Integrity and Metadata Protection in Data Retrieval

Kirill Nikitin

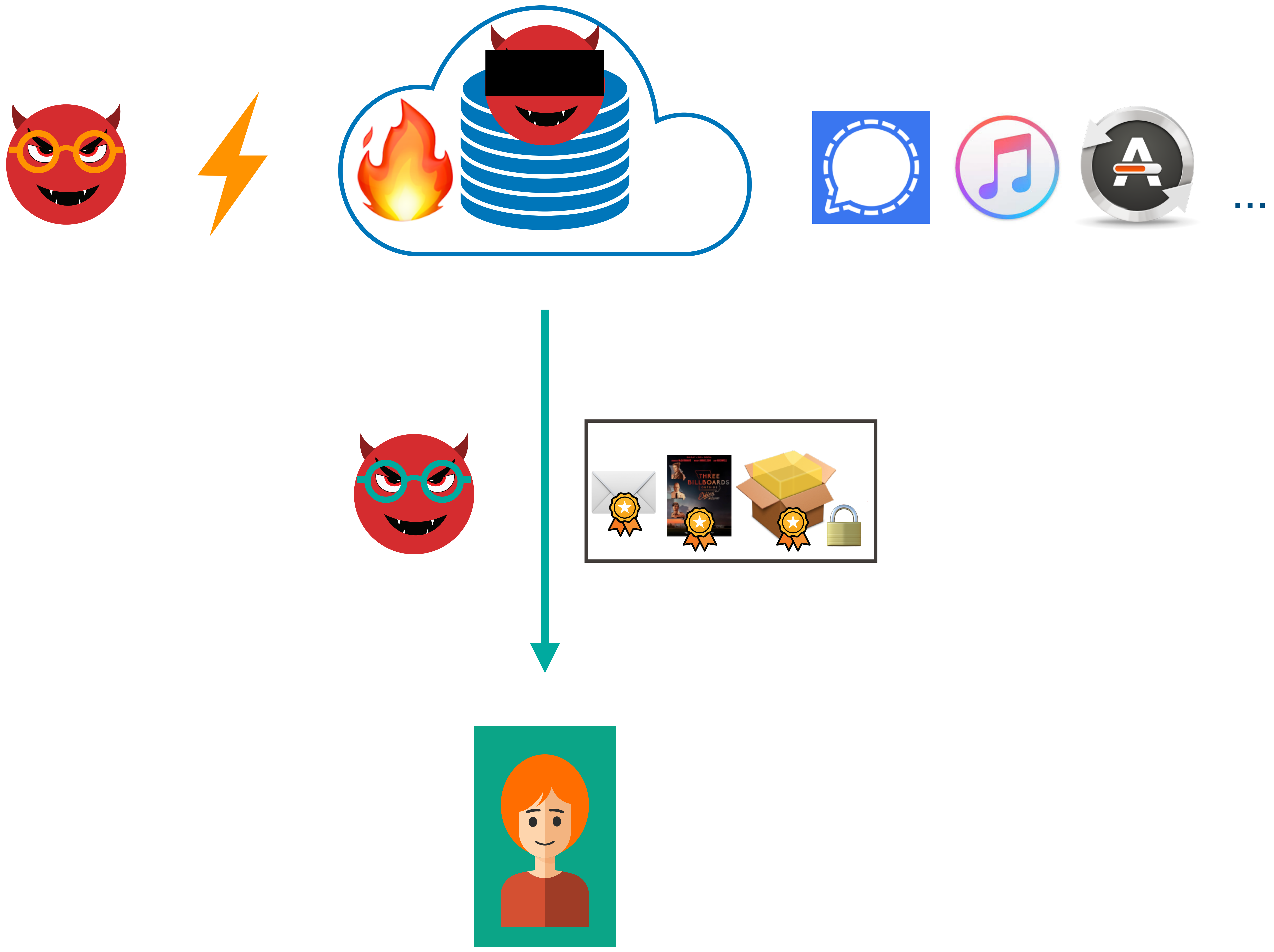
Decentralized and Distributed Systems Laboratory

PhD oral exam, 20.07.2021

Jury President:	Prof. Jean-Pierre Hubaux
Thesis Director:	Prof. Bryan Ford
Examiners:	Prof. Katerina Argyraki
	Prof. Justin Cappos
	Prof. Srdjan Capkun

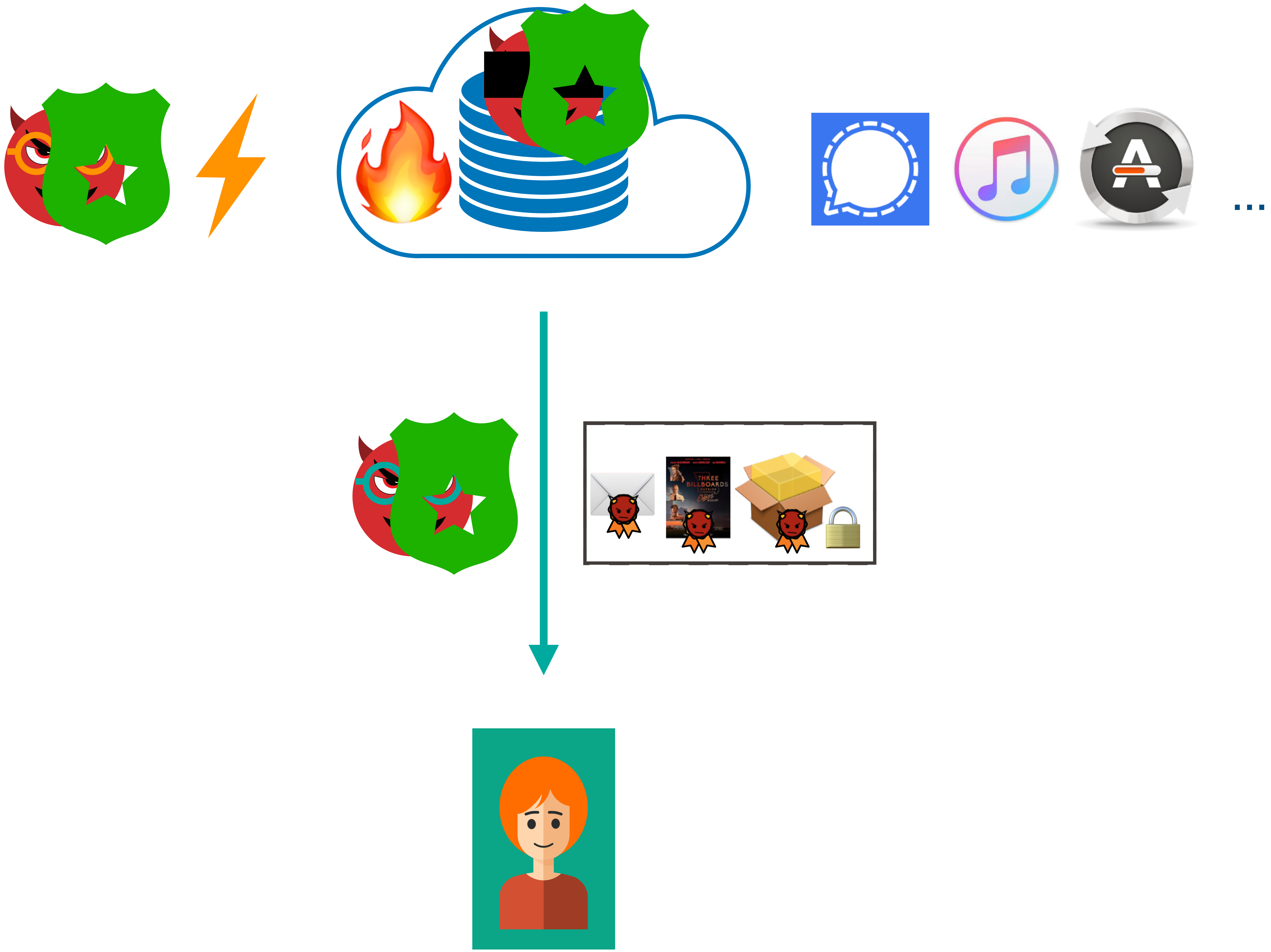
Kirill Nikitin

Users retrieve data all the time



Credit for the user and evil pictures here and graphics afterwards is to vecteezy.com

Current protection mechanisms do not suffice

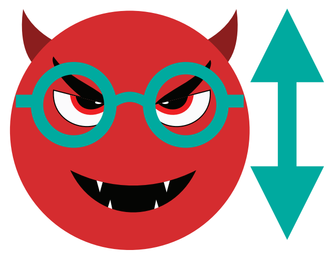


Credit for the user and evil pictures here and graphics afterwards is to vecteezy.com

This thesis

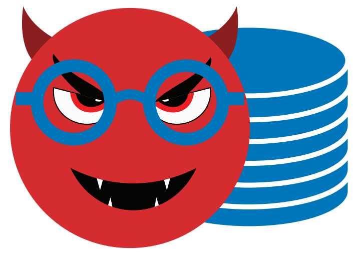
On-the-network attacker

- Protecting encryption metadata (Chapter 2) [1]



Malicious provider

- Data integrity in single-server private information retrieval (Chapter 3) [2]



Compromised provider

- Securing retrieval of software updates (Chapter 4) [3]

[1] K. Nikitin*, L. Barman*, W. Lueks, M. Underwood, J.-P. Hubaux, and B. Ford, "Reducing Metadata Leakage from Encrypted Files and Communication with PURBs", PETS 2019.

[2] S. Colombo*, K. Nikitin*, B. Ford, and H.Corrigan-Gibbs, "Verifiable Private Information Retrieval", Under submission.

[3] K. Nikitin, E. Kokoris-Kogias, P. Jovanovic, N. Gailly, L. Gasser, I. Khoffi, J. Cappos, and B. Ford, "CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds", USENIX Security 2017.

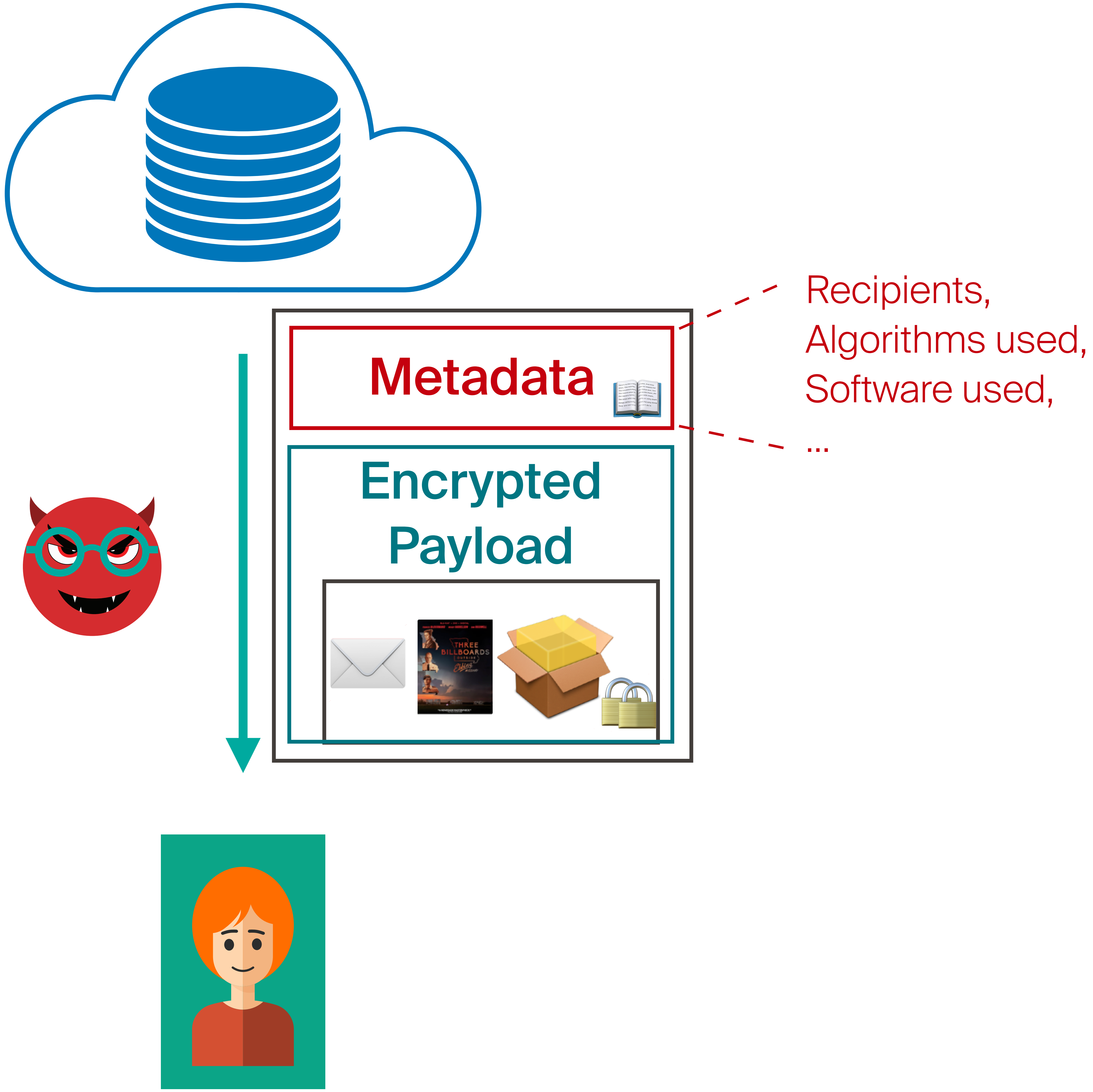
Roadmap

- ❖ Introduction
- ❖ Protecting encryption metadata (Chapter 2)
- ❖ Data integrity in single-server PIR (Chapter 3)
- ❖ Securing retrieval of software updates (Chapter 4)
- ❖ Conclusion

Roadmap

- Introduction
- Protecting encryption metadata (Chapter 2)
- Data integrity in single-server PIR (Chapter 3)
- Securing retrieval of software updates (Chapter 4)
- Conclusion

Metadata exposure in ciphertexts



OpenPGP Packet Format

```
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
```

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 encryption metadata necessary?

An OpenPGP message to Martin Vetterli encrypted with RSA-512 using an outdated format??

Small key? Outdated format? I might crack it!

Avoiding metadata leakage

- Can we design an application-level ciphertext format that avoids leakage of encryption metadata?
- Encryption metadata concretely:
  - The ciphertext's intended recipients
  - The encryption algorithm used
  - What application has produced the ciphertext
  - …

Kirill Nikitin

What If We Stripped Off All the Metadata?

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

What If We Stripped Off All the Metadata?

Encrypt the metadata instead!

```
                                          0f ff46
bd7f 1821 27a9 42c4 01b4 7ecd 433e 7f90
. . . . .  74b8 139c a802 6678 ba0d 1abd
---------------------------------------
          4b 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

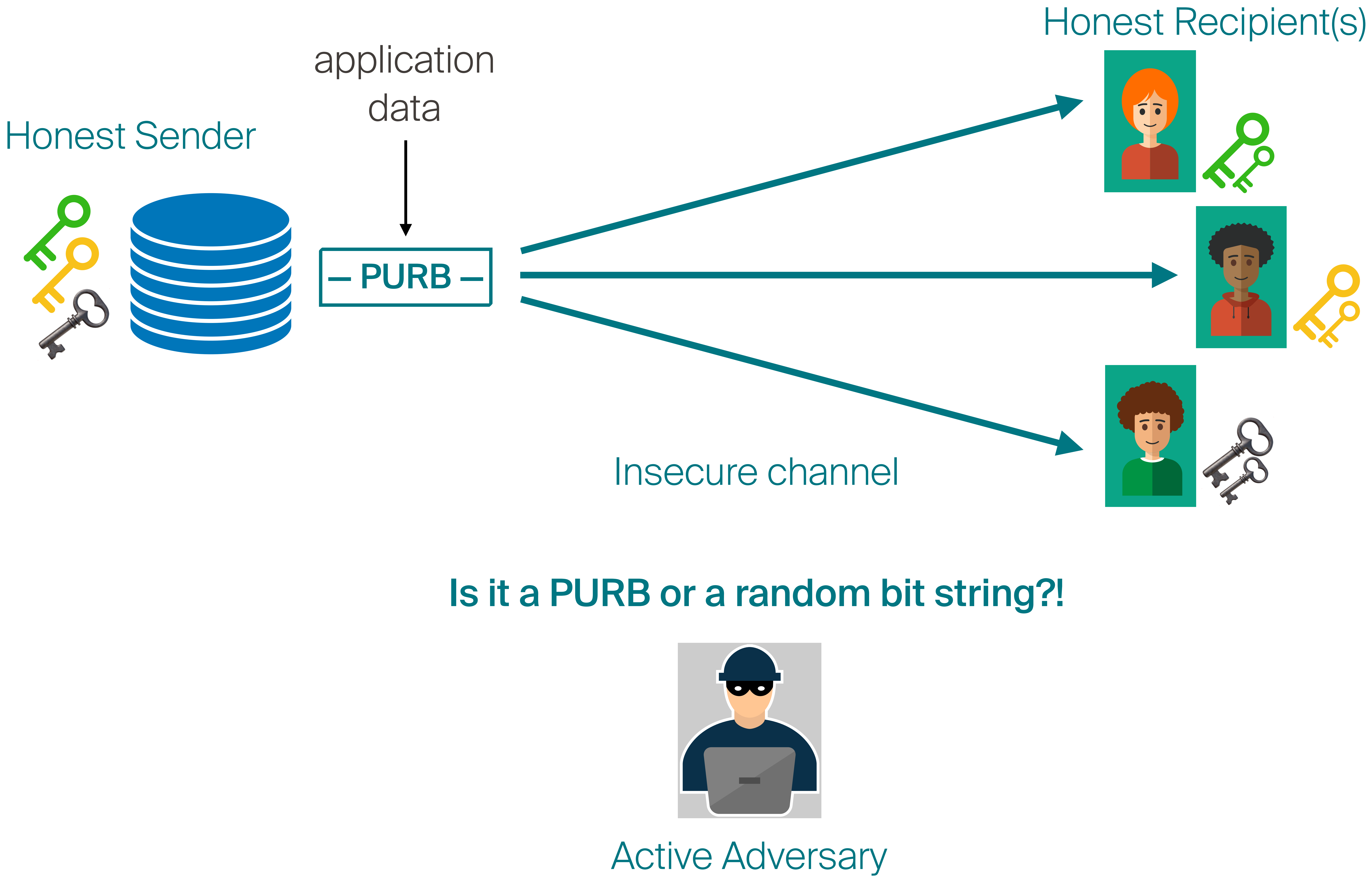
Encrypted Data →

- How does a recipient parse a ciphertext without any auxiliary information?
- What if the ciphertext is encrypted
  - To multiple recipients
  - By using multiple cryptographic algorithms

Padded Uniform Random Blobs (PURBs)

- A ciphertext format for application data without *any* metadata in clear
- The metadata can be found efficiently by trial decryptions following predefined logic
- Generic, i.e., still works efficiently with a large number of recipients and encryption algorithms used
- A PURB must be indistinguishable from a random bit string (IND$-CCA2)

Model



Kirill Nikitin

Data-encapsulation strawman

Similar to the Integrated Encryption Scheme [ABR01] (DH-based)

Recipient – public key g^y

pk	payload	MAC
$c_1 = g^x$	$c_2 = Enc_{g^{yx}}(data)$	$MAC_{g^{yx}}(c_1 \| c_2)$

↑

distinguishable from a random string

Data-encapsulation strawman

- The encoded public key is indistinguishable from a uniform random string
- Public encoding algorithms, e.g., Elligator [BHKL13], for different public-key types which all produce uniform strings

encoded pk	payload	MAC
$c_1 = \text{Hide}(g^x)$	$c_2 = \text{Enc}_{g^{yx}}(\text{data})$	$\text{MAC}_{g^{yx}}(c_1 \,\|\, c_2)$

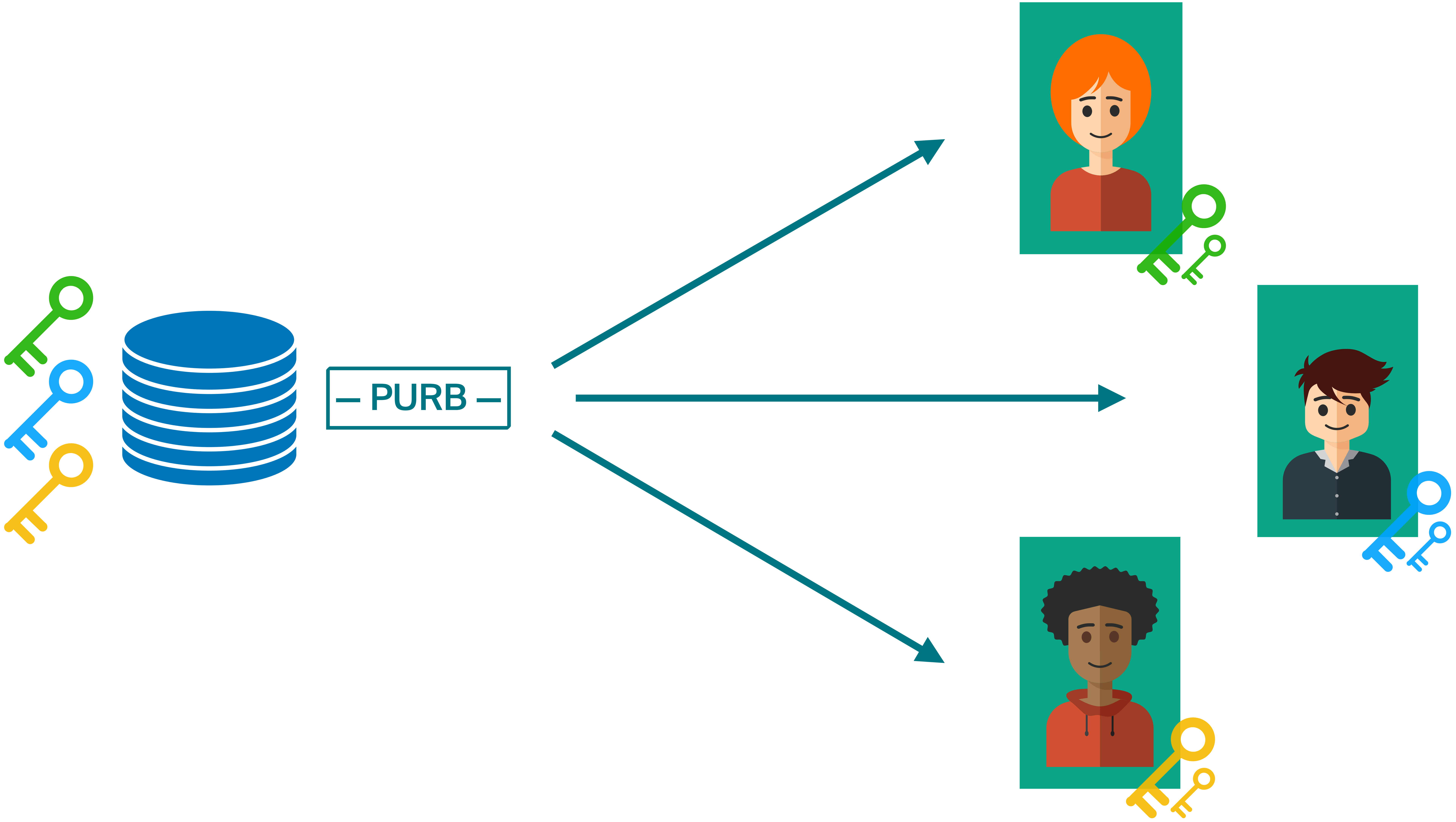
1. **Does not scale to multiple recipients (e.g., the issue of data duplication)**
2. **Does not accommodate multiple cryptographic algorithms**

Entry points

- The data are encrypted with an one-time session key K
- An *entry point* per recipient stores K and additional metadata, and signals the correctness of decryption

encoded pk	entry point	payload	MAC
$c_1 = \mathrm{Hide}(g^x)$	$c_2 = \mathrm{AE}_{g^{yx}}(K \,\|\|\, \mathrm{meta})$	$c_3 = \mathrm{Enc}_K(\mathrm{data})$	$\mathrm{MAC}_K(c_1\|\|c_2\|\|c_3)$

Multiple Recipients



Multiple Recipients

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

Sender creates an entry point (EP) per recipient, each with K and metadata but encrypted with g^{y1x}, g^{y2x}, g^{y3x} respectively.

EP_1	EP_2	EP_3
$AE_{g^{y1x}}(K\|\|meta)$	$AE_{g^{y2x}}(K\|\|meta)$	$AE_{g^{y3x}}(K\|\|meta)$

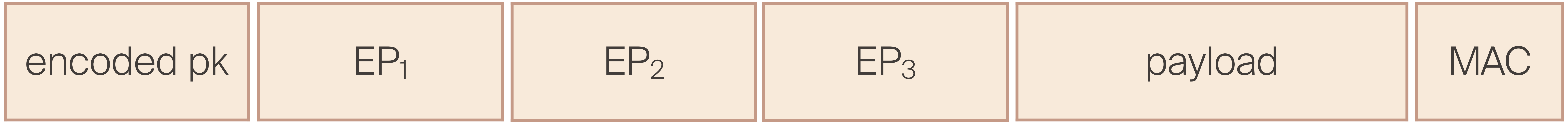
But how do we organize these entry points in the PURB?

Linear Approach Strawman

Entry points for the recipients – EP_1, EP_2, EP_3

Inefficient to decode

O(len(PURB))

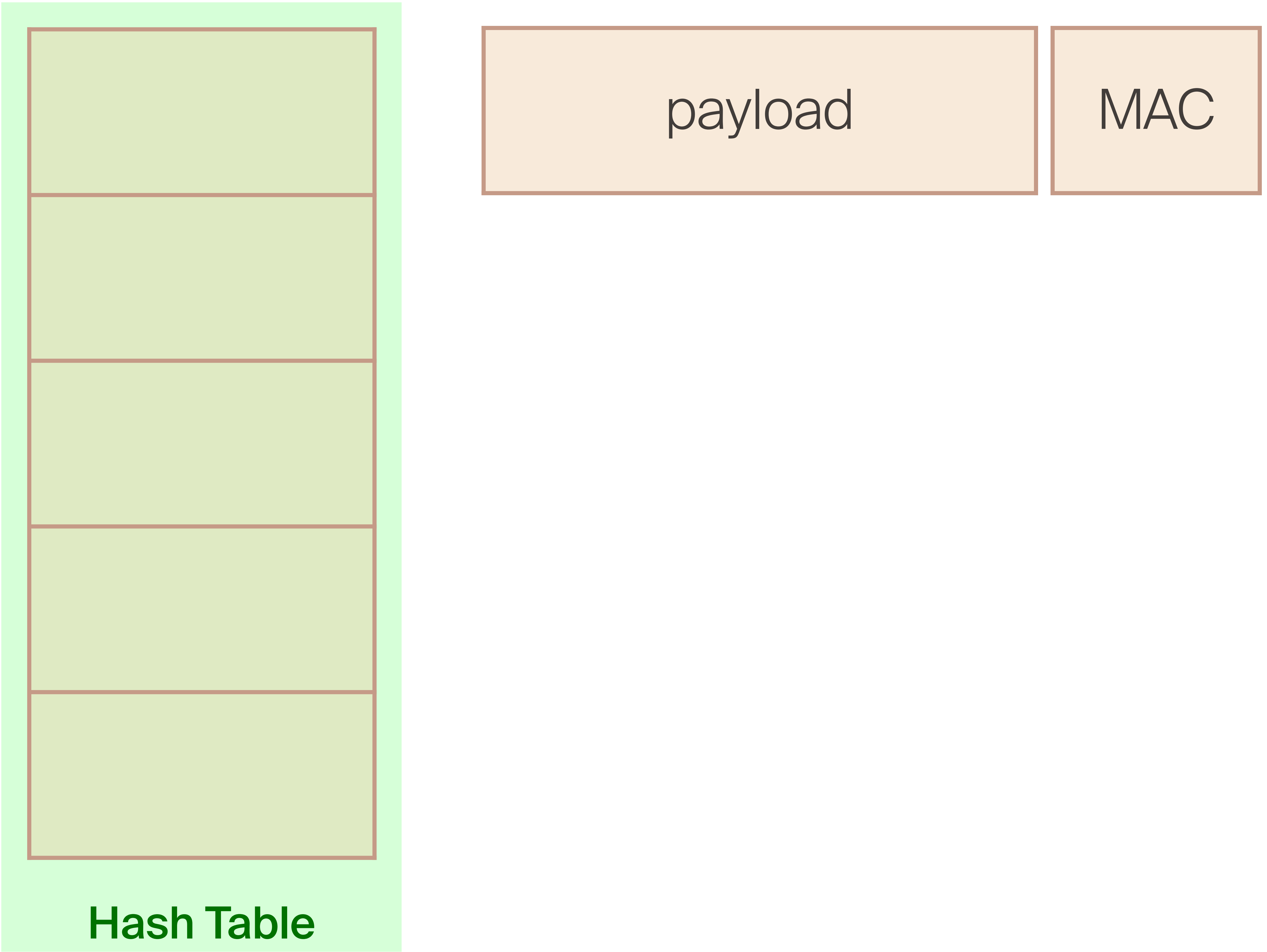


We create an entry point (EP) per recipient, each with K and metadata but encrypted with g^{y1x}, g^{y2x}, g^{y3x} respectively.

Similar to private broadcast encryption [BBW06]

Single Hash-Table Strawman

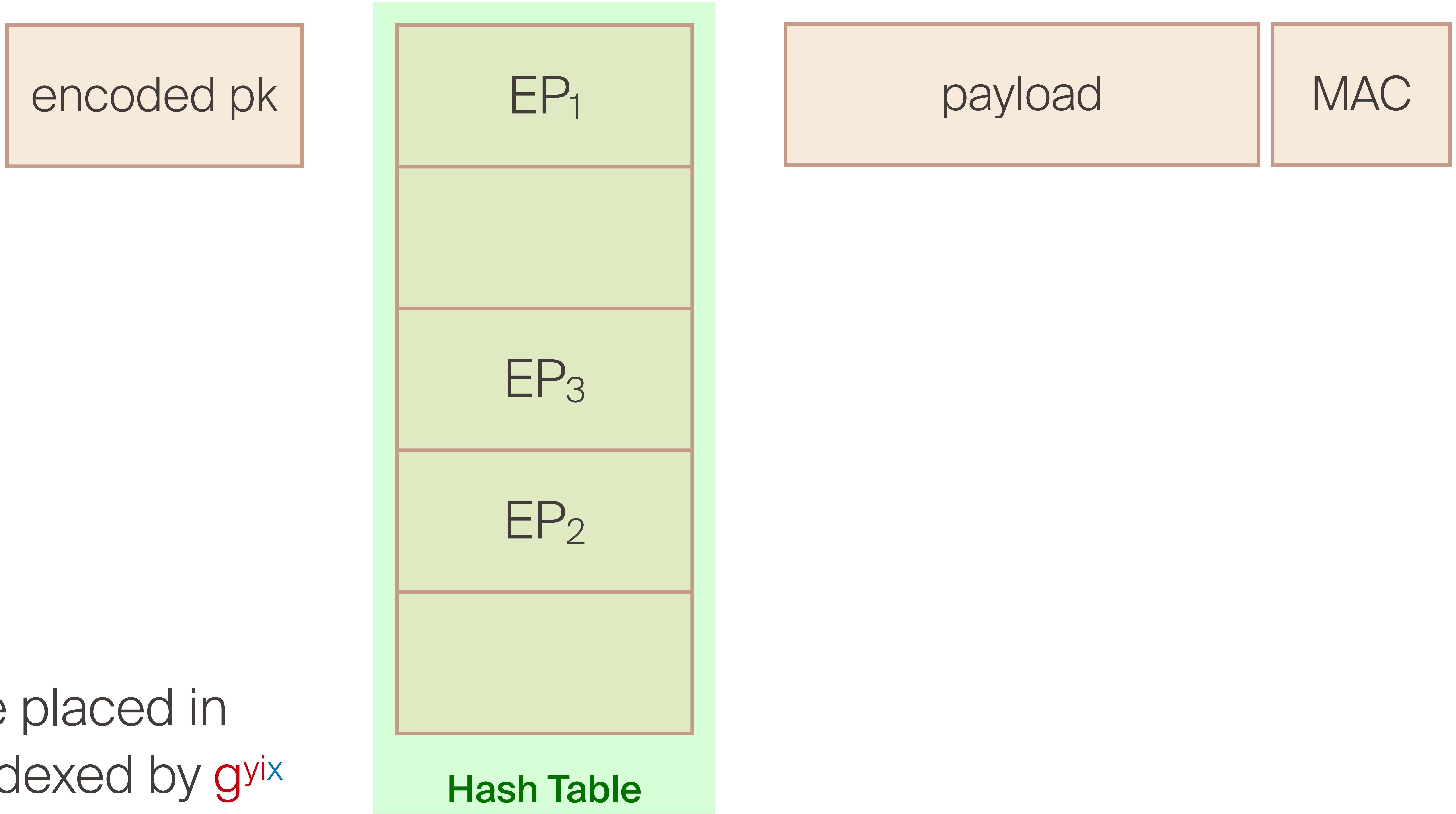
Entry points for the recipients – EP_1, EP_2, EP_3



Entry points are placed in a hash table, indexed by $g^{y_i x}$

Single Hash-Table Strawman

Entry points for the recipients – EP_1, EP_2, EP_3



Entry points are placed in a hash table, indexed by $g^{y_i x}$

Single Hash-Table Strawman

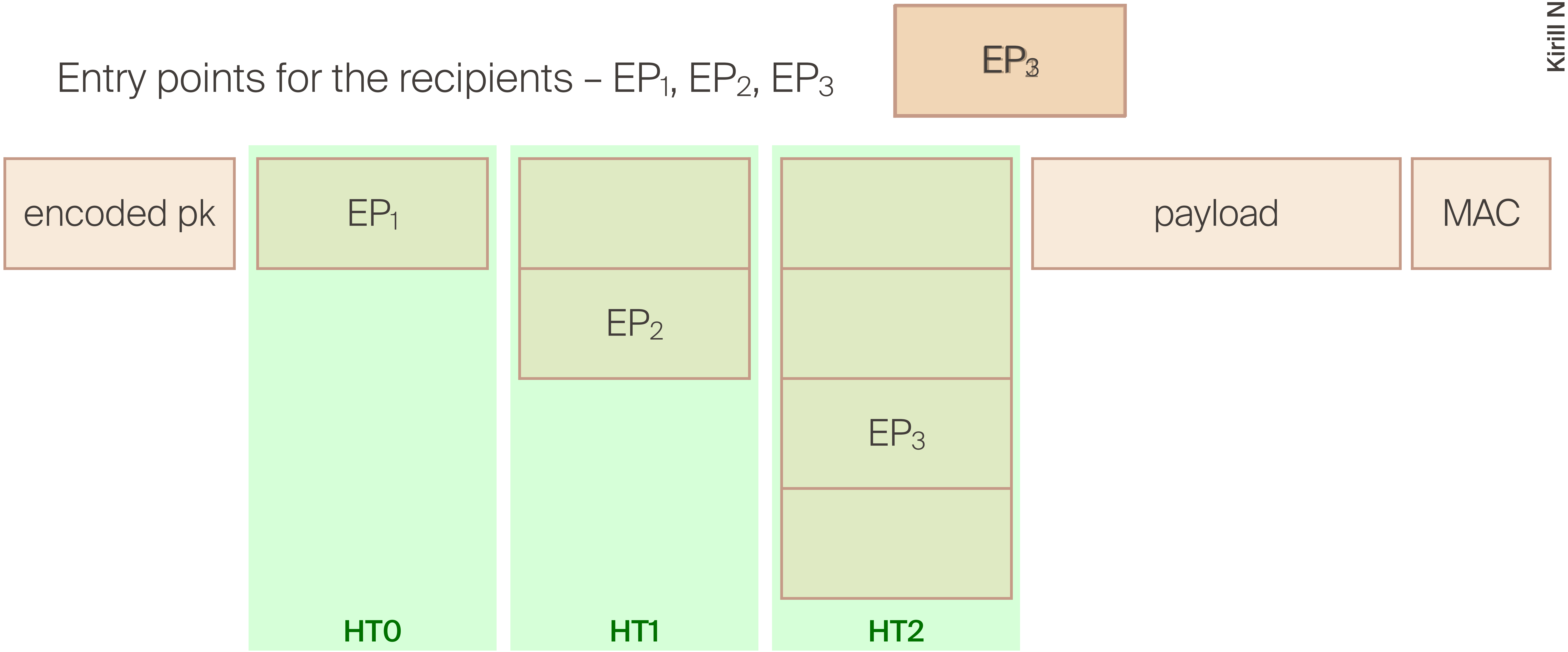
Entry points for the recipients – EP_1, EP_2, EP_3

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

Entry points are placed in a hash table, indexed by $g^{y_i x}$

Multiple Recipients: Our Solution

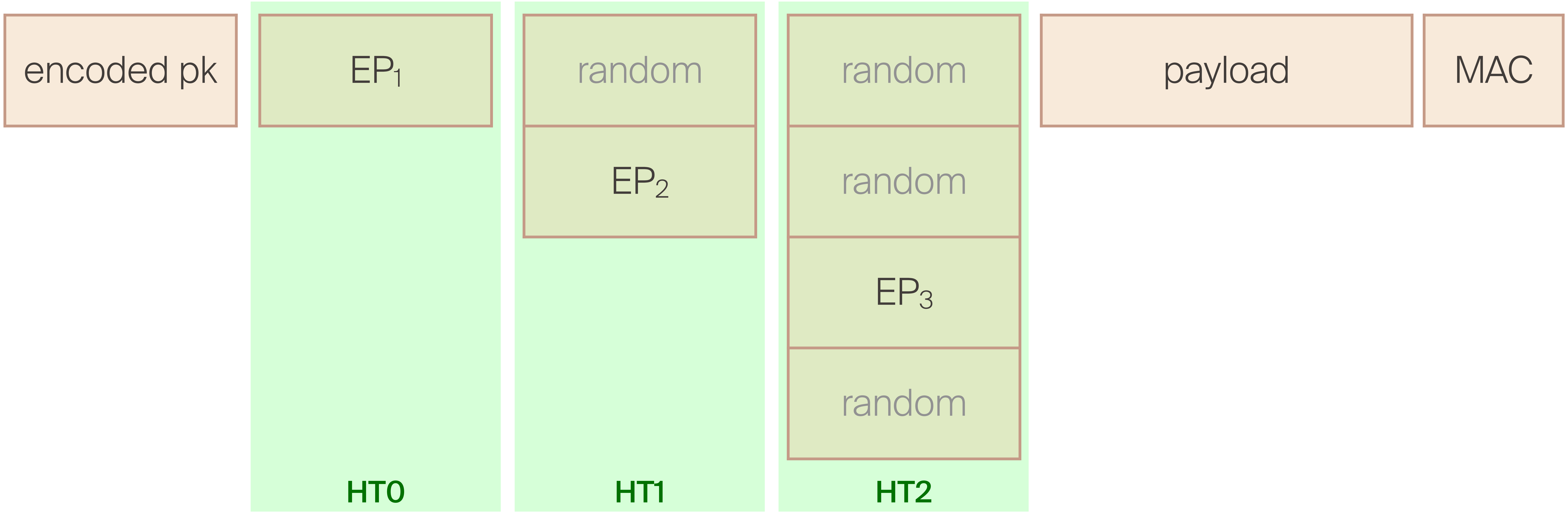
Entry points for the recipients – EP_1, EP_2, EP_3



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

Multiple Recipients: Our Solution

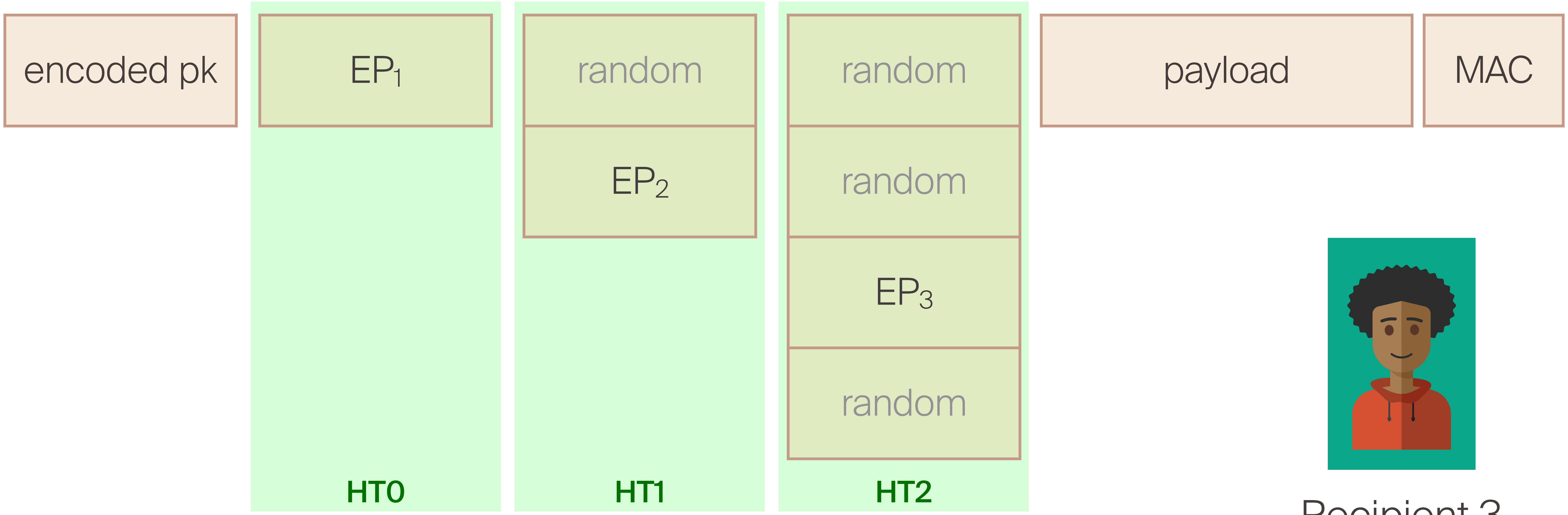
Entry points for the recipients – EP_1, EP_2, EP_3



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

Multiple Recipients: Decoding

Entry points for the recipients – EP_1, EP_2, EP_3



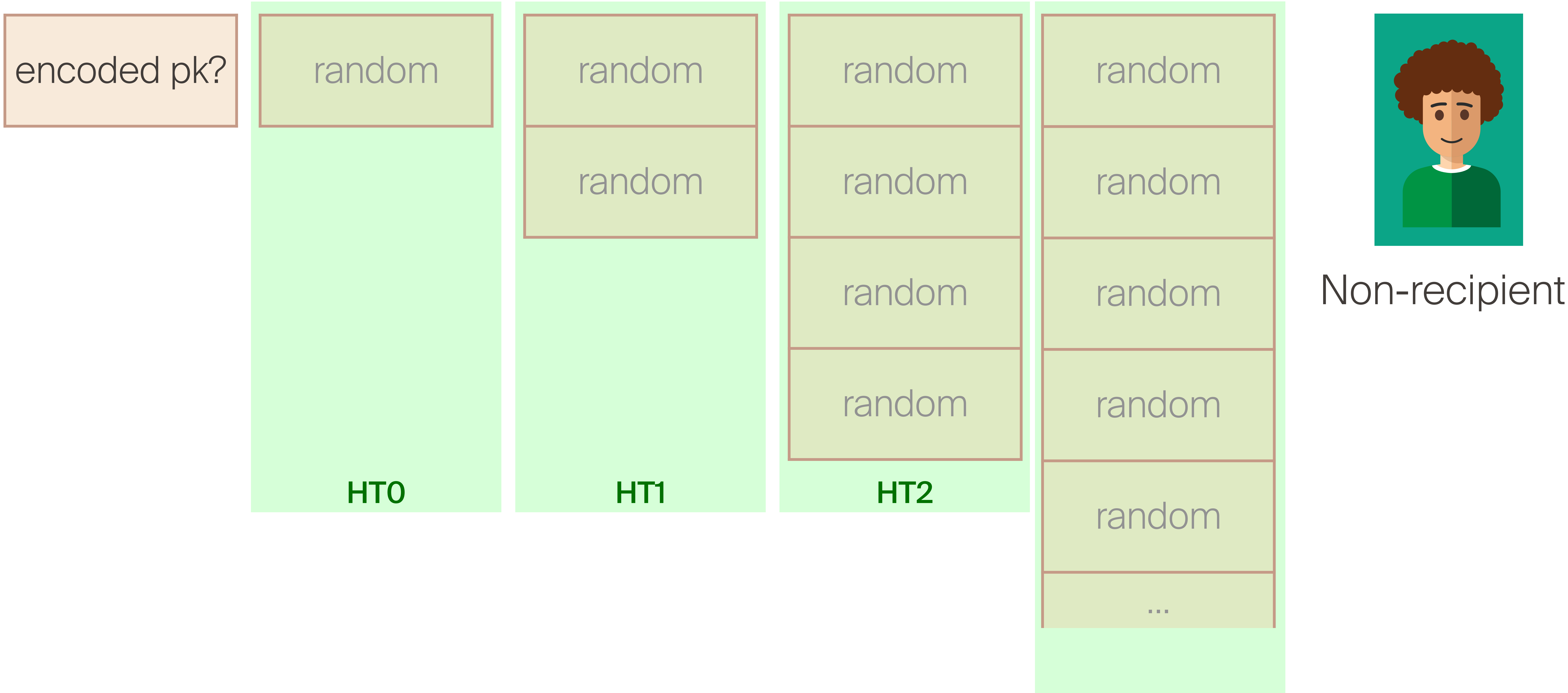
Recipient 3

Decoding in O(log len(PURB))

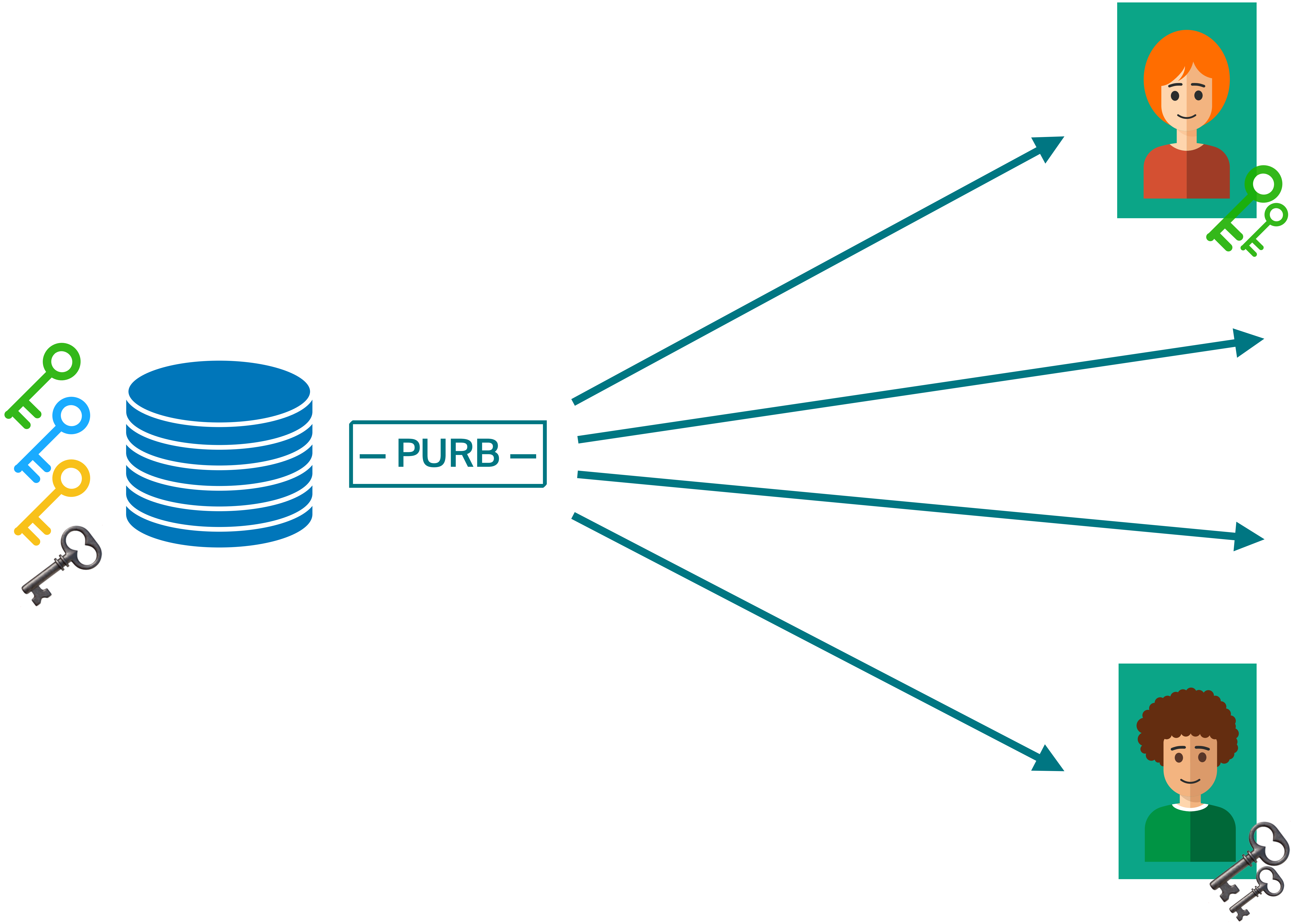
Entry points are placed in a series of growing hash-tables!

Multiple Recipients: Decoding

Entry points for the recipients – EP_1, EP_2, EP_3



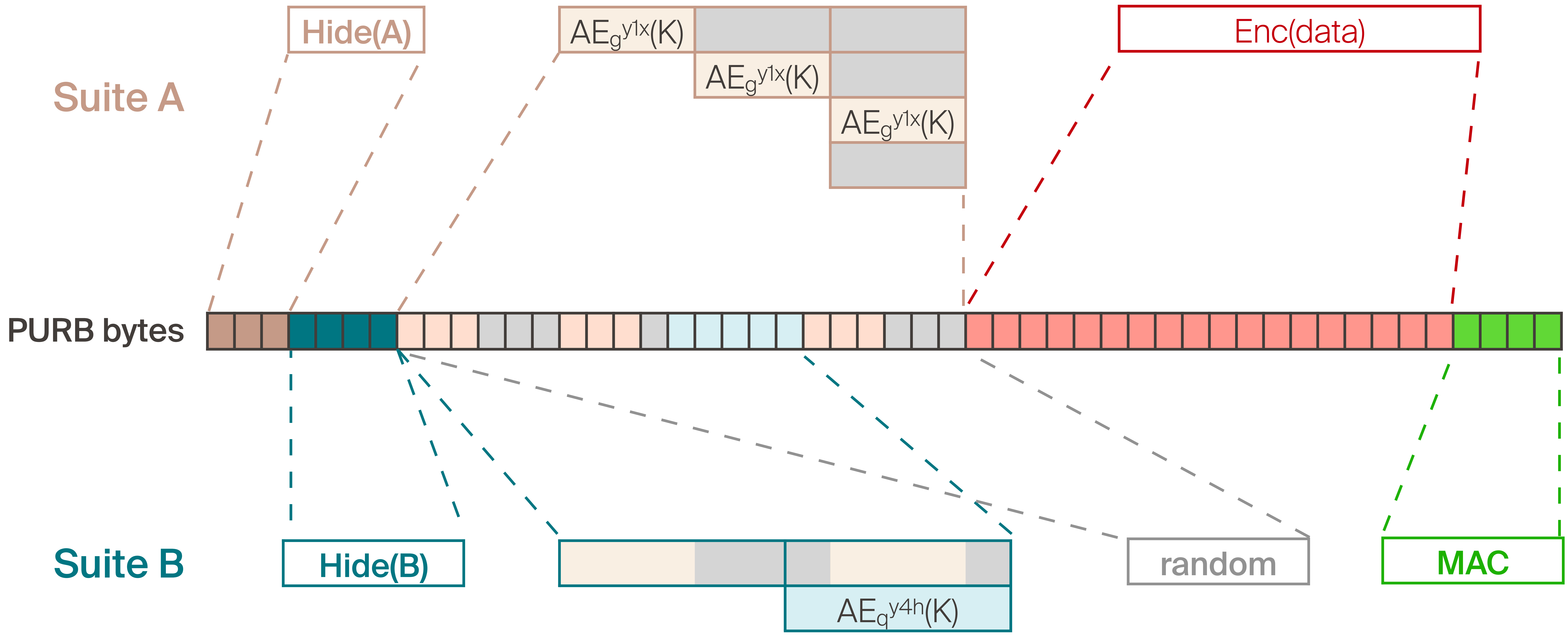
Multiple Suites



Multiple Suites

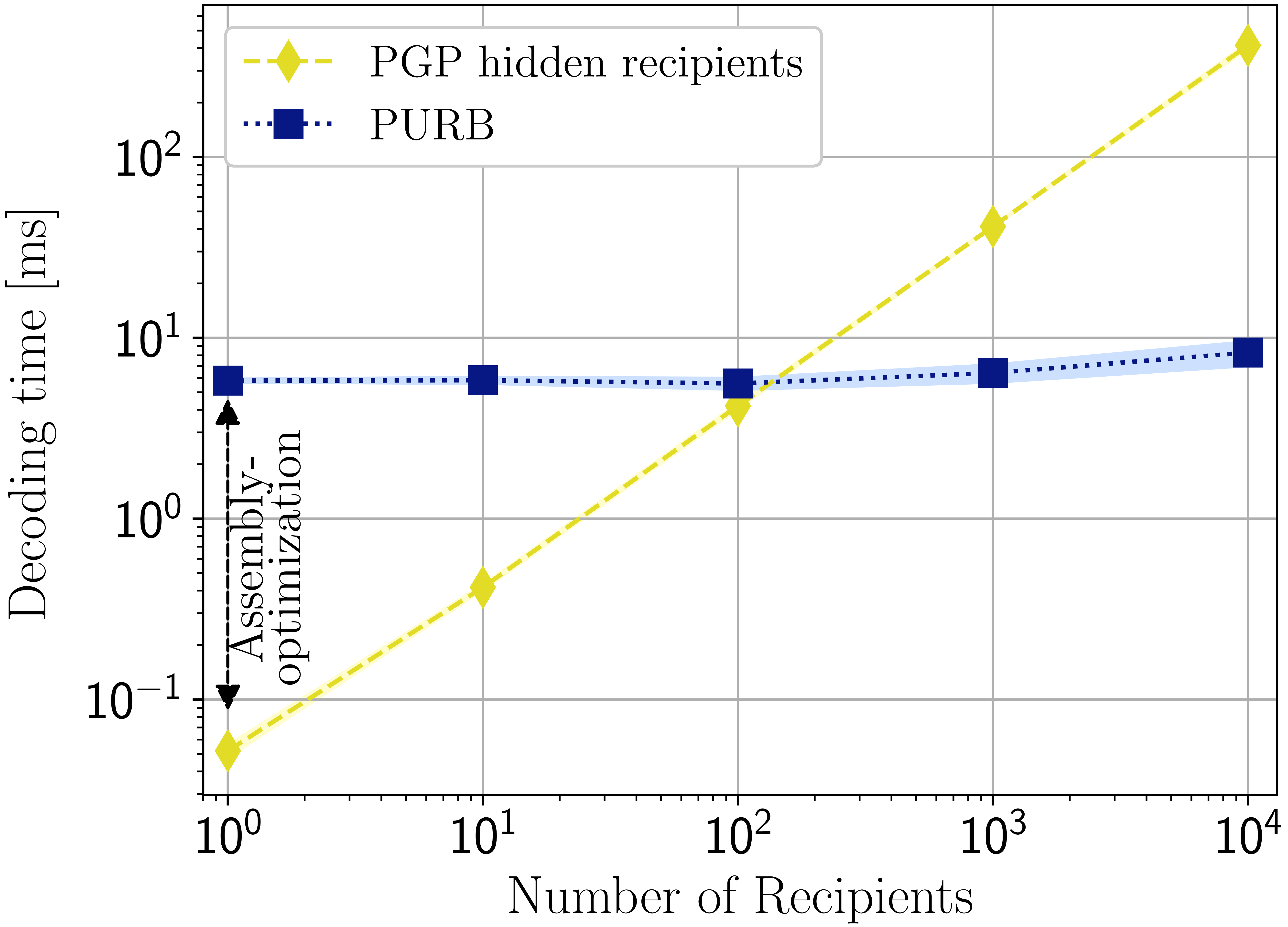
- Recipients use several distinct suites, based on public-key group (e.g., Curve25519 or Curve448) or entry point encryption.
- Each suite (an encoded public key and hash tables) becomes a distinct logical layer in a PURB, and these layers overlap!

Multi-suite PURB encoding



A recipient parses a multi-suite PURB in the same way as in the single-suite scenario!

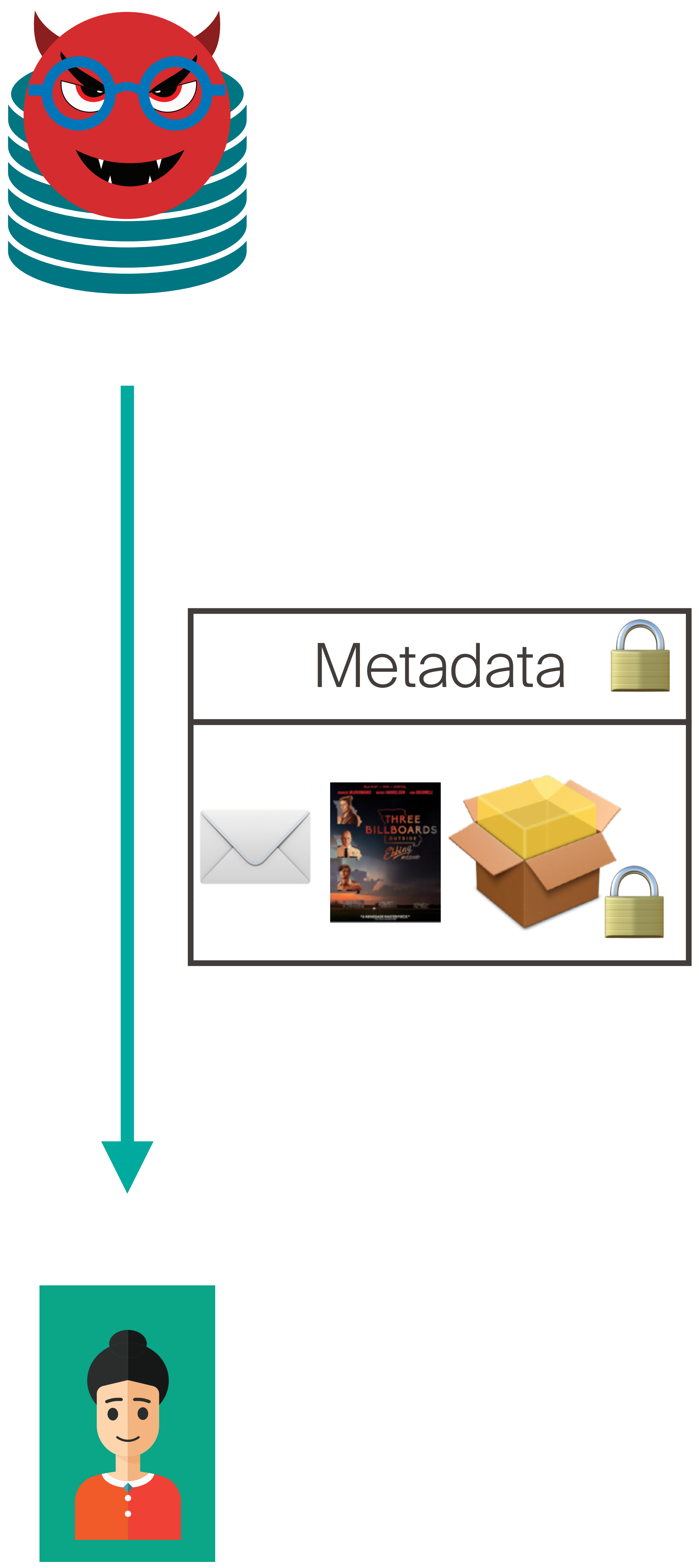
Evaluation of decoding performance



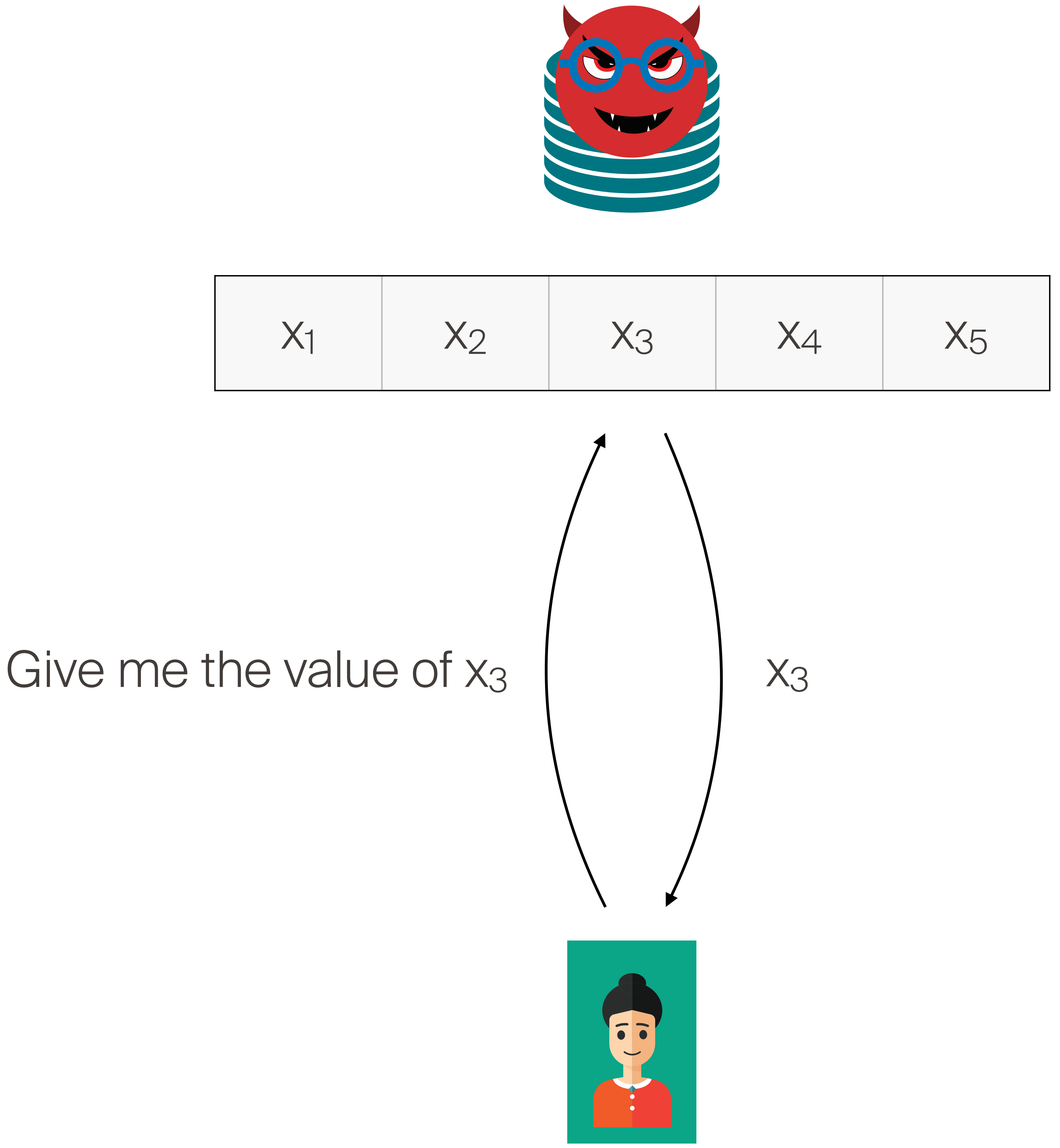
Roadmap

- Introduction
- Protecting encryption metadata (Chapter 2)
- Data integrity in single-server PIR (Chapter 3)
- Securing retrieval of software updates (Chapter 4)
- Conclusion

Service providers learn user’s choices

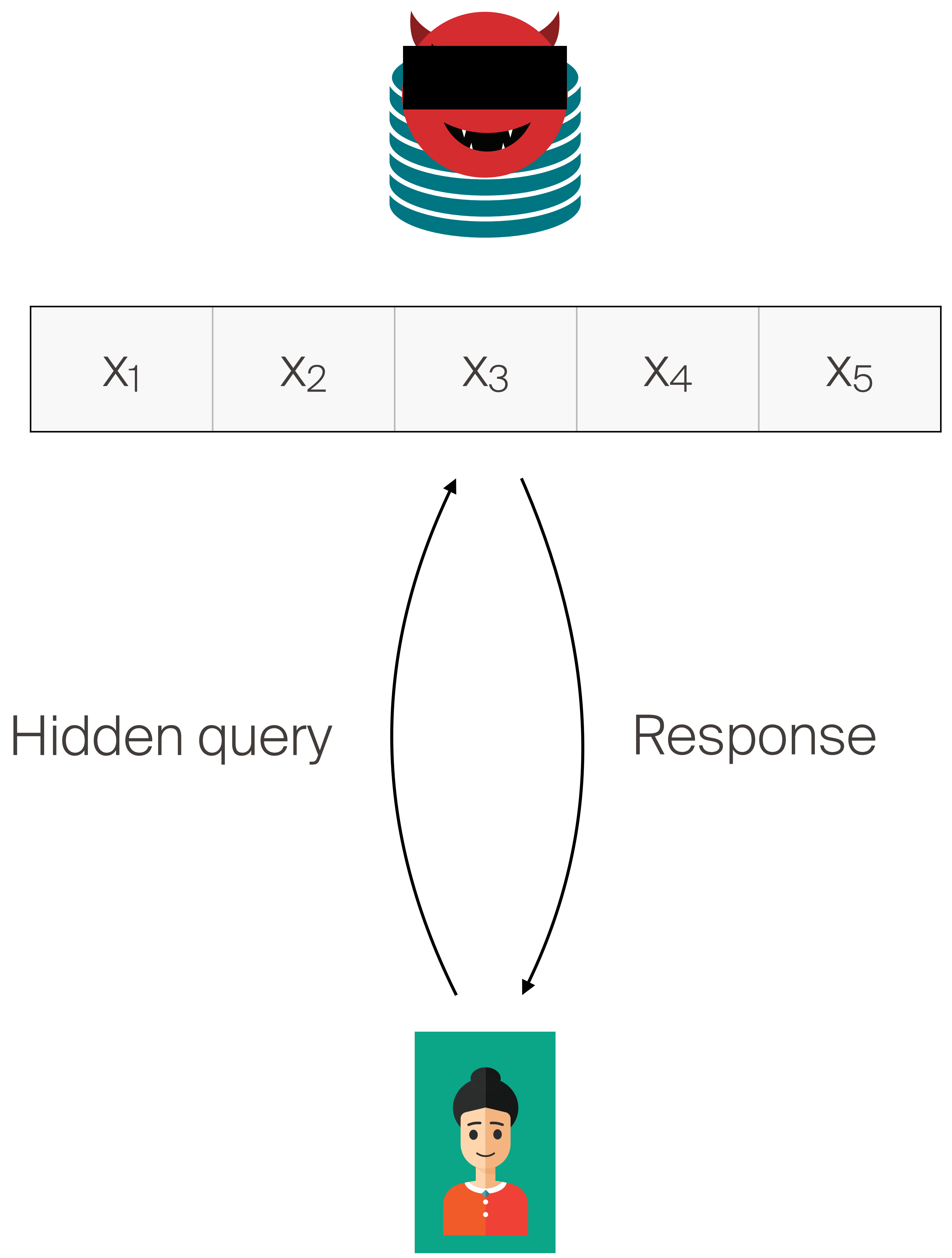


Service providers learn user’s choices



Kirill Nikitin

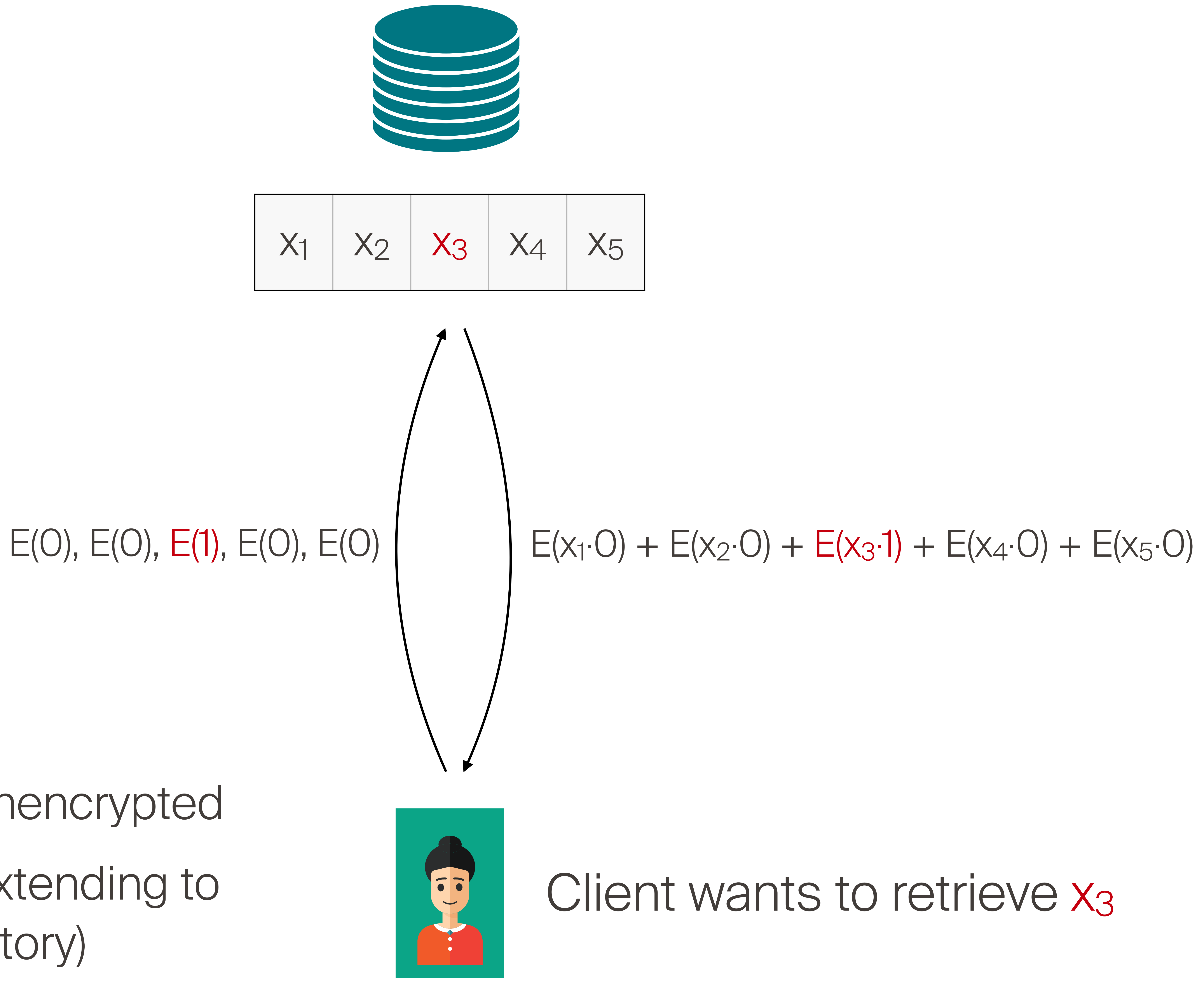
Private Information Retrieval (PIR)



Blind computation

Some applications:
software updates [Cap13]
online-presence service [BDG15]
anonymous messaging [AS16]
video streaming [GCM+16]
encrypted search [DFL+20]

The single-server PIR setting

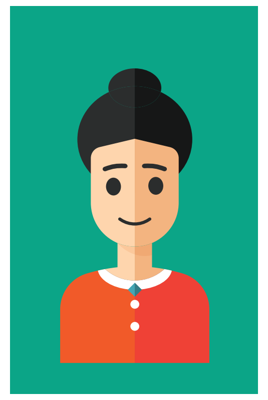


- The database is typically unencrypted
- Records x_i are often bits (extending to longer rows is a separate story)

Kirill Nikitin

Problem: No data integrity by default

y	y	y	y	y

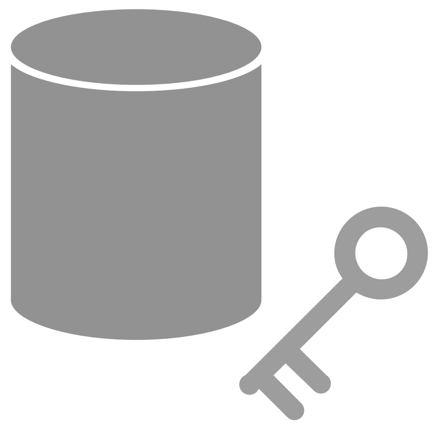


Kirill Nikitin

A typical way to get integrity

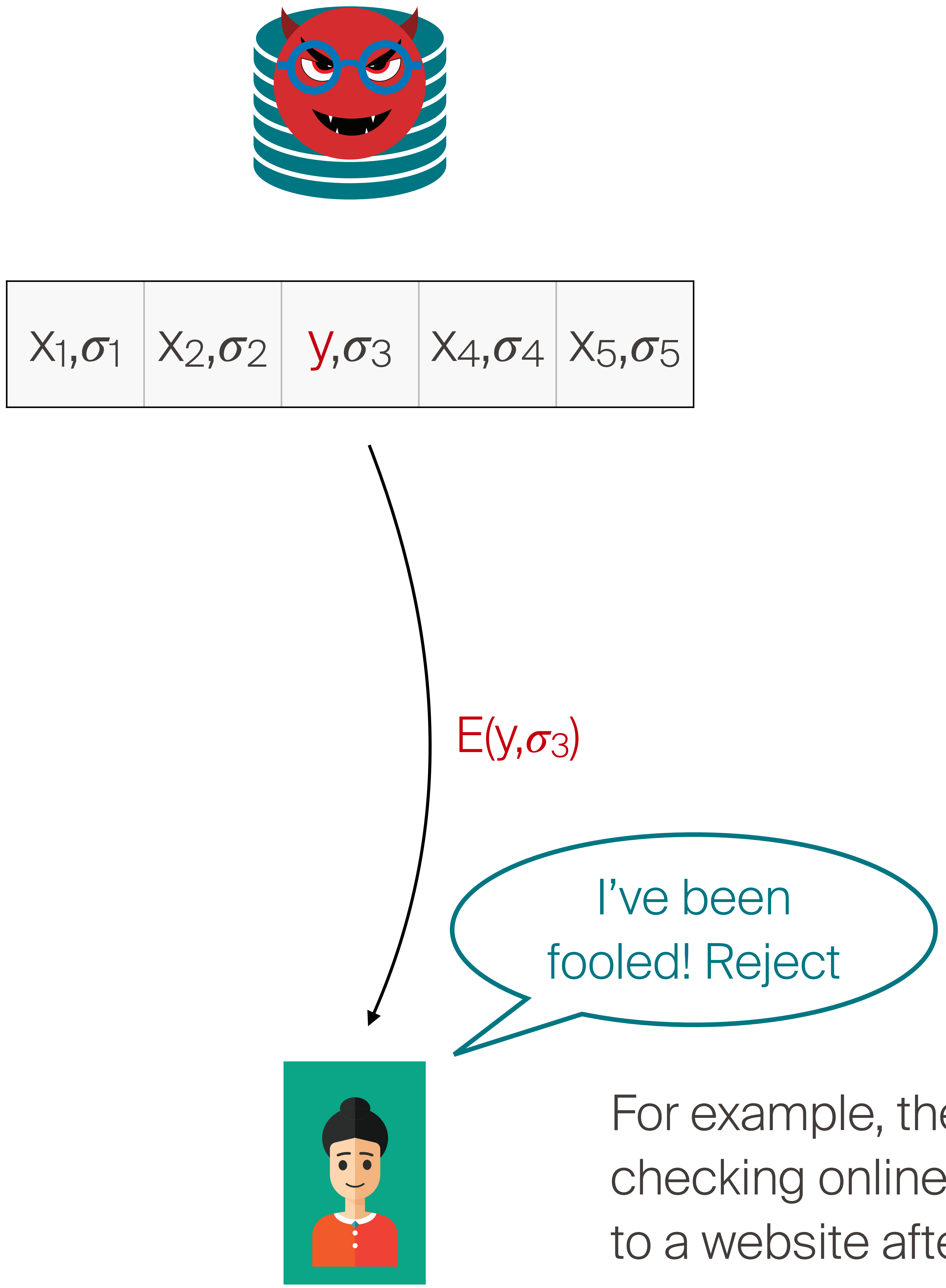
X_1,σ_1	X_2,σ_2	X_3,σ_3	X_4,σ_4	X_5,σ_5

Attach a digital signature to each record!



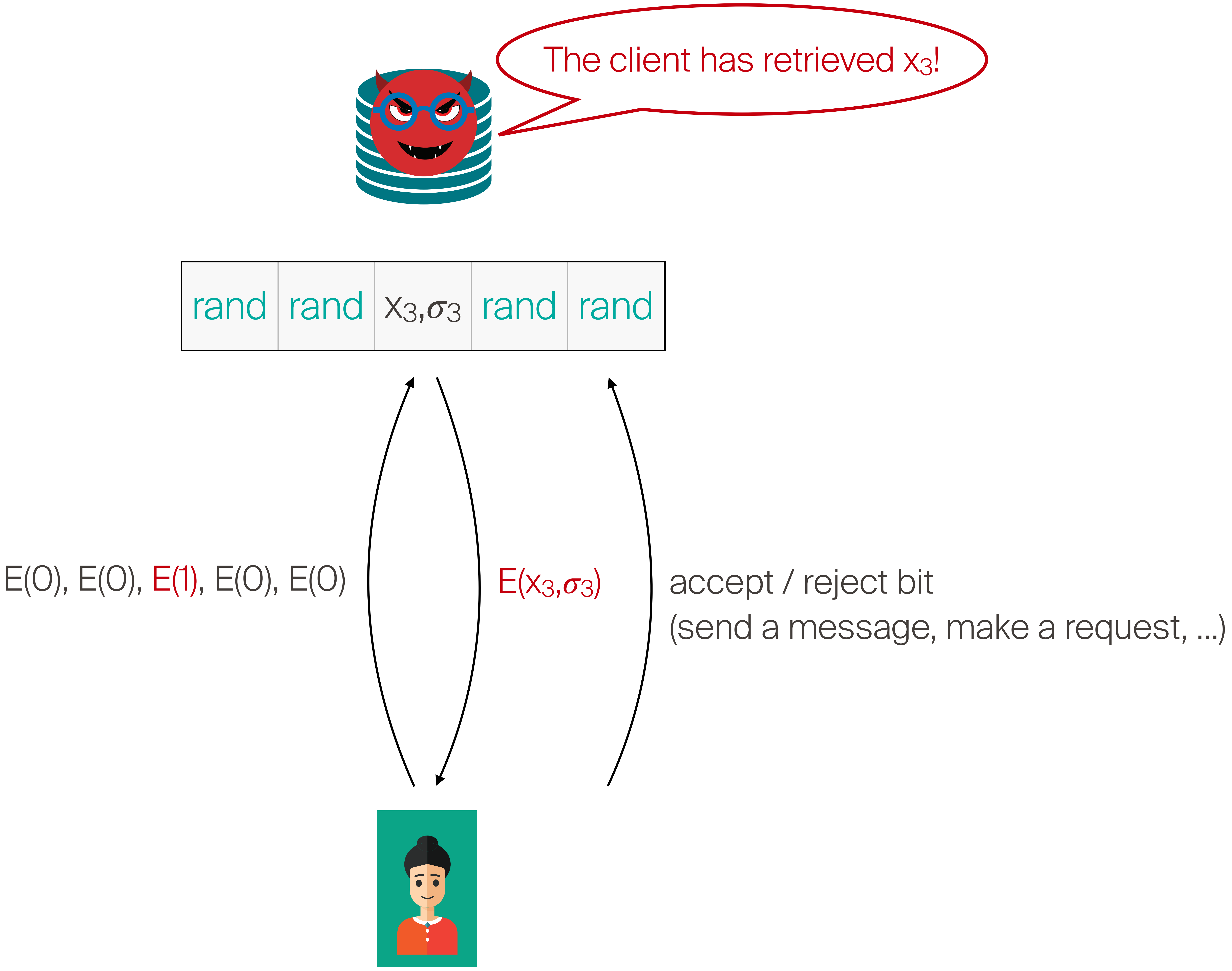
Signing data owner

When integrity breaks privacy



For example, the client starts communication after checking online presence of a friend, or connects to a website after retrieving a DNS record, etc

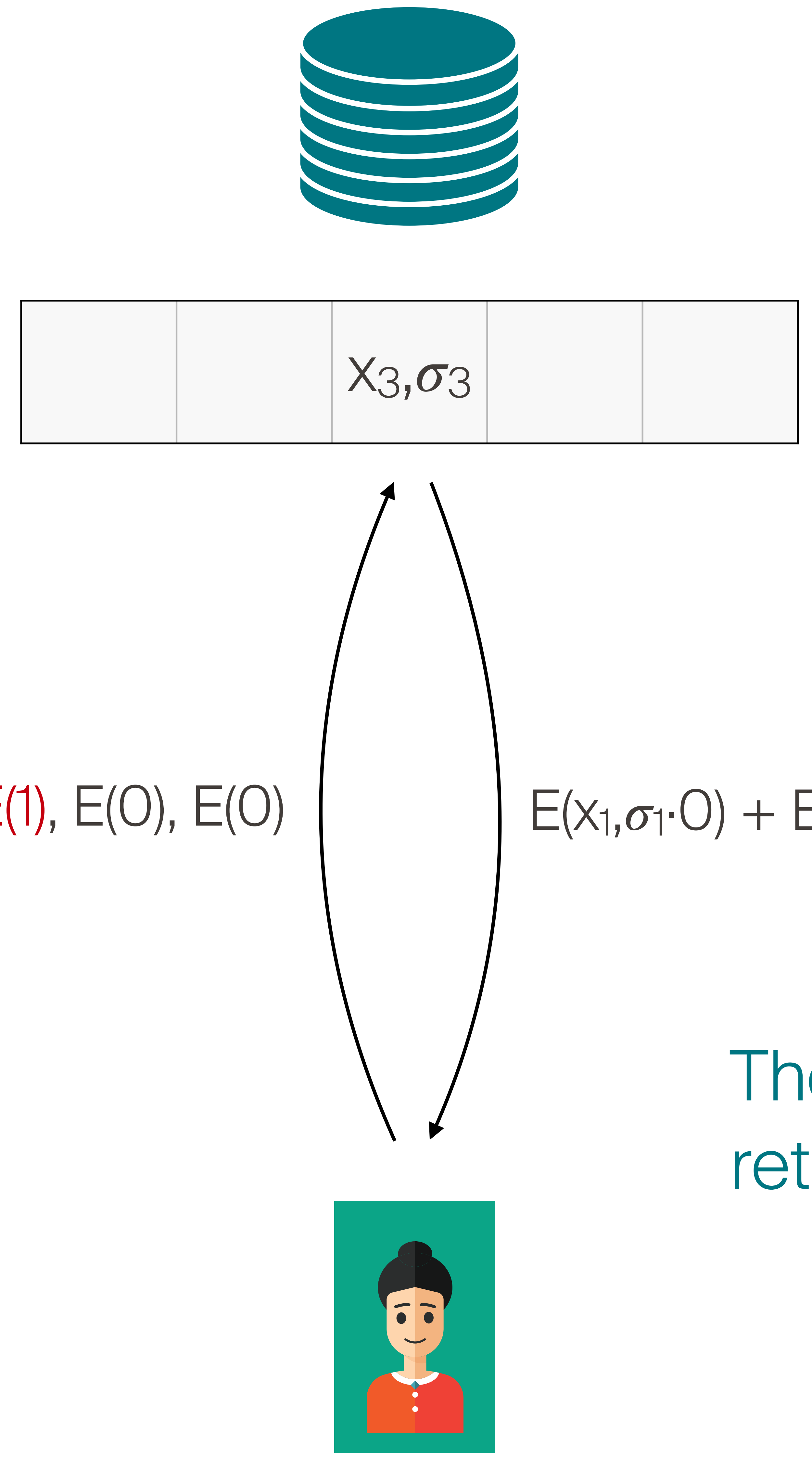
When integrity breaks privacy



Verifiable single-server PIR

- Provides privacy *and* integrity atomically
- Formally, adding the integrity property to the standard correctness and privacy
- Client detects any altering of the database, even for the records she is *not* retrieving
- Prior work on verifiable PIR [ZS14, WZ18] relied on heavy machinery (signatures of correct computation [PST13])

Verifiable single-server PIR: Challenge



$E(0), E(0), E(1), E(0), E(0)$

$E(x_1,\sigma_1 \cdot 0) + E(x_2,\sigma_2 \cdot 0) + E(x_3,\sigma_3 \cdot 1) + E(x_4,\sigma_4 \cdot 0) + E(x_5,\sigma_5 \cdot 0)$

The authentication information of non-retrieved records is cancelled out

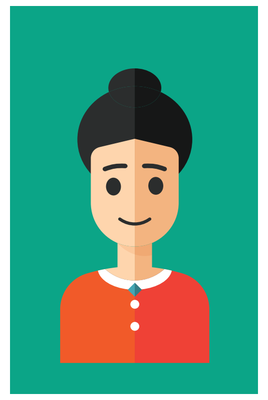
Verifiable single-server PIR

Public database digest

$d = g_1^{x1} \cdot g_2^{x2} \cdot g_3^{x3} \cdot g_4^{x4} \cdot g_5^{x5}$

g_i's are the hashes of the record indices to group elements

x_1	x_2	x_3	x_4	x_5



Verifiable single-server PIR

Public database digest

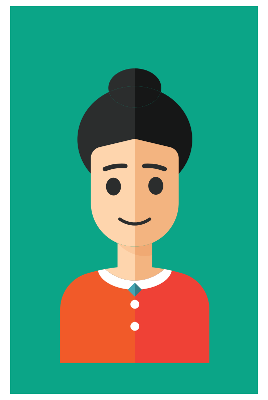
$d = g_1^{x1} \cdot g_2^{x2} \cdot g_3^{x3} \cdot g_4^{x4} \cdot g_5^{x5}$

g_i's are the hashes of the record indices to group elements

x_1	x_2	x_3	x_4	x_5

$g_1^r, g_2^r, g_3^{r+t}, g_4^r, g_5^r$

$a = g_1^{x1 \cdot r} \cdot g_2^{x2 \cdot r} \cdot g_3^{x3 \cdot (r+t)} \cdot g_4^{x4 \cdot r} \cdot g_5^{x5 \cdot r}$



If $a = d^r \cdot g_3^t$, $x_3 = 1$

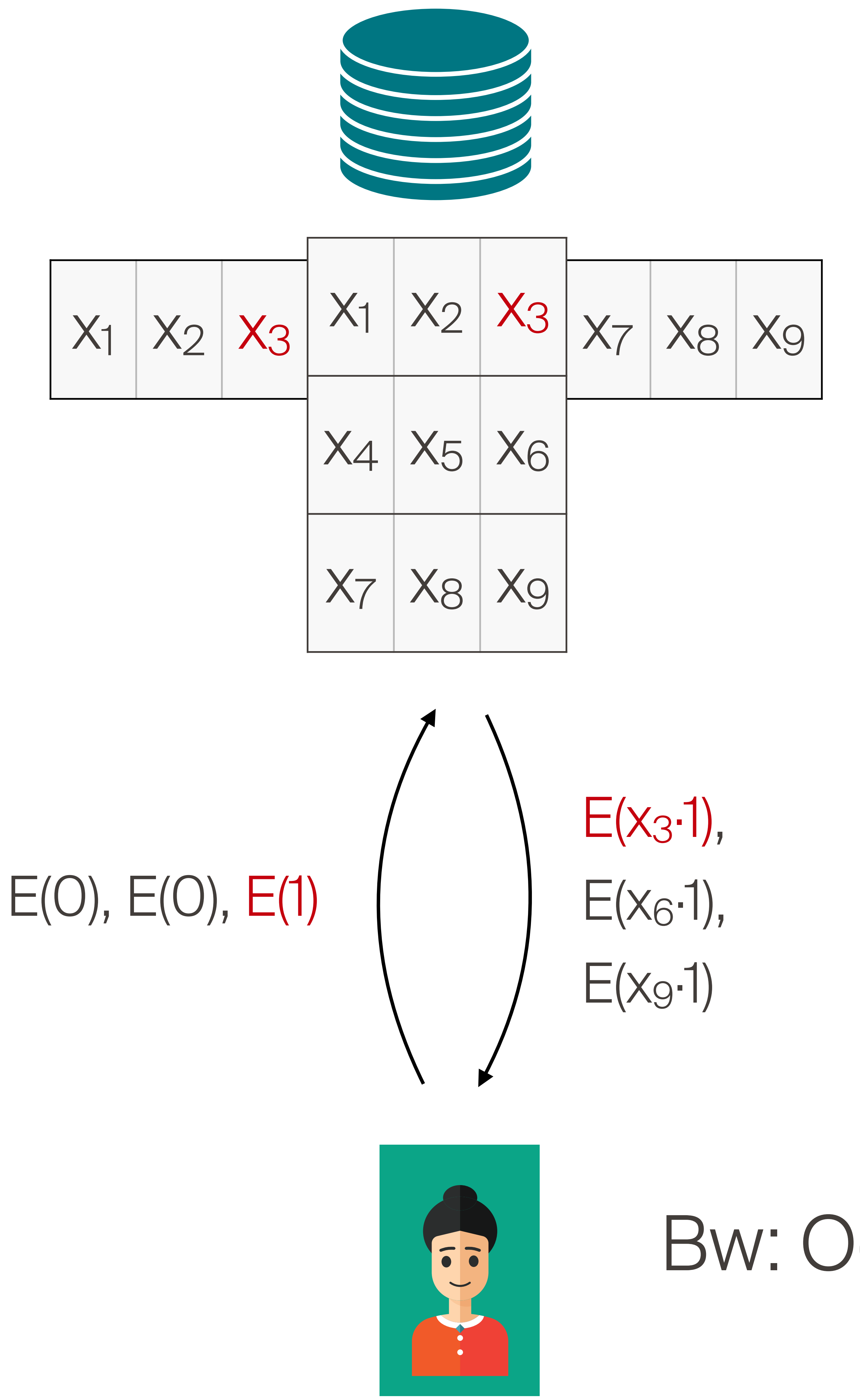
If $a = d^r \cdot 1_G$, $x_3 = 0$

Otherwise $\perp$

Reducing communication

Kirill Nikitin

Rebalancing



Bw: $O(n) \rightarrow O(\sqrt{n})$

Evaluation

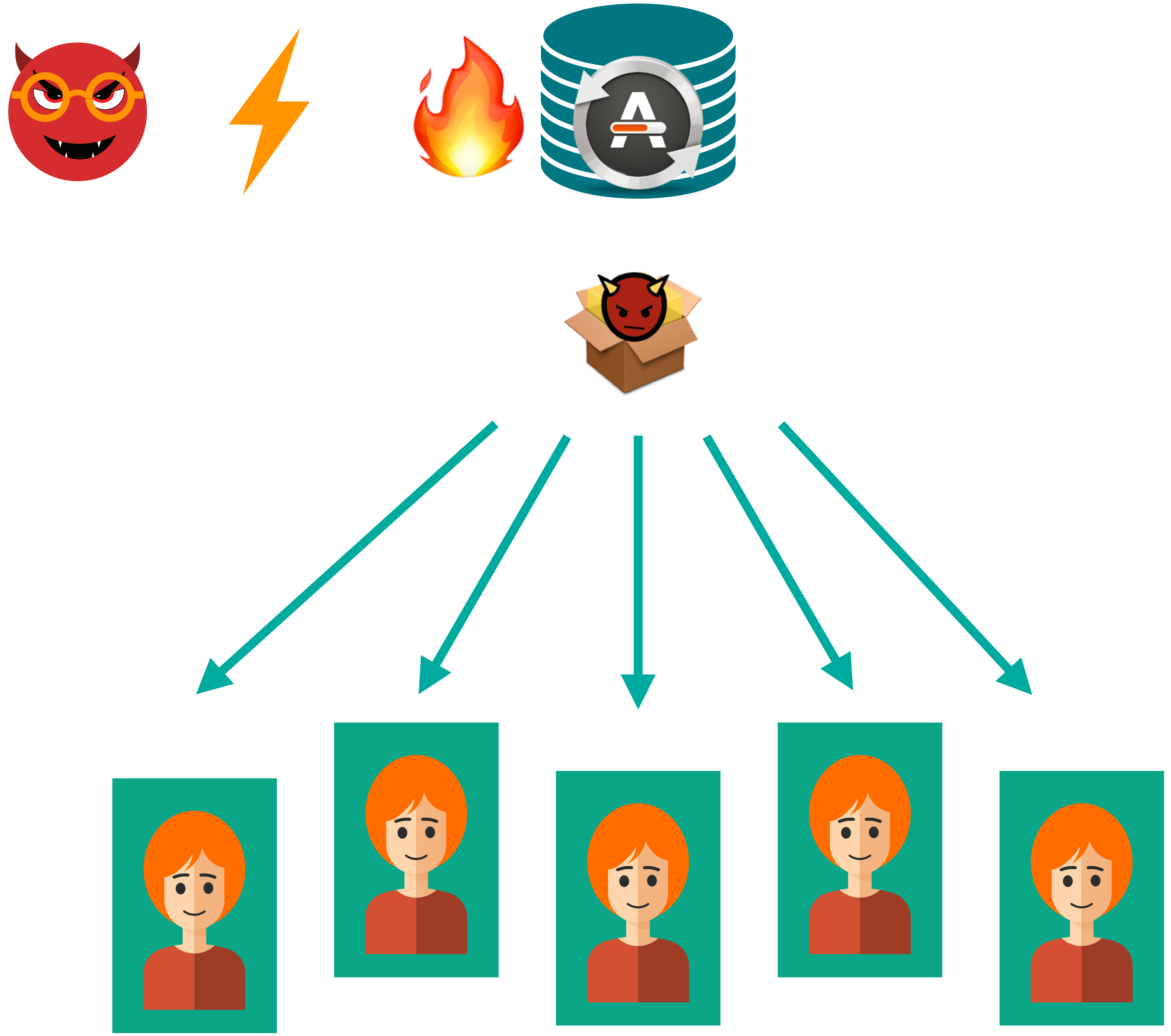
- The scenario of private contact discovery (retrieving 1 bit of data)
- Compare with state-of-the-art lattice-based PIR as a baseline

DB size [bits]	w/o integrity	Verifiable	Overhead
	Server CPU time [sec]		
1 M	1.2	16	13×
10 M	7	160	24×
100 M	60	1,561	26×
1 B	668	15,769	24×
	Bandwidth [MiB]		
1 M	1.5	0.06	0.04×
10 M	3.8	0.2	0.05×
100 M	11	0.6	0.06×
1 B	33	2.0	0.06×

Roadmap

- ❖ Introduction
- ❖ Protecting encryption metadata (Chapter 2)
- ❖ Data integrity in single-server PIR (Chapter 3)
- ❖ Securing retrieval of software updates (Chapter 4)
- ❖ Conclusion

Compromising a software-update system

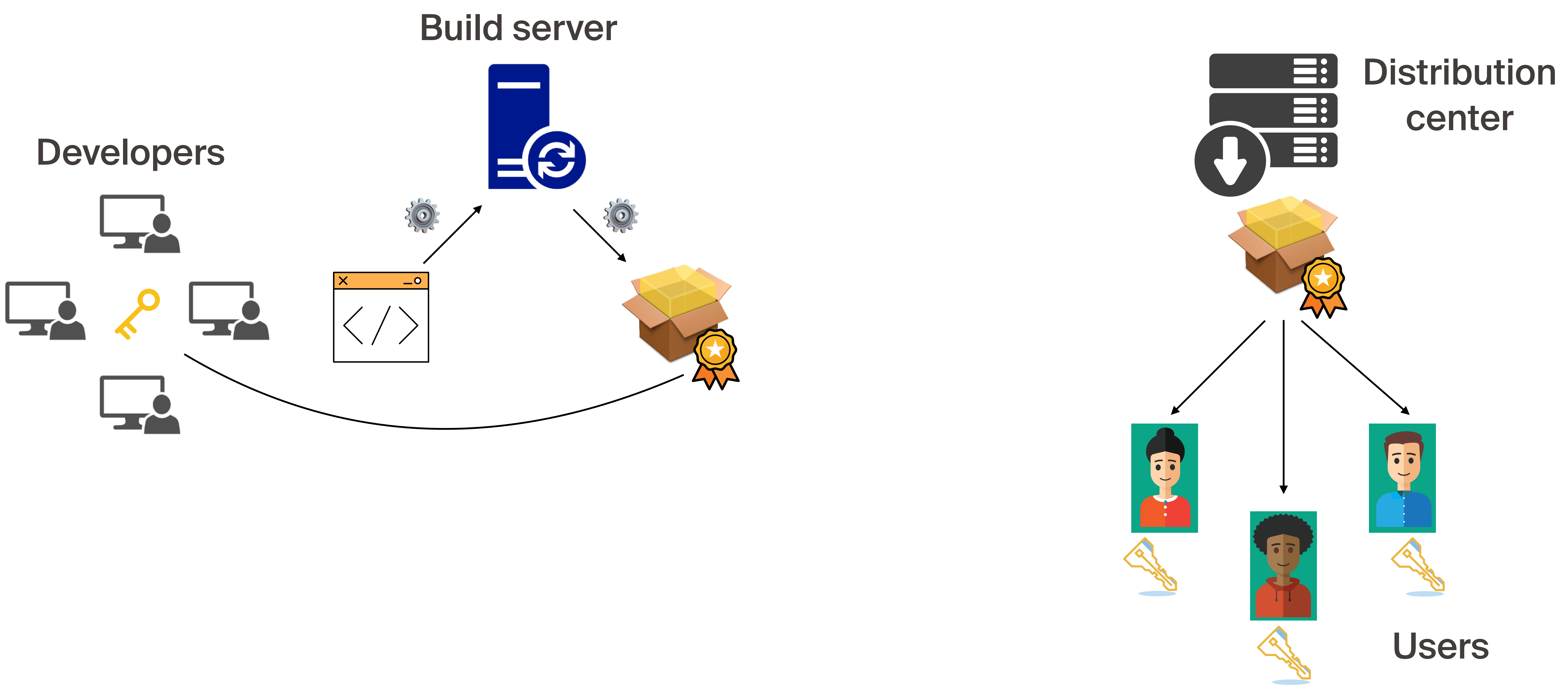


Compromised software-update systems

Software Release Pipeline

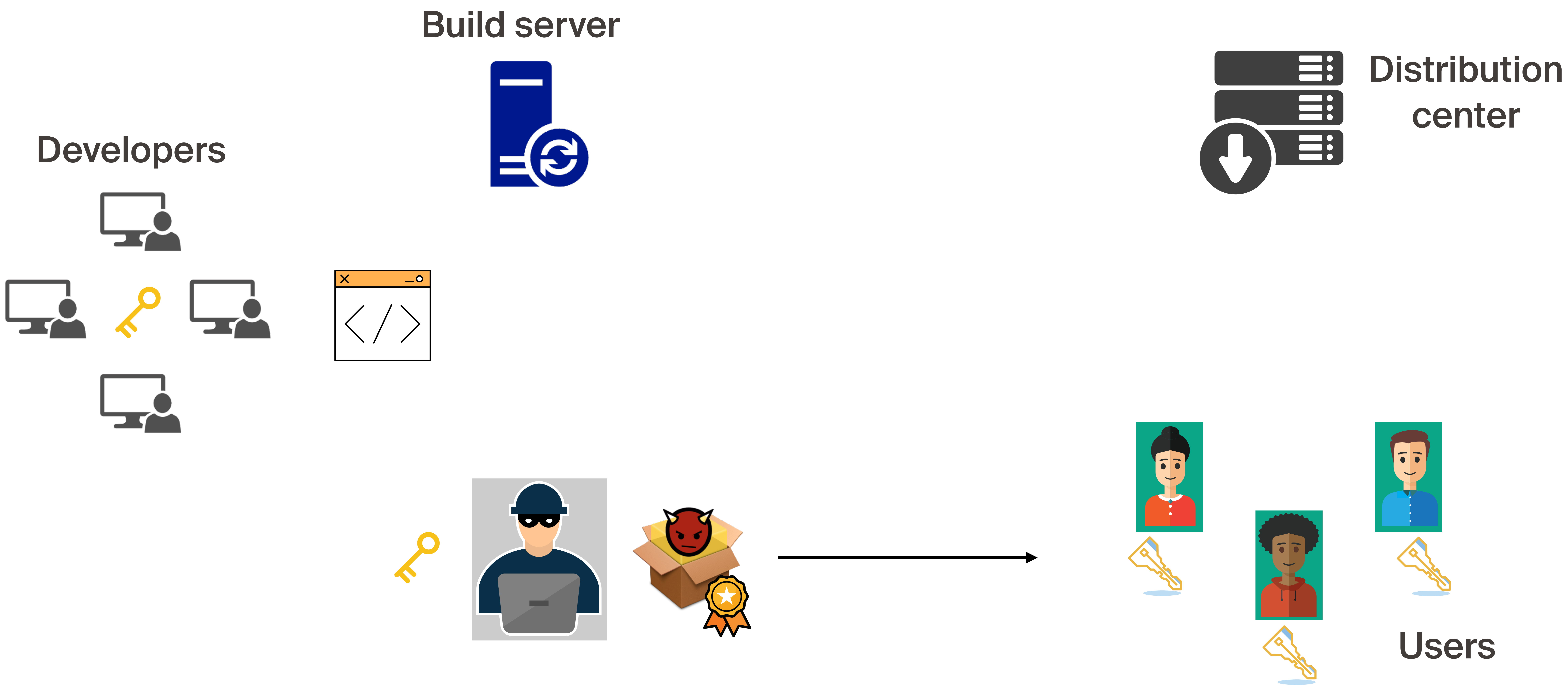
Kirill Nikitin

Development/Review – Building release binaries – Sign-off – Release distribution



Challenges

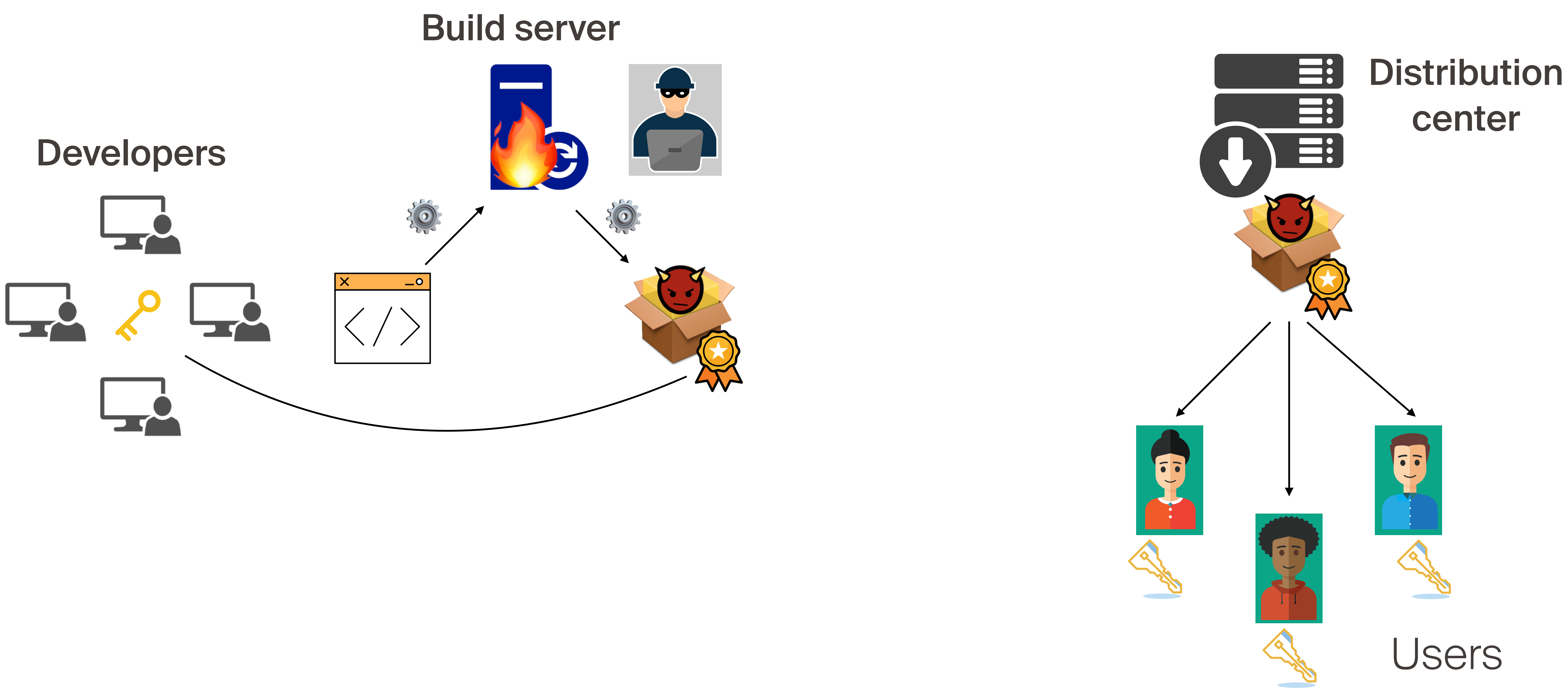
(1) Make software-update process resilient to partial key compromise



Challenges

Kirill Nikitin

(2) Prevent malicious substitution of a release binary during a build process



Challenges

(2) Prevent malicious substitution of a release binary during a build process

reproducible-builds.org

Provide a verifiable path from source code to binary.

debian
GNU/Linux

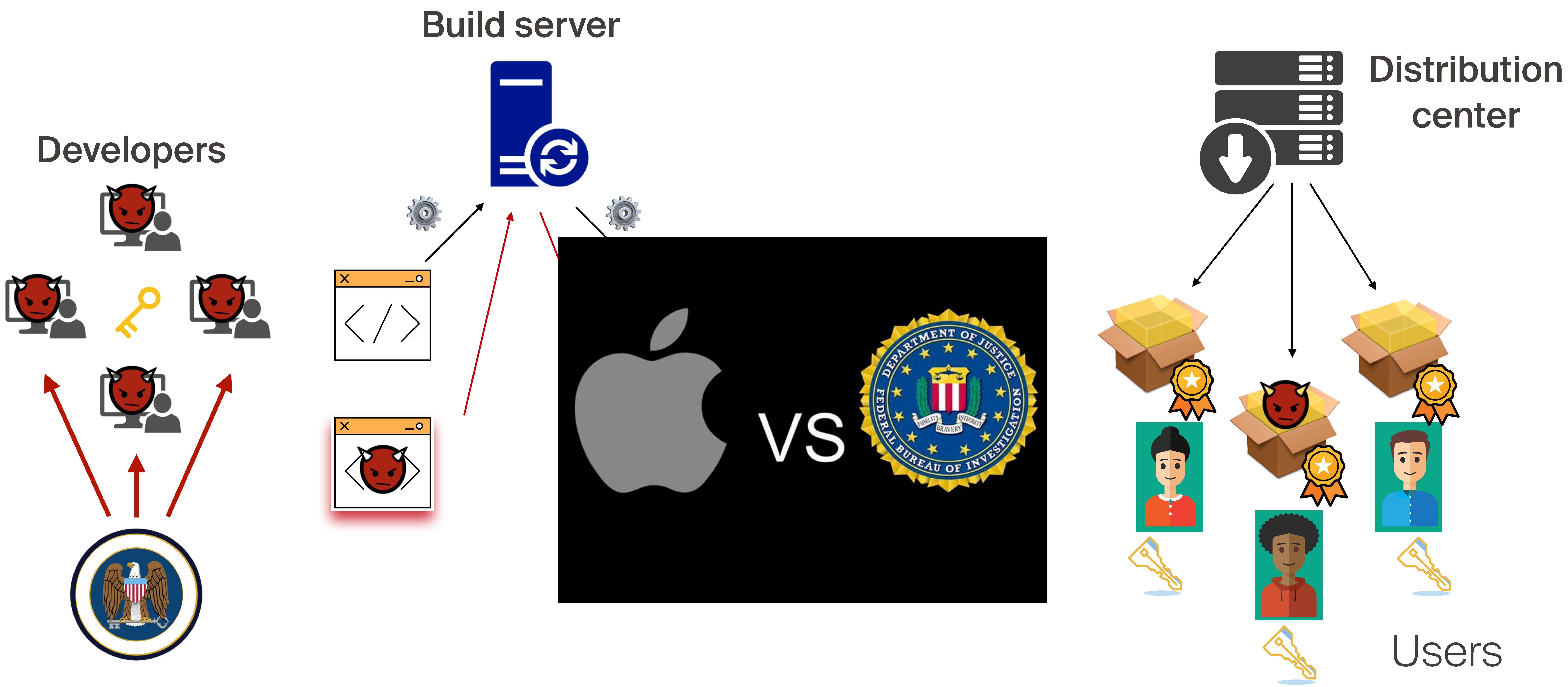
Over 90% of the source packages included in Debian 9 will build bit-for-bit identical binary packages

1. Regular users do not compile from source code
2. Reproducible compilation can take hours (e.g., Tor browser)
3. Closed-source software?

Kirill Nikitin

Challenges

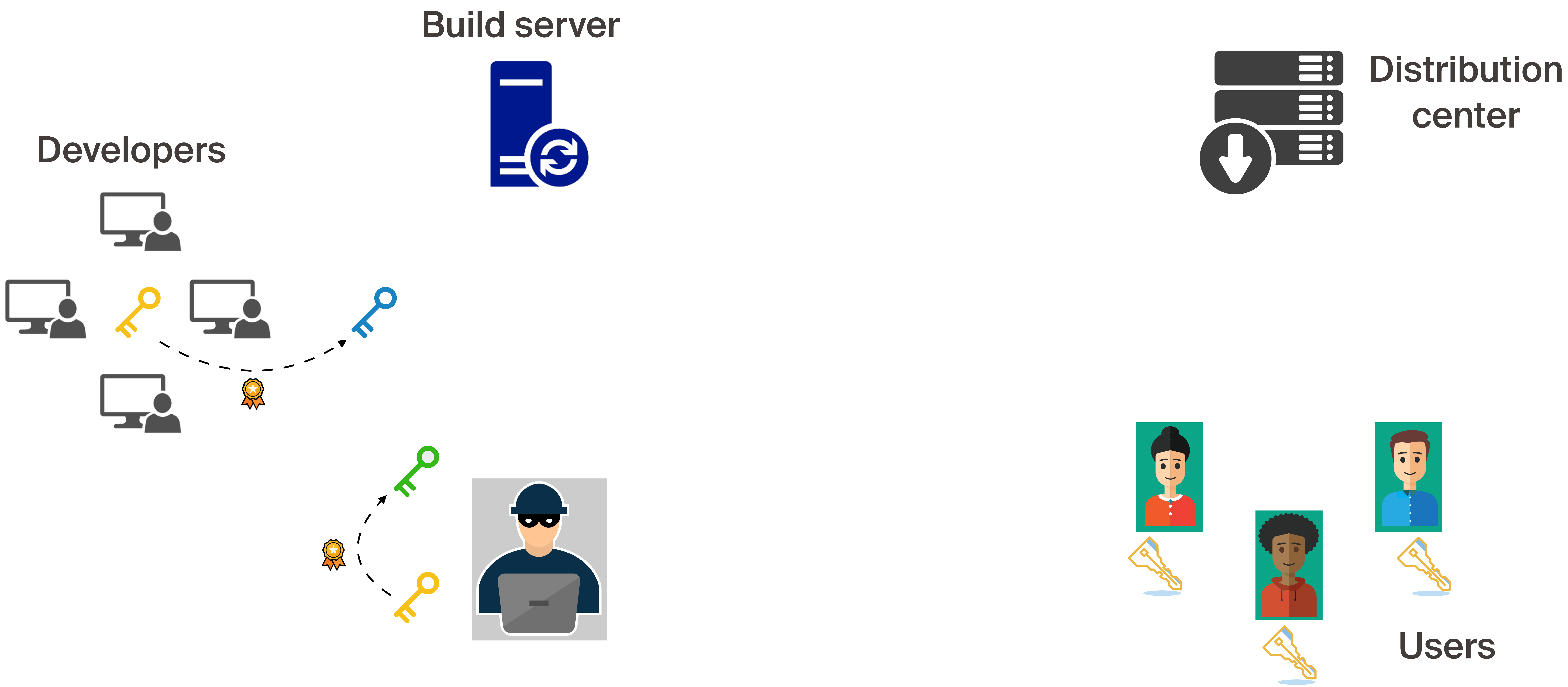
(3) Protect users from targeted attacks by coerced or bribed developers



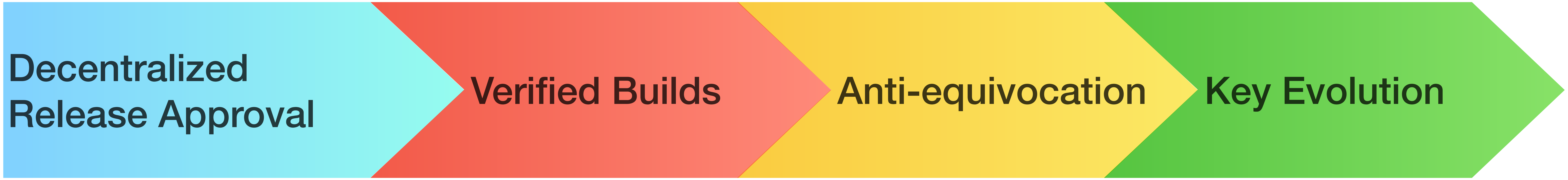
Kirill Nikitin

Challenges

(4) Enable developers to securely rotate their signing keys in case of renewal or compromise



CHAINIAC: Securing software-update retrieval

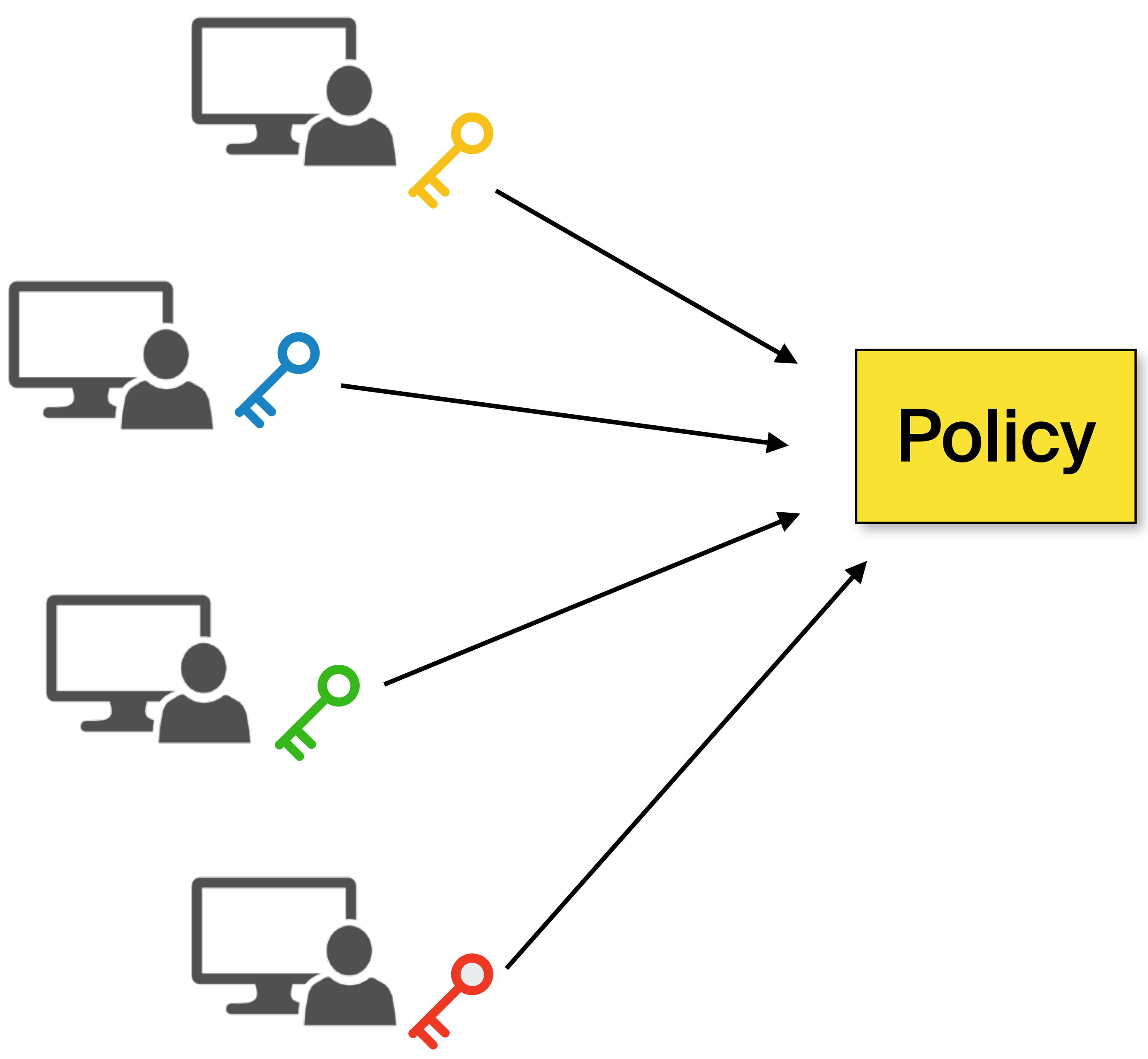


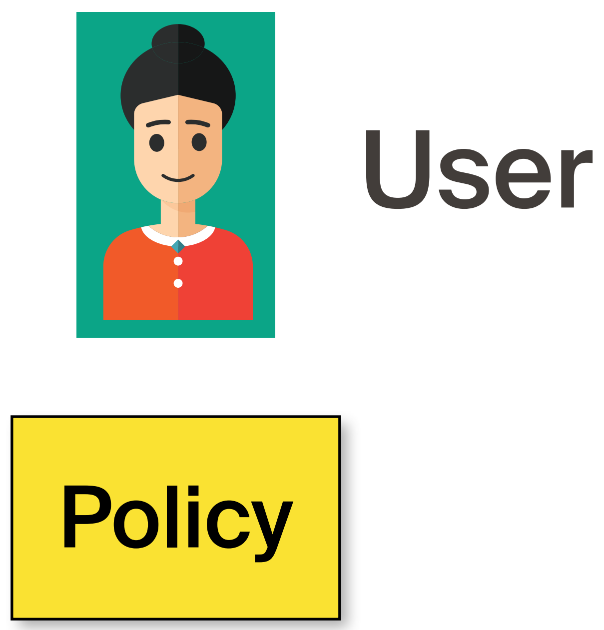
Kirill Nikitin

Decentralized release approval

(1) Make software-update process resilient to partial key compromise

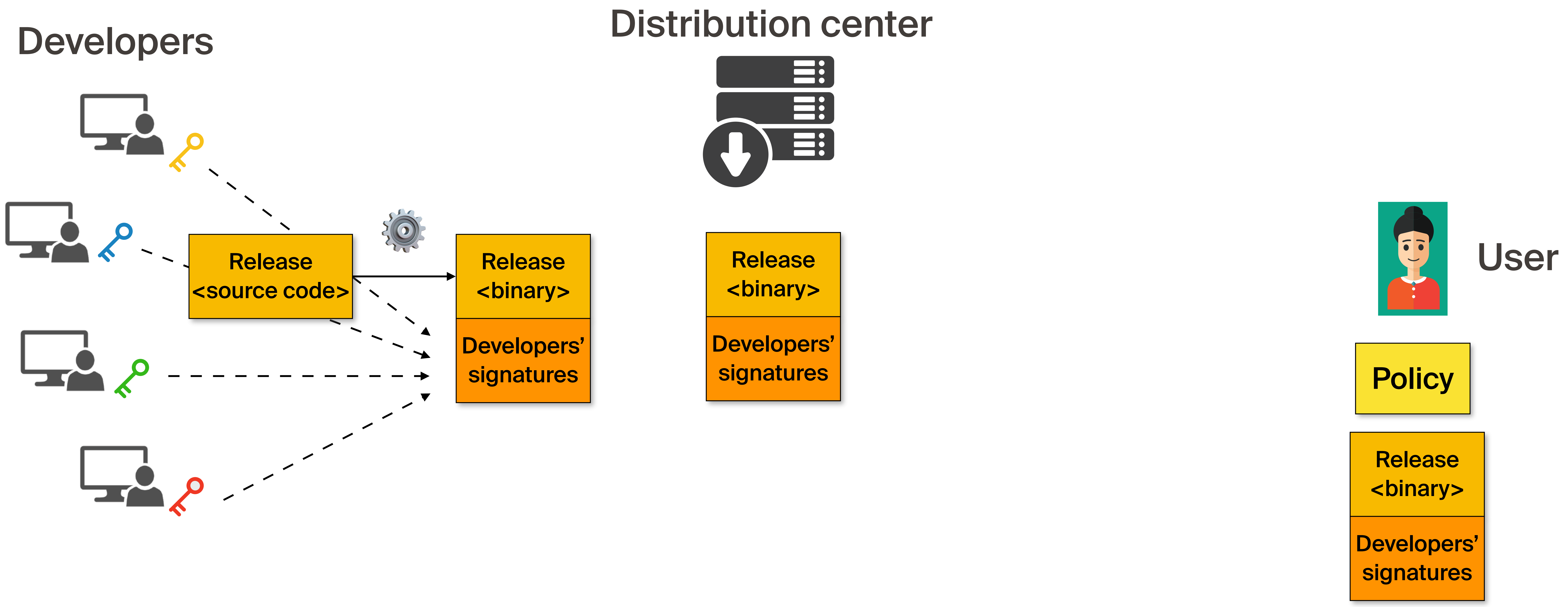
Developers





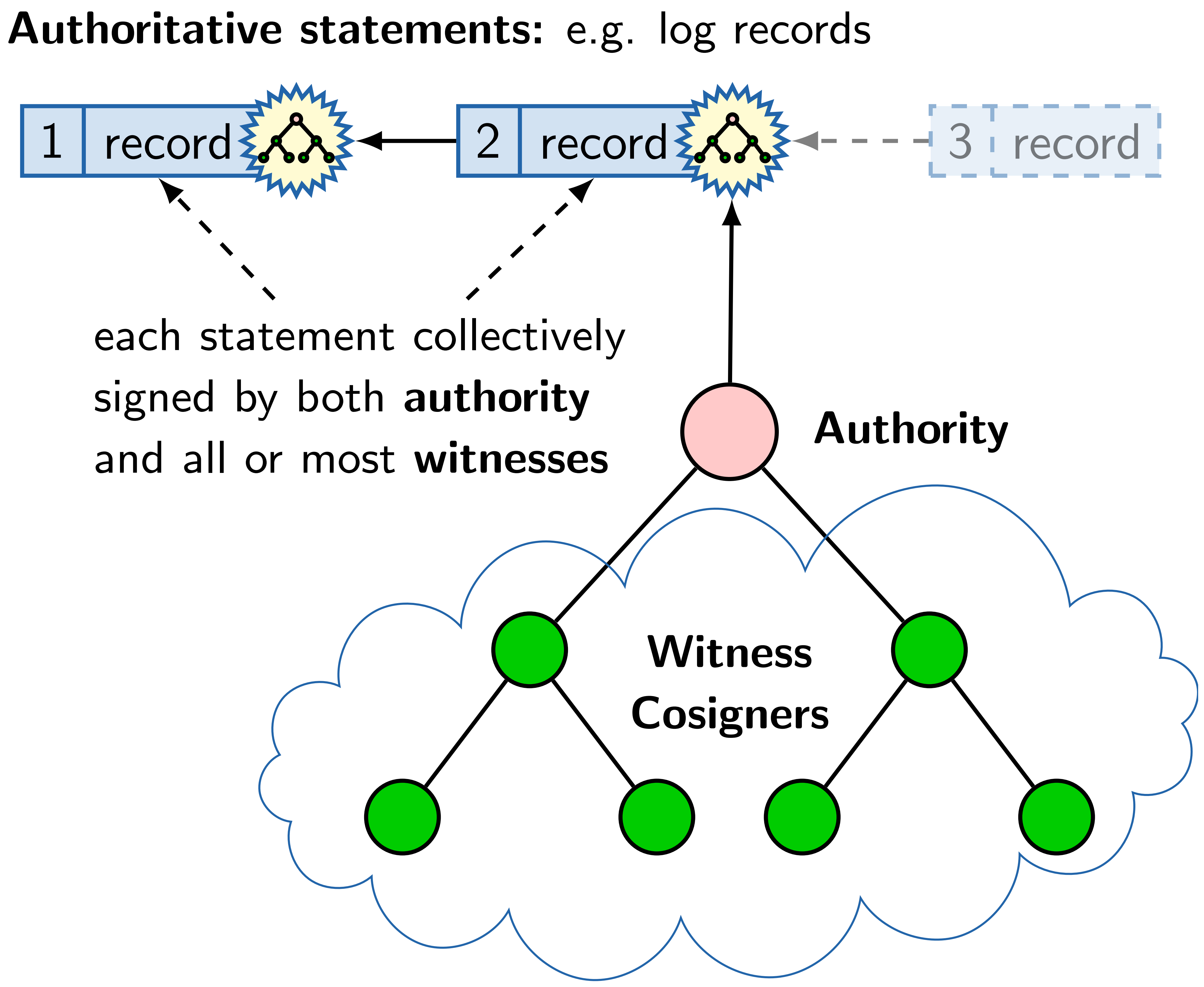
Decentralized release approval

(1) Make software-update process resilient to partial key compromise



Background

Collective Authority (Cothority), Collective Signing (CoSi), and BFT-CoSi



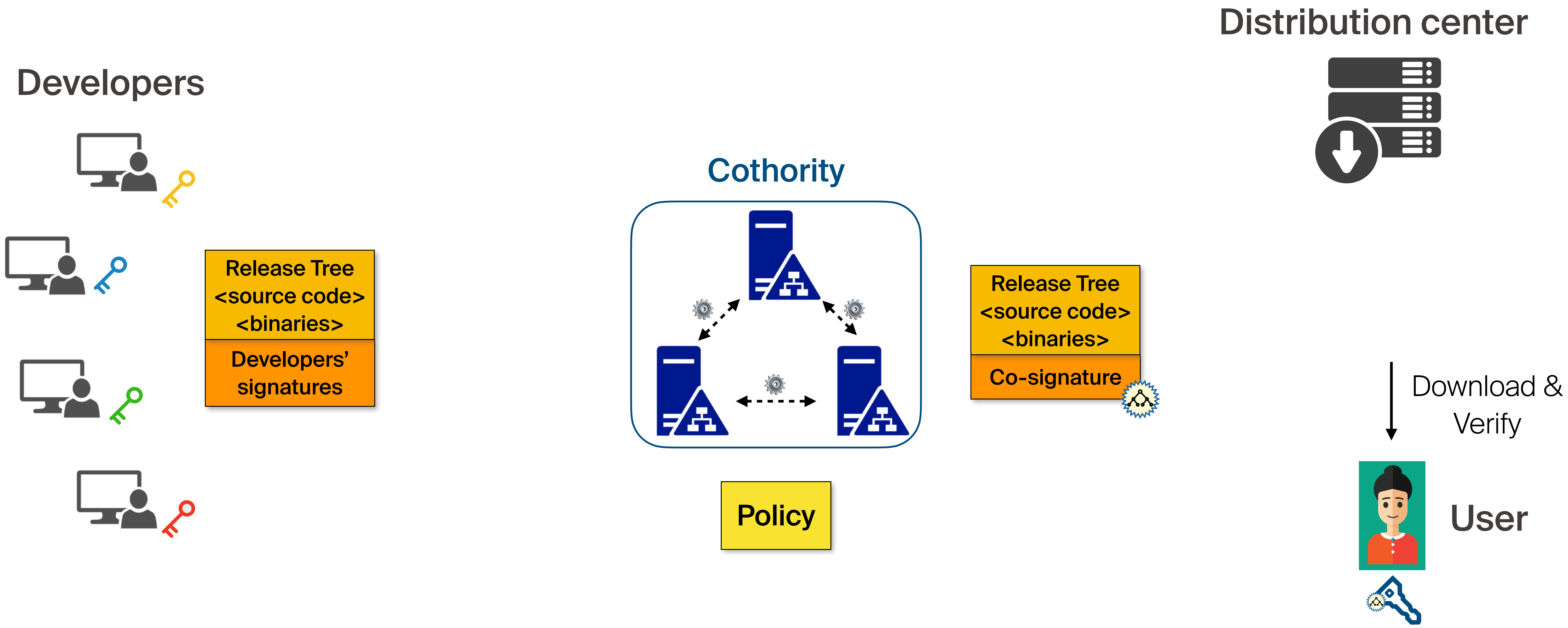
References

1. E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and Bryan Ford. Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning. S&P 2016.
2. E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. USENIX Security 2016.

Kirill Nikitin

Verified builds

(2) Prevent malicious substitution of a release binary during building process



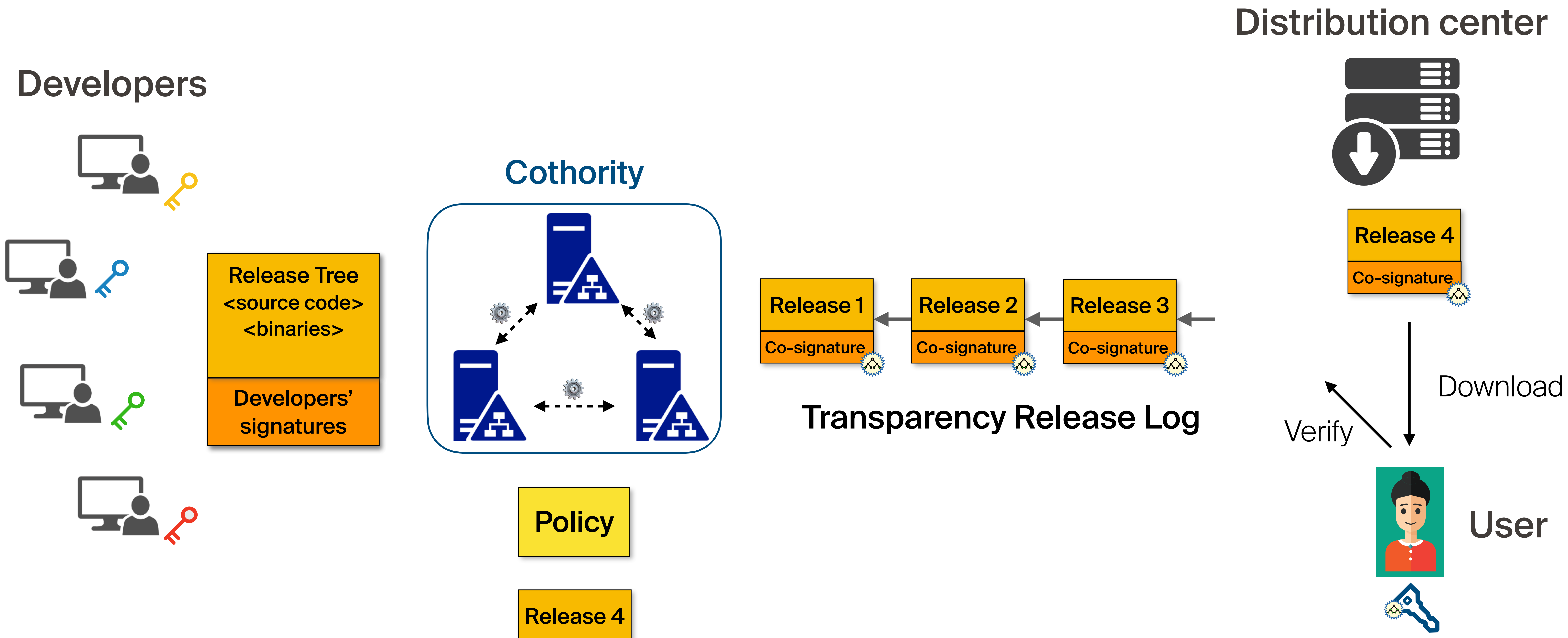
Verified builds

Release Policy File

- List of individual developer public keys
- Signing threshold
- Cothority public key
- Supported platforms for verified builds
- …

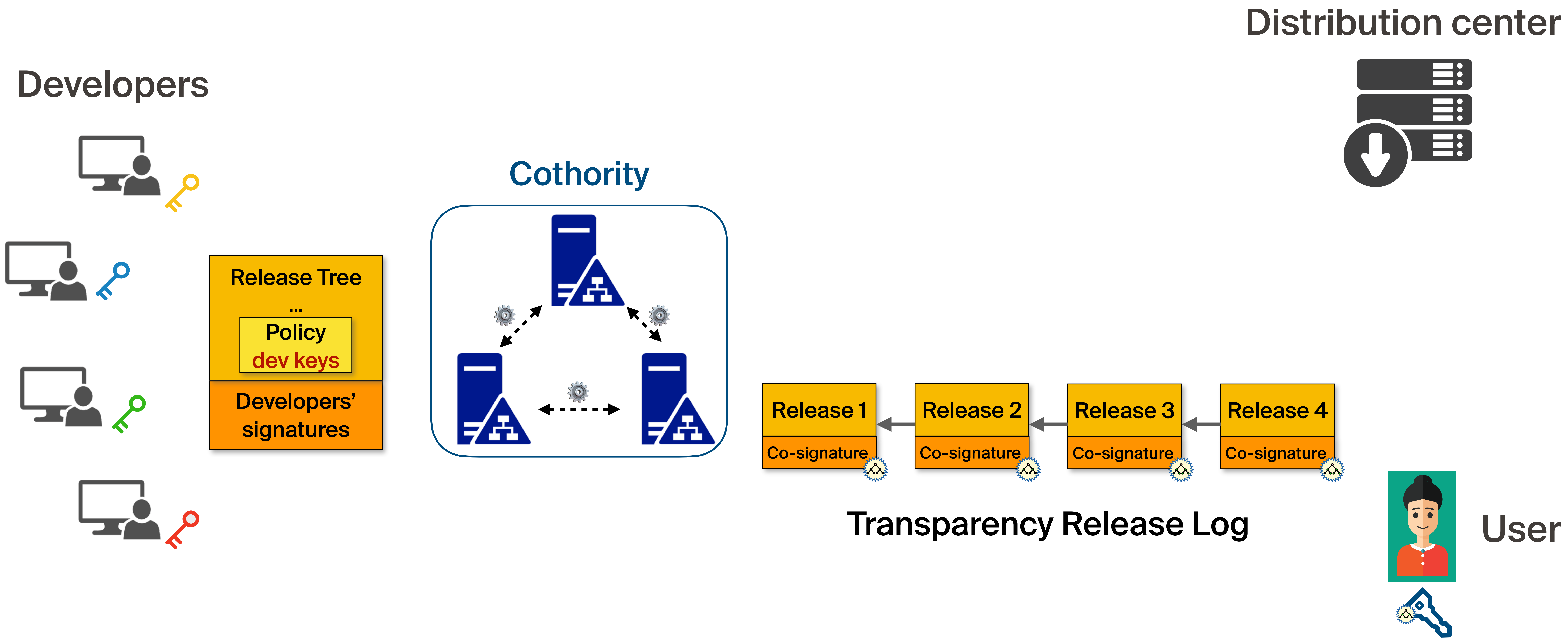
Anti-equivocation measures

(3) Protect users from targeted attacks by coerced or bribed developers



Anti-equivocation measures

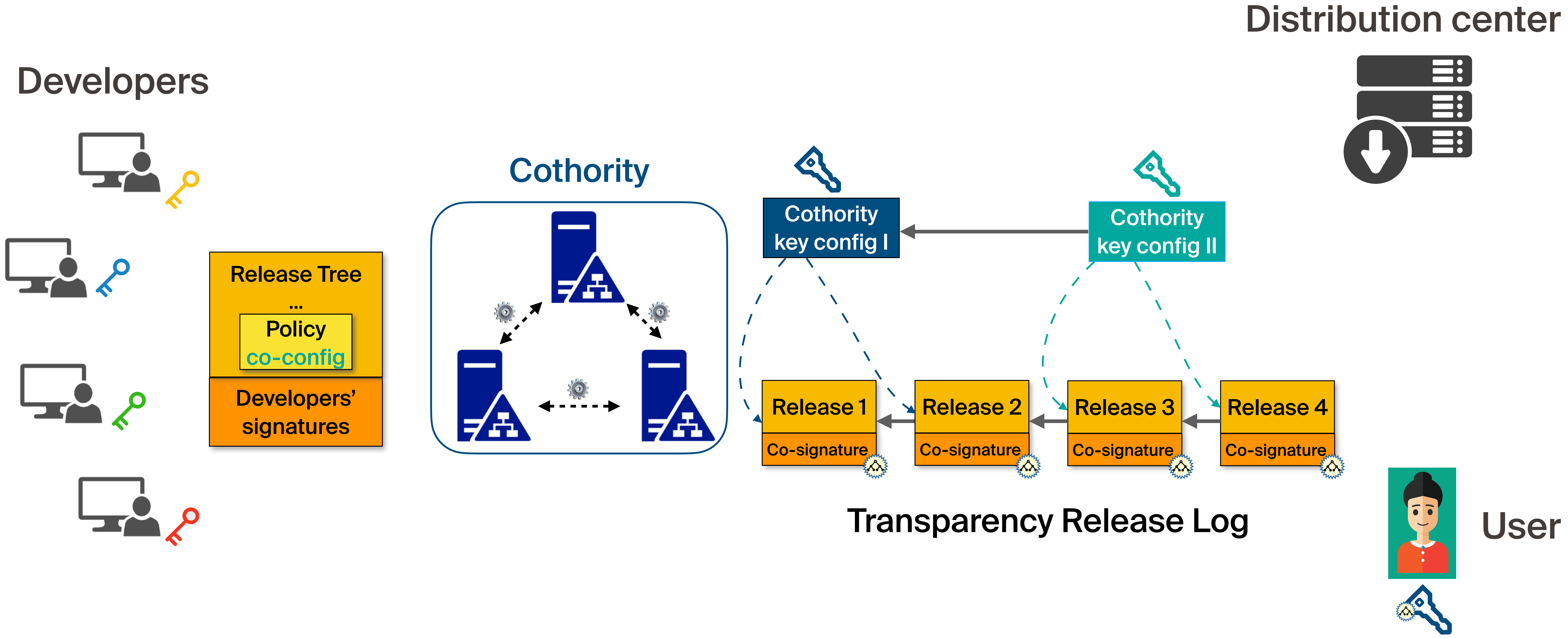
(4) Enable developers to securely rotate their keys



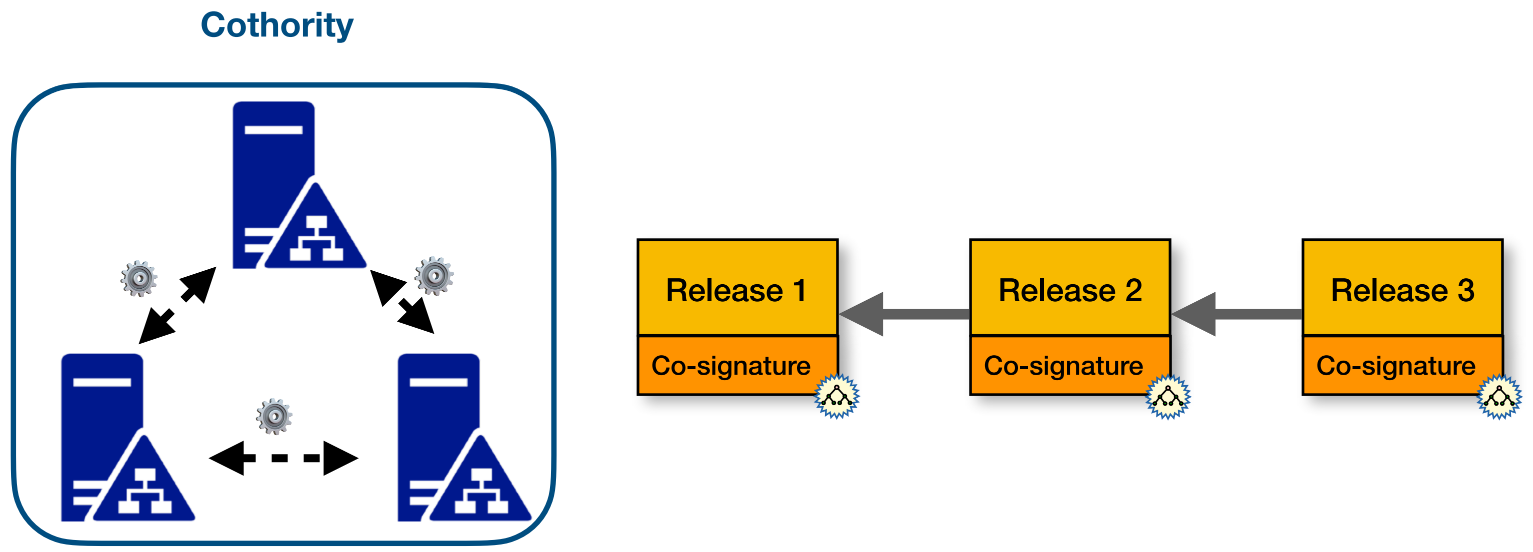
Kirill Nikitin

Anti-equivocation measures

(4) Enable cothority to securely rotate its keys

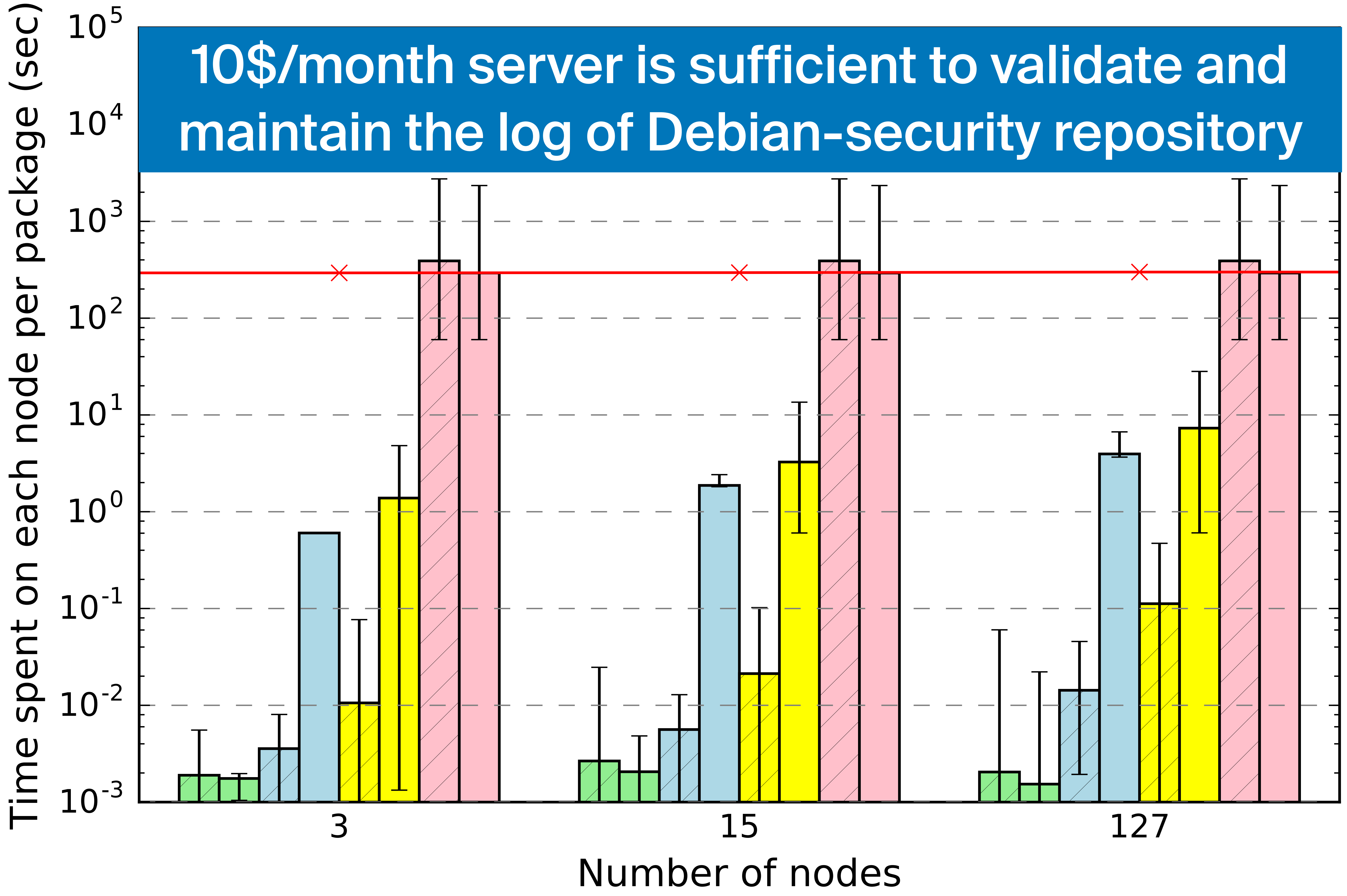


Evaluation



Kirill Nikitin

Cothority-node CPU cost of validating releases and maintaining release log



10$/month server is sufficient to validate and maintain the log of Debian-security repository

Roadmap

- Introduction
- Protecting encryption metadata (Chapter 2)
- Data integrity in single-server PIR (Chapter 3)
- Securing retrieval of software updates (Chapter 4)
- Conclusion

Contributions of the thesis

- Protecting encryption metadata (Chapter 2)
  - The concept of Padded Uniform Random Blobs: an encryption format without any cleartext markers
  - An encoding ind$-cca2-secure scheme for efficient generation and decoding of PURBs
    - An efficient way to combine encryptions of different types in a single ciphertext
    - The placement techniques (growing hash tables, public-key hiding) could find application in other privacy systems

Contributions of the thesis

- Verifiable single-server PIR (Chapter 3)
  - Selective failures in the PIR context
  - A PIR protocol with inherent database integrity
- Chainiac (Chapter 4)
  - Full use of decentralization for protecting software-update systems without deteriorating usability for end users
  - A practical system for real-world use

Future work

- Metadata protection
  - Protocols for secure communication, such as TLS
- Verifiable PIR
  - Better protocols (lower communication cost, larger database records)
  - Extensions to Oblivious RAM, encrypted search, etc
- Transparency and verifiability
  - From software updates to the Web

Future work

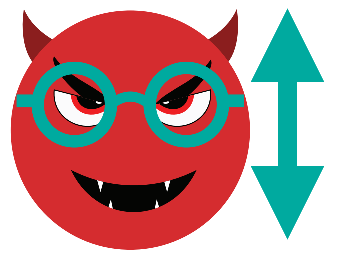
- More hybrid mechanisms that provably provide multiple security properties in an atomic way

Conclusion

Thank you!

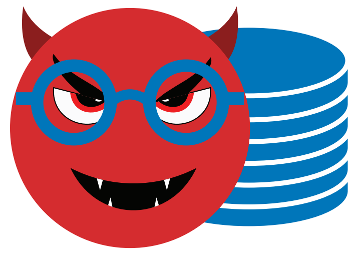
On-the-network attacker

- Protecting encryption metadata (Chapter 2) [1]



Malicious provider

- Data integrity in single-server private information retrieval (Chapter 3) [2]



Compromised provider

- Securing retrieval of software updates (Chapter 4) [3]

[1] K. Nikitin*, L. Barman*, W. Lueks, M. Underwood, J.-P. Hubaux, and B. Ford, "Reducing Metadata Leakage from Encrypted Files and Communication with PURBs", PETS 2019.

[2] S. Colombo*, K. Nikitin*, B. Ford, and H.Corrigan-Gibbs, "Verifiable Private Information Retrieval", Under submission.

[3] K. Nikitin, E. Kokoris-Kogias, P. Jovanovic, N. Gailly, L. Gasser, I. Khoffi, J. Cappos, and B. Ford, "CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds", USENIX Security 2017.