Hunting Invisible Salamanders:
Cryptographic (in)Security with Attacker-Controlled Keys

Paul Grubbs

Cornell Tech, New York University, University of Michigan

About Me

Now: PhD student in Computer Science at Cornell's NYC campus

This fall: starting postdoc at NYU

Next fall: starting as junior professor at Michigan EECS

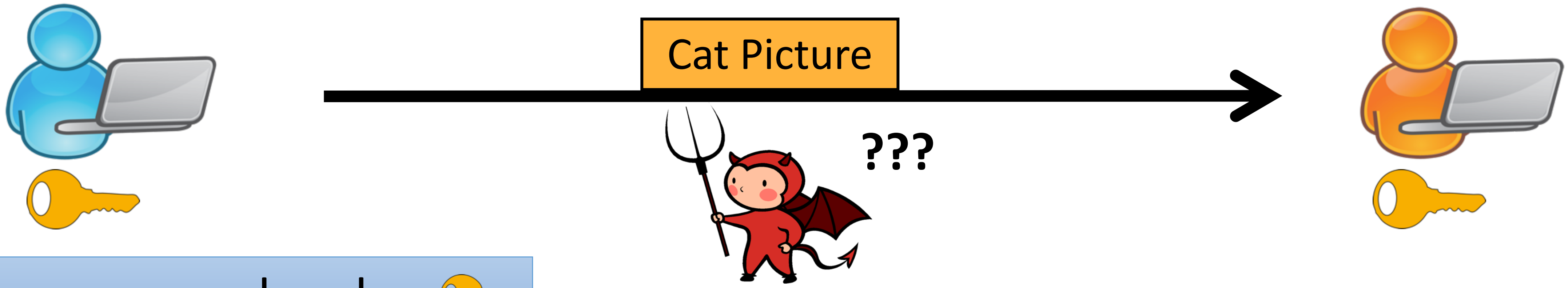
This Talk

Intended audience: those who design, implement, and use cryptography.
Others will find talk interesting and enjoyable but may lack some context.

This is a talk about cryptography.
Some of the slides involve math.

This symbol: if you don’t understand all the details, don’t worry about it!

Authenticated Encryption



Agree on random key

Encrypt message with using
authenticated encryption (AE)
(Galois/Counter Mode, Chacha20/Poly1305)

Core of protocols like TLS, IPSec, SSH

If key is random + hidden:
AE hides cat pictures,
prevents modifications

New Settings, New Needs

Attacker chooses key(s)

Encrypt message with using ***authenticated encryption (AE)*** (Galois/Counter Mode, Chacha20/Poly1305)

Increasingly important setting for AE:

- Password-Based Encryption/PAKE
- E2EE Group Messaging
- Abuse Reporting in Encrypted Messaging

Key isn't random + hidden!
What security do we need?
What security do we expect?

Overview

Describe "attacker-controlled keys" setting + examples, explain ***committing*** security property AE needs

Many widely-used AE schemes are ***not*** committing: can break for GCM, ChaCha20/Poly1305, others

Attacks resulting from non-committing AE:

- Inconsistent plaintexts in multi-receiver encryption
- Invisible salamanders in Facebook's message franking
- Key recovery via partitioning oracle attacks

Based on these research papers:

Message Franking via Committing Authenticated Encryption

G., Lu, Ristenpart. IACR CRYPTO17. https://eprint.iacr.org/2017/664

Fast Message Franking: From Invisible Salamanders to Encryptment

Dodis, *G.*, Ristenpart, Woodage. IACR CRYPTO18. https://eprint.iacr.org/2019/016

Partitioning Oracle Attacks

Len, *G.,* Ristenpart. In submission.

Overview

Describe “attacker-controlled keys” setting + examples, explain ***committing*** security property AE needs

Many widely-used AE schemes are ***not*** committing: can break for GCM, ChaCha20/Poly1305, others

Attacks resulting from non-committing AE:

- Inconsistent plaintexts in multi-receiver encryption
- Invisible salamanders in Facebook’s message franking
- Key recovery via partitioning oracle attacks

Based on these research papers:

Message Franking via Committing Authenticated Encryption

G., Lu, Ristenpart. IACR CRYPTO17. https://eprint.iacr.org/2017/664

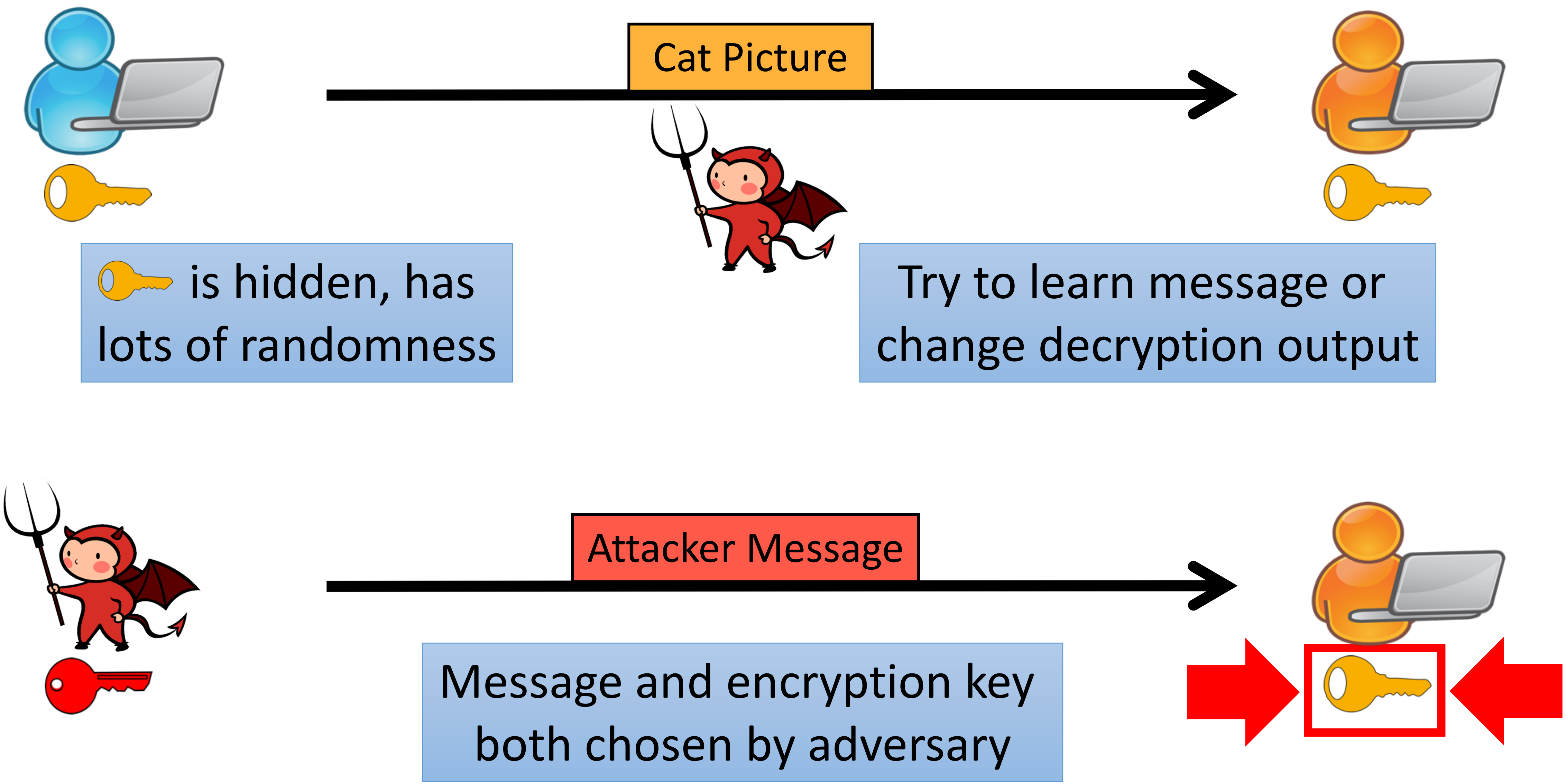
Fast Message Franking: From Invisible Salamanders to Encryptment

Dodis, *G.*, Ristenpart, Woodage. IACR CRYPTO18. https://eprint.iacr.org/2019/016

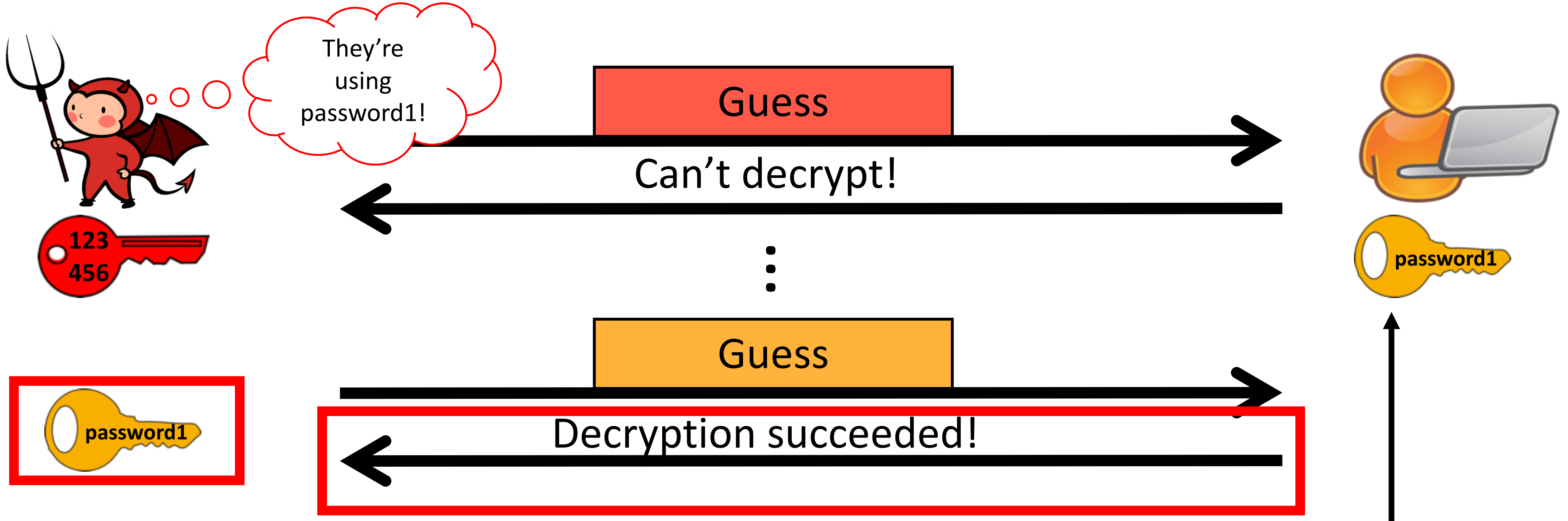
Partitioning Oracle Attacks

Len, *G.,* Ristenpart. In submission.

Attacker-Controlled Keys



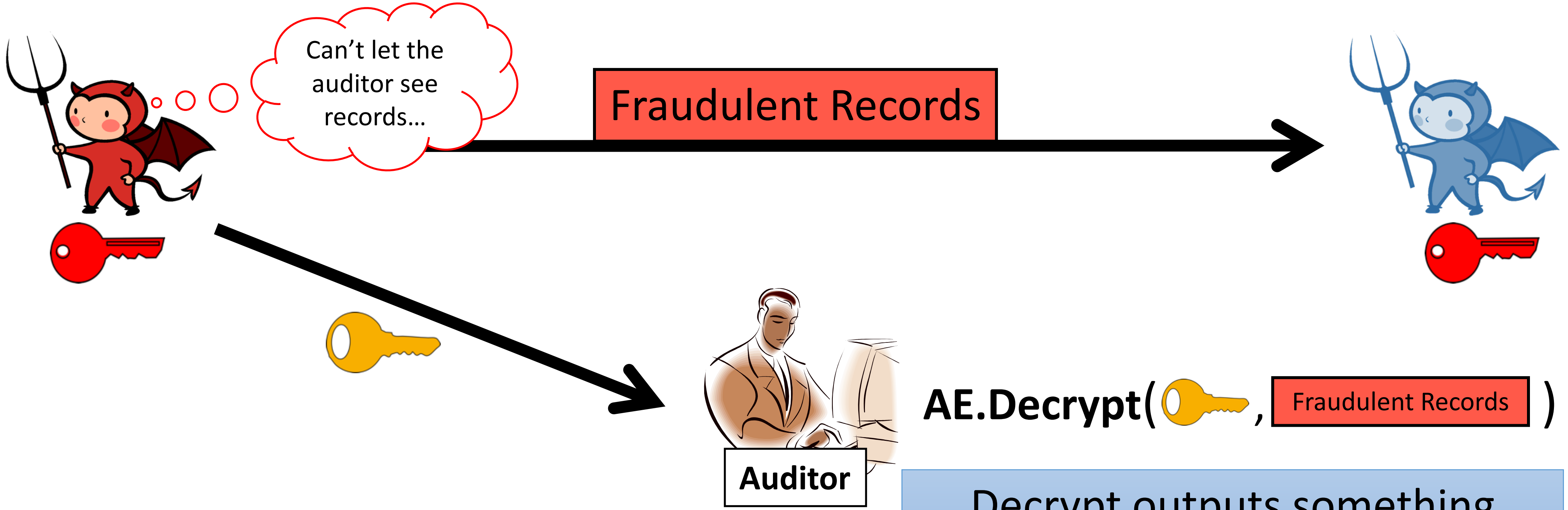
Example: Password-based AE



If attacker doesn't know decryption key, can learn using (online) brute-force attack

Brute-force feasible if key is not very random (e.g. password/PIN) or if side channel leaks key bits

Example: Reporting Plaintexts



Decrypt outputs something other than fraudulent records

Attacker chooses encryption and decryption key: tries to lie about ciphertext contents

Committing Security for AE

Useful to imagine AE as a lockbox

Intuition holds for hidden random key:

- Can't see inside (confidentiality)
- Can't change contents (integrity)

No matter the key, only one thing can come out when it's unlocked

Committing Security for AE

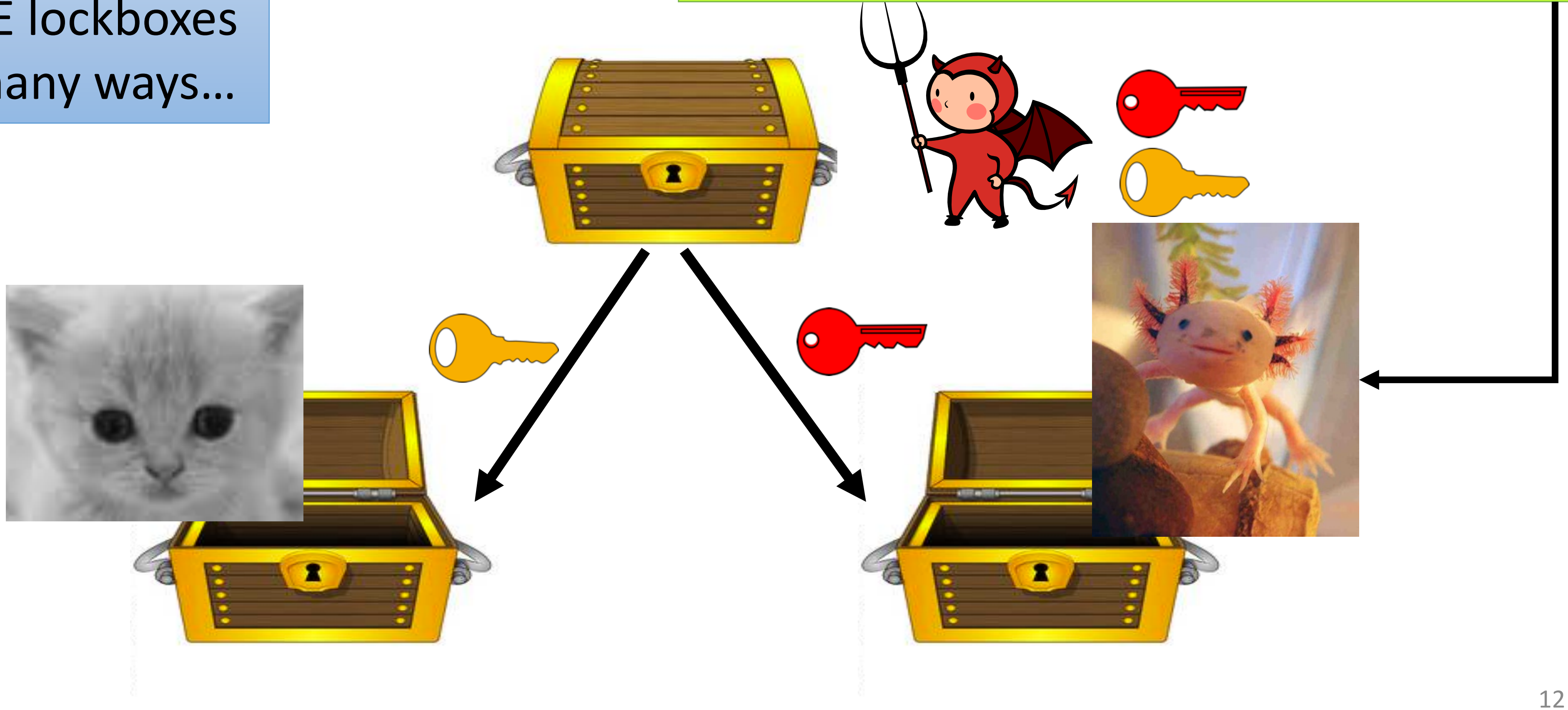
Useful to imagine AE as a lockbox

Intuition holds for hidden random key:

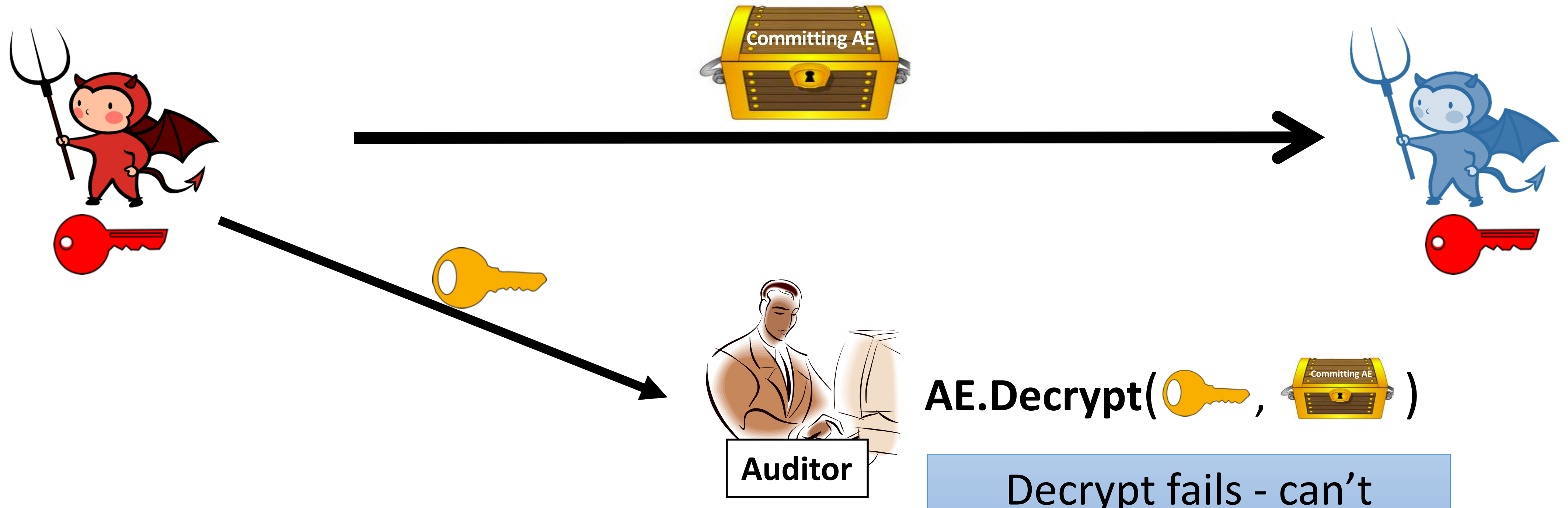
- Can’t see inside (confidentiality)
- Can’t change contents (integrity)

Without this, AE lockboxes could unlock many ways…

Committing security ***binds*** attacker to a single AE decryption, prevents ***invisible salamanders*** in ciphertexts



Reporting Salamanders



Decrypt fails - can’t report salamanders

If AE is committing, attacker can’t lie about plaintext by choosing different key

Overview

Describe "attacker-controlled keys" setting + examples, explain ***committing*** security property AE needs

Many widely-used AE schemes are ***not*** committing: can break for GCM, ChaCha20/Poly1305, others

Attacks resulting from non-committing AE:

- Inconsistent plaintexts in multi-receiver encryption
- Invisible salamanders in Facebook's message franking
- Key recovery via partitioning oracle attacks

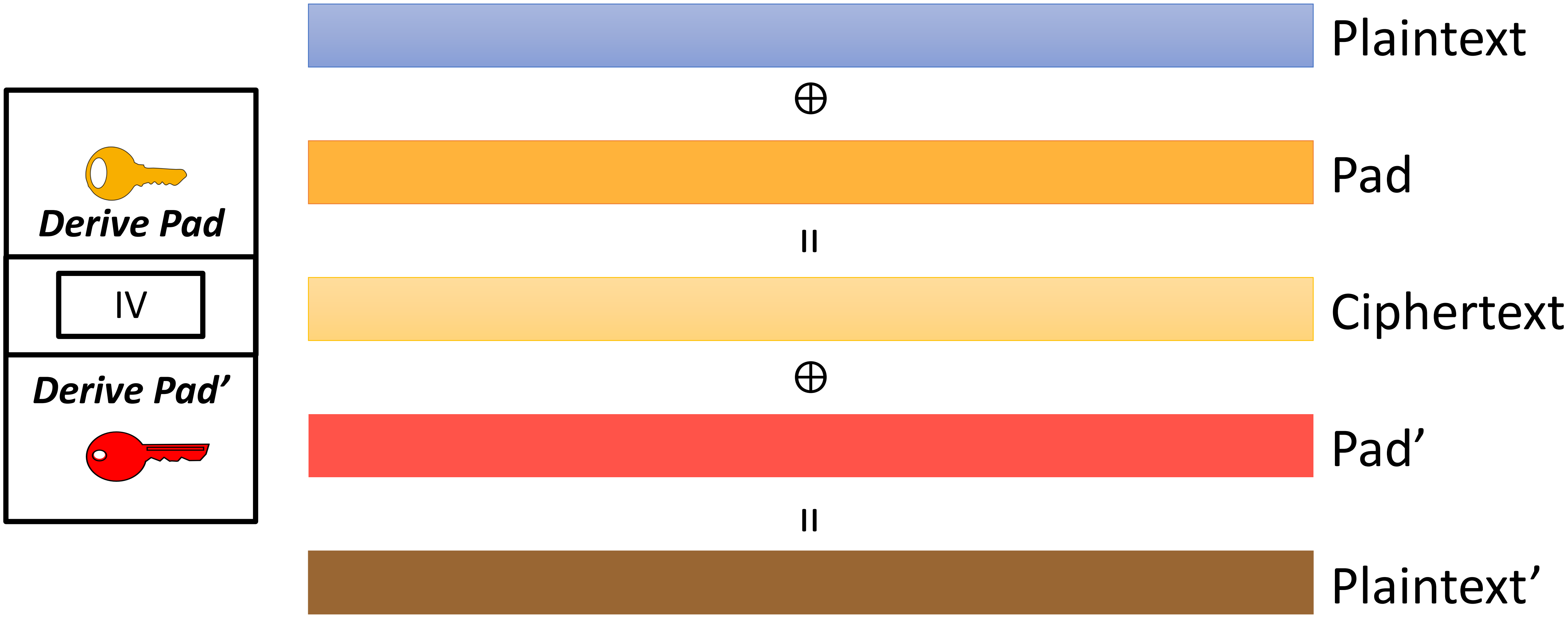
Based on these research papers:

Message Franking via Committing Authenticated Encryption
G., Lu, Ristenpart. IACR CRYPTO17. https://eprint.iacr.org/2017/664

Fast Message Franking: From Invisible Salamanders to Encryptment
Dodis, *G.*, Ristenpart, Woodage. IACR CRYPTO18. https://eprint.iacr.org/2019/016

Partitioning Oracle Attacks
Len, *G.,* Ristenpart. In submission.

Invisible Salamanders for CTR Mode



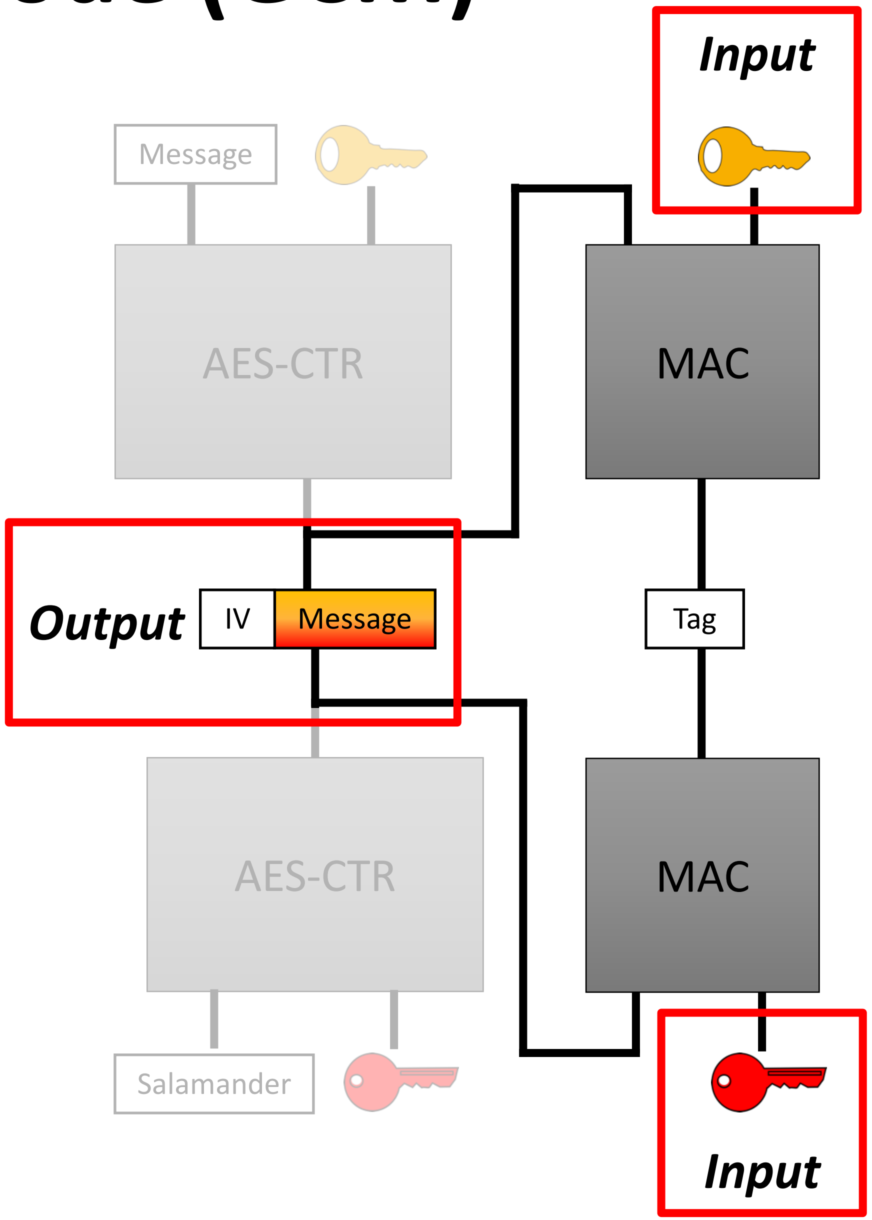
Galois/Counter Mode (GCM)

GCM is a fast, modern AE.
NIST/IEEE/ISO standard

Uses AES-CTR + message authentication code (MAC) to prevent tampering

Decryption recomputes, checks tag, fails if tags do not match

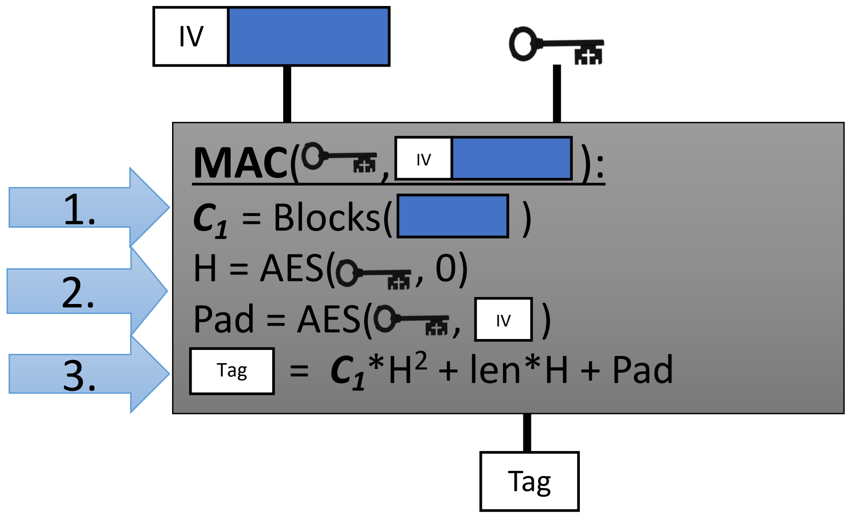
To get invisible salamanders for GCM, need to find IV Message with same MAC output Tag for [yellow key] and [red key]



Colliding GCM's MAC ⚠

MAC is polynomial evaluation + XOR.
Fast but not collision-resistant (cf. SHA-256)

1. Split ciphertext into blocks (coefficients)
2. Compute hash point (H) and pad (Pad)
3. Evaluate polynomial at H, then XOR Pad
   ('len' is encoded ciphertext length)



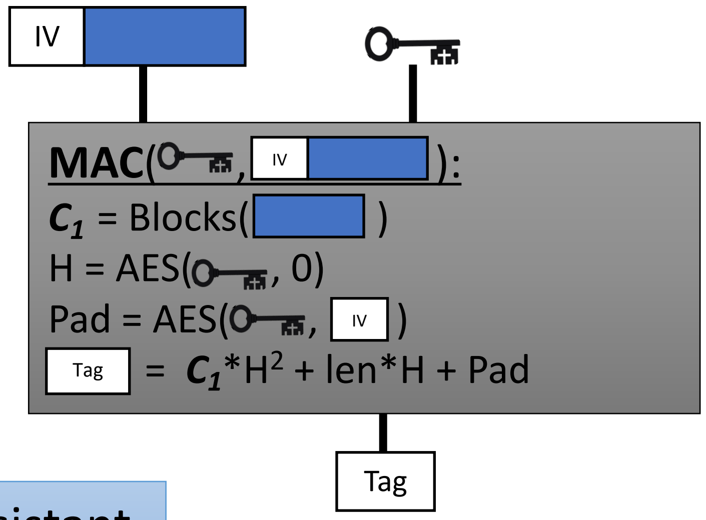
Colliding GCM's MAC ⚠

Tag is a "simple" algebraic function of ciphertext:
solve one equation to collide for two keys

1. Choose any [IV]
2. For key (yellow key), derive H_1, Pad_1
3. For key (red key), derive H_2, Pad_2
4. Set tag equations equal, solve for C_1:

$$C_1 * H_1^2 + len * H_1 + Pad_1$$
$$= C_1 * H_2^2 + len * H_2 + Pad_2$$

5. Let [Message] be C_1, recompute [Tag]
6. Output [IV][Message][Tag]



Not true for collision-resistant hashes like SHA-256

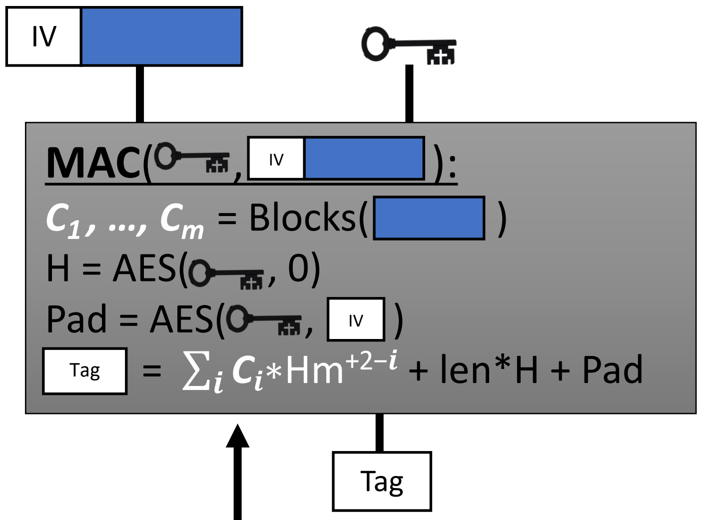
$$C_1 * (H_1^2 + H_2^2) = len * (H_1 + H_2) + Pad_1 + Pad_2 \qquad 2x = 6$$

$$C_1 = [len * (H_1 + H_2) + Pad_1 + Pad_2] * (H_1^2 + H_2^2)^{-1} \qquad x = 3$$

One equation, one unknown!

From Two Keys to Many

Colliding GCM's MAC on two keys is pretty easy.
Can even collide many (>>2) keys: use *interpolation*



$$\begin{bmatrix} H_1^{m+1} & \cdots & H_1^2 \\ \vdots & \ddots & \vdots \\ H_k^{m+1} & \cdots & H_k^2 \end{bmatrix} \begin{bmatrix} C_1 \\ \vdots \\ C_m \end{bmatrix} = \begin{bmatrix} Tag + Pad_1 + lens * H_1 \\ \vdots \\ Tag + Pad_k + lens * H_k \end{bmatrix}$$

As many variables as ciphertext blocks: can solve when m ≥ k in $O(k^2)$ time

One equation per key

$$C_1 * H_1^2 + len * H_1 + Pad_1 = C_1 * H_2^2 + len * H_2 + Pad_2$$

$$C_1 * (H_1^2 + H_2^2) = len * (H_1 + H_2) + Pad_1 + Pad_2$$

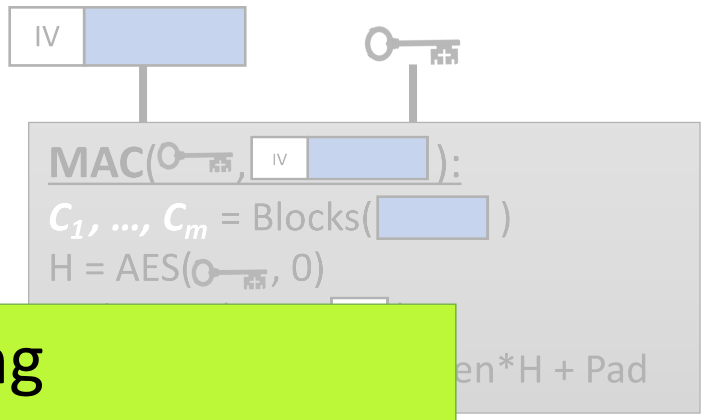
$$C_1 = [len * (H_1 + H_2) + Pad_1 + Pad_2] * (H_1^2 + H_2^2)^{-1}$$

Polynomial MACs are very common:
Poly1305 (libsodium, NaCL), GCM-SIV, etc.

</math>

Colliding GCM's MAC on two keys is pretty easy.
Can even collide many (>>2) keys: use *interpolation*

$$\begin{bmatrix} H_1^{m+1} & \cdots & H_1^2 \\ \vdots & \ddots & \vdots \\ H_k^{m+1} & & \end{bmatrix} \begin{bmatrix} C_1 \\ \vdots \\ \end{bmatrix} = \begin{bmatrix} Tag + Pad_1 + lens * H_1 \\ \vdots \\ \end{bmatrix}$$



1. Widely-used AE schemes are not committing
   (though they are fine for use in TLS/IPSec/SSH!)
2. Crafting invisible salamanders for them is easy
3. One ciphertext can have 100,000s+ invisible salamanders
   (E.g., my colleague generated one with 131,072 correct decryptions)

As man…
can s…

$C_1*H_1^2 + len*H_1 + Pad_1 = C_1*H_2^2 + len*H_2 + Pad_2$

$C_1*(H_1^2 + H_2^2) = len*(H_1 + H_2) + Pad_1 + Pad_2$

$C_1 = [len*(H_1 + H_2) + Pad_1 + Pad_2]*(H_1^2 + H_2^2)^{-1}$

Polynomial MACs are very common:
Poly1305 (libsodium, NaCL), GCM-SIV, etc.

Overview

Describe "attacker-controlled keys" setting + examples, explain ***committing*** security property AE needs

Many widely-used AE schemes are ***not*** committing: can break for GCM, ChaCha20/Poly1305, others

Attacks resulting from non-committing AE:

- Inconsistent plaintexts in multi-receiver encryption
- Invisible salamanders in Facebook's message franking
- Key recovery via partitioning oracle attacks

Based on these research papers:

Message Franking via Committing Authenticated Encryption

G., Lu, Ristenpart. IACR CRYPTO17. https://eprint.iacr.org/2017/664

Fast Message Franking: From Invisible Salamanders to Encryptment

Dodis, *G.*, Ristenpart, Woodage. IACR CRYPTO18. https://eprint.iacr.org/2019/016

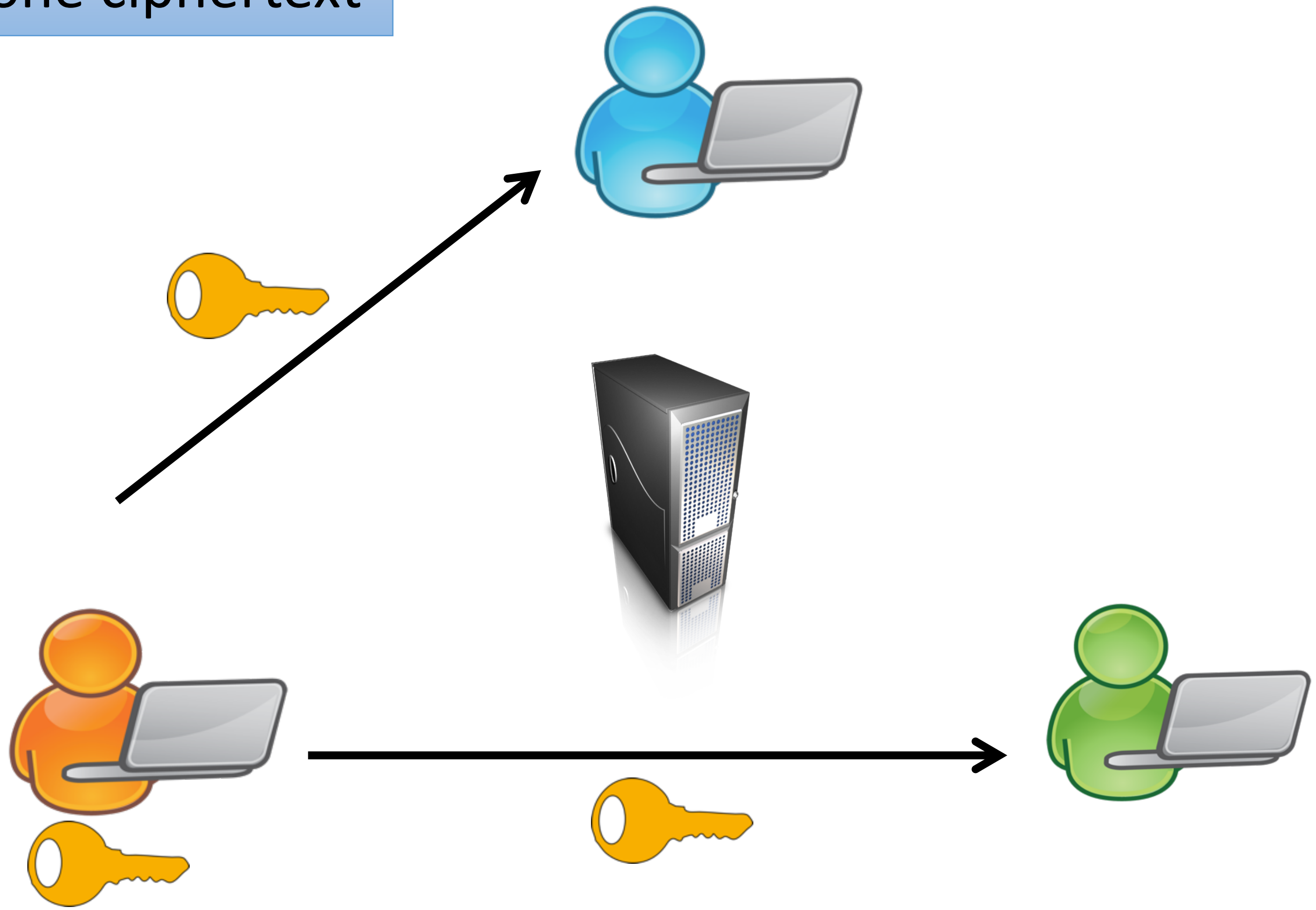
Partitioning Oracle Attacks

Len, *G.,* Ristenpart. In submission.

Multi-Receiver Encryption

In group messaging applications, senders must encrypt and send messages to group

Keys shared pairwise; only one ciphertext

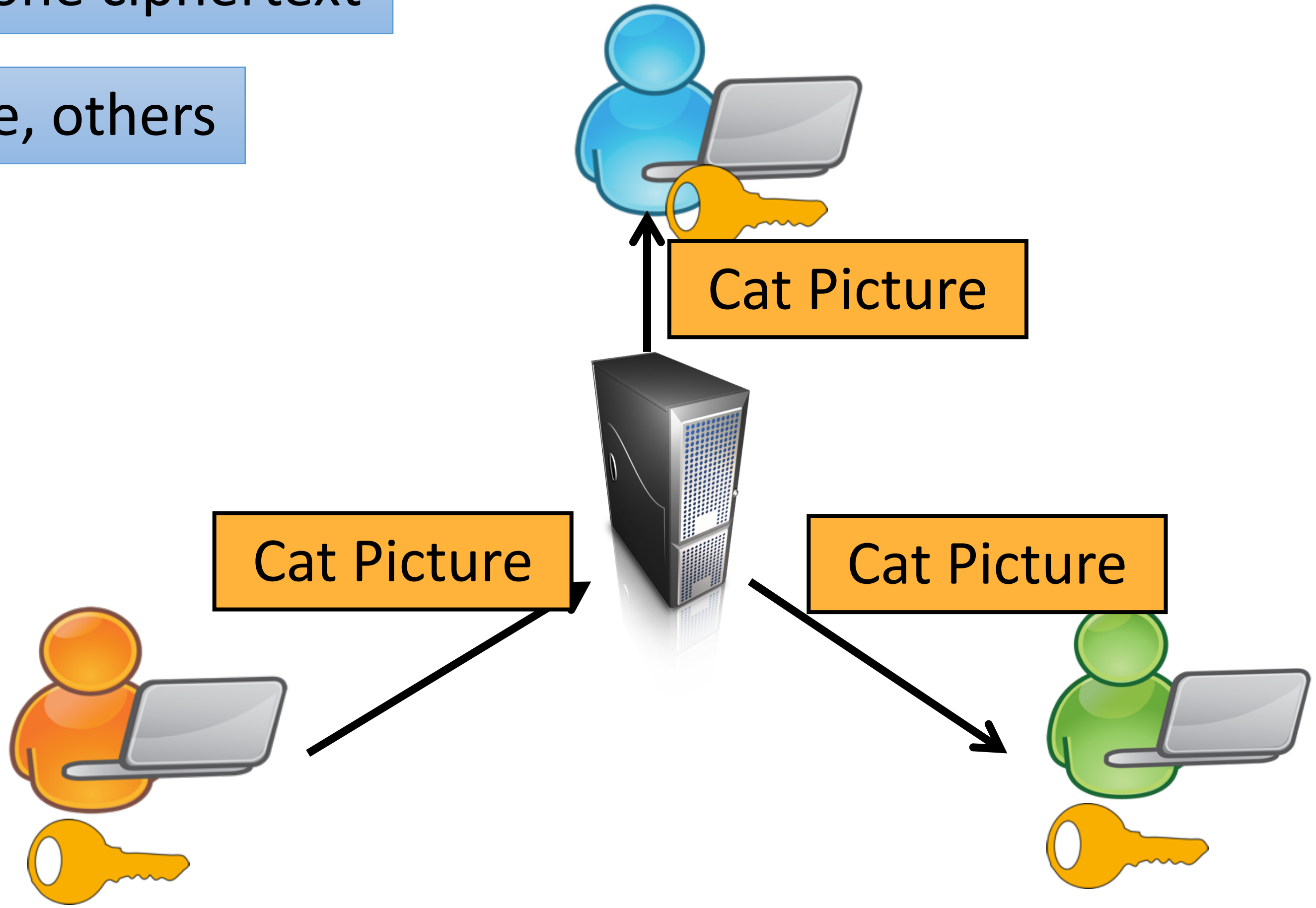


Multi-Receiver Encryption

In group messaging applications, senders must encrypt and send messages to group

Keys shared pairwise; only one ciphertext

Used by Whatsapp, Keybase, others



Multi-Receiver Encryption

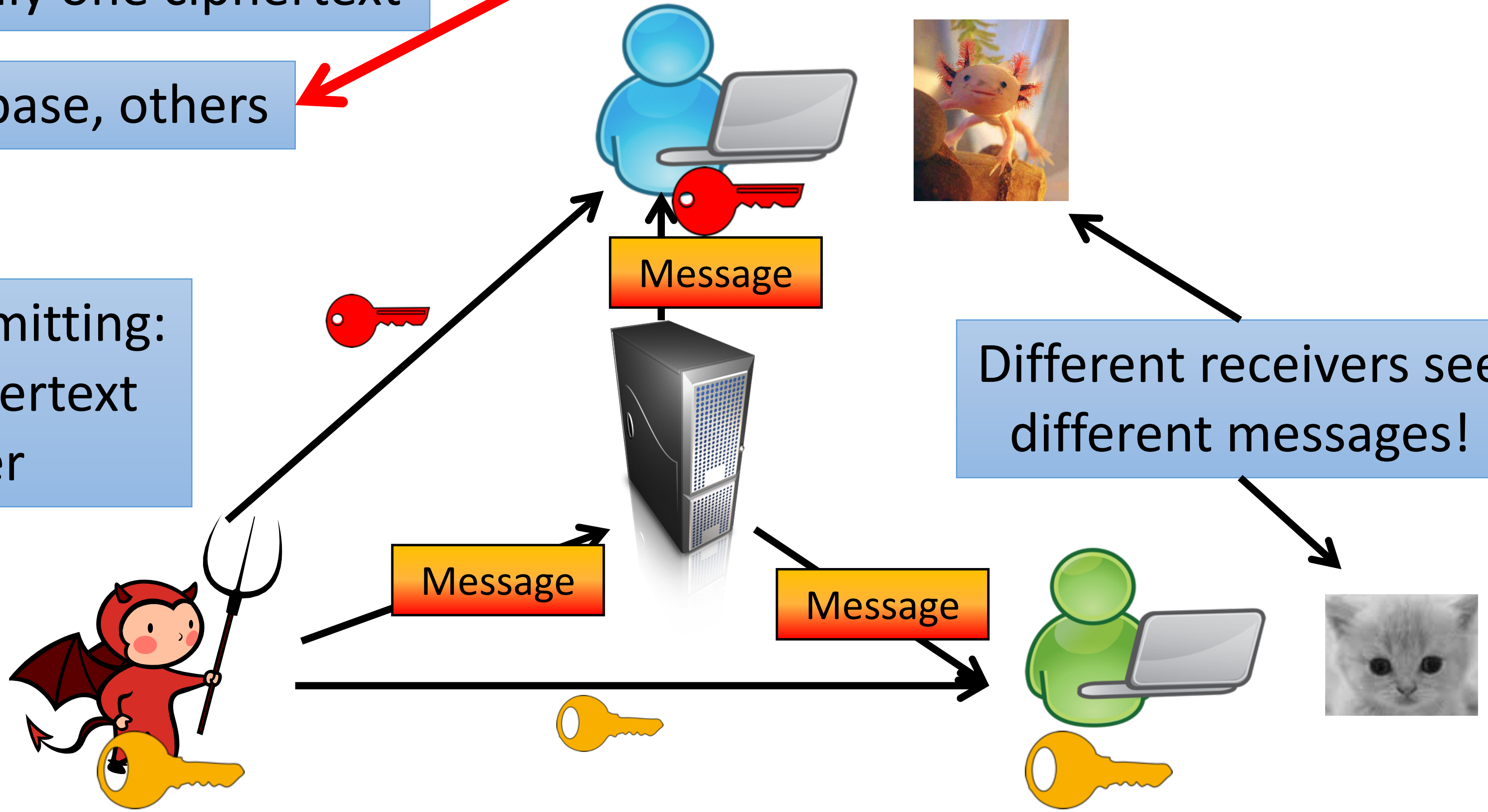
In group messaging applications, senders must encrypt and send messages to group

Keys shared pairwise; only one ciphertext

Used by Whatsapp, Keybase, others

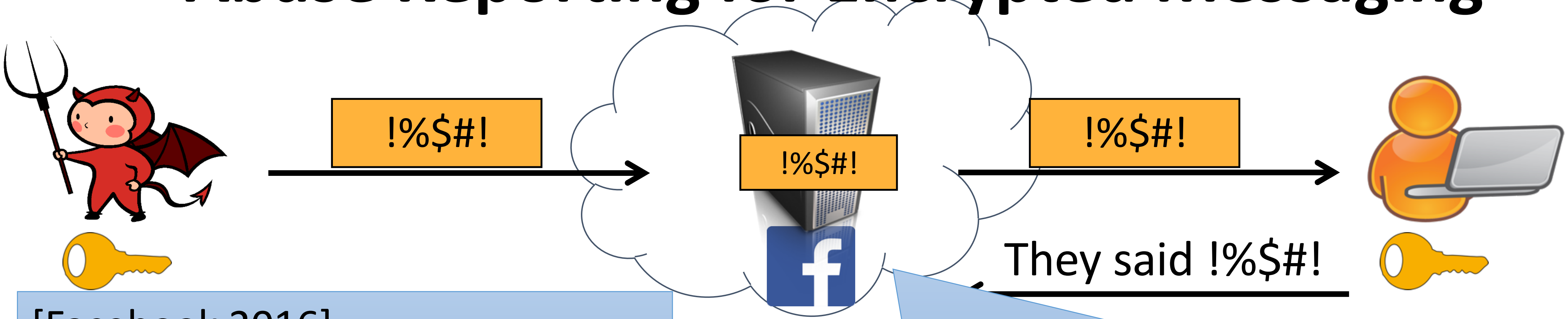
Theoretical attack. Unclear if these are vulnerable (homework!)

If encryption is not committing: send different keys, ciphertext with invisible salamander



Different receivers see different messages!

Abuse Reporting for Encrypted Messaging

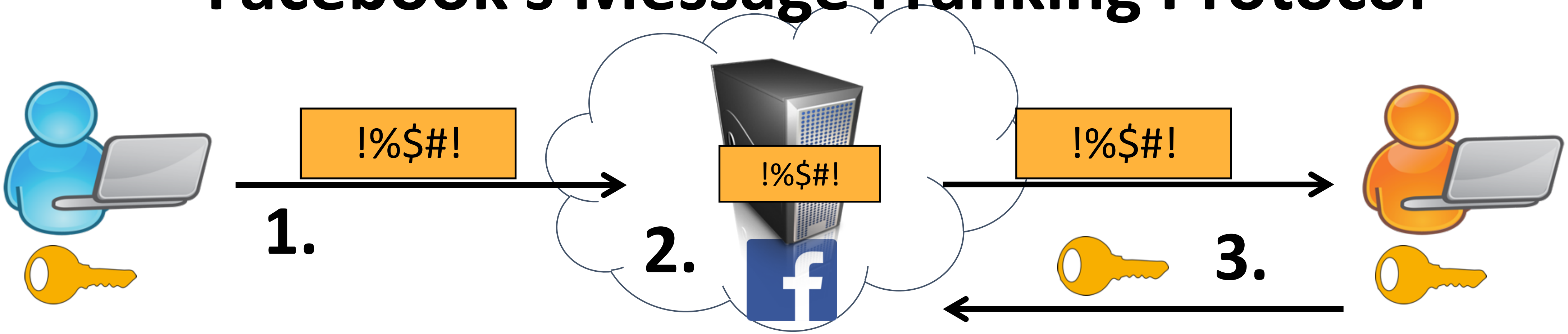


[Facebook 2016]:
Reporting via ad-hoc proof of contents: ***message franking***

Service can't tell if "!%$#!" was sent.
Need secure reporting of message content

Attack: use of non-committing encryption means any sender could have sent unreportable content

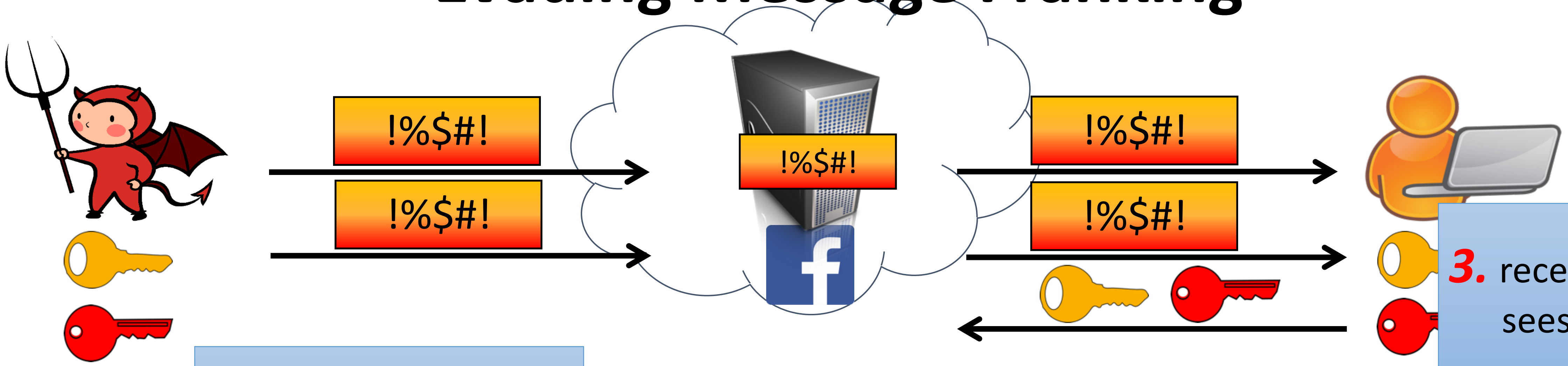
Facebook's Message Franking Protocol



Message franking:

1. GCM Encrypt w/ **sender-chosen** per-message key
2. Facebook stores, forwards ciphertexts
3. Report all recent keys, FB decrypts **unique** ciphertexts

Evading Message Franking



3. receiver sees both

2. Send !%$#! twice with [yellow key] / [red key]

4. Only the innocuous image appears in report to Facebook!

1. Craft ***GCM*** ciphertext !%$#! :

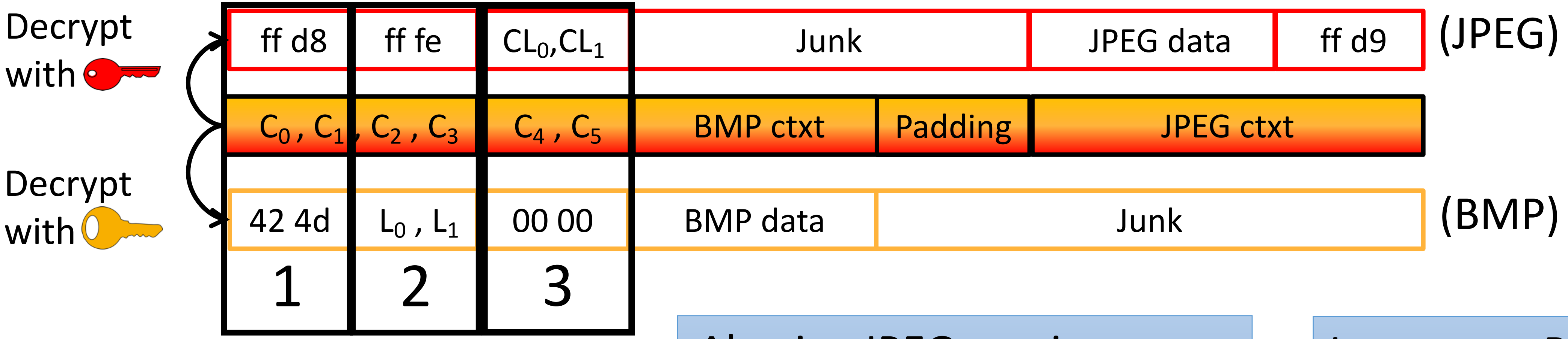
- Decrypts under [yellow key] to innocuous image
- Decrypts under [red key] to abusive image

Message franking:

1. Encrypt w/**sender-chosen** per-message key
2. Facebook stores, forwards ciphertexts
3. Report all recent keys, FB decrypts **unique** ciphertexts

Crafting the Ciphertext

Proof of concept: ciphertext which decrypts to valid JPEG under [red key] and valid BMP under [yellow key]

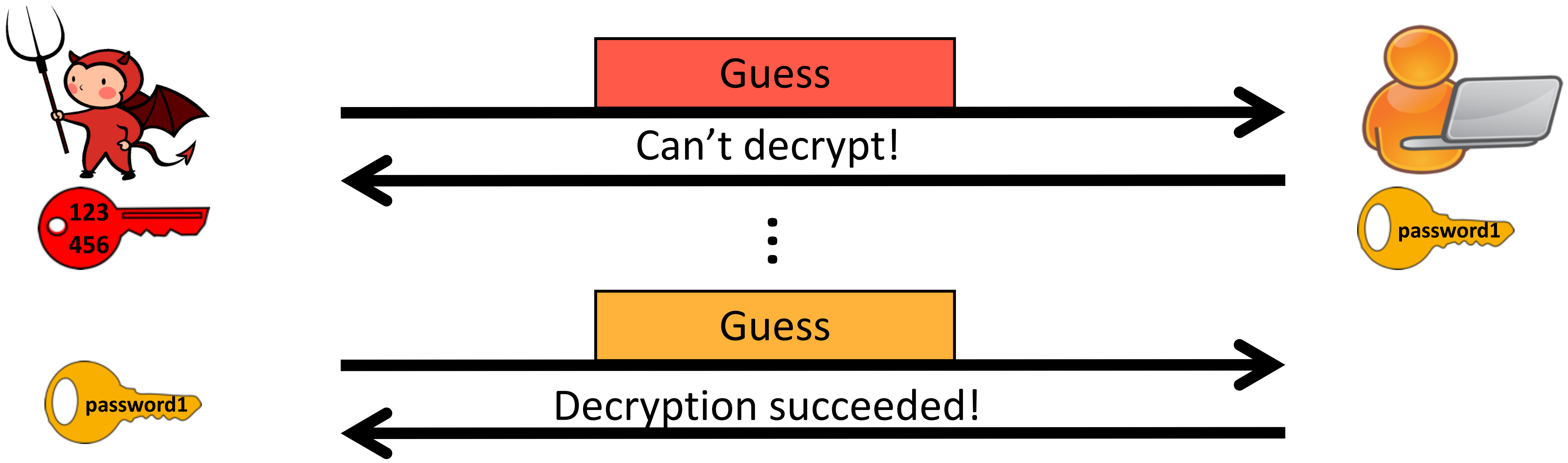


1. Image headers
2. BMP length and comment header
3. Comment length

Abusive JPEG receiver sees, but not in abuse report

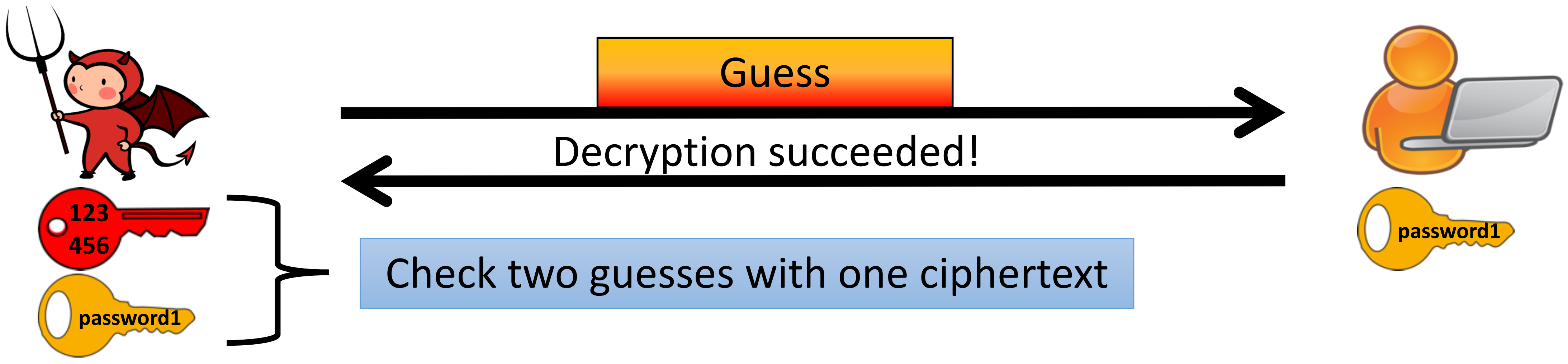
Innocuous BMP in abuse report

Partitioning Oracles



Use of non-committing AE with passwords can lead to partitioning oracles: speedup for online brute-force key recovery for AE

Partitioning Oracles



Use of non-committing AE with passwords can lead to partitioning oracles: speedup for online brute-force key recovery for AE

Worst-case *exponential* reduction in guesses!
E.g., one million passwords = only 20 guesses

Found partitioning oracle attacks on:
- Shadowsocks UDP proxying
- Incorrect OPAQUE prototypes

Latent vulnerabilities elsewhere

Preventing Invisible Salamanders

Committing AE schemes do exist!
E.g., CTR-then-HMAC (done correctly)

Not standardized, nor widely available in libraries
(also can be less efficient than non-committing AE)

Needed only if attacker-controlled keys are part of threat model

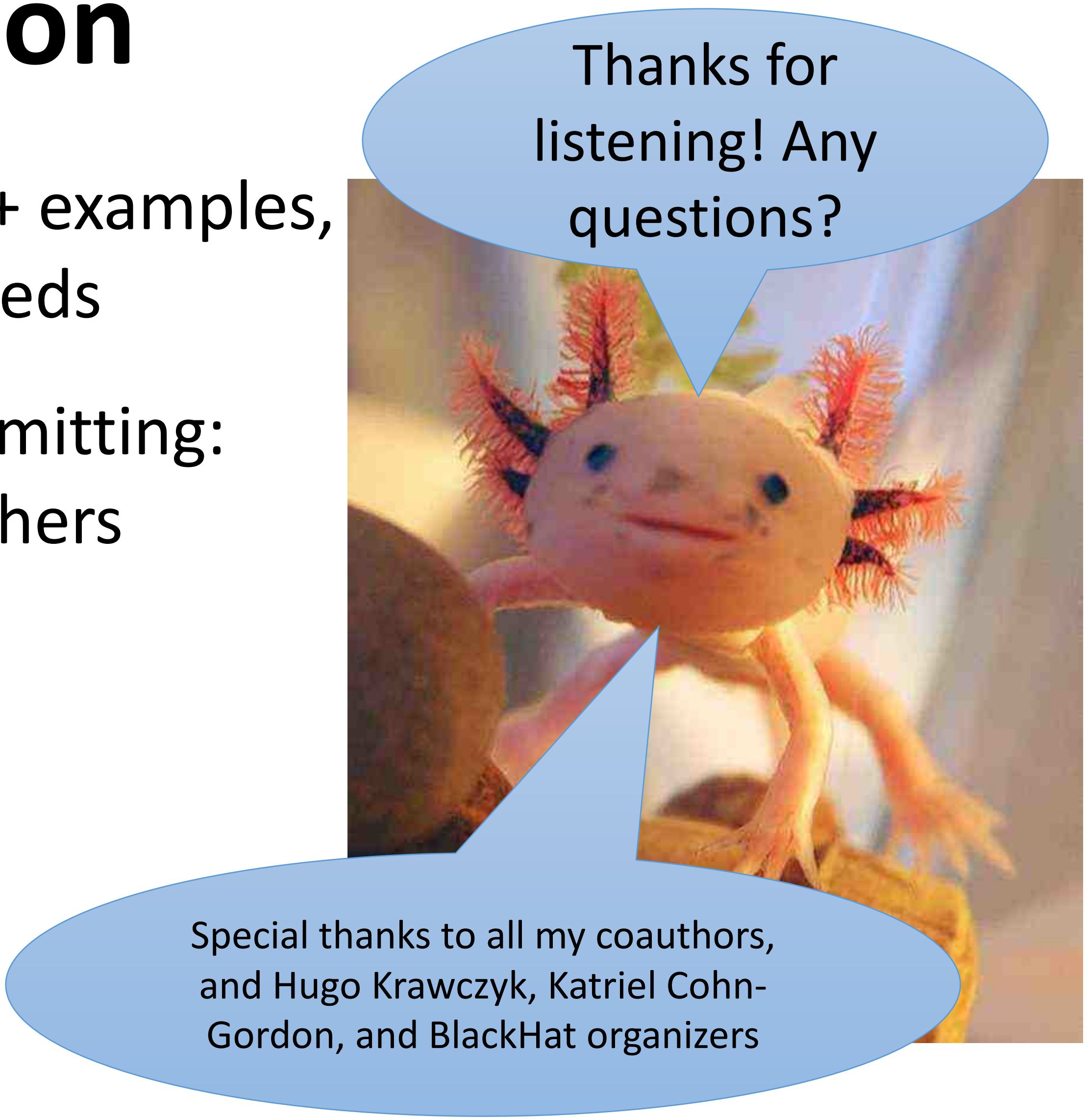
Conclusion

Describe “attacker-controlled keys” setting + examples, explain ***committing*** security property AE needs

Many widely-used AE schemes are ***not*** committing: can break for GCM, ChaCha20/Poly1305, others

Attacks resulting from non-committing AE:

- Inconsistent plaintexts in multi-receiver encryption
- Invisible salamanders in Facebook’s message franking
- Key recovery via partitioning oracle attacks



Based on these research papers:

Message Franking via Committing Authenticated Encryption

G., Lu, Ristenpart. IACR CRYPTO17. https://eprint.iacr.org/2017/664

Fast Message Franking: From Invisible Salamanders to Encryptment

Dodis, *G.*, Ristenpart, Woodage. IACR CRYPTO18. https://eprint.iacr.org/2019/016

Partitioning Oracle Attacks

Len, *G.,* Ristenpart. In submission.