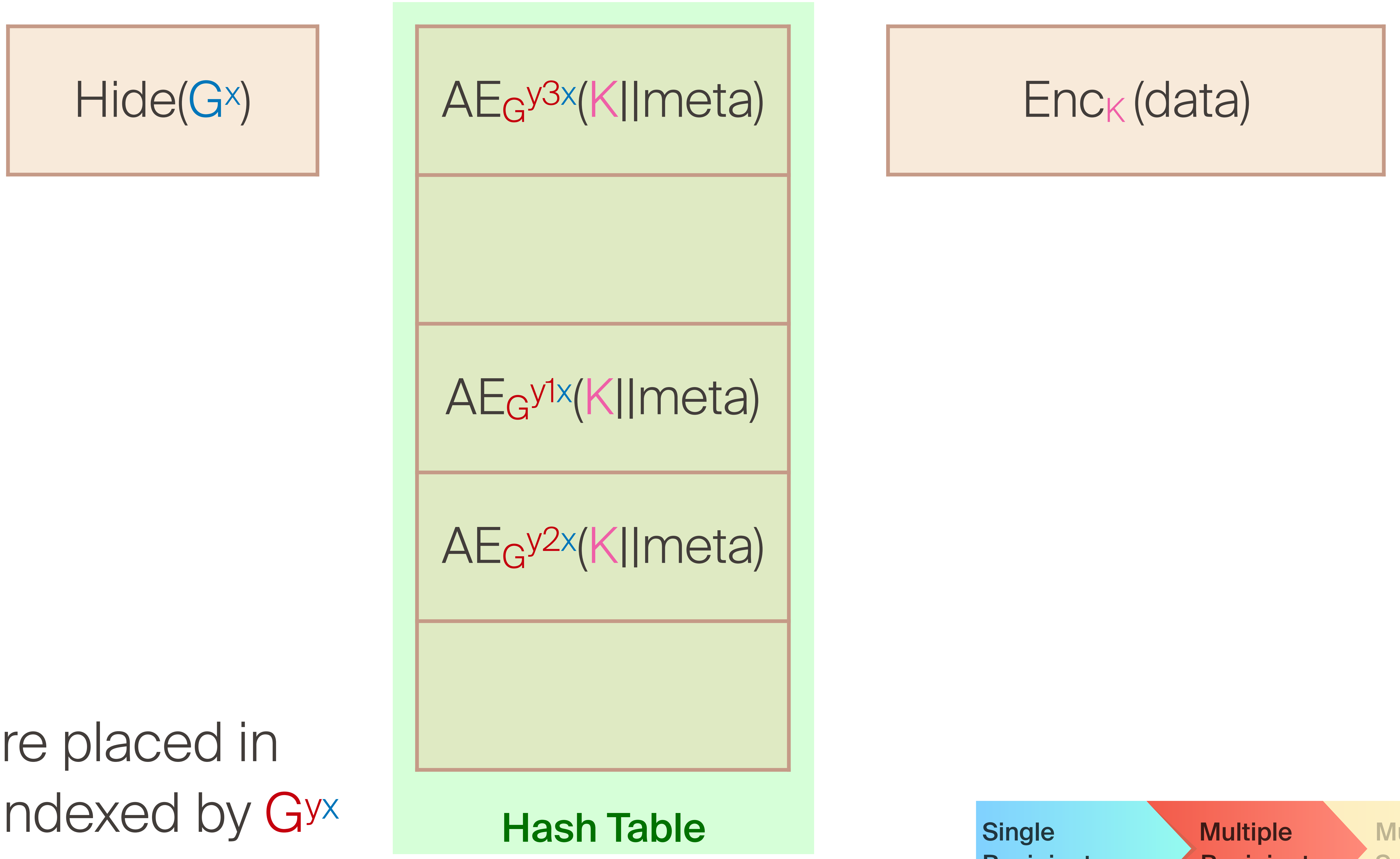


Enc_K(data)

Entry points are placed in a hash table, indexed by G^{yx}

Single Hash-Table Strawman

Recipients – public keys G^{y1}, G^{y2}, G^{y3}.



Entry points are placed in a hash table, indexed by G^{yx}

Single Hash-Table Strawman

Recipients – public keys G^{y1}, G^{y2}, G^{y3}.

Hide(G^x)

Hash Table
$AE_{G^{y3x}}(K\|\|meta)$
random
$AE_{G^{y1x}}(K\|\|meta)$
$AE_{G^{y2x}}(K\|\|meta)$
random

Enc_K (data)

Entry points are placed in a hash table, indexed by G^{yx}

Single Hash-Table Strawman

Recipients – public keys G^{y1}, G^{y2}, G^{y3}.

Hide(G^x)

$AE_{G^{y3x}}(K||meta)$

random

$AE_{G^{y1x}}(K||meta)$

$AE_{G^{y2x}}(K||meta)$

random

Hash Table

Enc_K (data)

1. **Space waste**
2. **Bound on N of recipients**

Entry points are placed in a hash table, indexed by G^{yx}

Multiple Recipients: Our Solution

Recipients – public keys G^{y1}, G^{y2}, G^{y3}.

Hide(G^x)

Enc_K (data)

Entry points are placed in a series of growing hash-tables!

Multiple Recipients: Our Solution

Recipients – public keys G^{y1}, G^{y2}, G^{y3}.

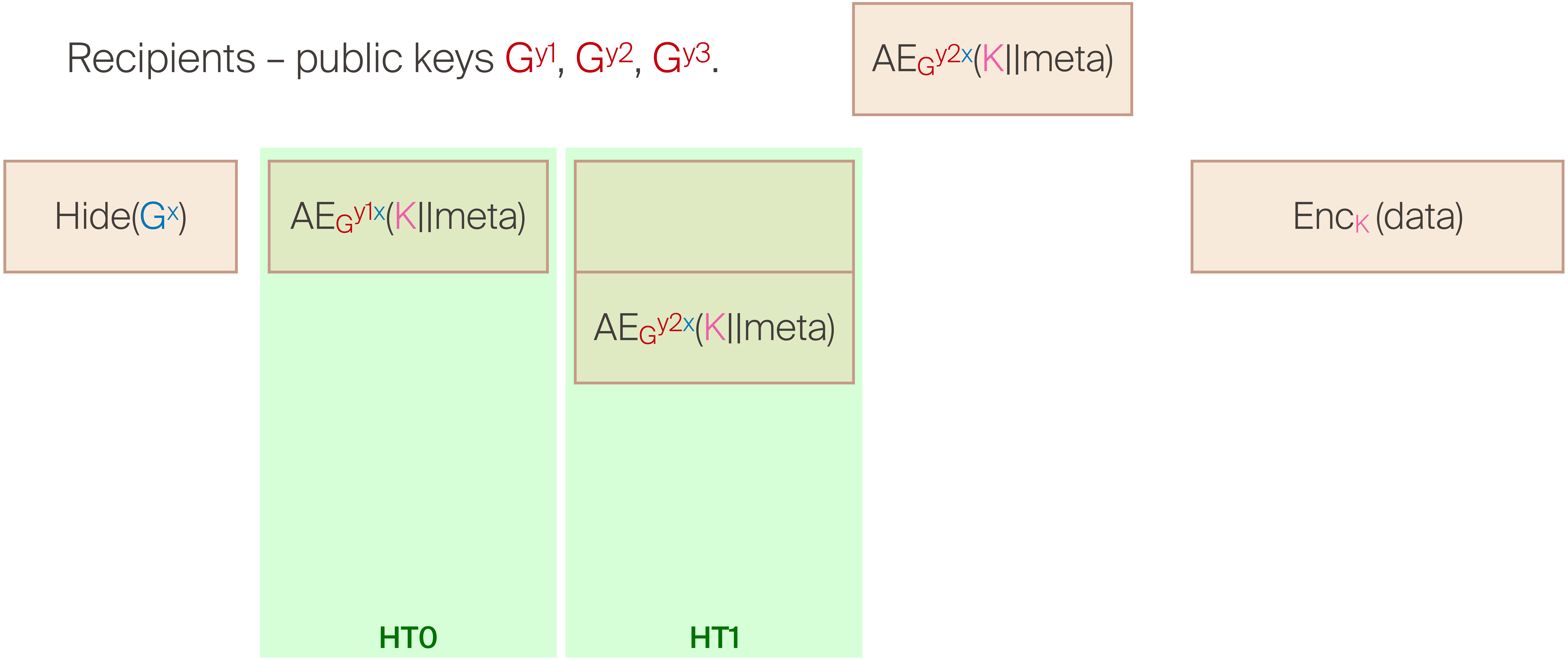
$AE_{G^{y1x}}(K||meta)$

Hide(G^x)	$AE_{G^{y1x}}(K||meta)$	Enc_K (data)
	HT0	

Entry points are placed in a series of growing hash-tables!

Multiple Recipients: Our Solution

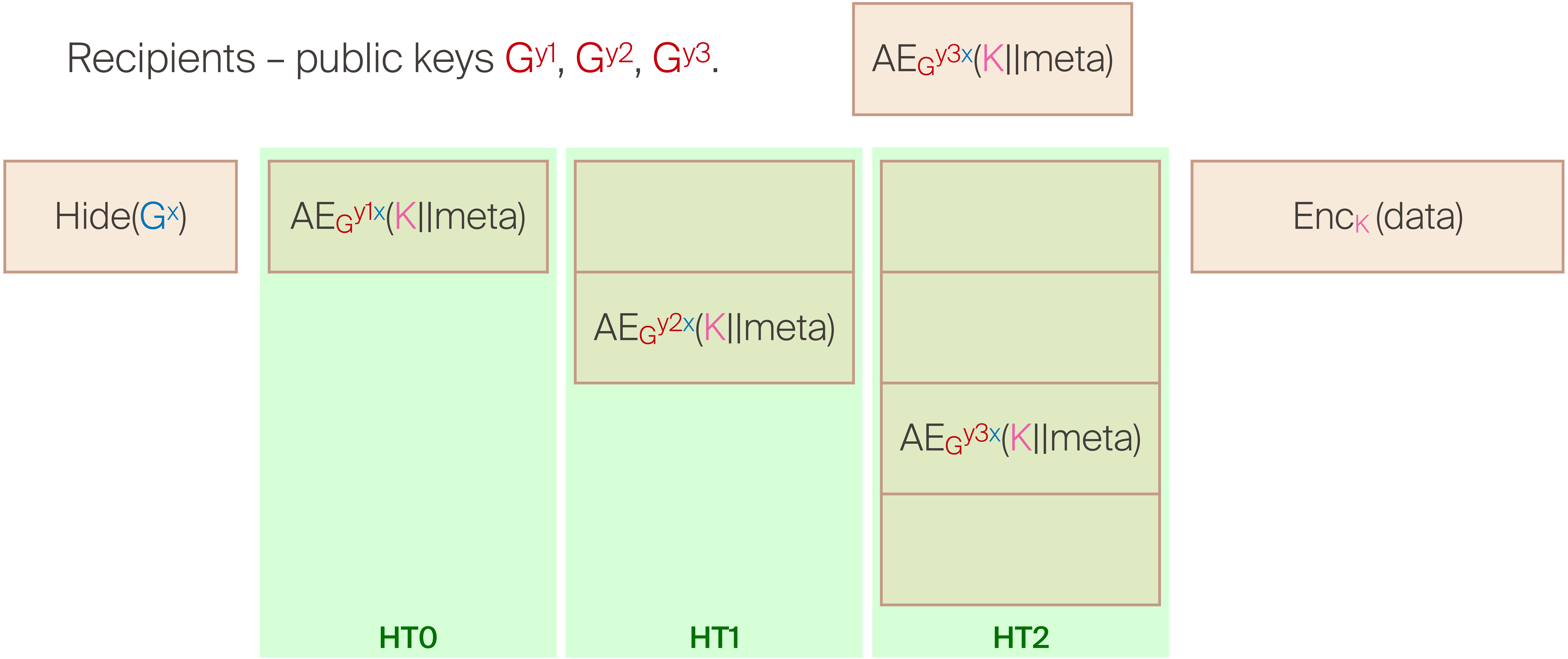
Recipients – public keys G^{y1}, G^{y2}, G^{y3}.



Entry points are placed in a series of growing hash-tables!

Multiple Recipients: Our Solution

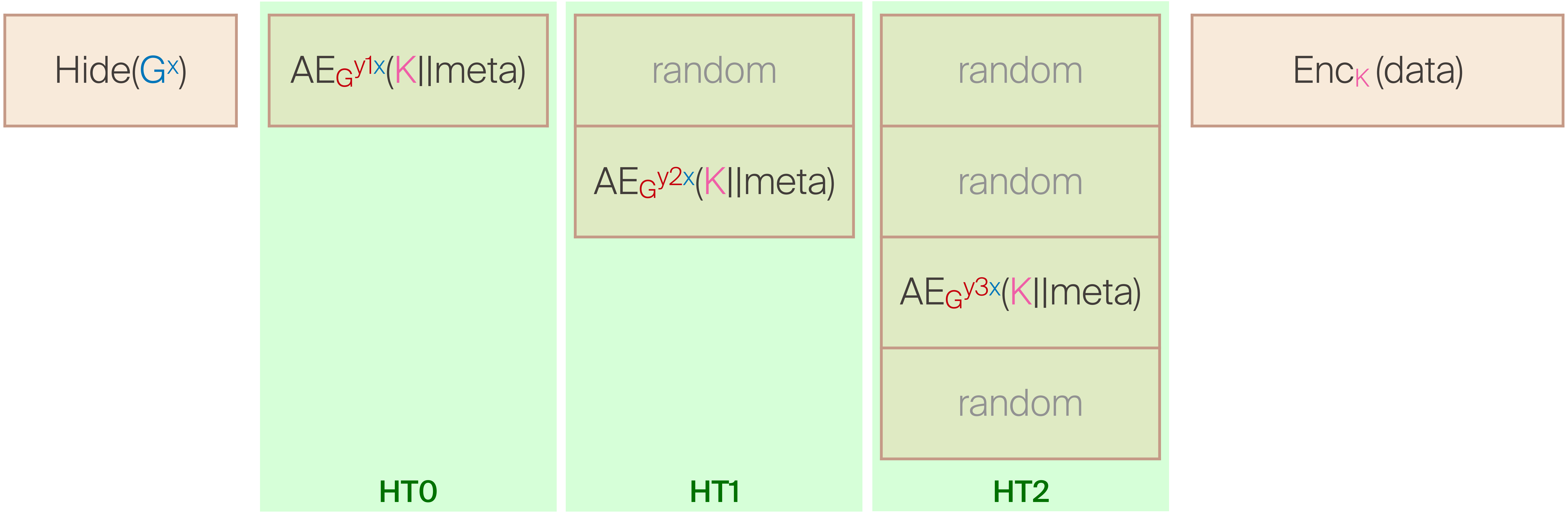
Recipients – public keys G^{y1}, G^{y2}, G^{y3}.



Entry points are placed in a series of growing hash-tables!

Multiple Recipients: Our Solution

Recipients – public keys G^{y1}, G^{y2}, G^{y3}.



Entry points are placed in a series of growing hash-tables!

Multiple Recipients: Our Solution

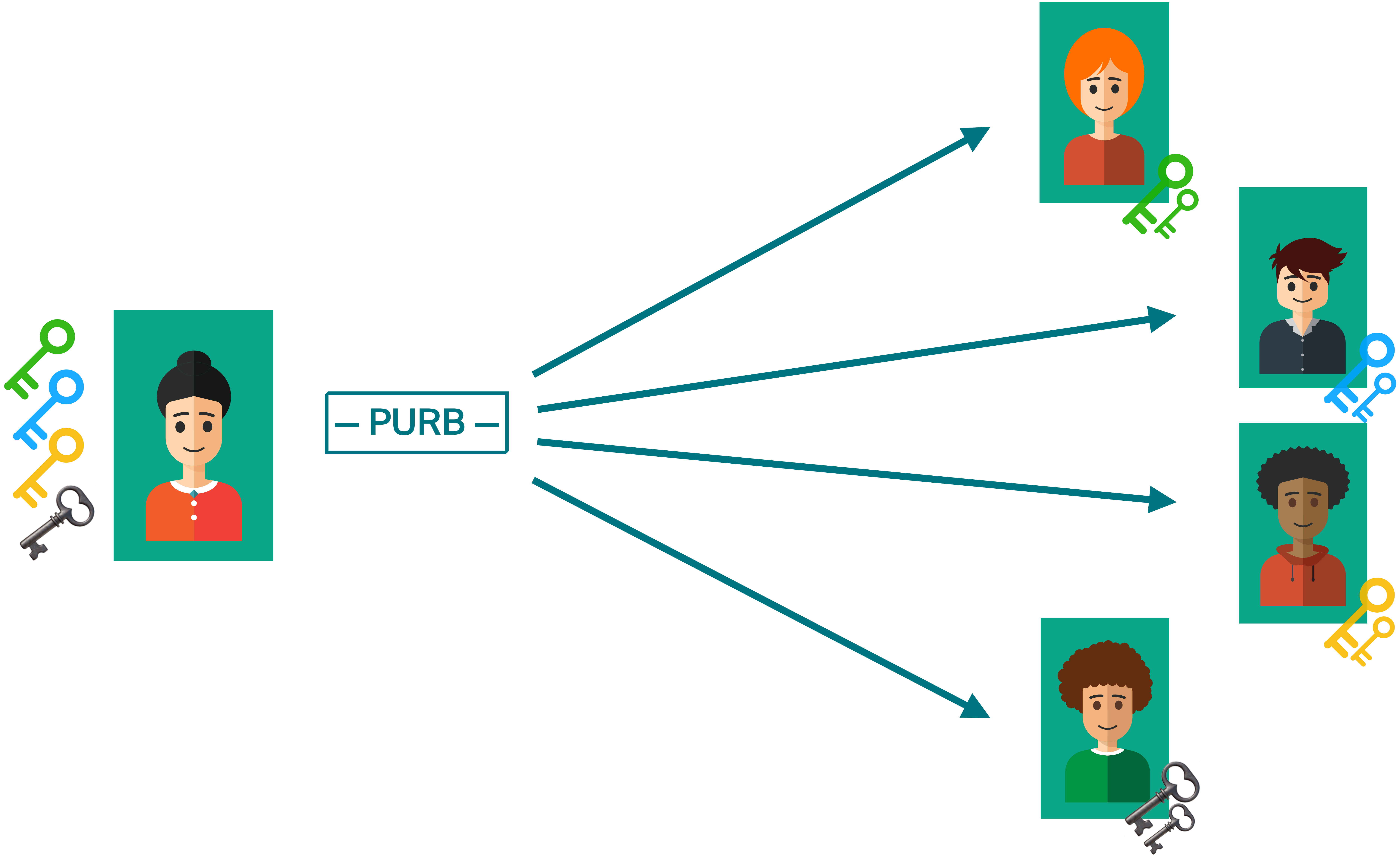
Recipients – public keys G^{y1}, G^{y2}, G^{y3}.

Hide(G^x)	HT0	HT1	HT2	Enc_K (data)
	$AE_{G^{y1x}}$(K||meta)	random	random	
		$AE_{G^{y2x}}$(K||meta)	random	
			$AE_{G^{y3x}}$(K||meta)	
			random	

Decoding in Log_2 len(PURB)

Entry points are placed in a series of growing hash-tables!

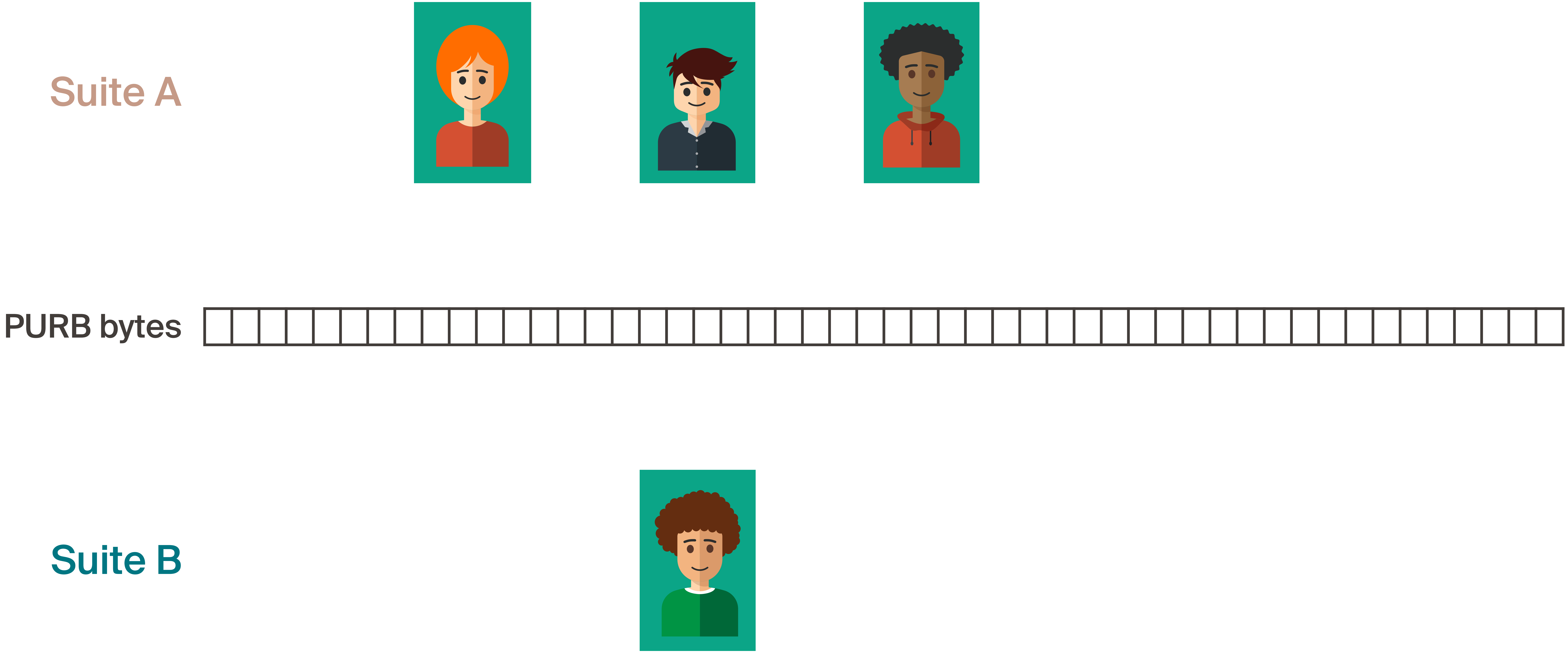
Multiple Suites



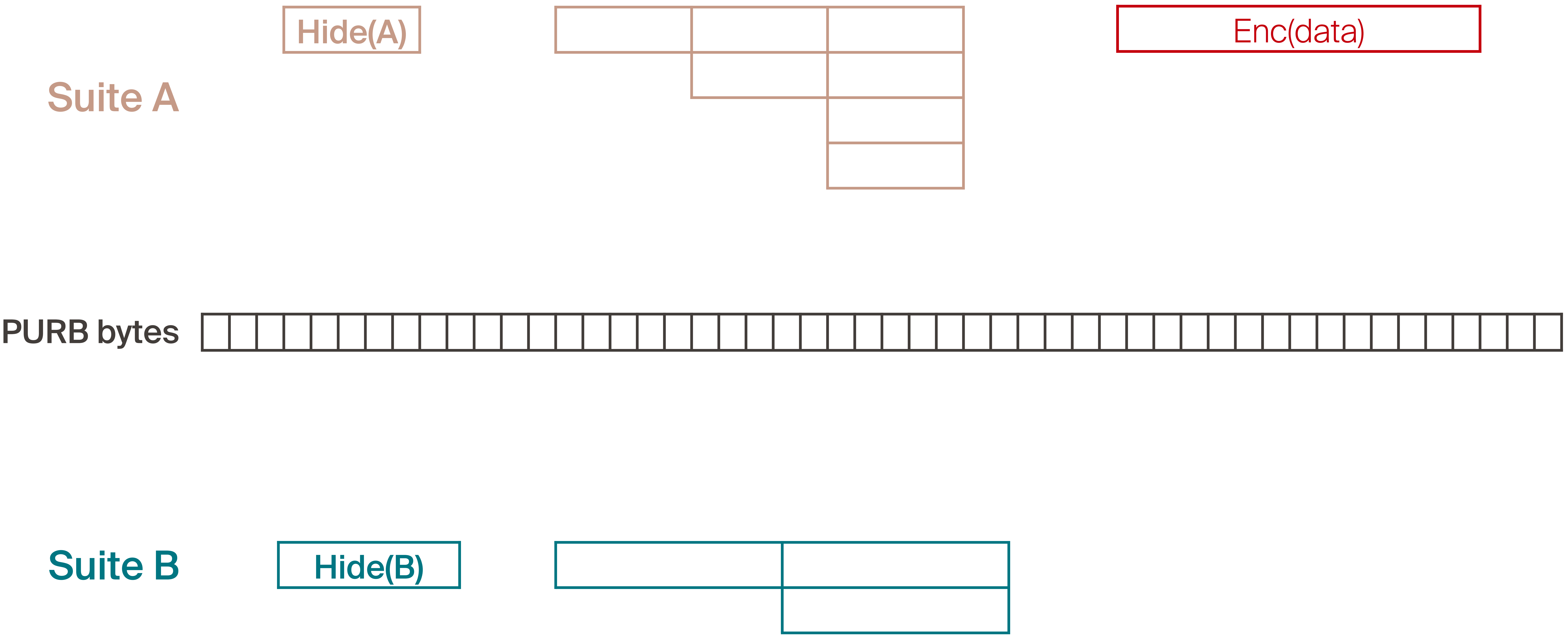
Multiple Suites

- Recipients use several distinct suites, based on public-key group (e.g., Curve25519 or Curve448) or entry point length.
- Each suite (an encoded public key and hash tables) becomes a distinct logical layer in a PURB, and these layers overlap!

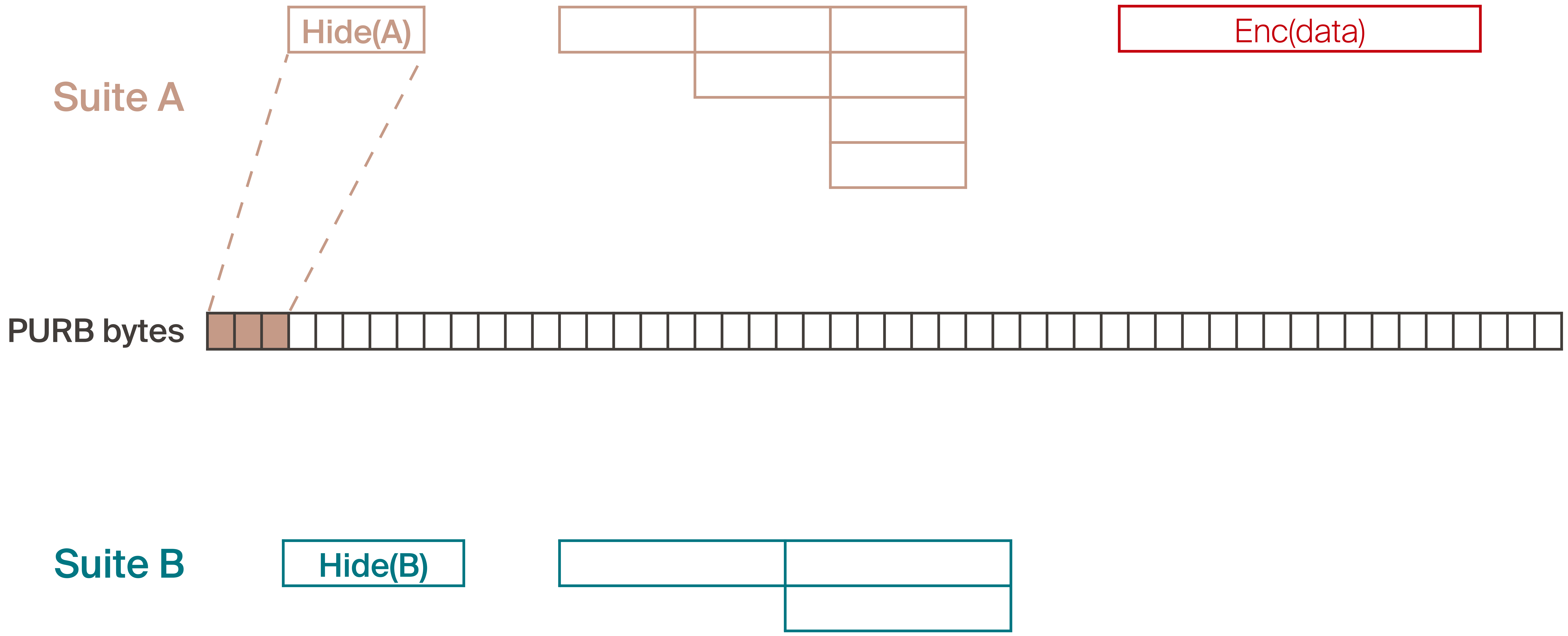
Multiple Suites: Layout



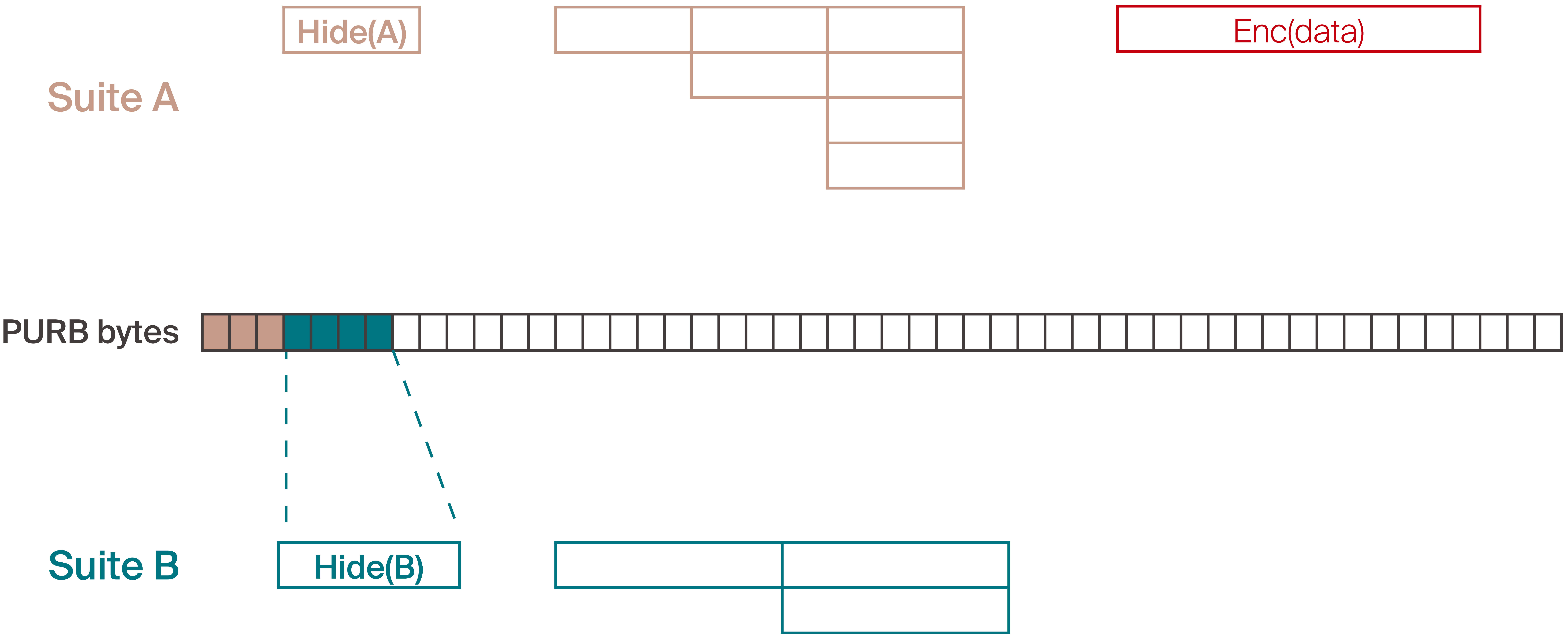
Multiple Suites: Layout



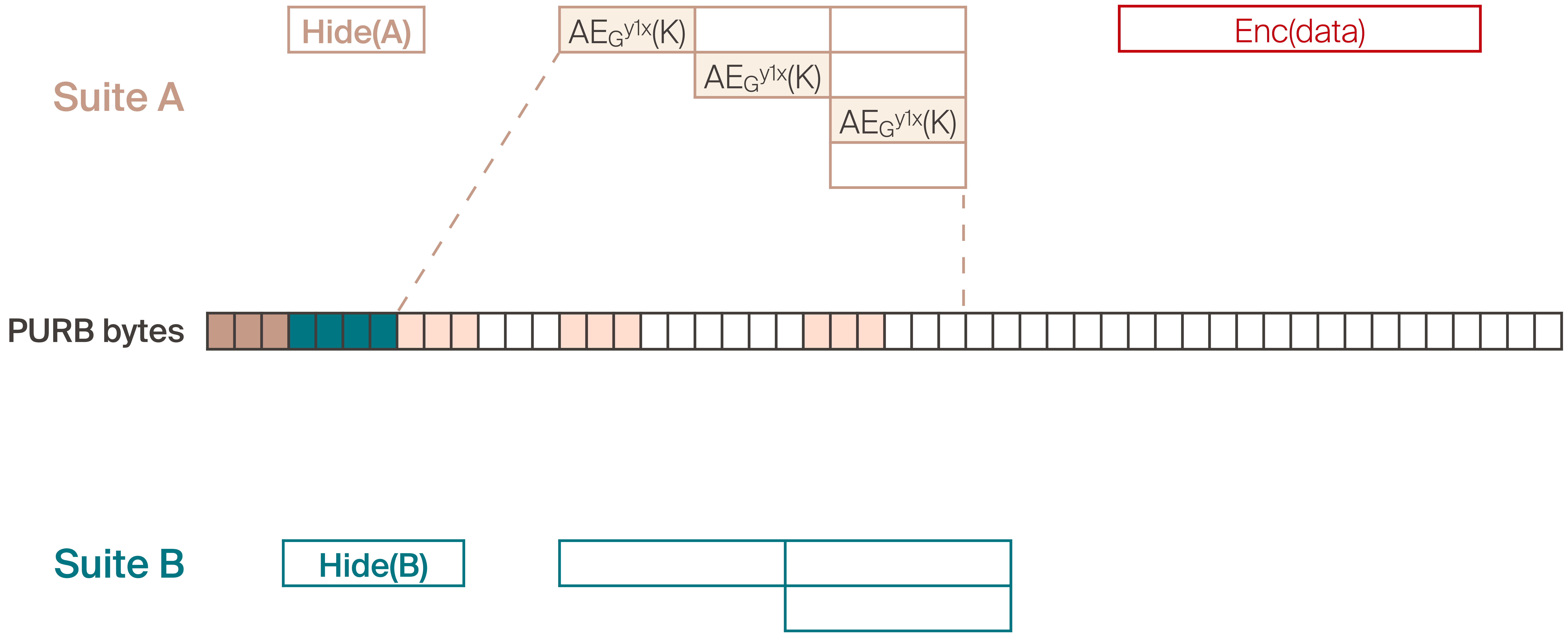
Multiple Suites: Layout



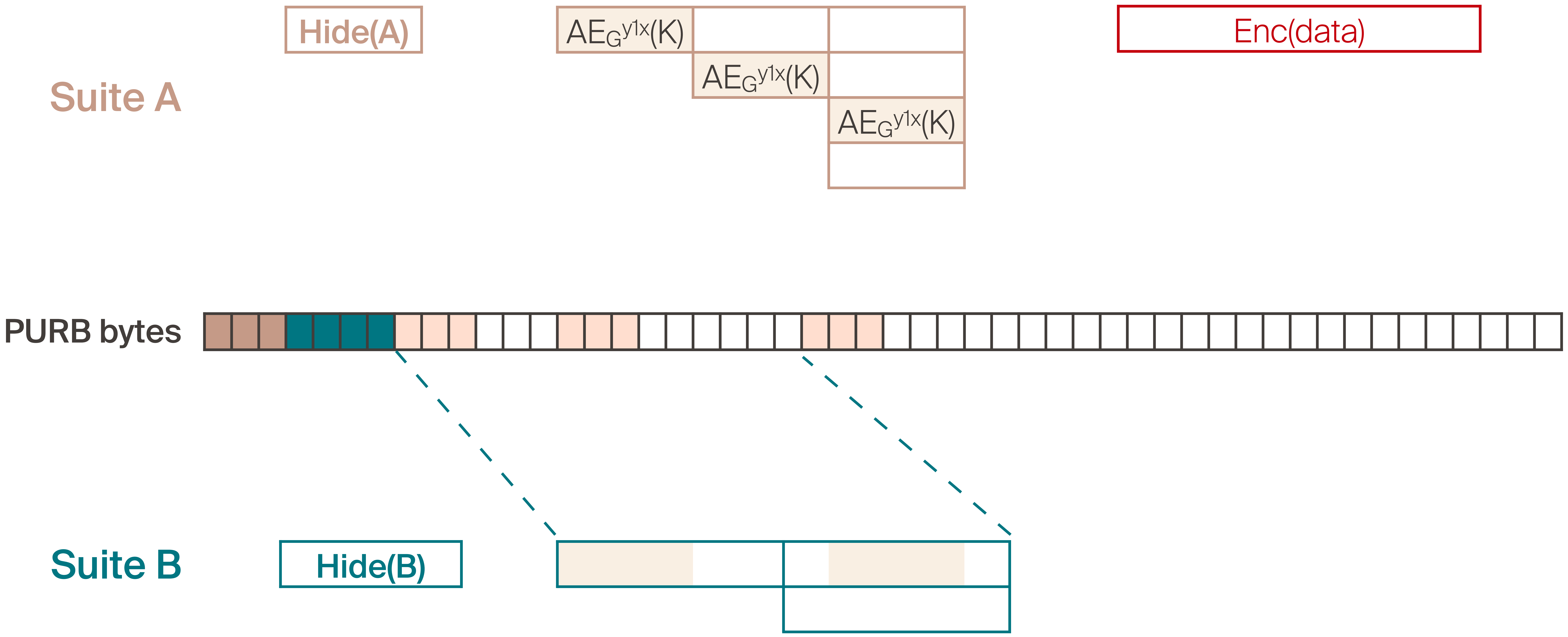
Multiple Suites: Layout



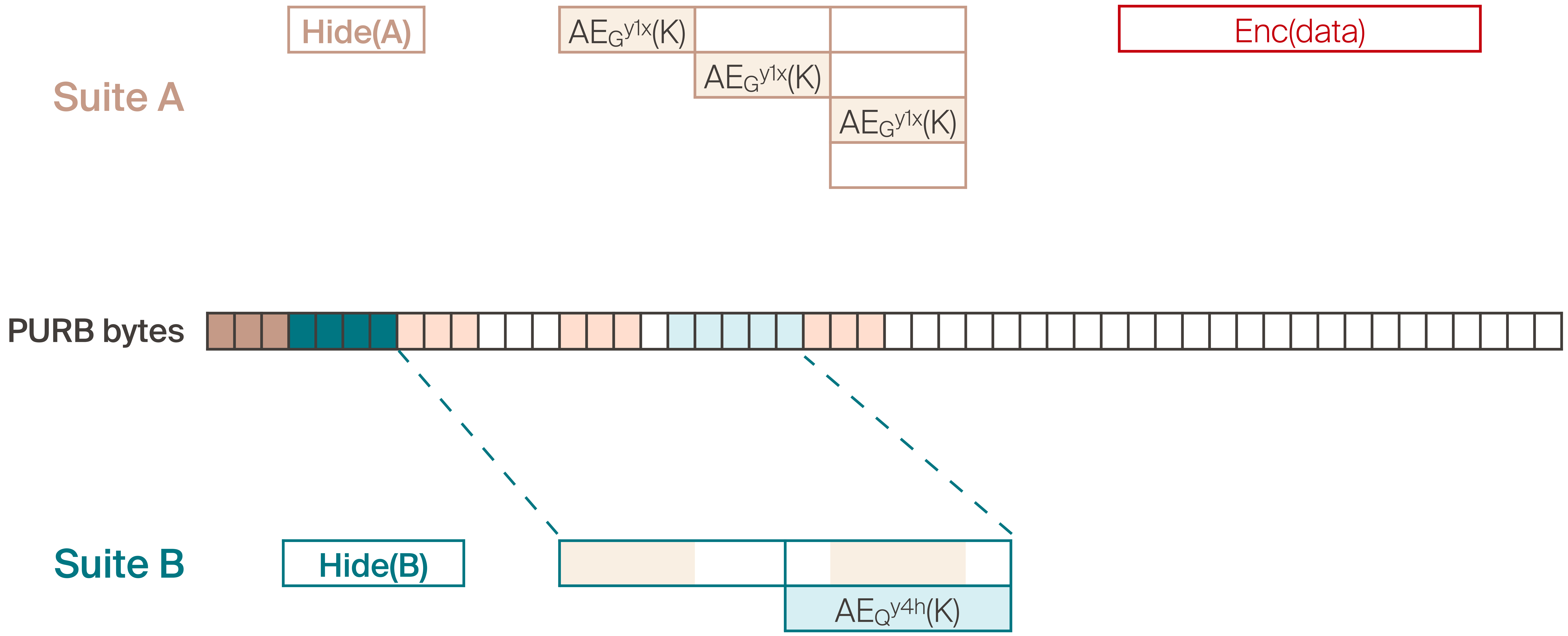
Multiple Suites: Layout



Multiple Suites: Layout

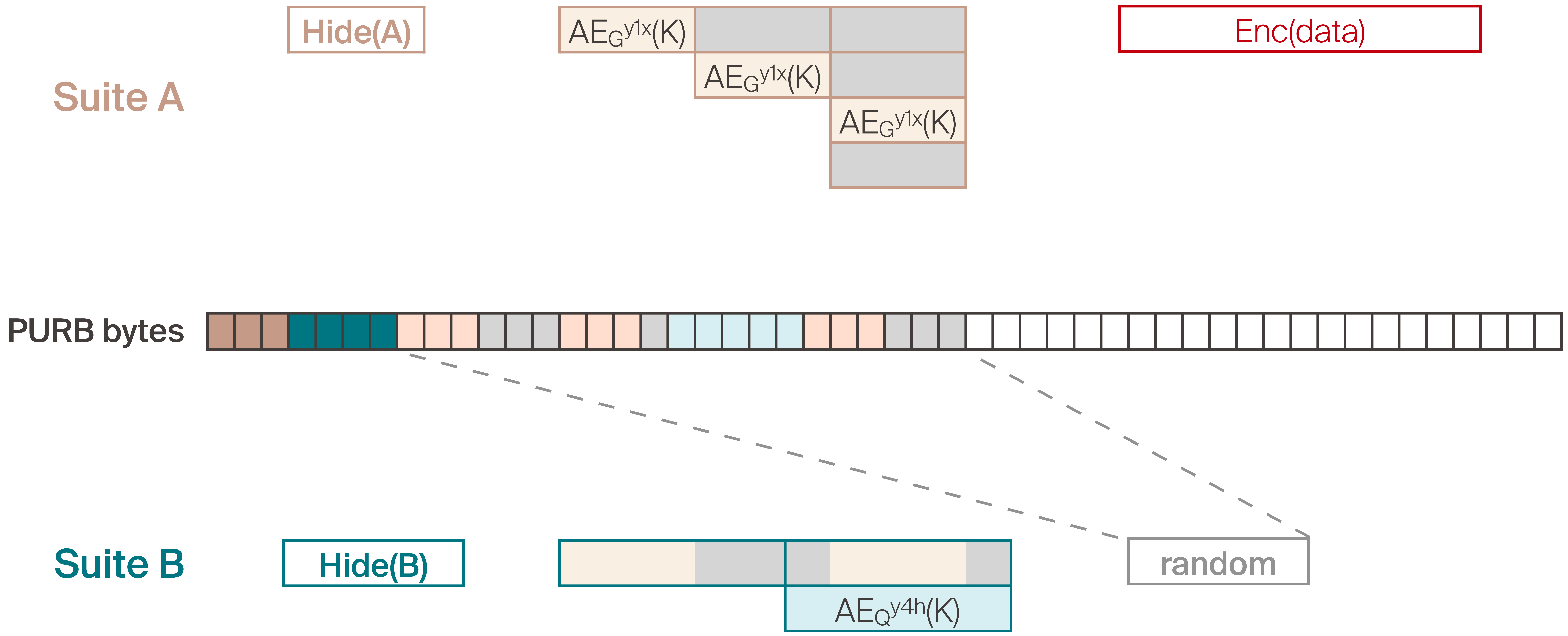


Multiple Suites: Layout

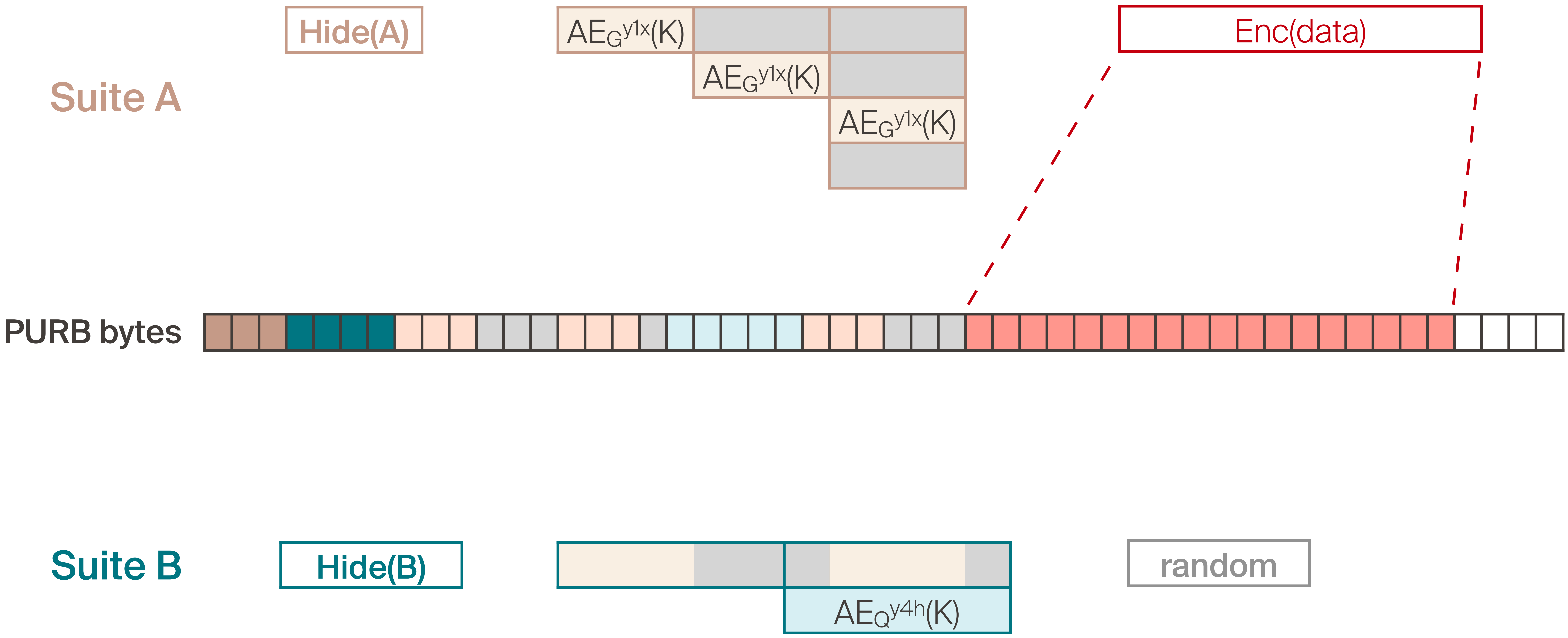


Single Recipient | Multiple Recipients | Multiple Suites | Non-malleability

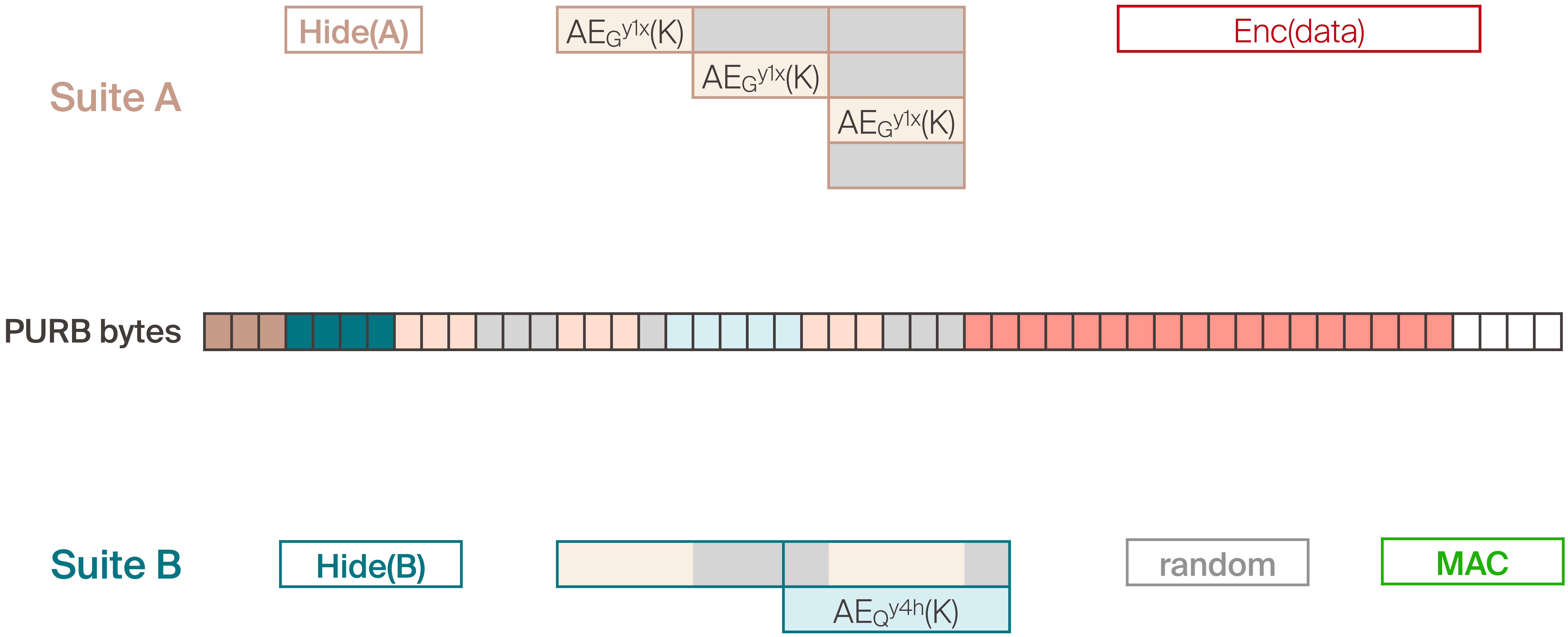
Multiple Suites: Layout



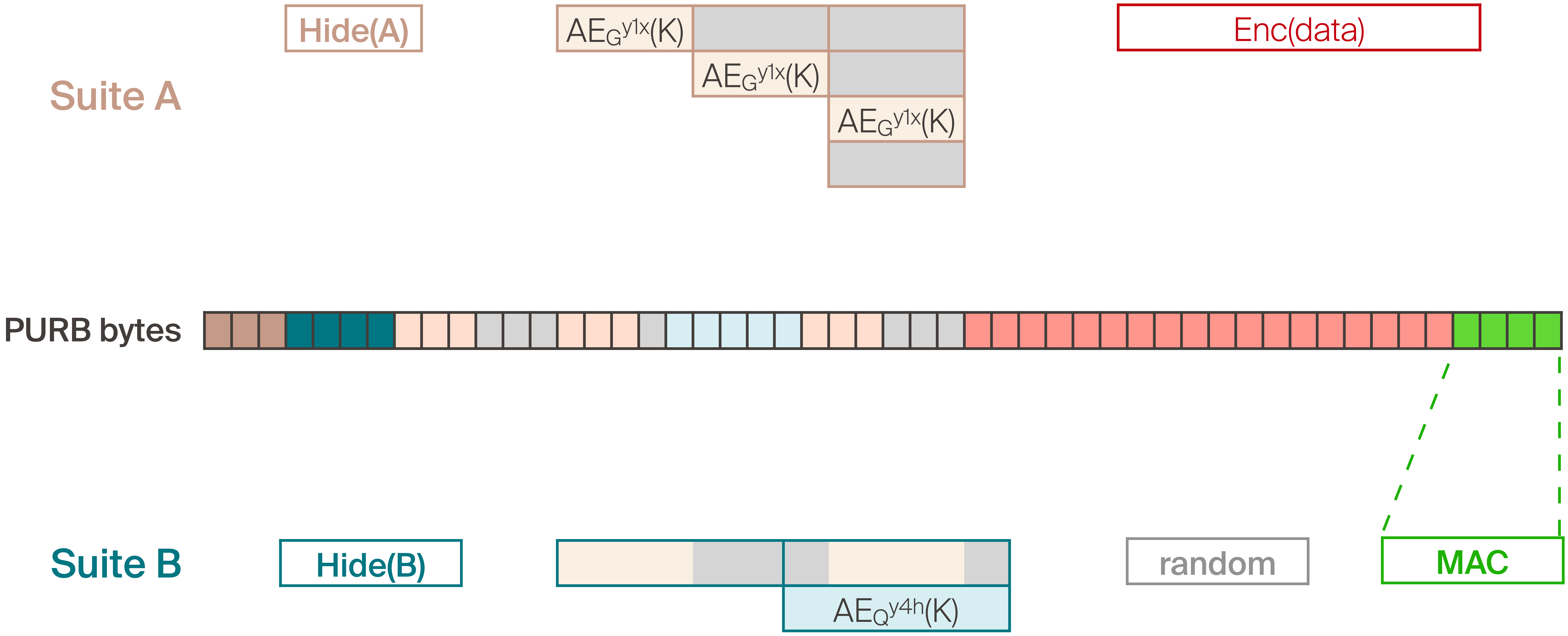
Multiple Suites: Layout



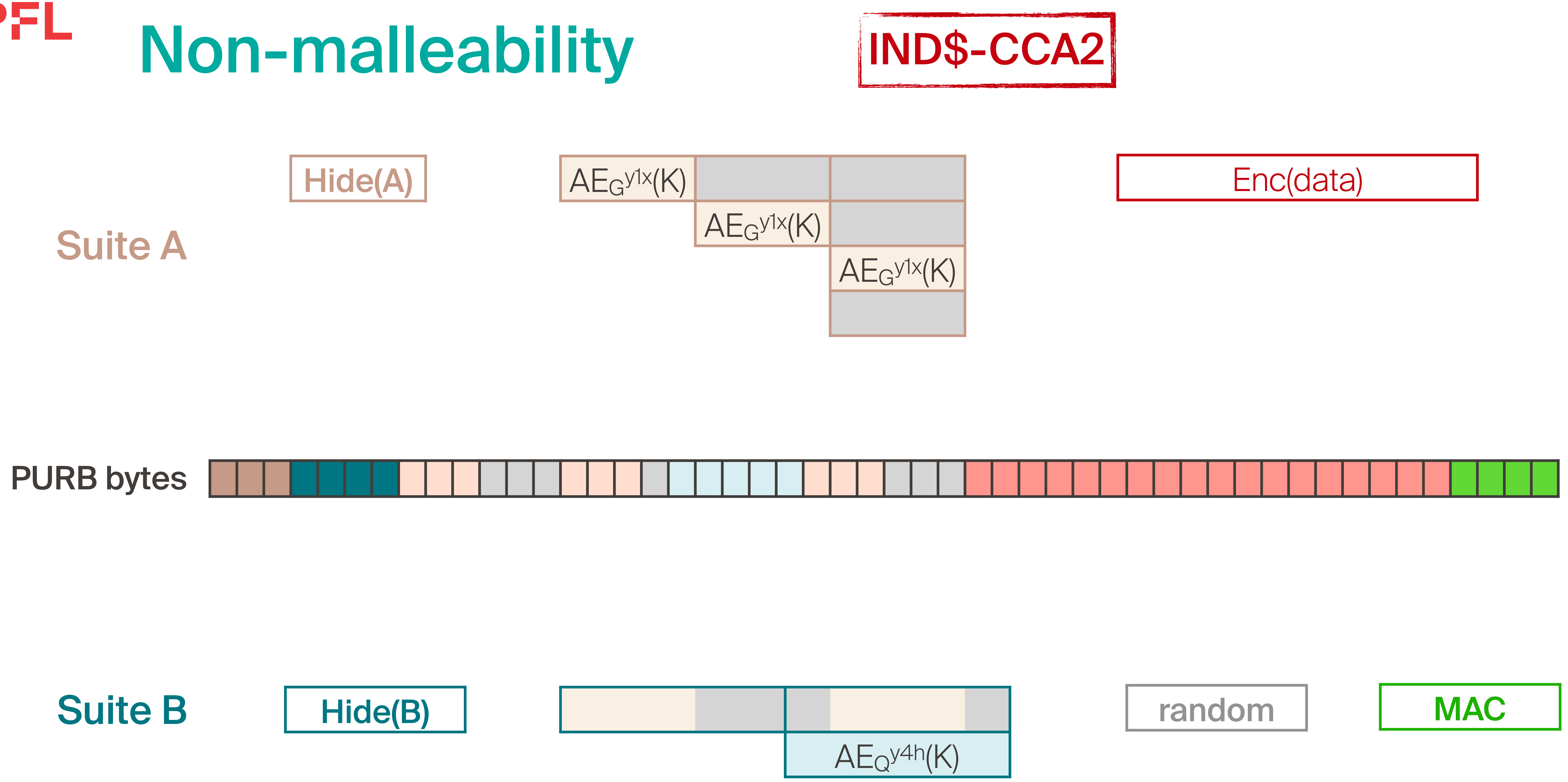
Non-malleability



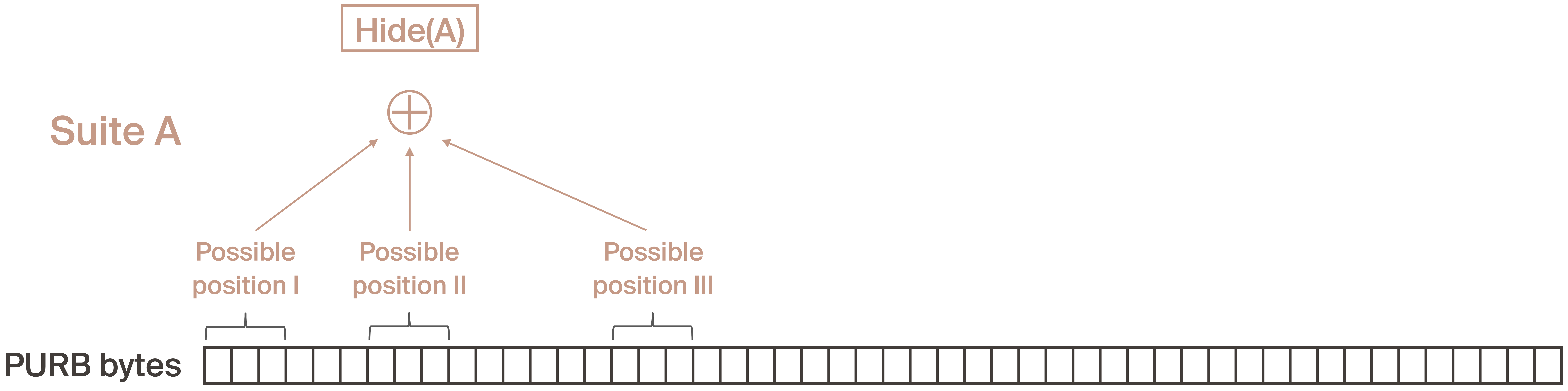
Non-malleability



Non-malleability

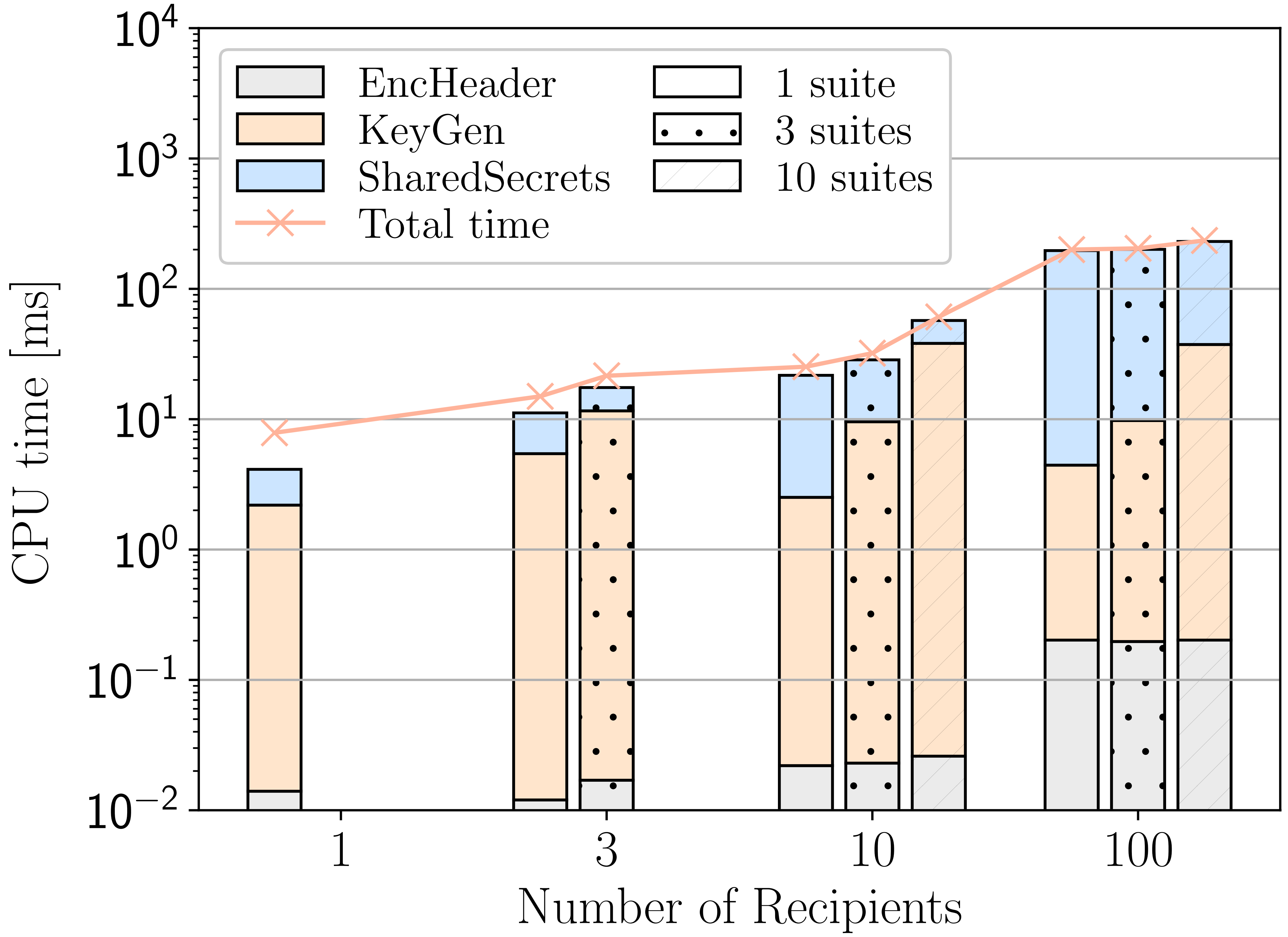


Finding Public Keys Efficiently

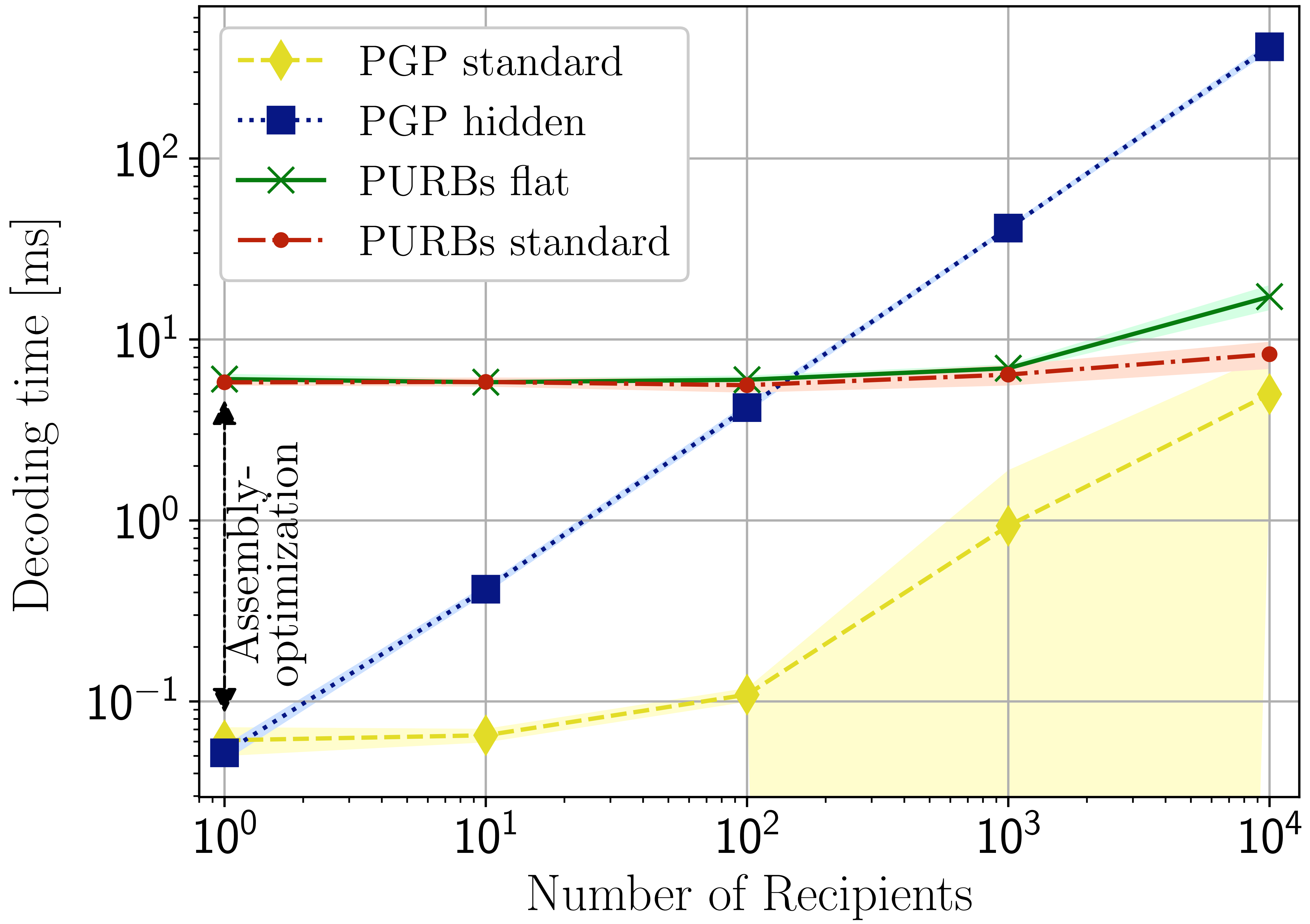


See the paper for the details

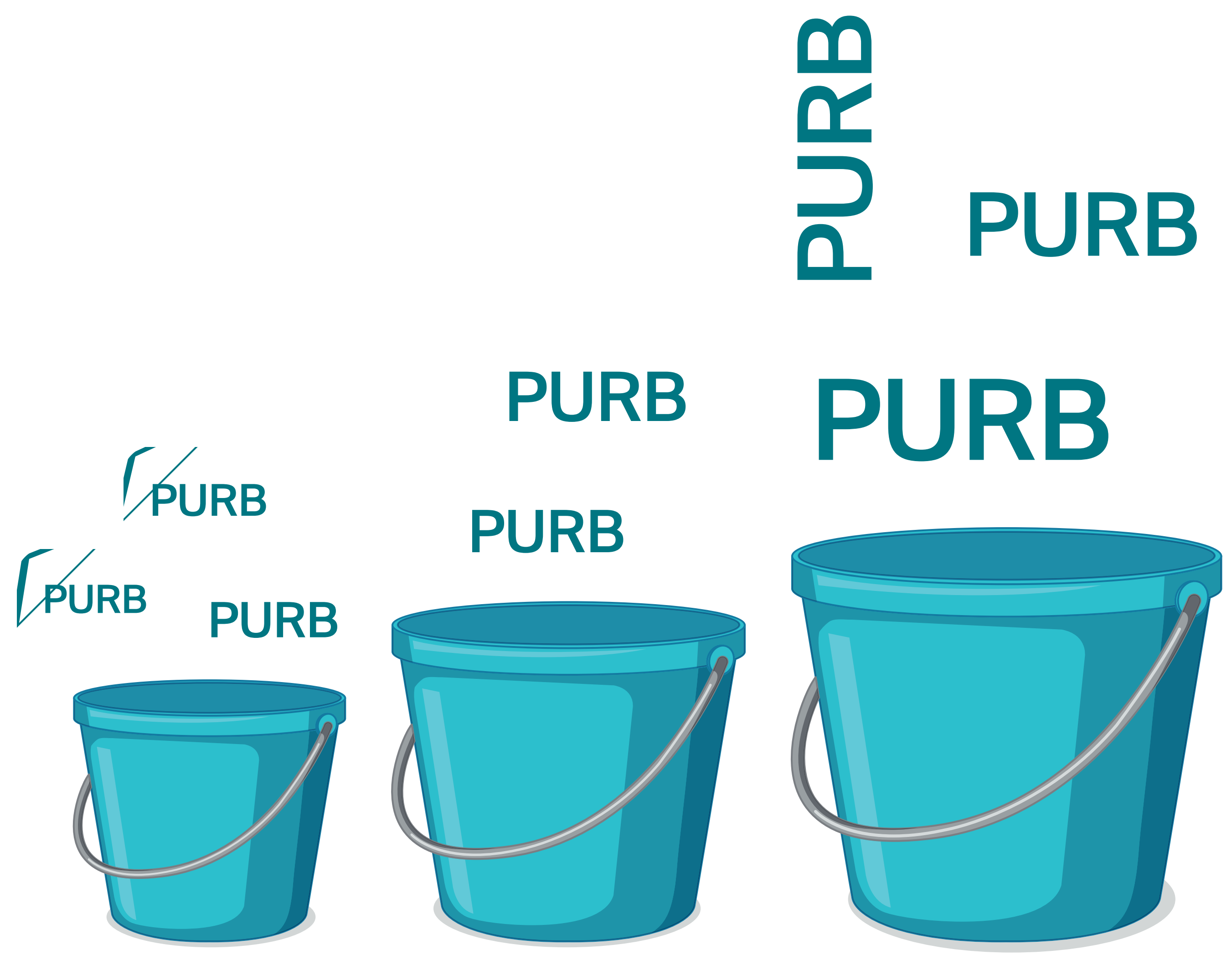
Encoding and Decoding of PURBs



Encoding and Decoding of PURBs



Padmé: reducing leakage about the size

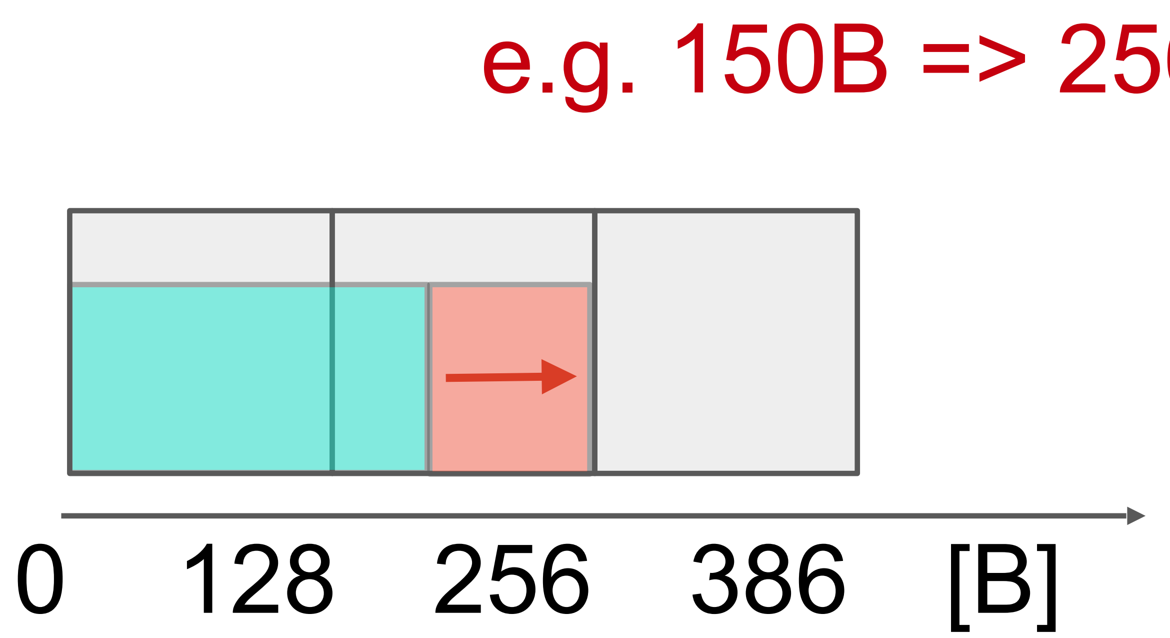


Padmé

- The total size is an important metadata, used in many attacks:
  - Website Fingerprinting
  - Traffic-Analysis
  - Attacks against HTTPS

- Design a padding function to improve "size privacy"

Naïve approach: (constant) block-padding



Naïve approach: (constant) block-padding

Problem: no good value for block size

Example: b = 1 MB

512 B

17 GB

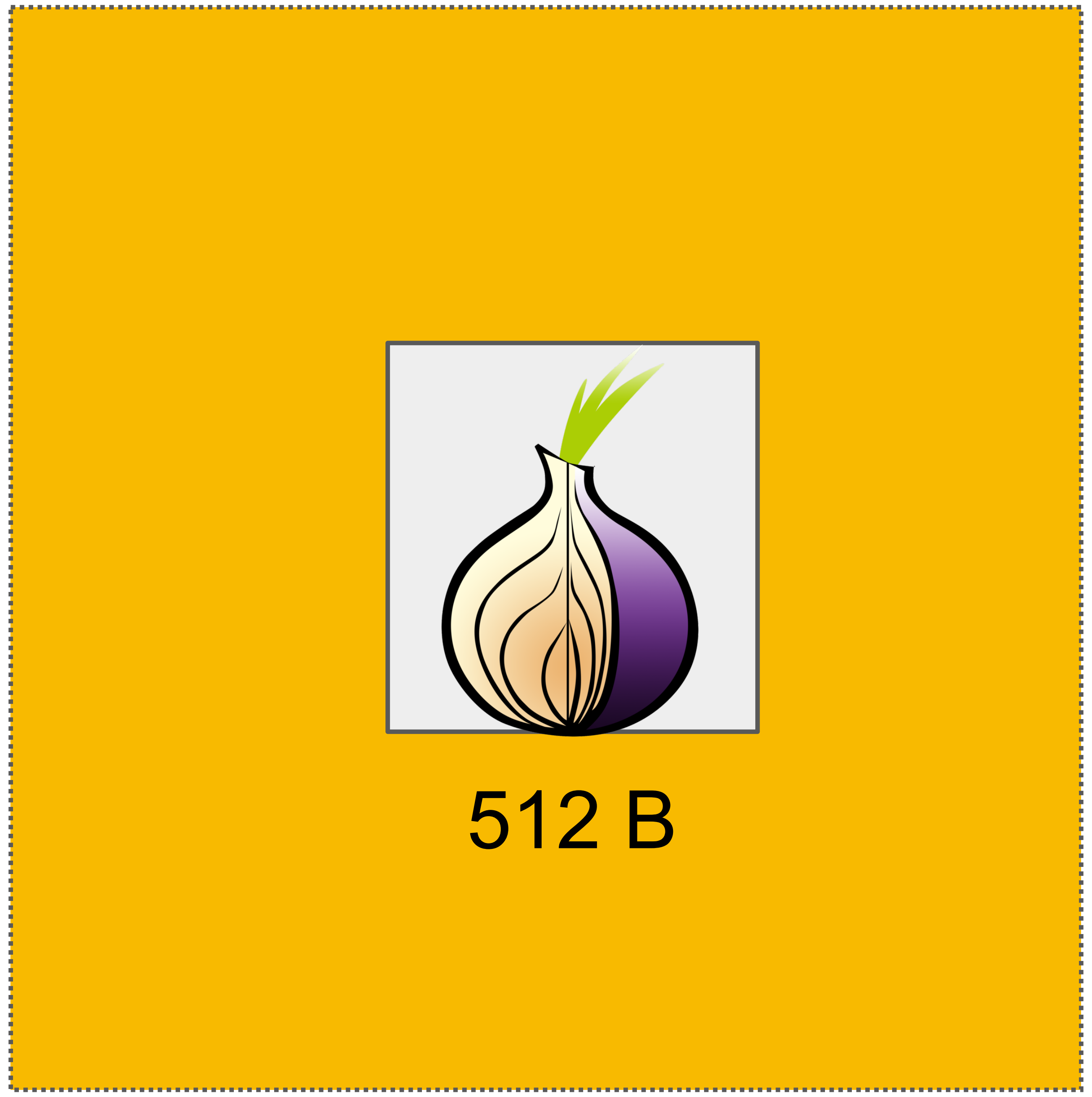
Ludovic Barman

Naïve approach: (constant) block-padding

Problem: no good value for block size

Example: b = 1 MB

little confusion / privacy

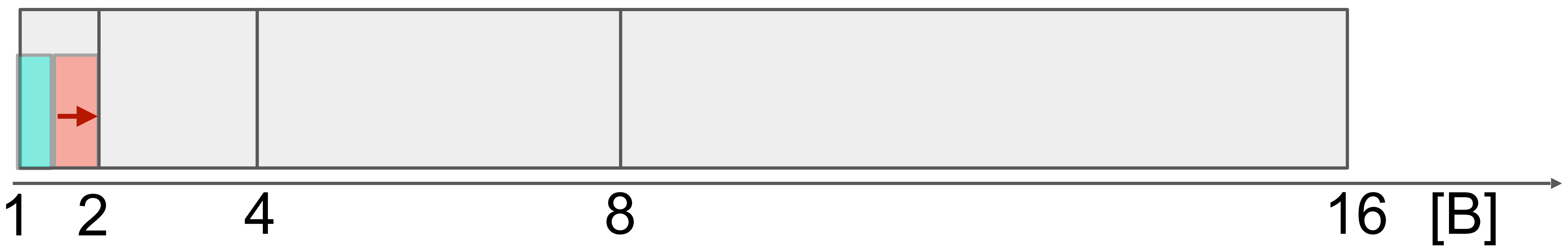


2000x overhead

Padding relative to the object size

Variable block size:

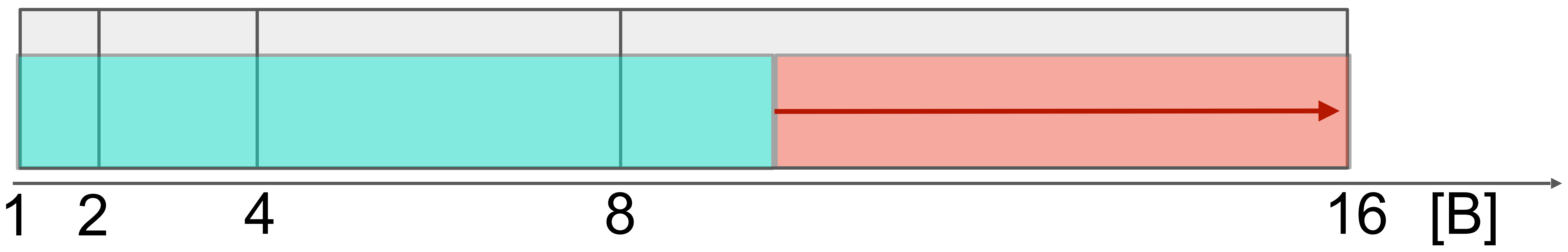
small objects: small overhead



Padding relative to the object size

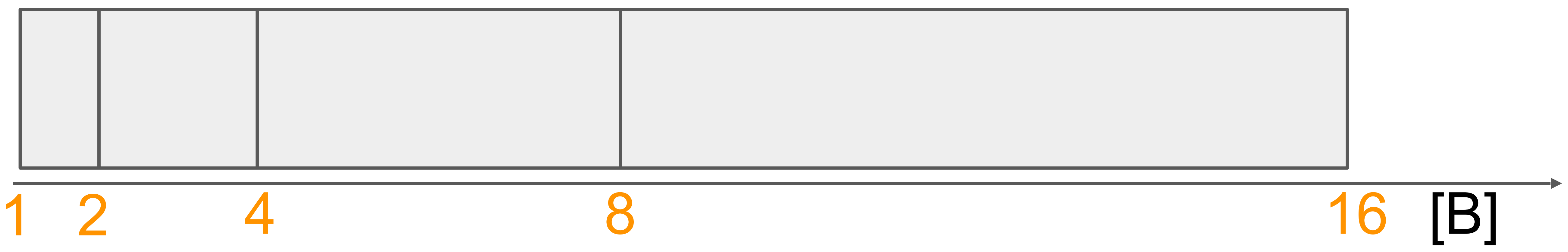
Variable block size:

large objects: larger privacy



Padding relative to the object size

Variable block size:



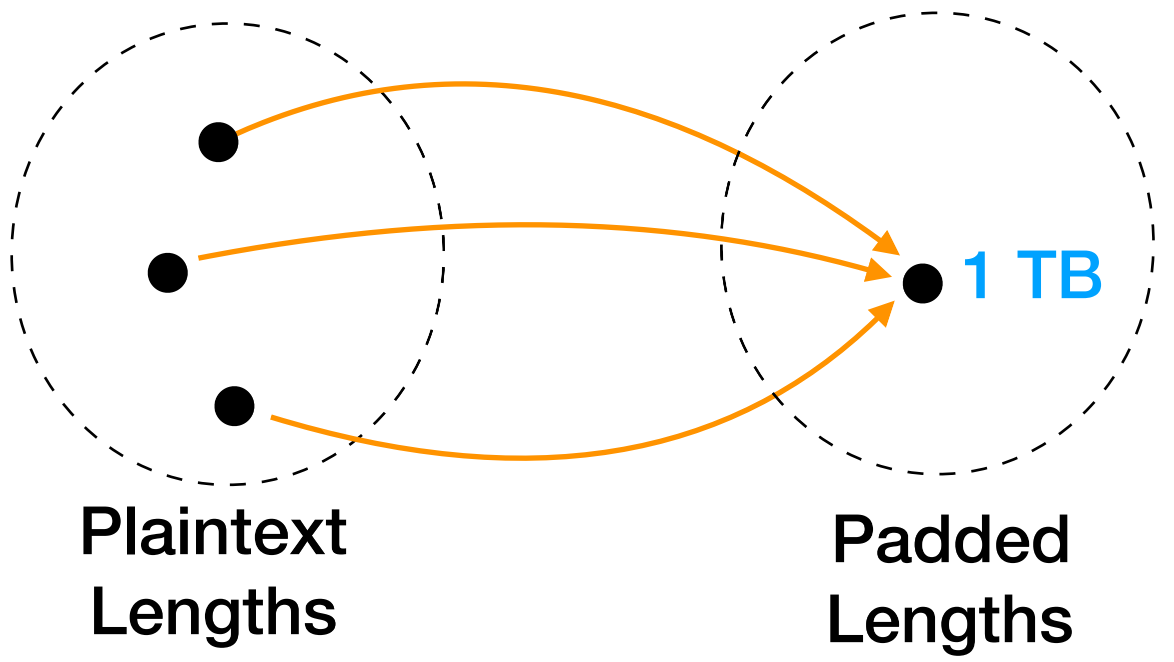
Padding to the "Next Power of 2"

Quantifying leakage of a padding function

- Let $f: \mathbb{N} \rightarrow \mathbb{N}$ be the padding function. Let C be the image set of f.

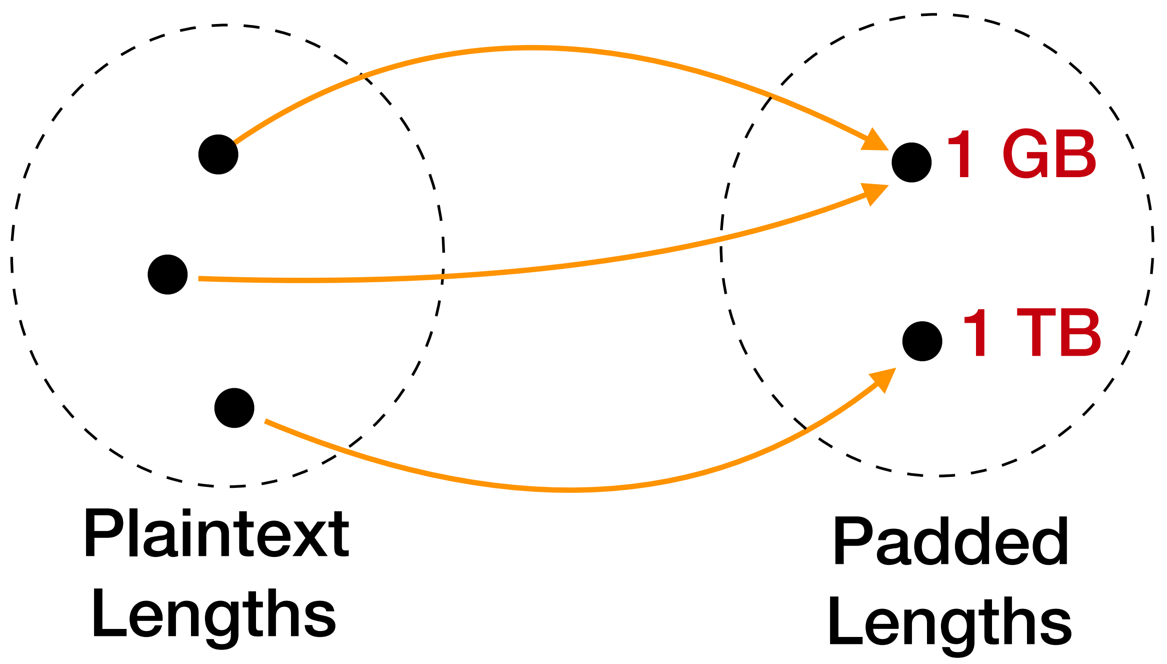
$$\boxed{\text{Leakage [bits]} = \log_2(\,|C|\,)}$$

$f(p) = 1\ \text{TB}$



Leakage: $\log_2(1) = 0$ bit

$$f(p) = \begin{cases} 1\ \text{GB} & p \leq 1\text{GB} \\ 1\ \text{TB} & \end{cases}$$



Leakage: $\log_2(2) = 1$ bit

Padding to the nearest power of 2

- Leakage: $O(\log \underbrace{\log(M)}_{\text{size of the image set}})$, where M is the biggest plaintext possible
- Much better than with constant block-size, which leaks O(log(M))
- Interestingly, not padding at all also leaks O(log(M))

- Max overhead: +100%
    - e.g., 16.1 GB => 32 GB

Reducing the overhead of padding

Next power of 2: Blocks have the form 2^0, 2^1, 2^2, etc. Represent them like floating points:

$$\underbrace{\boxed{\quad\text{Exponent}\quad}}_{\log_2(M)\text{ bits}}\underbrace{\boxed{\quad\text{Mantissa}\quad}}_{\text{all 0's}}$$

Padmé: Instead of 0's, allow *some* values in the mantissa:

=> Smaller blocks => Smaller overhead 

Padmé

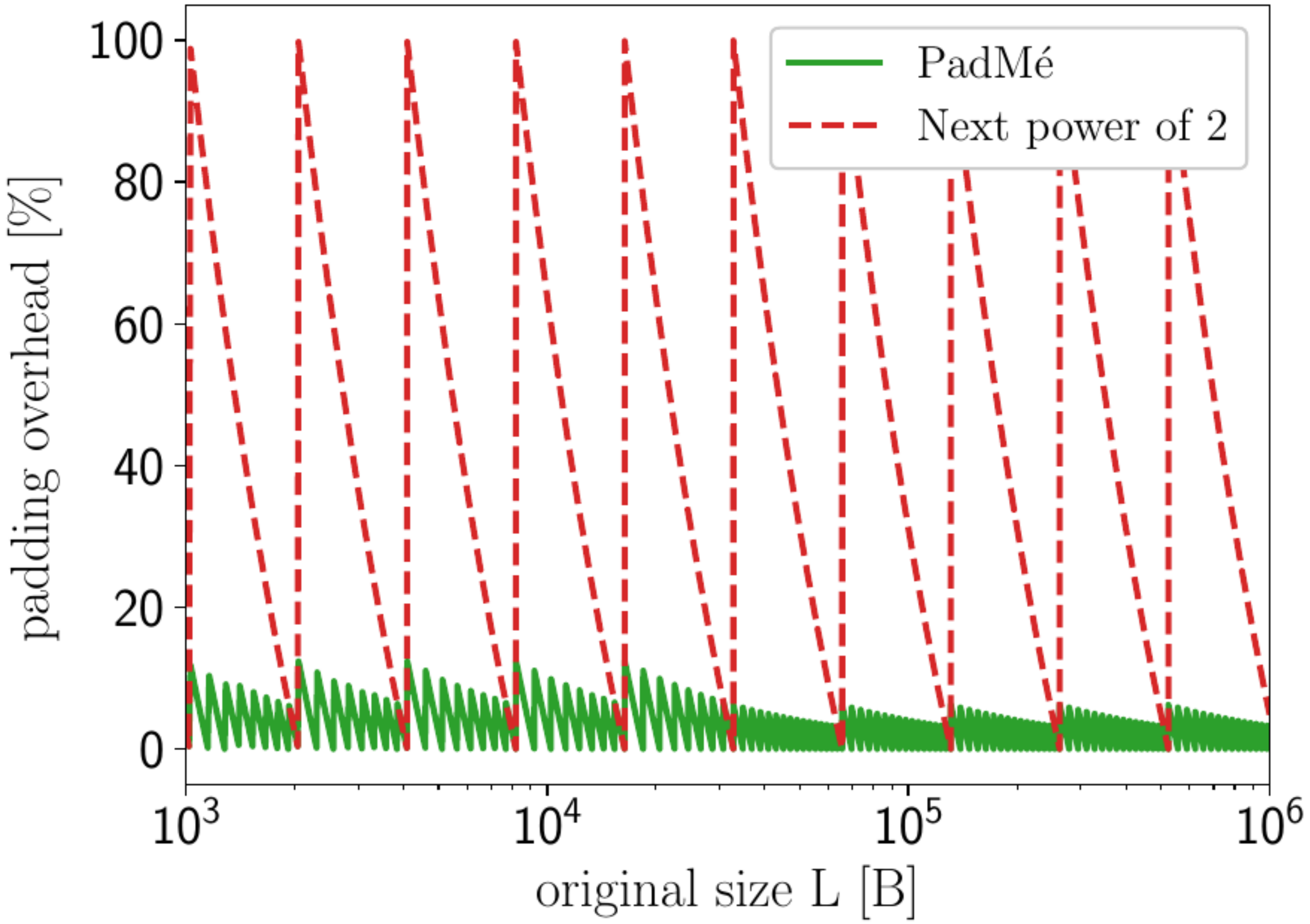
Pad to the next length L which respects:

Exponent	Mantissa
$\log_2(L)$ bits	$\leq \log_2(\log_2(L))$ bits

Intuition: the exponent can be anything, but the mantissa cannot be "too precise"

Doubles leakage => still in O(log log(M)) 

Padmé's overhead



$$\text{max overhead} = \frac{1}{2 \log_2 L} \%$$

Slowly decreases with L

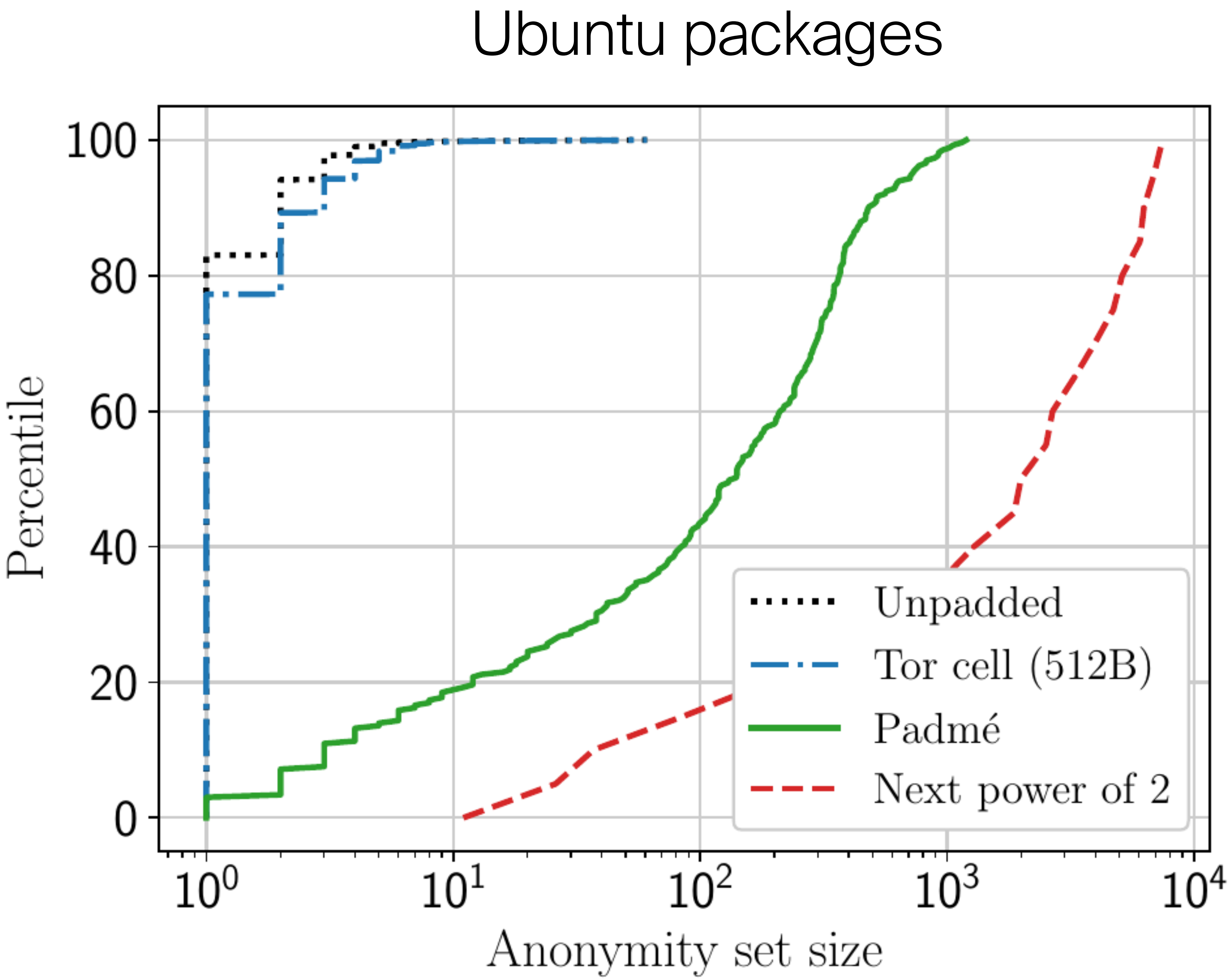
Max 12% ∀ L

Max ~6% for L ≥ ~1 MB

Max ~3% for L ≥ ~1 GB

EPFL

Padmé's "size privacy"



57'000 objects collected from apt lists

Mean overhead:

Next power of 2: **+44%**

Padmé: **+2.3%**

Ludovic Barman

Conclusion

- Padded Uniform Random Blobs (PURBs): ciphertext format with no metadata leakage except length, which is minimized.
- Encoding + Padding schemes.
- Applications: Email, Group Chat, Disk Encryption, Initiation of Protocols.
- To the best of our knowledge, the first video with pets @ PETS.

https://purbs.net

https://github.com/dedis/purb

kirill.nikitin@epfl.ch 👨🏼‍🏫
@ni_kirill

ludovic.barman@epfl.ch 👨🏻‍🏫
@lbarman_ch