Ask Your Cryptographer if Context-Committing AEAD Is Right for You

Mihir Bellare, John Chan, Paul Grubbs, Viet Tung Hoang,
Sanketh Menda, Julia Len, Thomas Ristenpart, and Phillip Rogaway

Authenticated Encryption with Associated Data (AEAD)

Sender

Recipient

Authenticated Encryption with Associated Data (AEAD)

Context:

- Key K
- Nonce N (e.g., random IV or counter)
- Associated data A (e.g., network header)

K, **N**, **A**

Sender

Shared out of band.

K, **N**, **A**

Recipient

Authenticated Encryption with Associated Data (AEAD)

Context:
- Key K
- Nonce N (e.g., random IV or counter)
- Associated data A (e.g., network header)

Shared out of band.

K, **N**, **A**

Sender

C = AEAD.Enc(K,N,A,M)

K, **N**, **A**

Recipient

Authenticated Encryption with Associated Data (AEAD)

Context:
- Key K
- Nonce N (e.g., random IV or counter)
- Associated data A (e.g., network header)

Shared out of band.

K, N, A

Sender

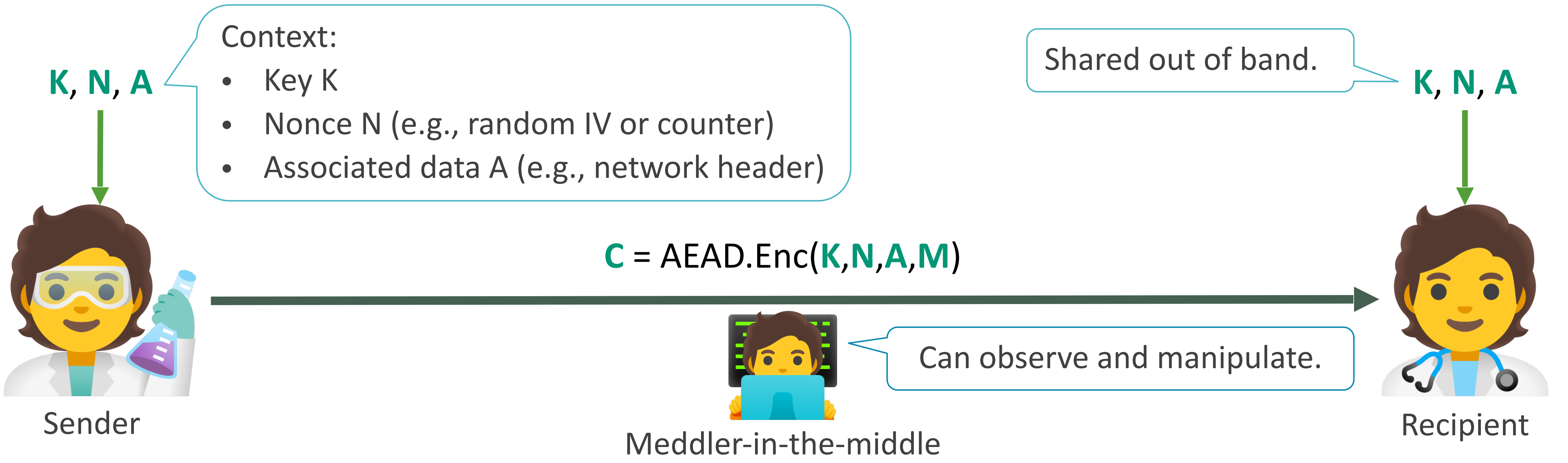
C = AEAD.Enc(K,N,A,M)

K, N, A

Recipient

Standardized
1. AES-GCM
2. ChaCha20/Poly1305
3. AES-GCM-SIV

Authenticated Encryption with Associated Data (AEAD)



Standardized

1. AES-GCM
2. ChaCha20/Poly1305
3. AES-GCM-SIV

Provably Secure

1. Confidentiality
2. Authenticity

Recent Attacks on AEAD

Recent Attacks on AEAD

Fast Message Franking:
From Invisible Salamanders to Encryptment*

Yevgeniy Dodis[1], Paul Grubbs[2,†], Thomas Ristenpart[2], Joanne Woodage[3,†]

[1] New York University [2] Cornell Tech

[3] Royal Holloway, University of London

Abstract

Message franking enables cryptographically verifiable reporting of abusive content in end-to-end encrypted messaging. Grubbs, Lu, and Ristenpart recently formalized the needed underlying primitive, what they call compactly committing authenticated encryption (AE), and analyzed the security of a number of approaches. But all known secure schemes are still slow compared to the fastest standard AE schemes. For this reason Facebook Messenger uses AES-GCM for franking of attachments such as images or videos.

We show how to break Facebook's attachment franking scheme: a malicious user can send an objectionable image to a recipient but that recipient cannot report it as abuse. The core problem stems from use of fast but non-committing AE, and so we build the fastest compactly committing AE schemes to date. To do so we introduce a new primitive, called encryptment, which captures the essential properties needed. We prove that, unfortunately, schemes with performance profile similar to AES-GCM won't work. Instead, we show how to efficiently transform Merkle-Damgärd-style hash functions into secure encryptments, and how to efficiently build compactly committing AE from encryptment. Ultimately our main construction allows franking using just a single computation of SHA-256 or SHA-3. Encryptment proves useful for a variety of other applications, such as remotely keyed AE and concealments, and our results imply the first single-pass schemes in these settings as well.

Recent Attacks on AEAD

Fast Message Franking: From Invisible Salamanders to Encryptment*

Yevgeniy Dodis[1], Paul Grubbs[2,†], Thomas Ristenpart[2], Joanne Woodage[3,†]

[1] New York University [2] Cornell Tech

[3] Royal Holloway, University of London

Abstract

Message franking enables cryptographically verifiable reporting of abusive content in end-to-end encrypted messaging. Grubbs, Lu, and Ristenpart recently formalized the needed underlying primitive, what they call compactly committing authenticated encryption (AE), and analyzed the security of a number of approaches. But all known secure schemes are still slow compared to the fastest standard AE schemes. For this reason Facebook Messenger uses AES-GCM for franking of attachments such as images or videos.

We show how to break Facebook's attachment franking scheme: a malicious user can send an objectionable image to a recipient but that recipient cannot report it as abuse. The core problem stems from use of fast but non-committing AE, and so we build the fastest compactly committing AE schemes to date. To do so we introduce a new primitive, called encryptment, which captures the essential properties needed. We prove that, unfortunately, schemes with performance profile similar to AES-GCM won't work. Instead, we show how to efficiently transform Merkle-Damgård-style hash functions into secure encryptments, and how to efficiently build compactly committing AE from encryptment. Ultimately our main construction allows franking using just a single computation of SHA-256 or SHA-3. Encryptment proves useful for a variety of other applications, such as remotely keyed AE and concealments, and our results imply the first single-pass schemes in these settings as well.

Key Material A Blog about Security and Cryptography About Contact Search

CRYPTOGRAPHY

Invisible Salamanders in AES-GCM-SIV

By Sophie Schmieg September 7, 2020 No Comments

By now, many people have run across the Invisible Salamander paper about the interesting property of AES-GCM, that allows an attacker to construct a ciphertext that will decrypt with a valid tag under two different keys, provided both keys are known to the attacker. On some level, finding properties like this isn't too surprising: AES-GCM was designed to be an AEAD, and nowhere in the AEAD definition does it state anything about what attackers with access to the keys can do, since the usual assumption is that attackers don't have that access, since any Alice-Bob-Message model would be meaningless in that scenario.

Recent Attacks on AEAD

Fast Message Franking:
From Invisible Salamanders to Encryptment*

Yevgeniy Dodis[1], Paul Grubbs[2,†], Thomas Ristenpart[2], Joanne Woodage[3,†]

[1] New York University [2] Cornell Tech

[3] Royal Holloway, University of London

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE
ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL
ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

How to Abuse and Fix Authenticated Encryption Without Key Commitment

Ange Albertini[1], Thai Duong[1], Shay Gueron[2,3], Stefan Kölbl[1], Atul Luykx[1], and Sophie Schmieg[1]

[1]Security Engineering Research, Google
[2]University of Haifa
[3]Amazon

Abstract

Authenticated encryption (AE) is used in a wide variety of applications, potentially in settings for which it was not originally designed. Recent research tries to understand what happens when AE is not used as prescribed by its designers. A question given relatively little attention is whether an AE scheme guarantees "key commitment": ciphertext should only decrypt to a valid plaintext under the key used to generate the

is insecure when they see a proof-of-concept exploit. Similar efforts are deemed necessary to demonstrate the exploitability of cryptographic algorithms such as SHA-1 [SBK+17].

The vast majority of applications should default to using authenticated encryption (AE) [BN00, KY00], a well-studied primitive which avoids the pitfalls of unauthenticated SKE with relatively small performance overhead. AE schemes are used in widely adopted protocols like TLS [Res18], standard-

Key Material A Blog about Security and Cryptography About Contact Search

CRYPTOGRAPHY

Invisible Salamanders in AES-GCM-SIV

By Sophie Schmieg September 7, 2020 No Comments

By now, many people have run across the Invisible Salamander paper about the interesting property of AES-GCM, that allows an attacker to construct a ciphertext that will decrypt with a valid tag under two different keys, provided both keys are known to the attacker. On some level, finding properties like this isn't too surprising: AES-GCM was designed to be an AEAD, and nowhere in the AEAD definition does it state anything about what attackers with access to the keys can do, since the usual assumption is that attackers don't have that access, since any Alice-Bob-Message model would be meaningless in that scenario.

Recent Attacks on AEAD

Fast Message Franking:
From Invisible Salamanders to Encryptment*

Yevgeniy Dodis[1], Paul Grubbs[2,†], Thomas Ristenpart[2], Joanne Woodage[3,†]

[1] New York University [2] Cornell Tech

[3] Royal Holloway, University of London

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE | ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL | ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

How to Abuse and Fix Authenticated Encryption Without Key Commitment

Ange Albertini[1], Thai Duong[1], Shay Gueron[2,3], Stefan Kölbl[1], Atul Luykx[1], and Sophie Schmieg[1]

[1]Security Engineering Research, Google
[2]University of Haifa
[3]Amazon

Abstract

Authenticated encryption (AE) is used in a wide variety of applications, potentially in settings for which it was not originally designed. Recent research tries to understand what happens when AE is not used as prescribed by its designers. A question given relatively little attention is whether an AE scheme guarantees "key commitment": ciphertext should only decrypt to a valid plaintext under the key used to generate the

is insecure when they see a proof-of-concept exploit. Similar efforts are deemed necessary to demonstrate the exploitability of cryptographic algorithms such as SHA-1 [SBK+17].

The vast majority of applications should default to using authenticated encryption (AE) [BN00, KY00], a well-studied primitive which avoids the pitfalls of unauthenticated SKE with relatively small performance overhead. AE schemes are used in widely adopted protocols like TLS [Res18], standard-

Key Material A Blog about Security and Cryptography   About   Contact   Search

CRYPTOGRAPHY

Invisible Salamanders in AES-GCM-SIV

By Sophie Schmieg   September 7, 2020   No Comments

The proceedings version of this paper appears at USENIX Security 2021. This is the full version.

Partitioning Oracle Attacks

Julia Len   Paul Grubbs   Thomas Ristenpart

Cornell Tech

Abstract

In this paper we introduce *partitioning oracles*, a new class of decryption error oracles which, conceptually, take a ciphertext as input and output whether the decryption key belongs to some known subset of keys. We introduce the first partitioning oracles which arise when encryption schemes are not committing with respect to their keys. We detail novel adaptive chosen ciphertext attacks that exploit partitioning oracles to efficiently recover passwords and de-anonymize anonymous communications. The attacks utilize efficient key multi-collision algorithms — a cryptanalytic goal that we define — against widely used authenticated en-

This is perhaps because attacks exploiting lack of robustness have arisen in relatively niche applications like auction protocols [23] or recently as an integrity issue in moderation for encrypted messaging [22, 30].

We introduce partitioning oracle attacks, a new type of CCA. These are similar to previous attacks considered in the password-authenticated key exchange (PAKE) literature [11, 72, 98]; we provide a unifying attack framework that transcends PAKE and show partitioning oracle attacks that exploit weaknesses in widely used non-committing AEAD schemes. Briefly, a partitioning oracle arises when an adversary can: (1) efficiently craft ciphertexts that suc-

Recent Attacks on AEAD

Fast Message Franking:
From Invisible Salamanders to Encryptment*

Yevgeniy Dodis[1], Paul Grubbs[2,†], Thomas Ristenpart[2], Joanne Woodage[3,†]

[1] New York University [2] Cornell Tech

[3] Royal Hol

Key Material A Blog about Security and Cryptography About Contact Search

CRYPTOGRAPHY

Invisible Salamanders in AES-GCM-SIV

No Comments

These attacks work in new threat models!

ecurity 2021. This is the full version.

How to Abuse and Fix Authenticated Encryption Without Key Commitment

Ange Albertini[1], Thai Duong[1], Shay Gueron[2,3], Stefan Kölbl[1], Atul Luykx[1], and Sophie Schmieg[1]

[1]Security Engineering Research, Google
[2]University of Haifa
[3]Amazon

Abstract

Authenticated encryption (AE) is used in a wide variety of applications, potentially in settings for which it was not originally designed. Recent research tries to understand what happens when AE is not used as prescribed by its designers. A question given relatively little attention is whether an AE scheme guarantees "key commitment": ciphertext should only decrypt to a valid plaintext under the key used to generate the

is insecure when they see a proof-of-concept exploit. Similar efforts are deemed necessary to demonstrate the exploitability of cryptographic algorithms such as SHA-1 [SBK+17].

The vast majority of applications should default to using authenticated encryption (AE) [BN00, KY00], a well-studied primitive which avoids the pitfalls of unauthenticated SKE with relatively small performance overhead. AE schemes are used in widely adopted protocols like TLS [Res18], standard-

Partitioning Oracle Attacks

Julia Len Paul Grubbs Thomas Ristenpart

Cornell Tech

Abstract

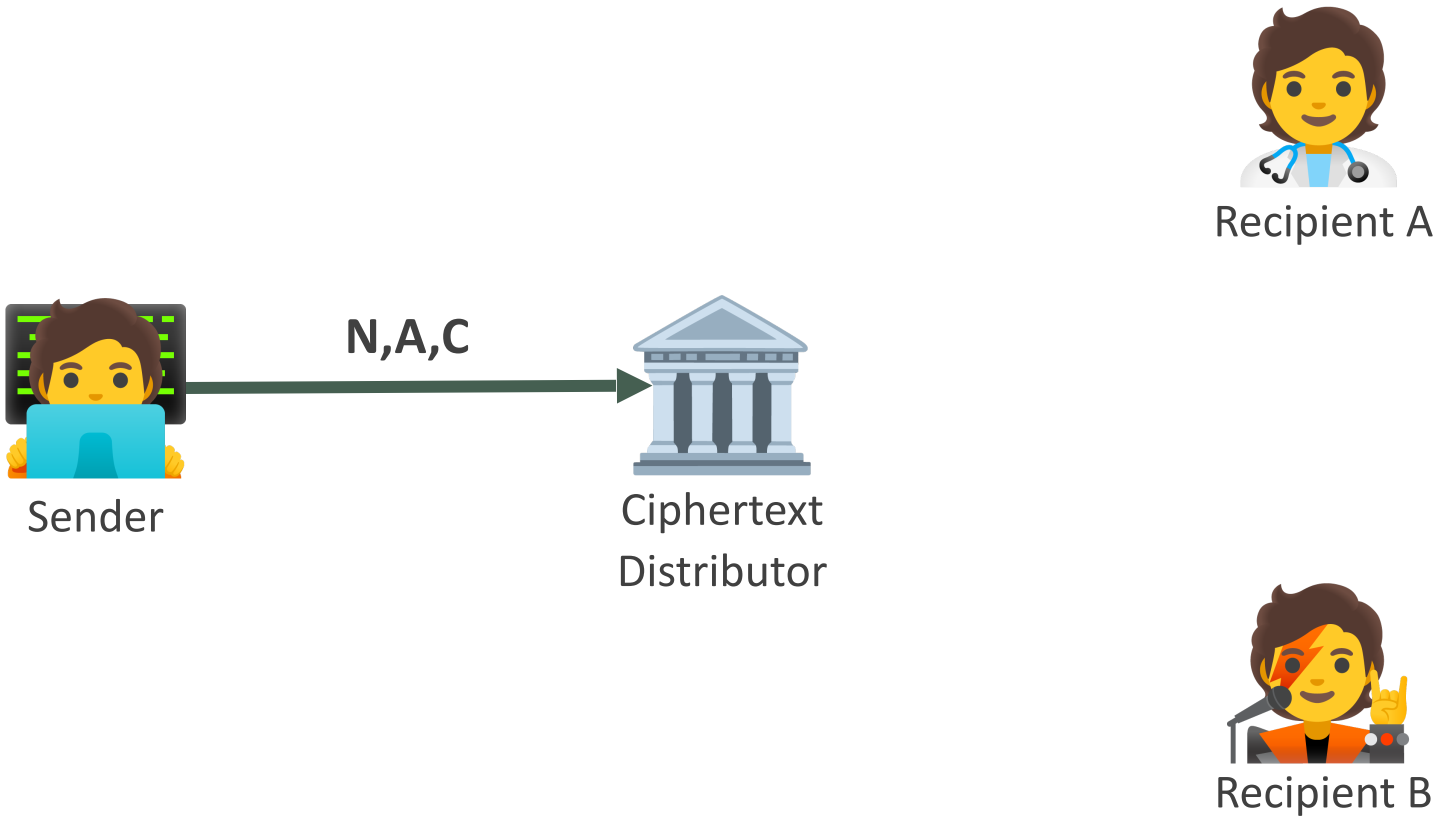
In this paper we introduce *partitioning oracles*, a new class of decryption error oracles which, conceptually, take a ciphertext as input and output whether the decryption key belongs to some known subset of keys. We introduce the first partitioning oracles which arise when encryption schemes are not committing with respect to their keys. We detail novel adaptive chosen ciphertext attacks that exploit partitioning oracles to efficiently recover passwords and de-anonymize anonymous communications. The attacks utilize efficient key multi-collision algorithms — a cryptanalytic goal that we define — against widely used authenticated en

This is perhaps because attacks exploiting lack of robustness have arisen in relatively niche applications like auction protocols [23] or recently as an integrity issue in moderation for encrypted messaging [22, 30].

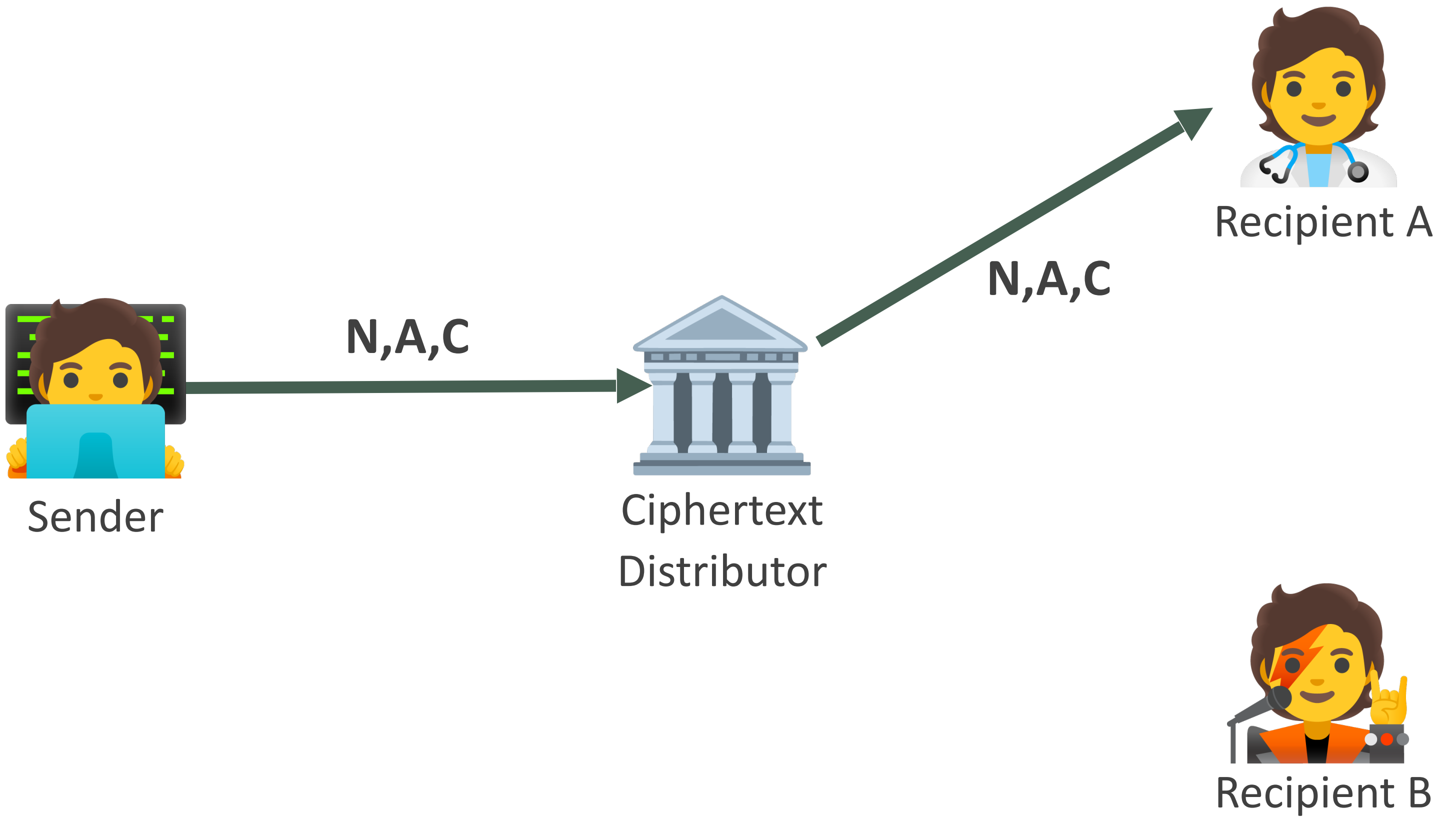
We introduce partitioning oracle attacks, a new type of CCA. These are similar to previous attacks considered in the password-authenticated key exchange (PAKE) literature [11, 72, 98]; we provide a unifying attack framework that transcends PAKE and show partitioning oracle attacks that exploit weaknesses in widely used non-committing AEAD schemes. Briefly, a partitioning oracle arises when an adversary can: (1) efficiently craft ciphertexts that suc-

Threat #1: Multi-Recipient Integrity [FOR17, GLR17]

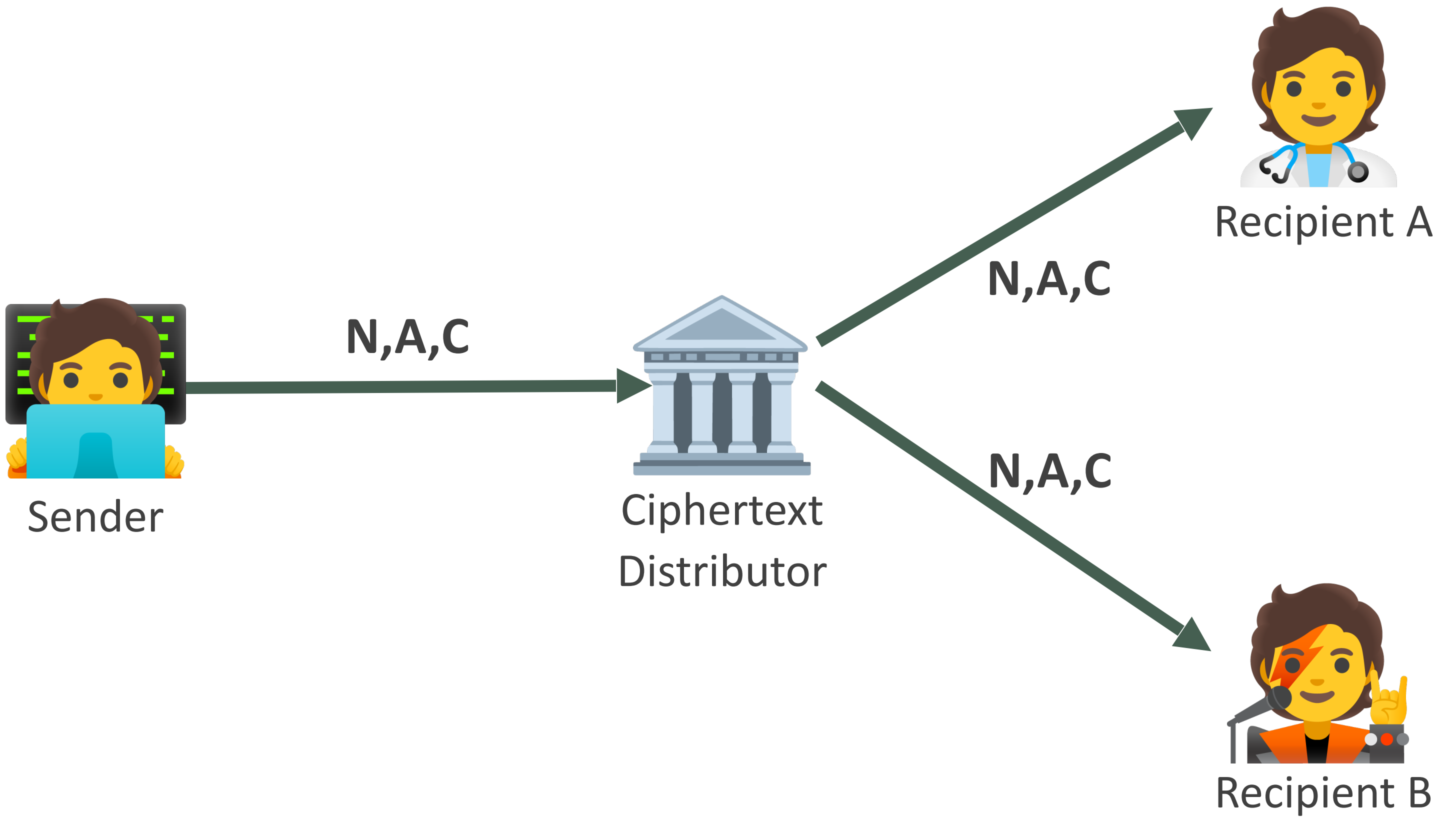
Threat #1: Multi-Recipient Integrity [FOR17, GLR17]



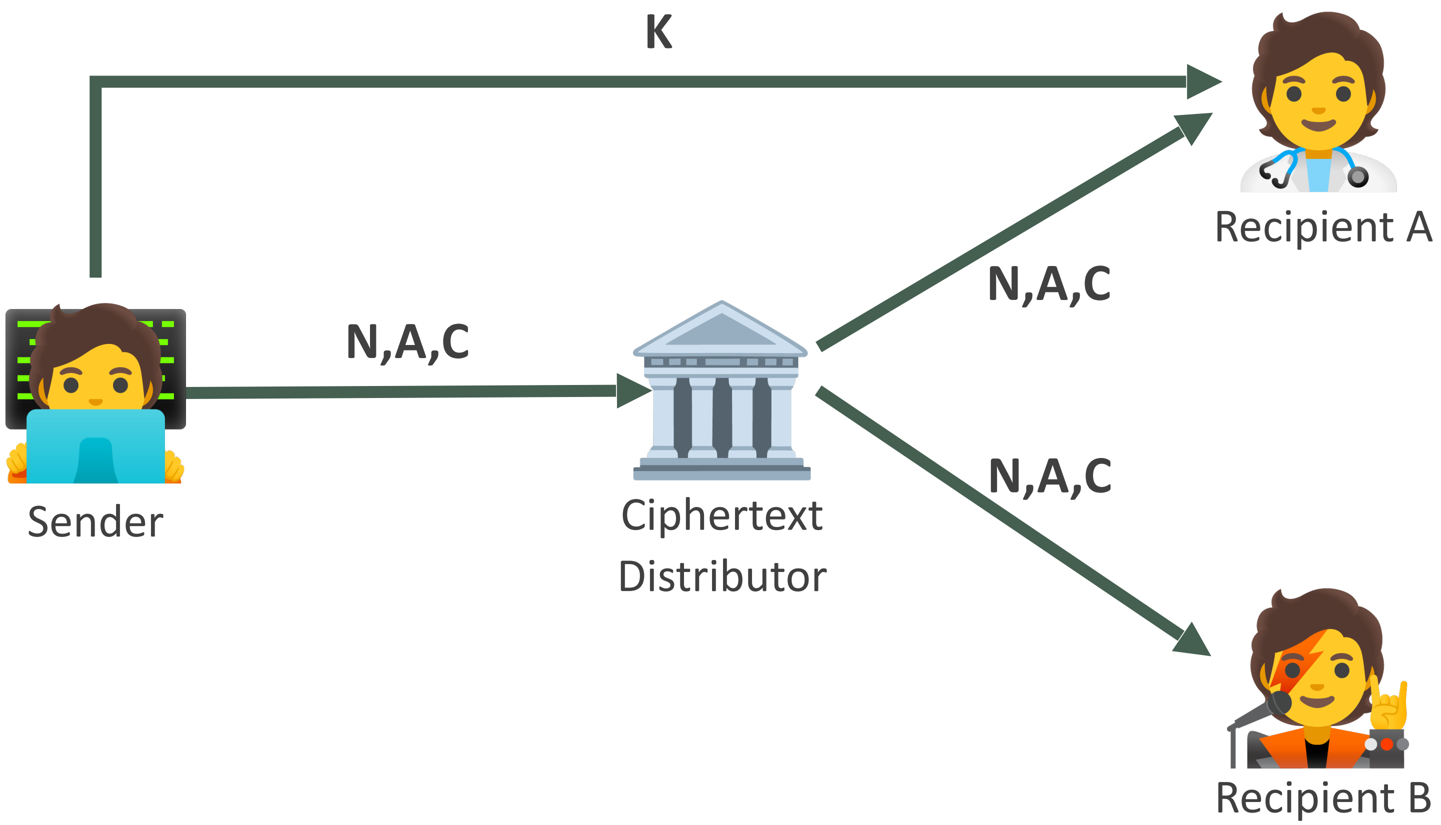
Threat #1: Multi-Recipient Integrity [FOR17, GLR17]



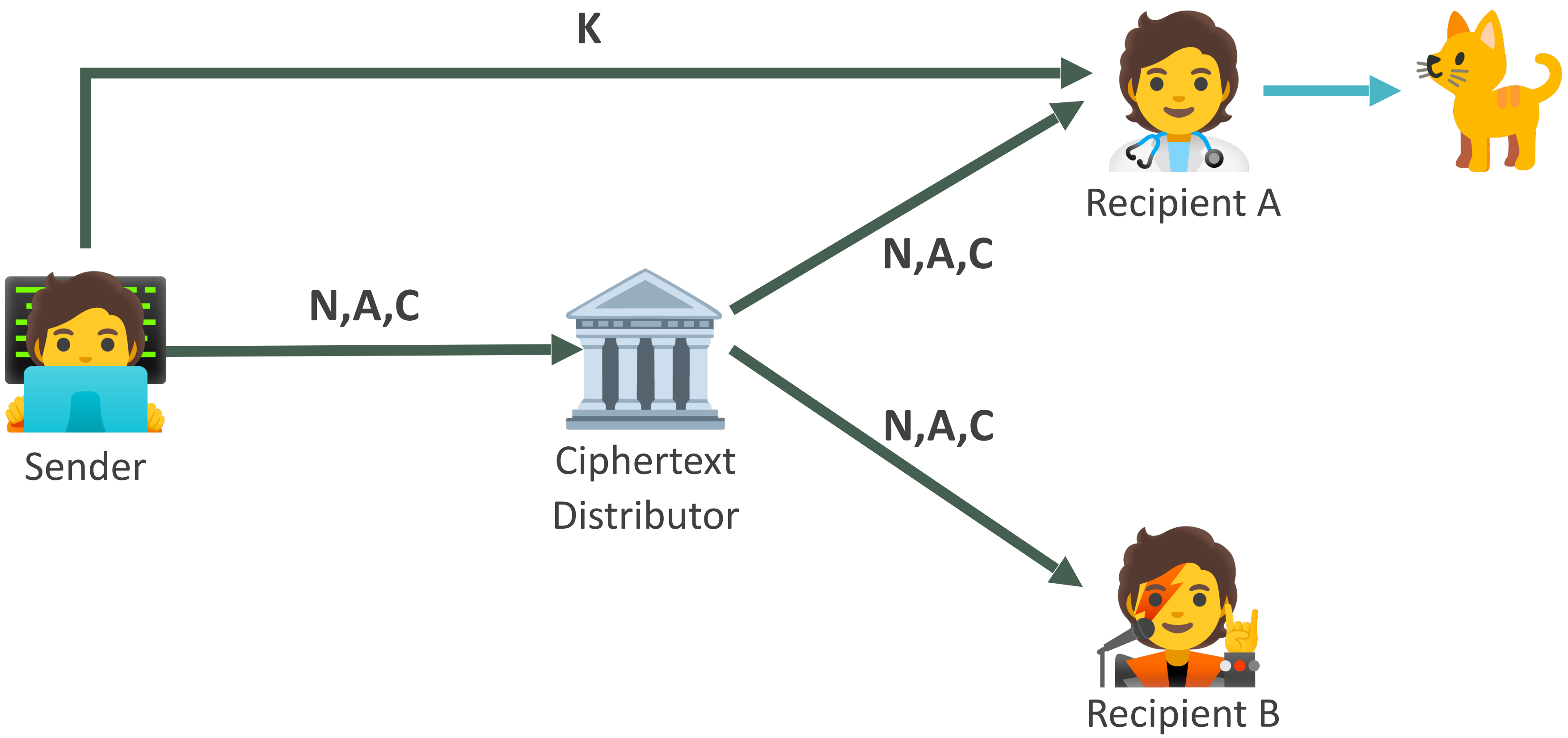
Threat #1: Multi-Recipient Integrity [FOR17, GLR17]



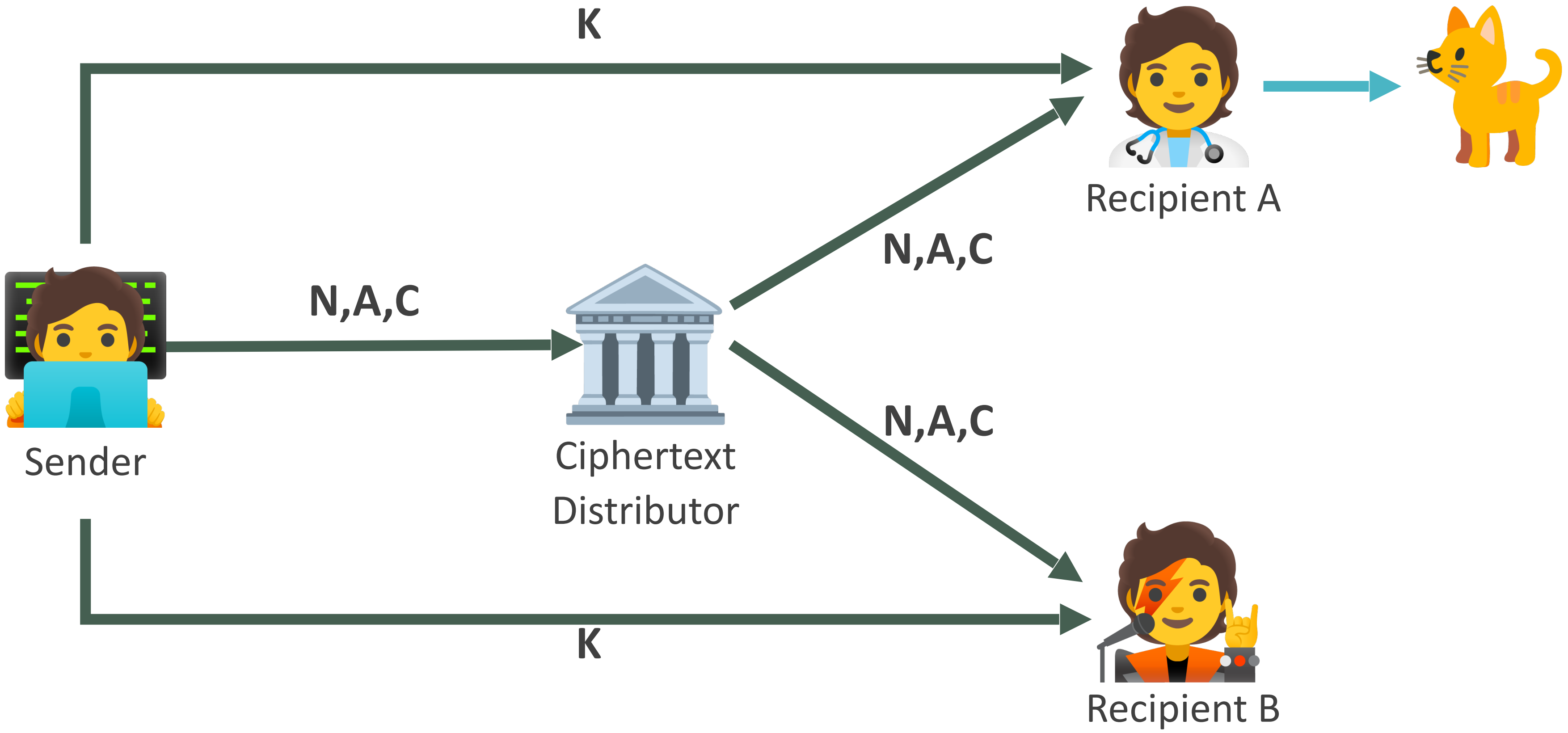
Threat #1: Multi-Recipient Integrity [FOR17, GLR17]



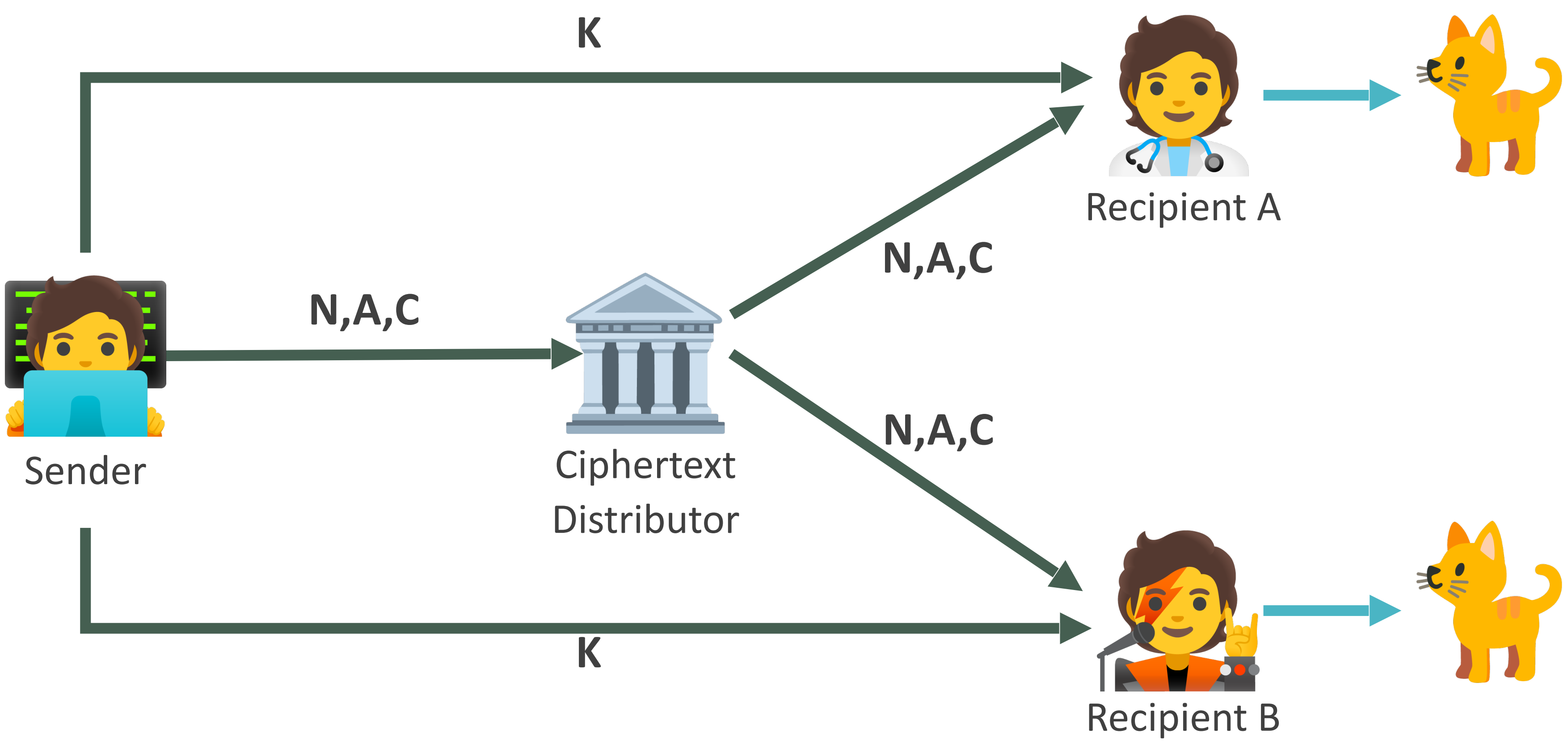
Threat #1: Multi-Recipient Integrity [FOR17, GLR17]



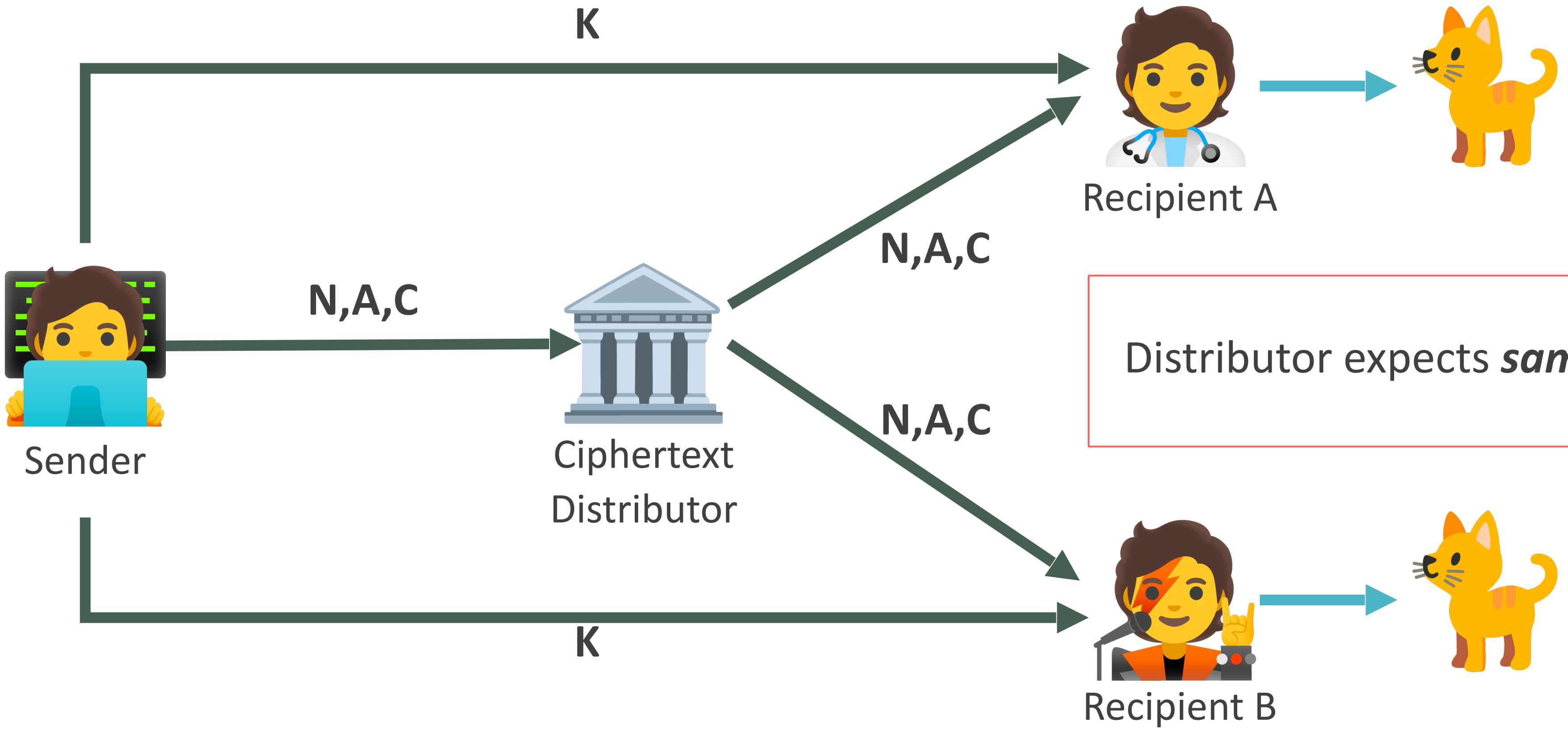
Threat #1: Multi-Recipient Integrity [FOR17, GLR17]



Threat #1: Multi-Recipient Integrity [FOR17, GLR17]

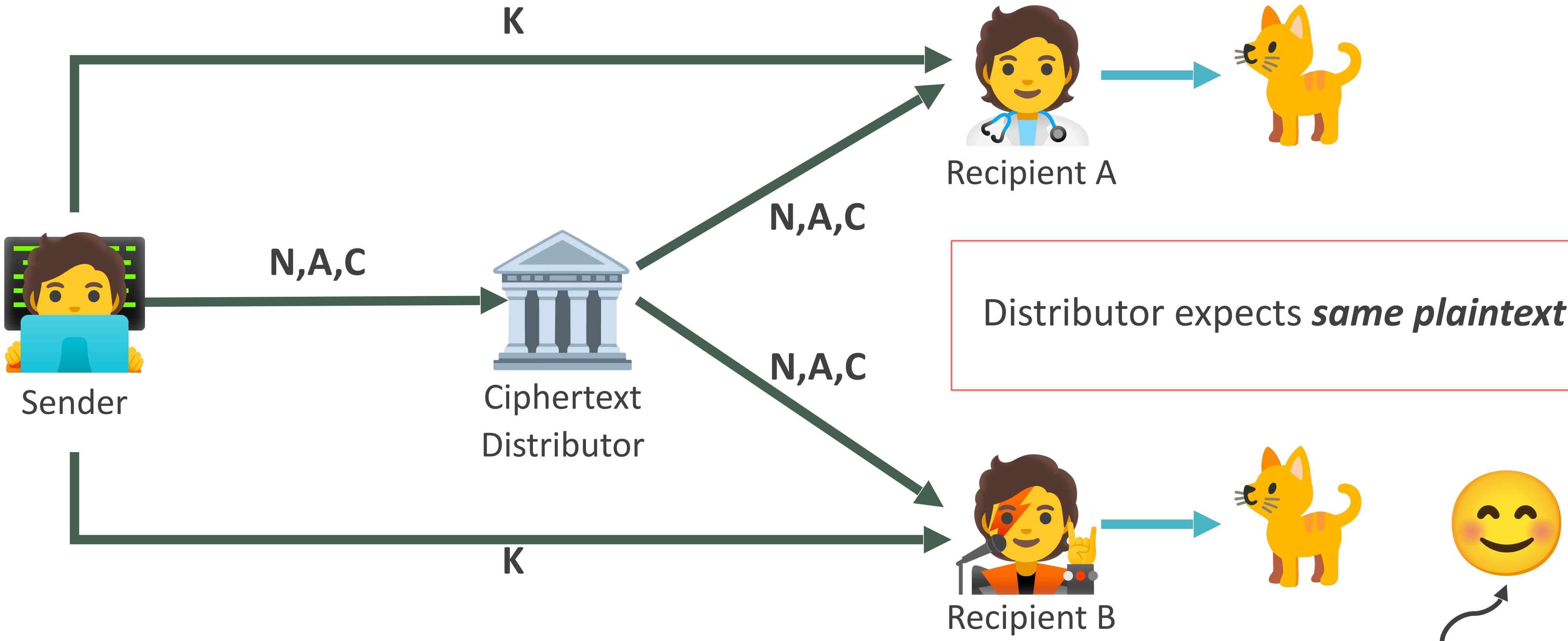


Threat #1: Multi-Recipient Integrity [FOR17, GLR17]



Distributor expects ***same plaintext received***

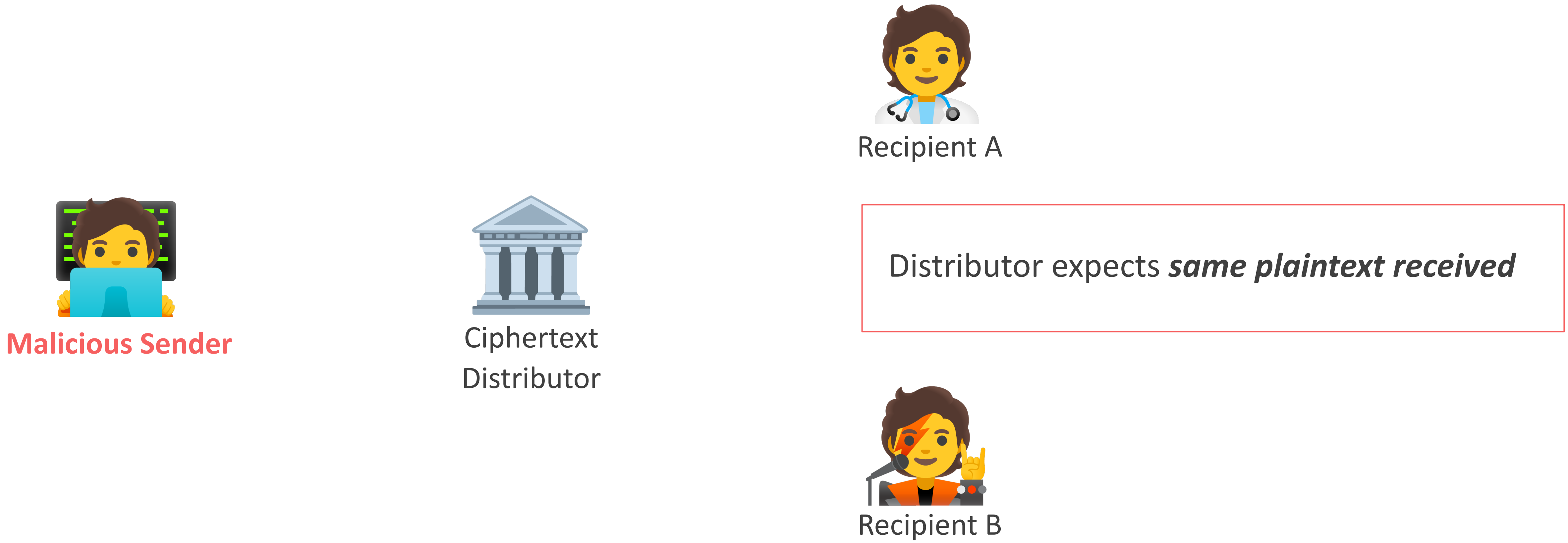
Threat #1: Multi-Recipient Integrity [FOR17, GLR17]



Distributor expects ***same plaintext received***

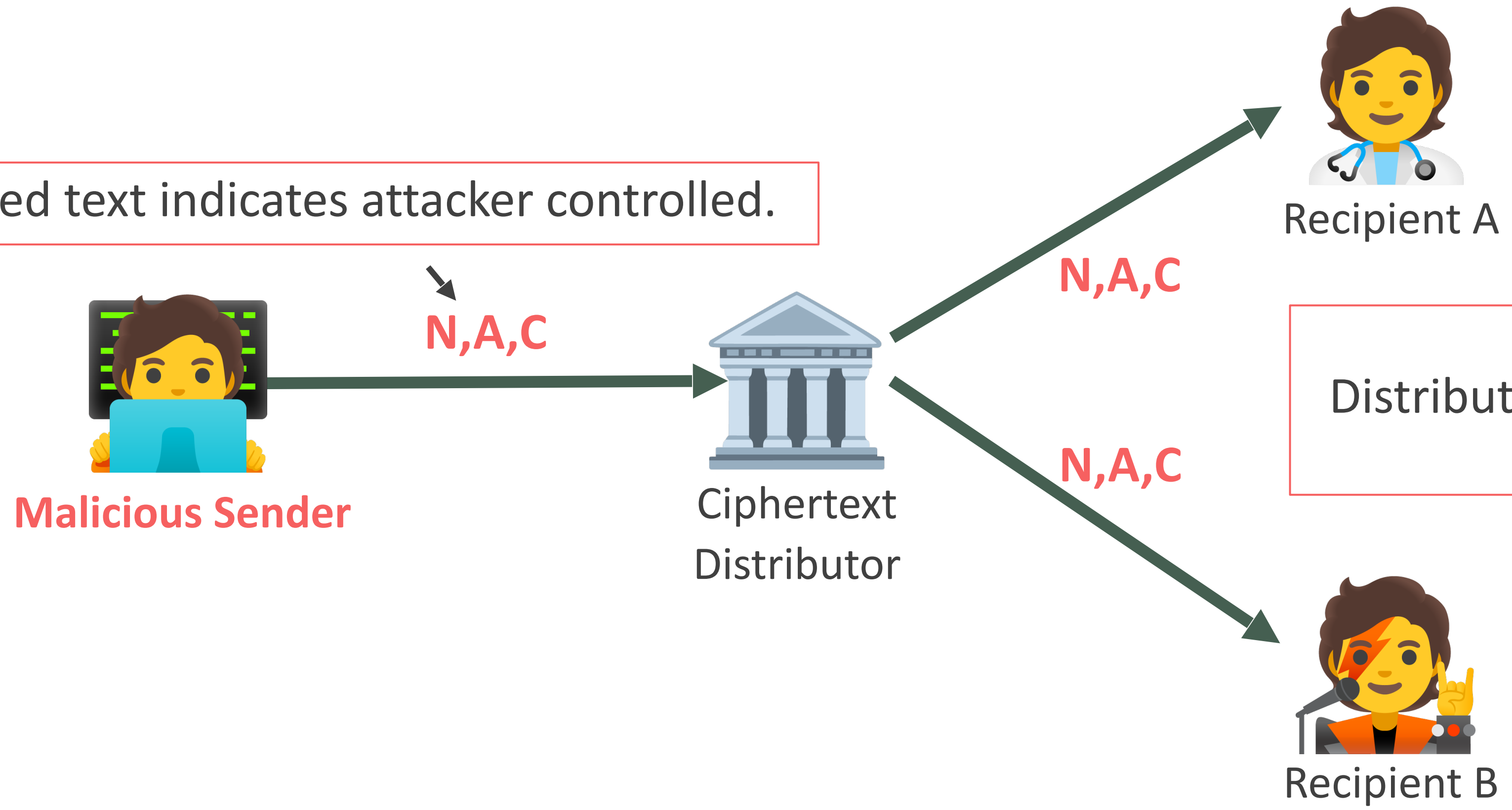
Smile indicates intended behavior.

Threat #1: Multi-Recipient Integrity [FOR17, GLR17]



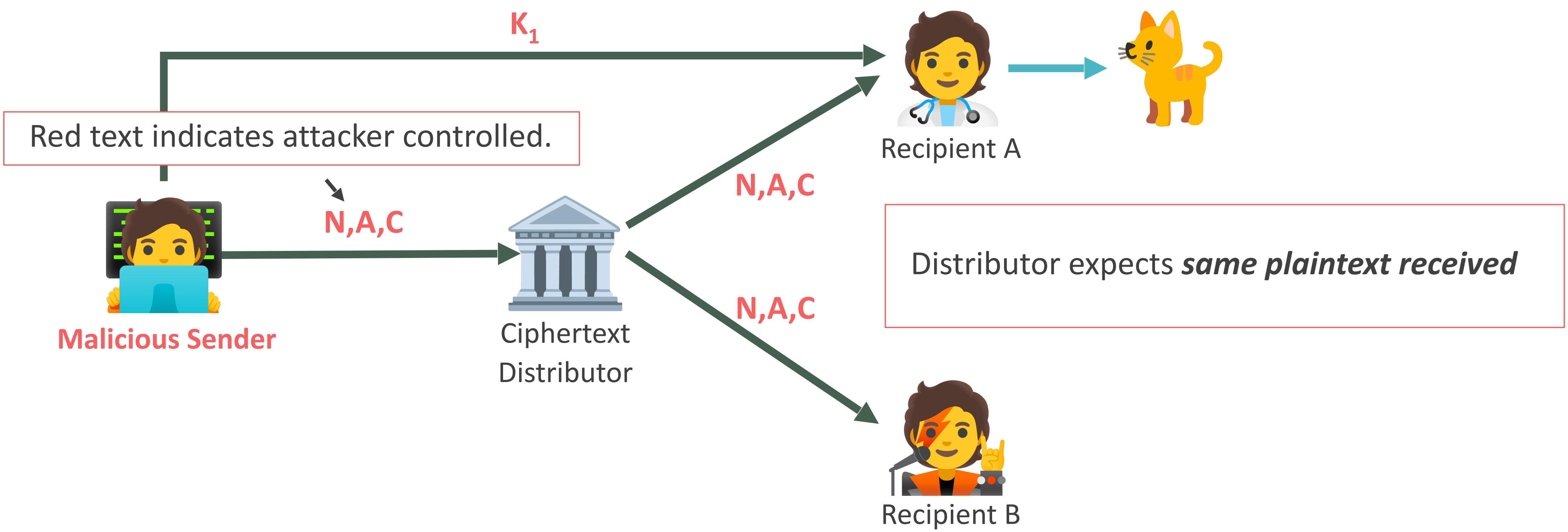
Threat #1: Multi-Recipient Integrity [FOR17, GLR17]

Red text indicates attacker controlled.

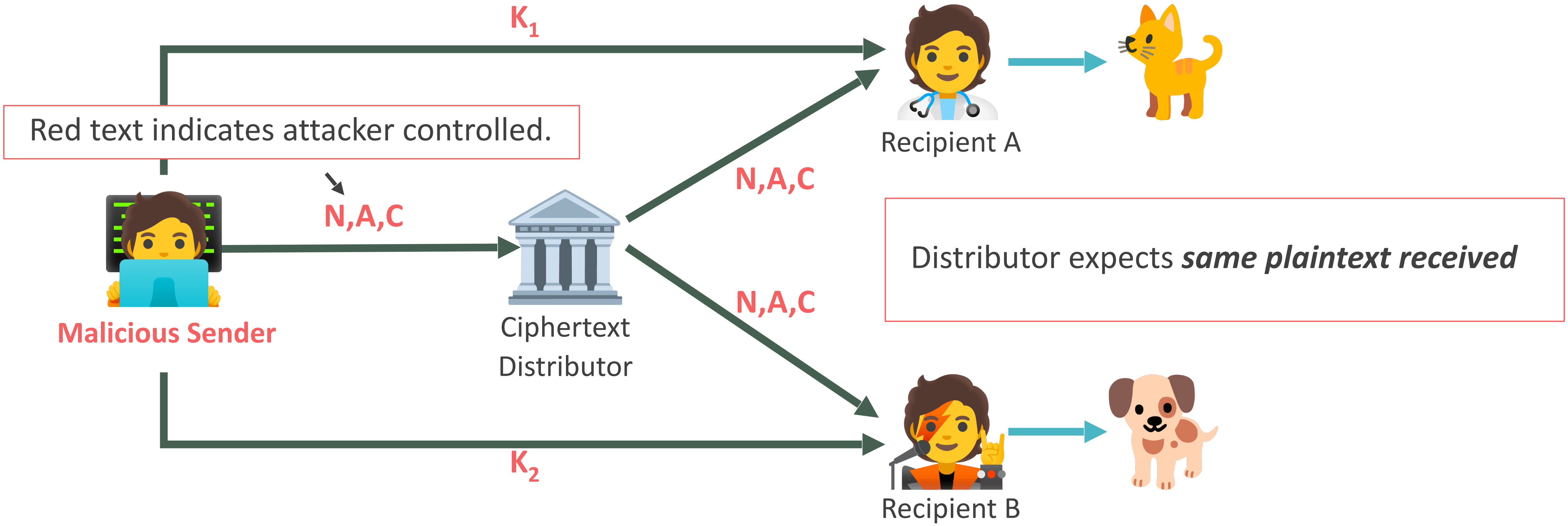


Distributor expects ***same plaintext received***

Threat #1: Multi-Recipient Integrity [FOR17, GLR17]

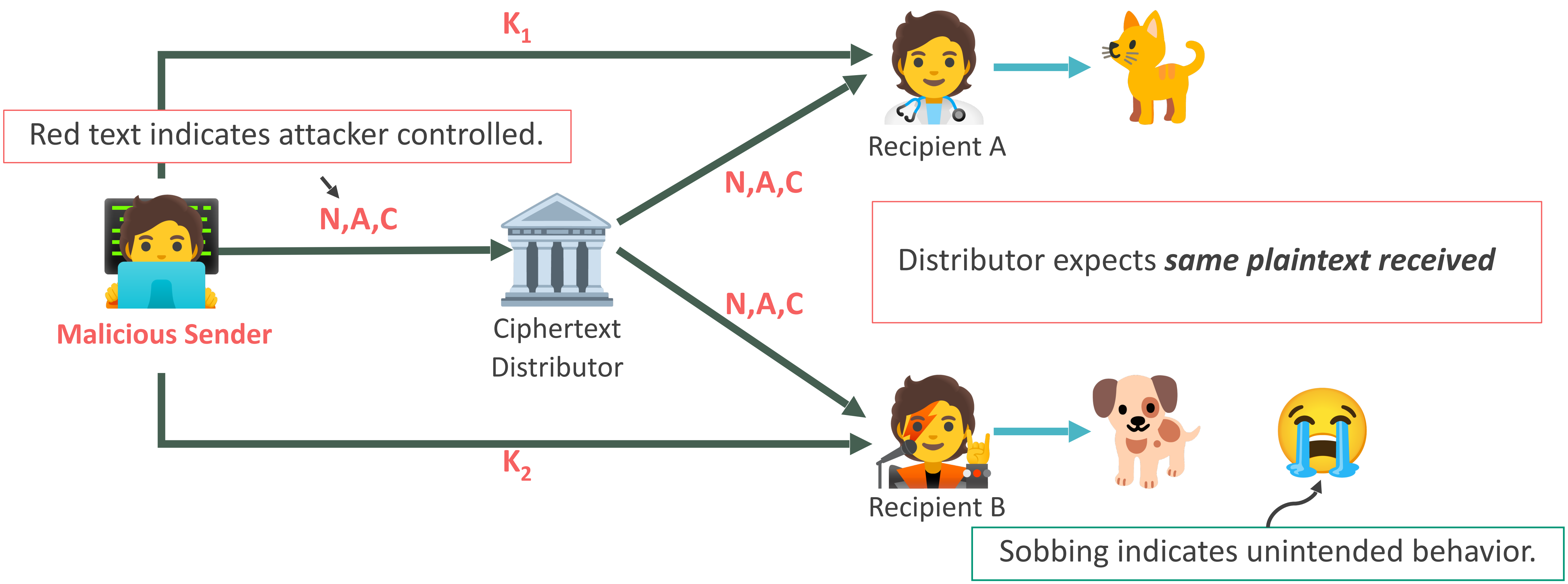


Threat #1: Multi-Recipient Integrity [FOR17, GLR17]



But malicious sender can arrange for different plaintexts to be received!

Threat #1: Multi-Recipient Integrity [FOR17, GLR17]



But malicious sender can arrange for different plaintexts to be received!

In-use AEAD schemes are not *key committing* [FOR17, GLR17]

For AEAD = XXX computationally efficient to find

$K_1 \neq K_2$ and N, A, C

such that decryption

M_1 = AEAD.Dec(K_1,N,A,C)
M_2 = AEAD.Dec(K_2,N,A,C)

succeeds and $M_1 \neq M_2$

In-use AEAD schemes are not *key committing* [FOR17, GLR17]

For AEAD = XXX computationally efficient to find

$K_1 \neq K_2$ and N, A, C

such that decryption

$$M_1 = \text{AEAD.Dec}(K_1, N, A, C)$$
$$M_2 = \text{AEAD.Dec}(K_2, N, A, C)$$

succeeds and $M_1 \neq M_2$

XXX	Attack Citation
AES-GCM	[GLR17]
ChaCha20/Poly1305	[LGR20]
AES-GCM-SIV	[Sch20, LGR20]
AES-OCB3	[ADGKLS20]
AES-SIV	[MLGR23]
XSalsa20/Poly1305	[LGR20]

In-use AEAD schemes are not *key committing* [FOR17, GLR17]

For AEAD = XXX computationally efficient to find

$K_1 \neq K_2$ and N, A, C

such that decryption

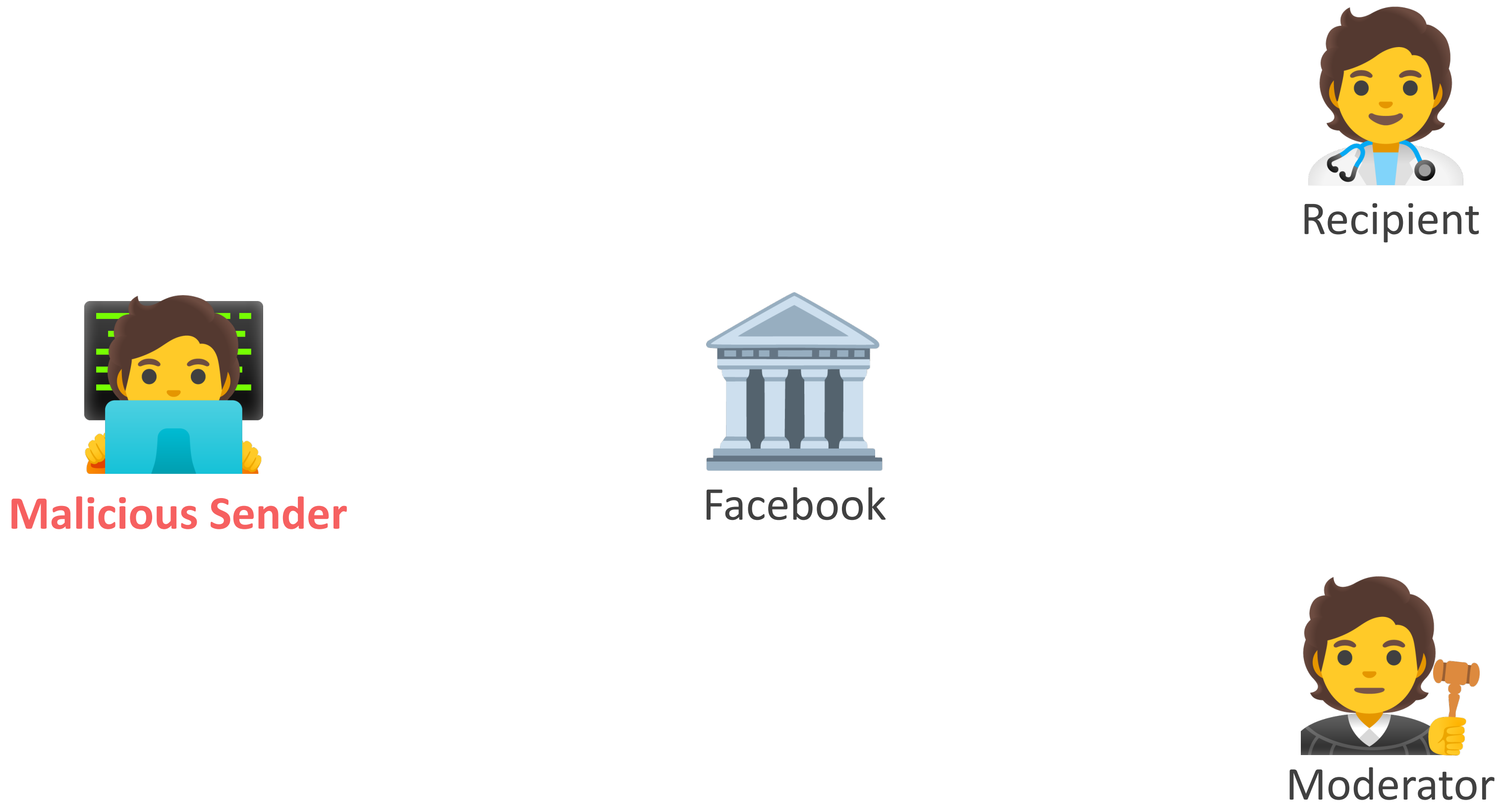
$$M_1 = \text{AEAD.Dec}(K_1,N,A,C)$$
$$M_2 = \text{AEAD.Dec}(K_2,N,A,C)$$

succeeds and $M_1 \neq M_2$

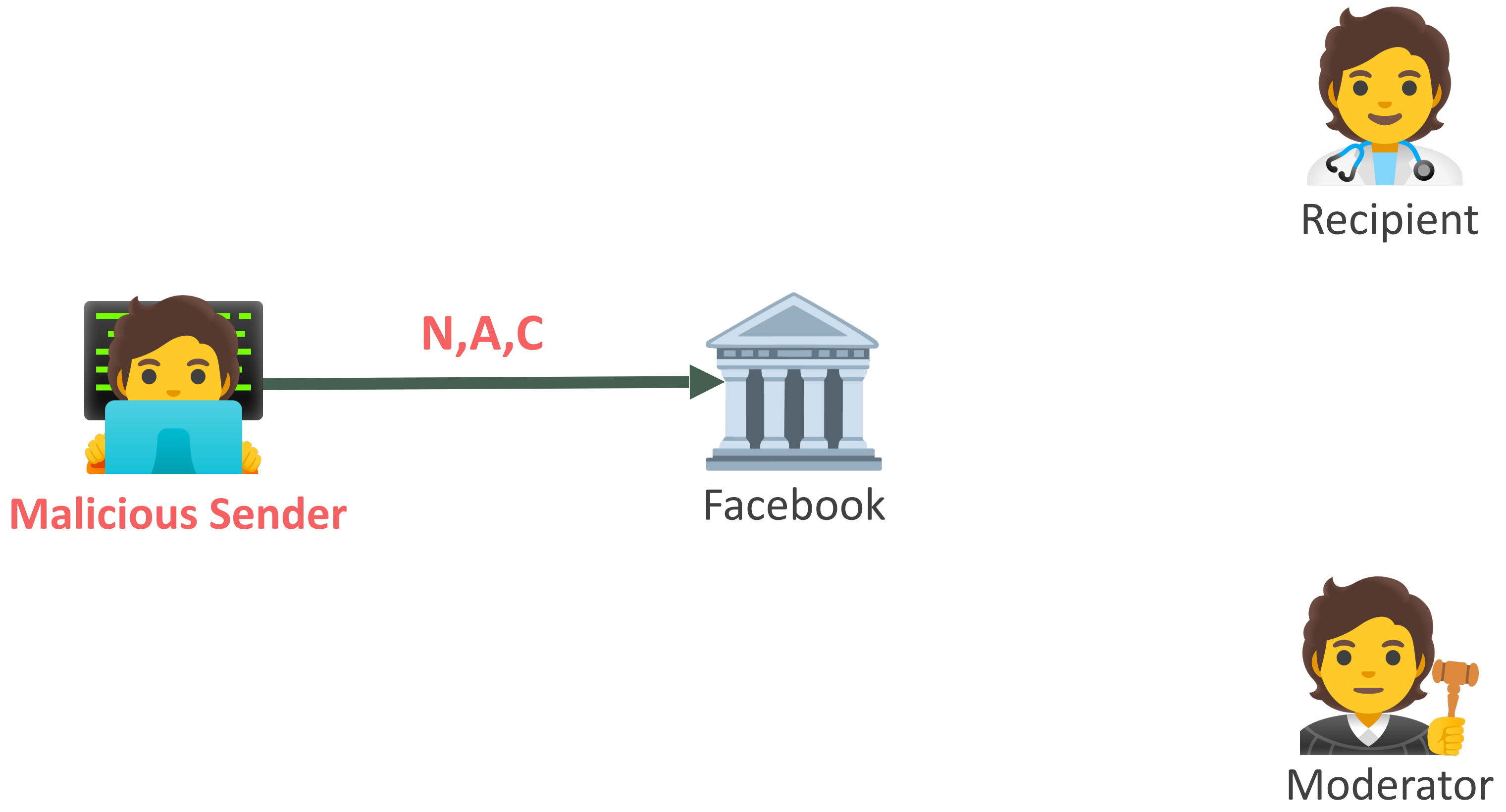
XXX	Attack Citation
AES-GCM	[GLR17]
ChaCha20/Poly1305	[LGR20]
AES-GCM-SIV	[Sch20, LGR20]
AES-OCB3	[ADGKLS20]
AES-SIV	[MLGR23]
XSalsa20/Poly1305	[LGR20]

Attacks are fast and practically damaging

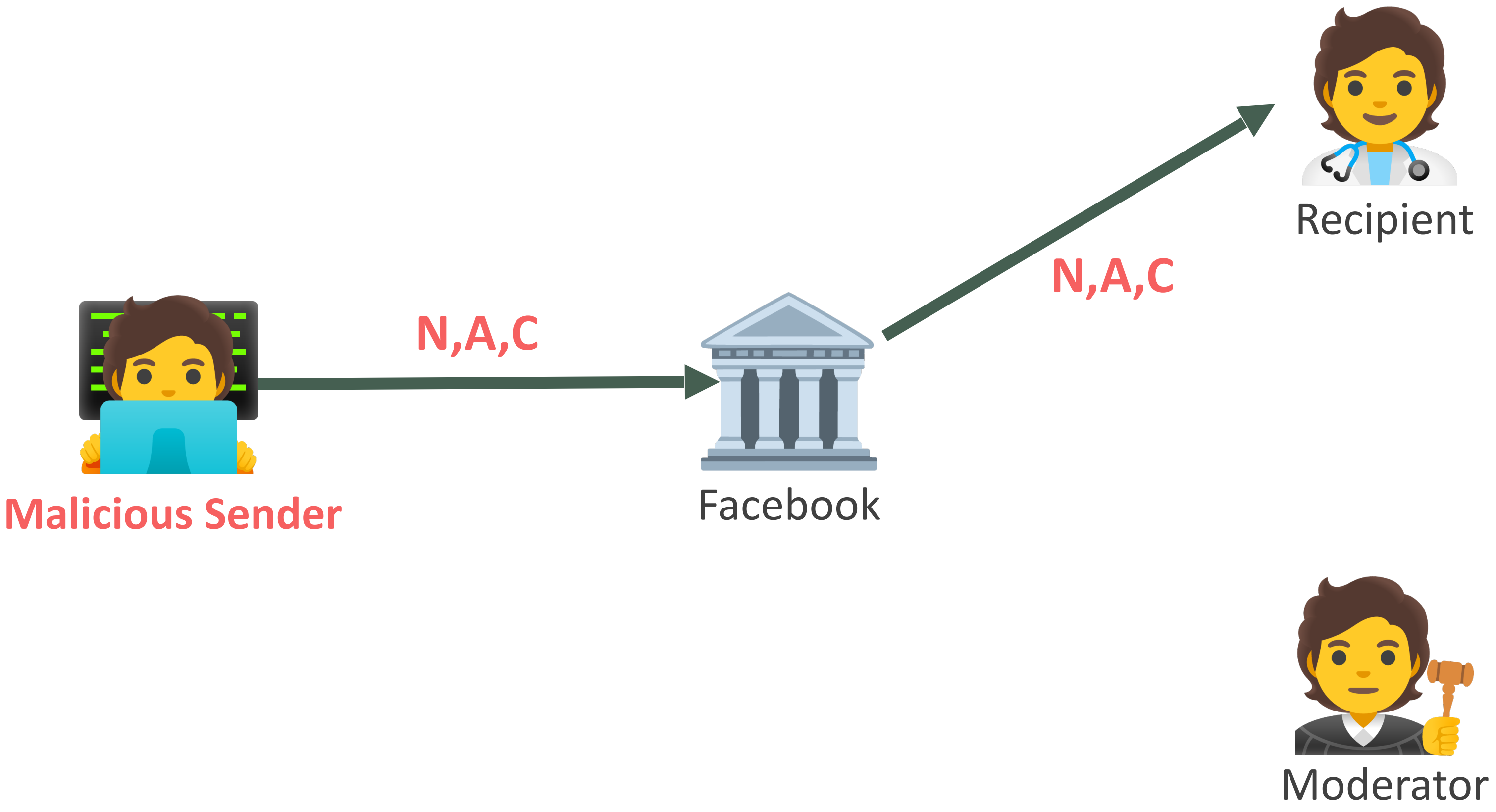
Threat #1: Invisible Salamanders Attack [DGRW19]



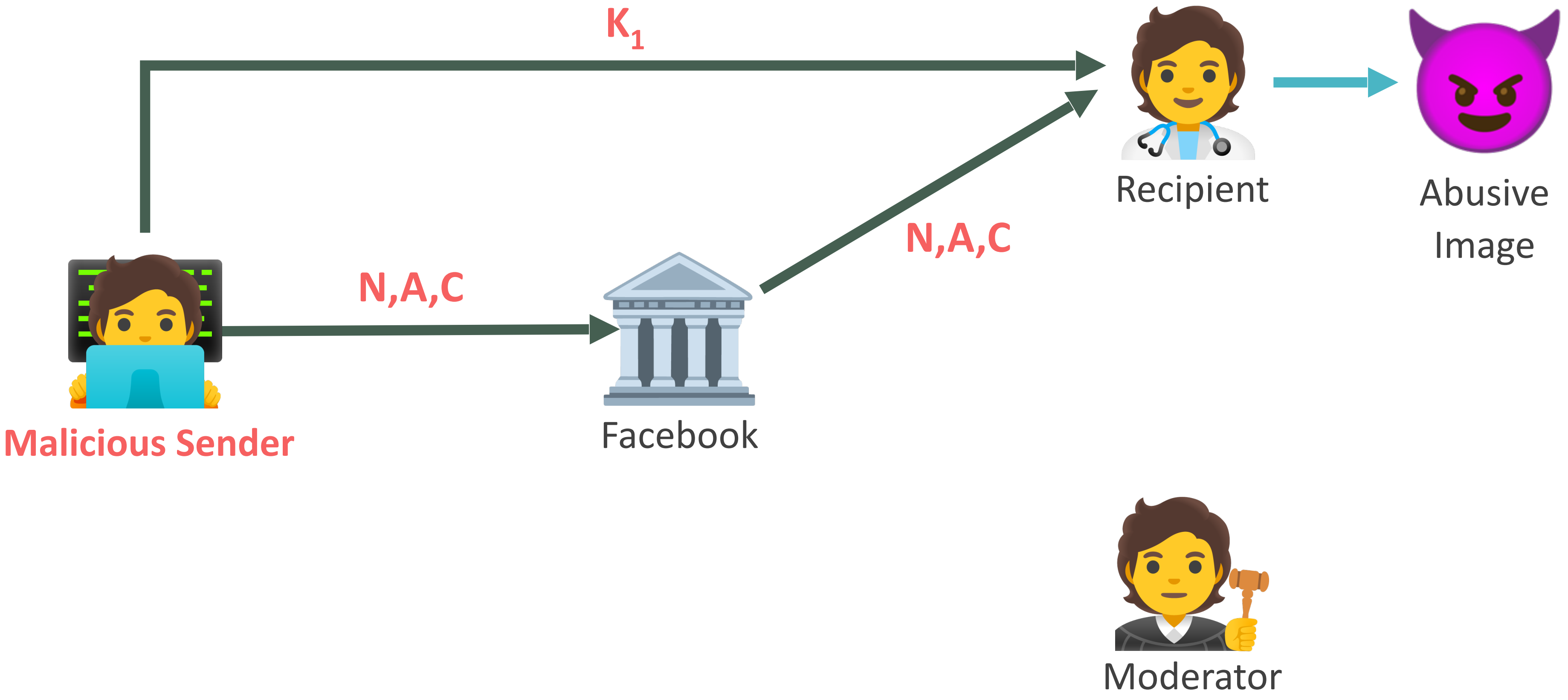
Threat #1: Invisible Salamanders Attack [DGRW19]



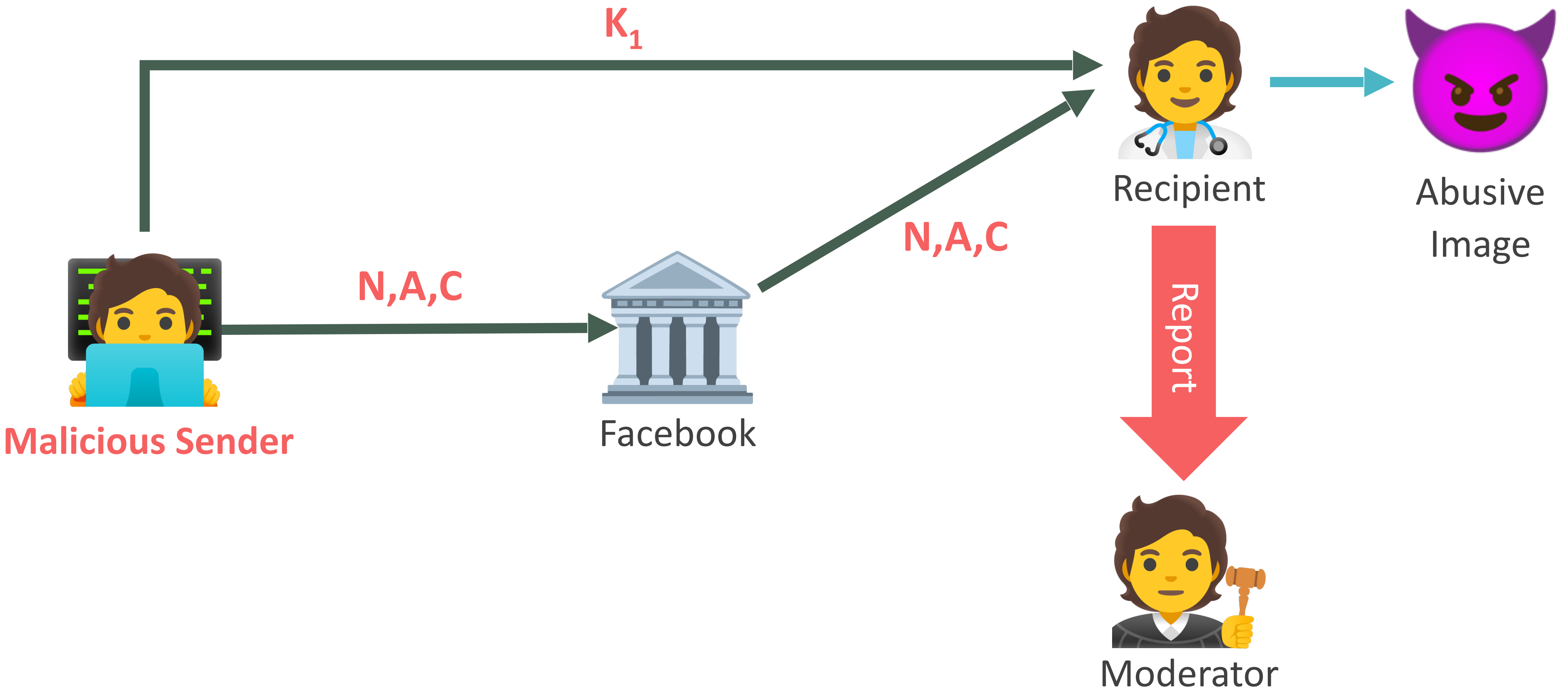
Threat #1: Invisible Salamanders Attack [DGRW19]



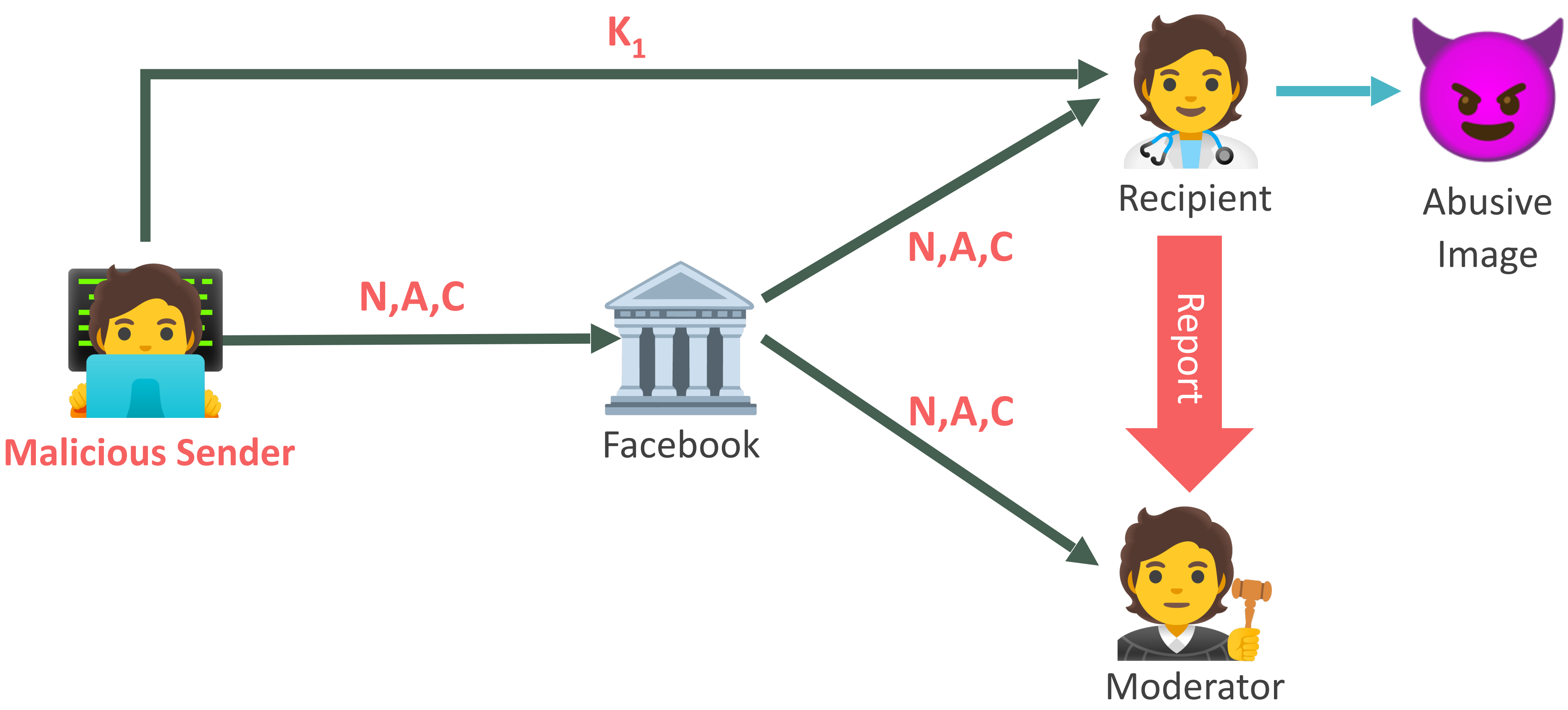
Threat #1: Invisible Salamanders Attack [DGRW19]



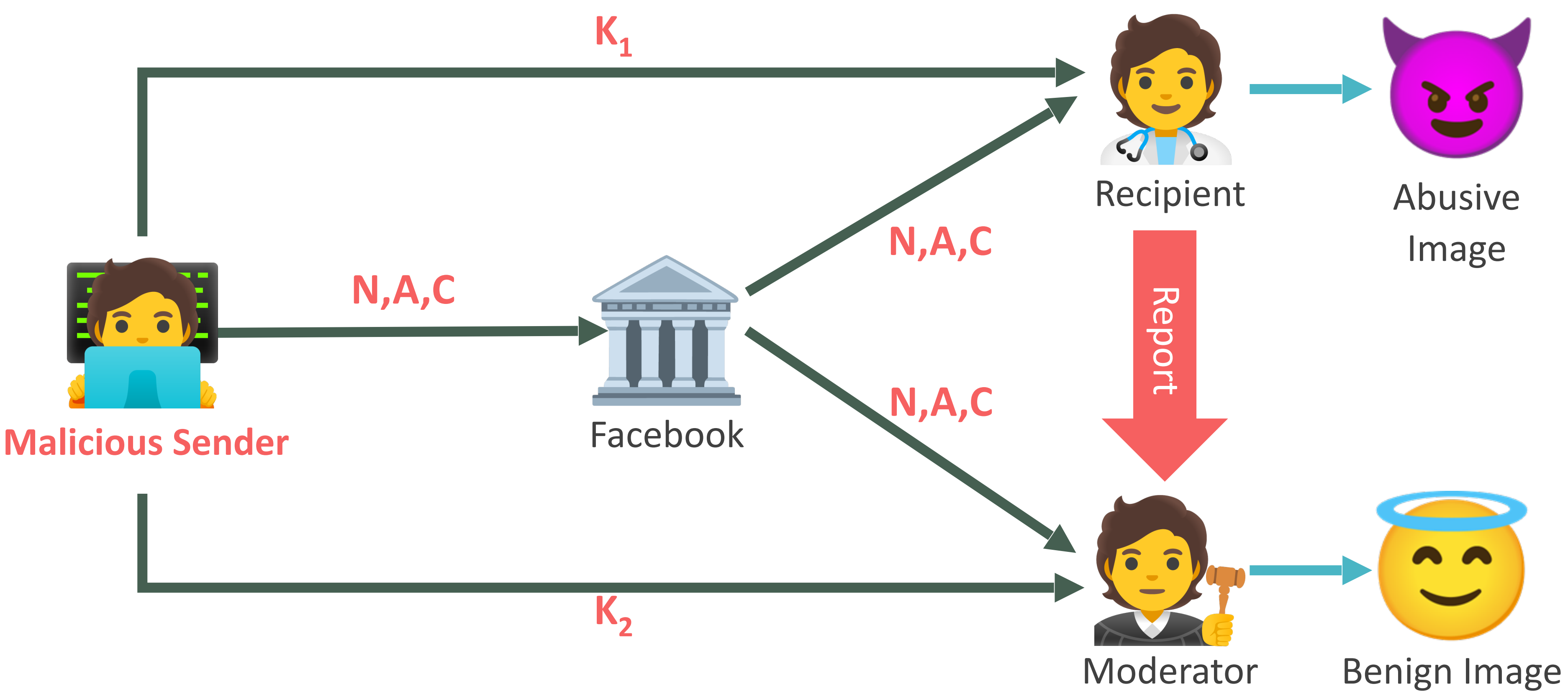
Threat #1: Invisible Salamanders Attack [DGRW19]



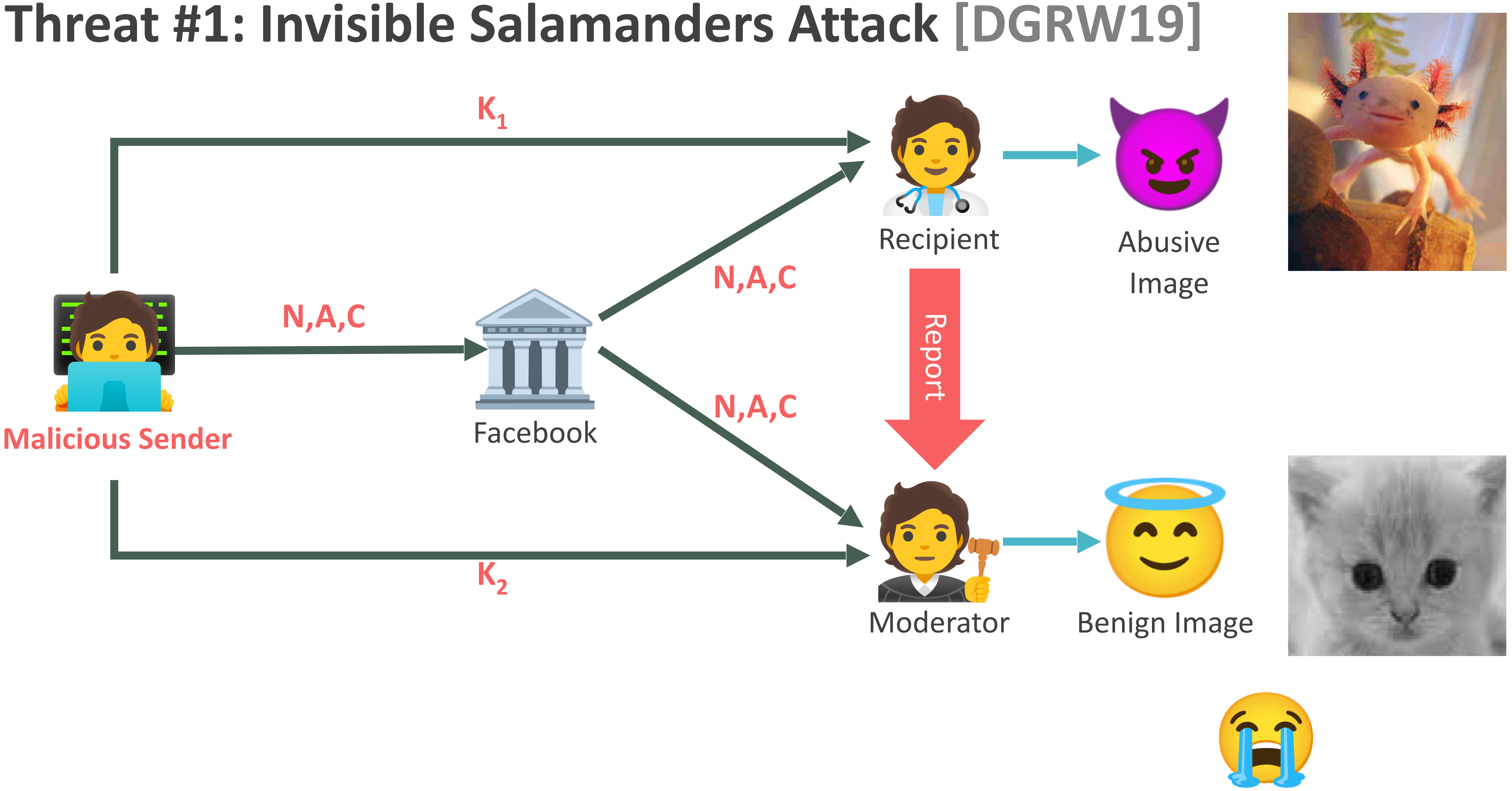
Threat #1: Invisible Salamanders Attack [DGRW19]



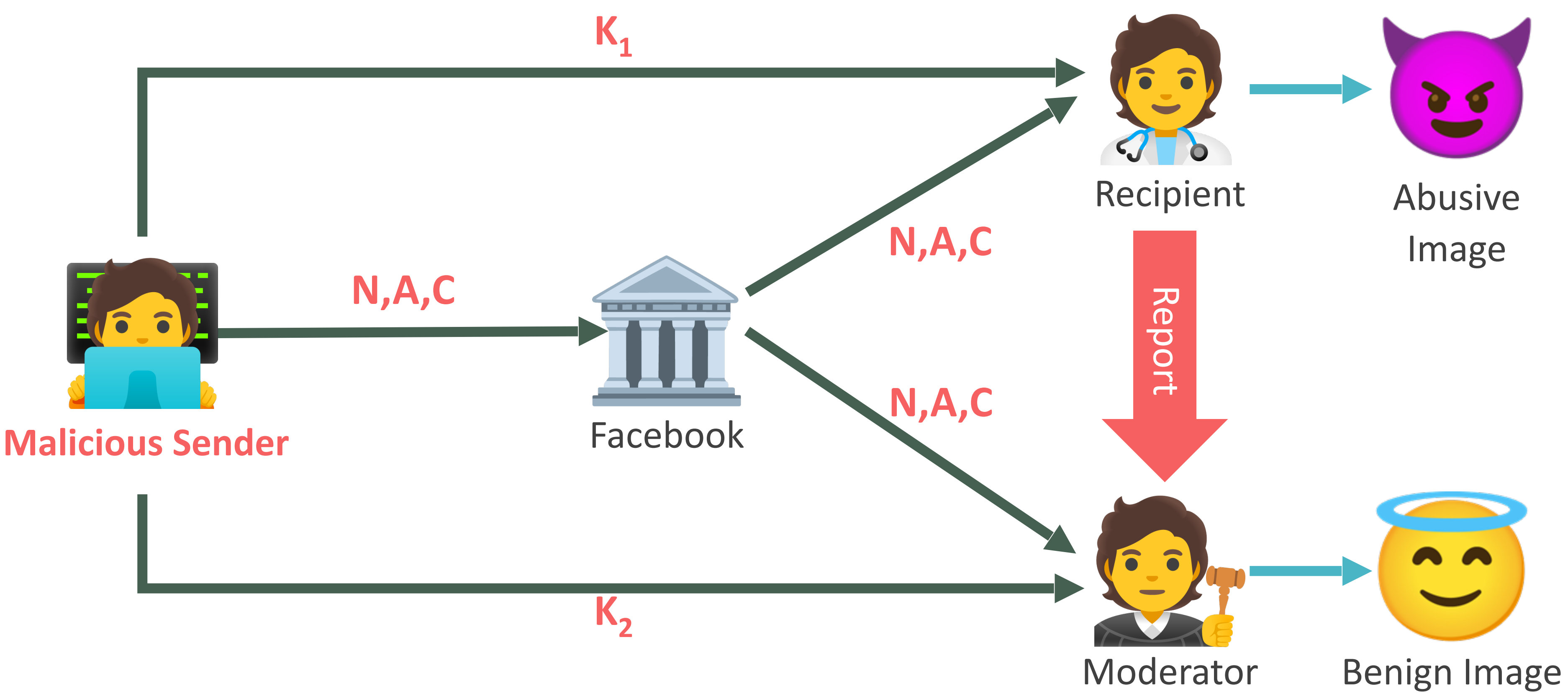
Threat #1: Invisible Salamanders Attack [DGRW19]



Threat #1: Invisible Salamanders Attack [DGRW19]



Threat #1: Invisible Salamanders Attack [DGRW19]



Multi-recipient integrity vulnerabilities also found in

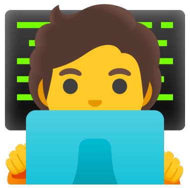
- AWS Encryption SDK
- pre-release product reviewed at Google

[ADGKLS20]

Threat #2: Partitioning oracle attacks [LGR20]

Server uses password to decrypt ciphertexts

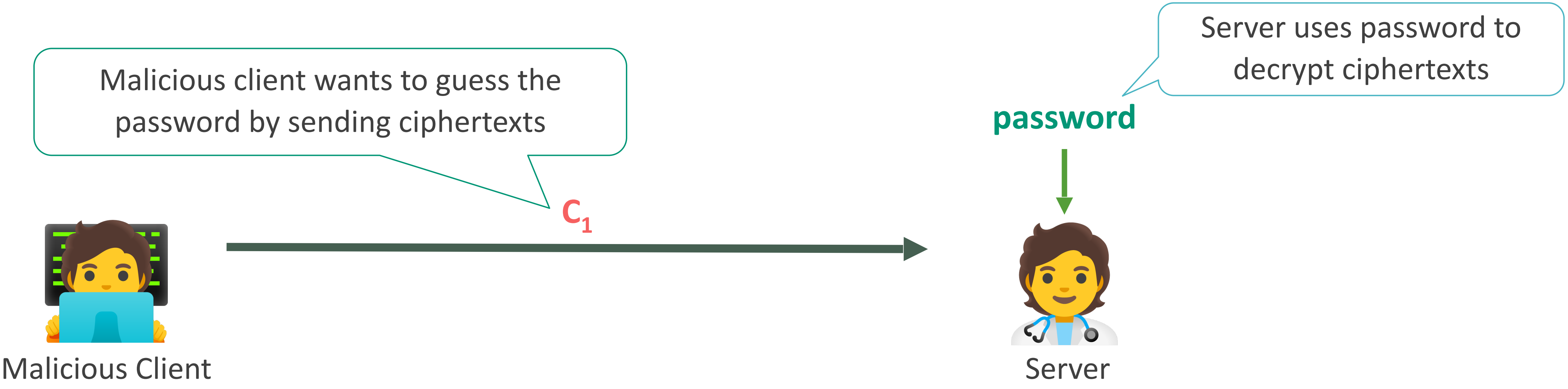
password



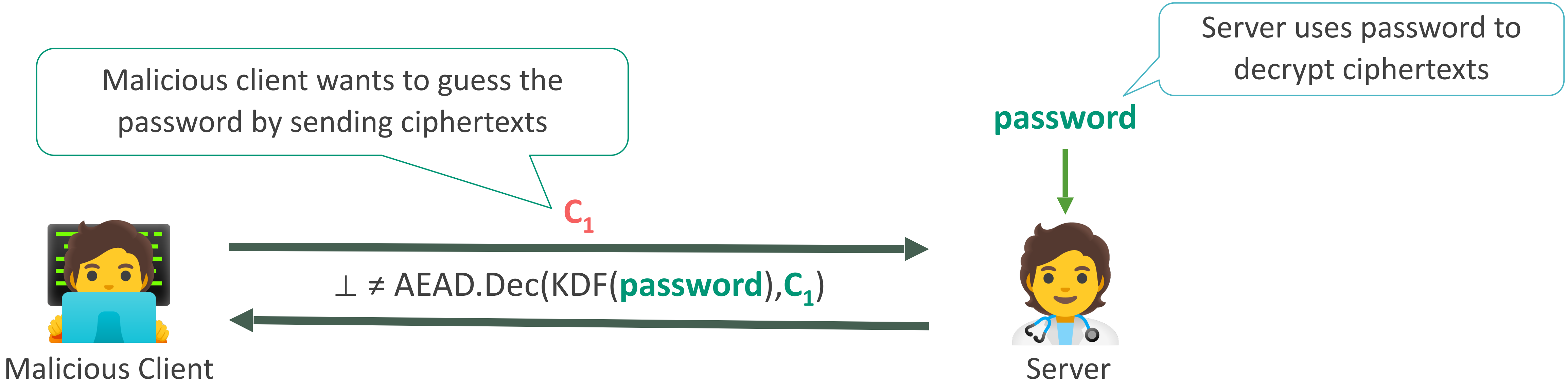
Malicious Client

Server

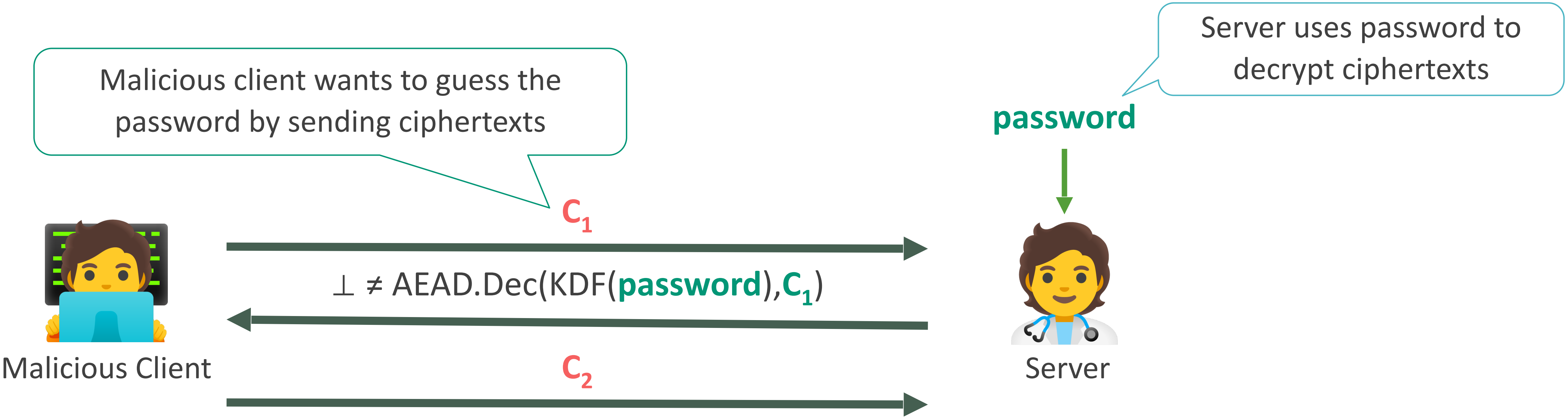
Threat #2: Partitioning oracle attacks [LGR20]



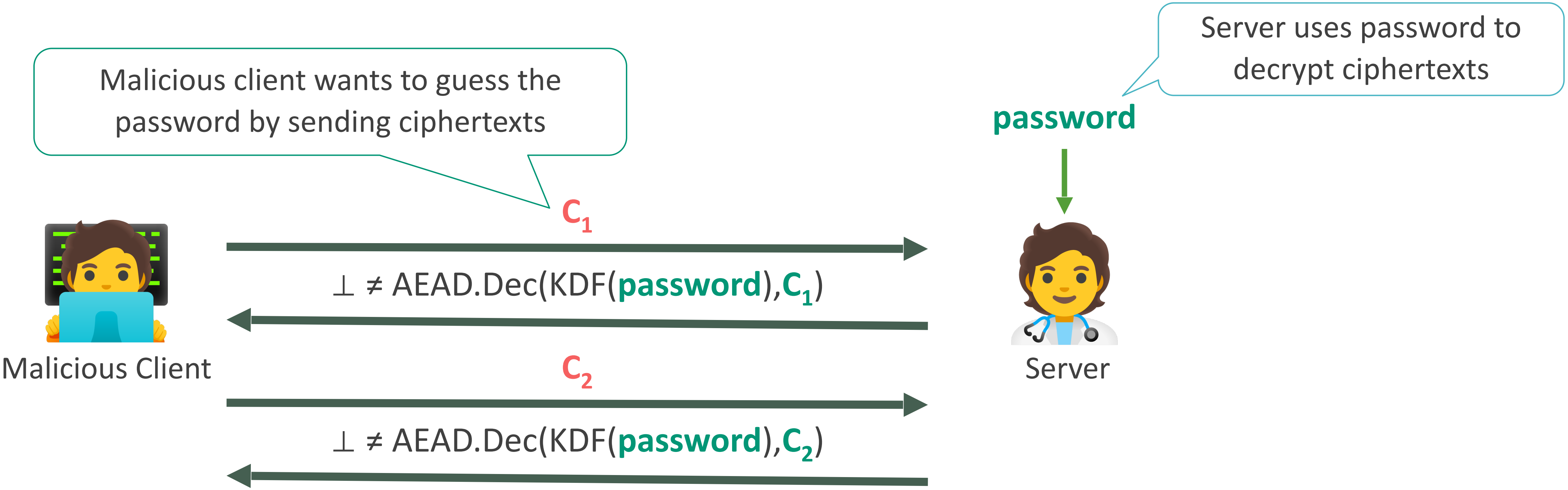
Threat #2: Partitioning oracle attacks [LGR20]



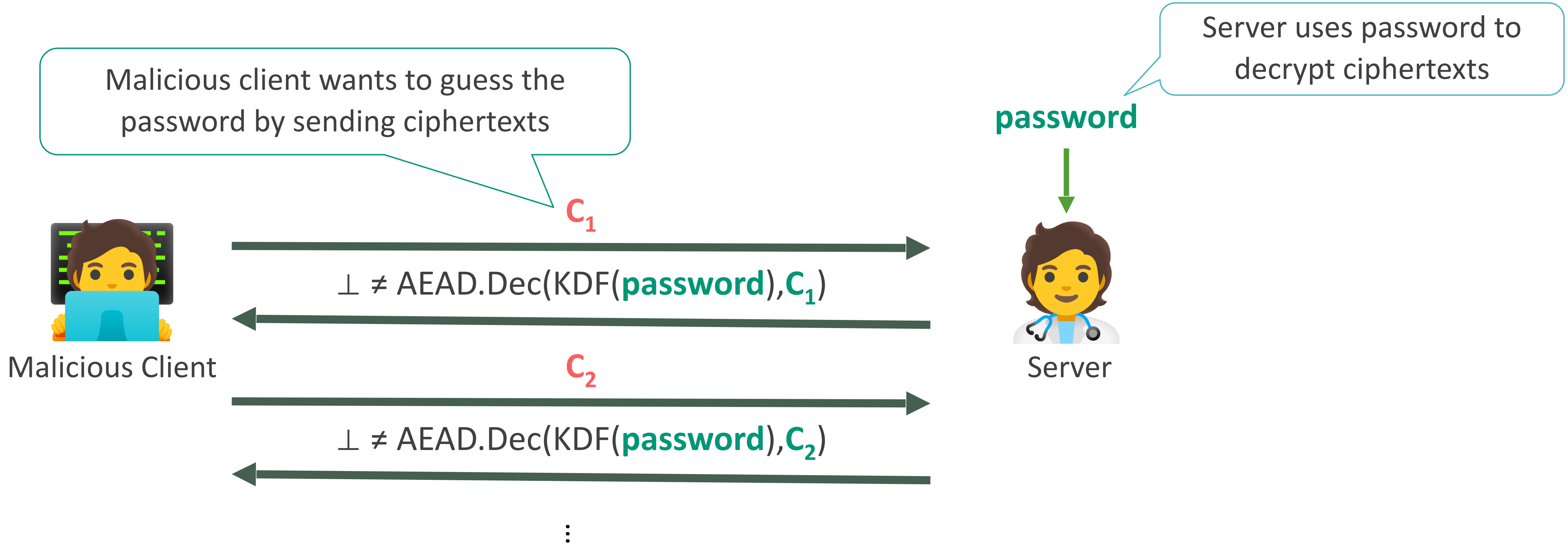
Threat #2: Partitioning oracle attacks [LGR20]



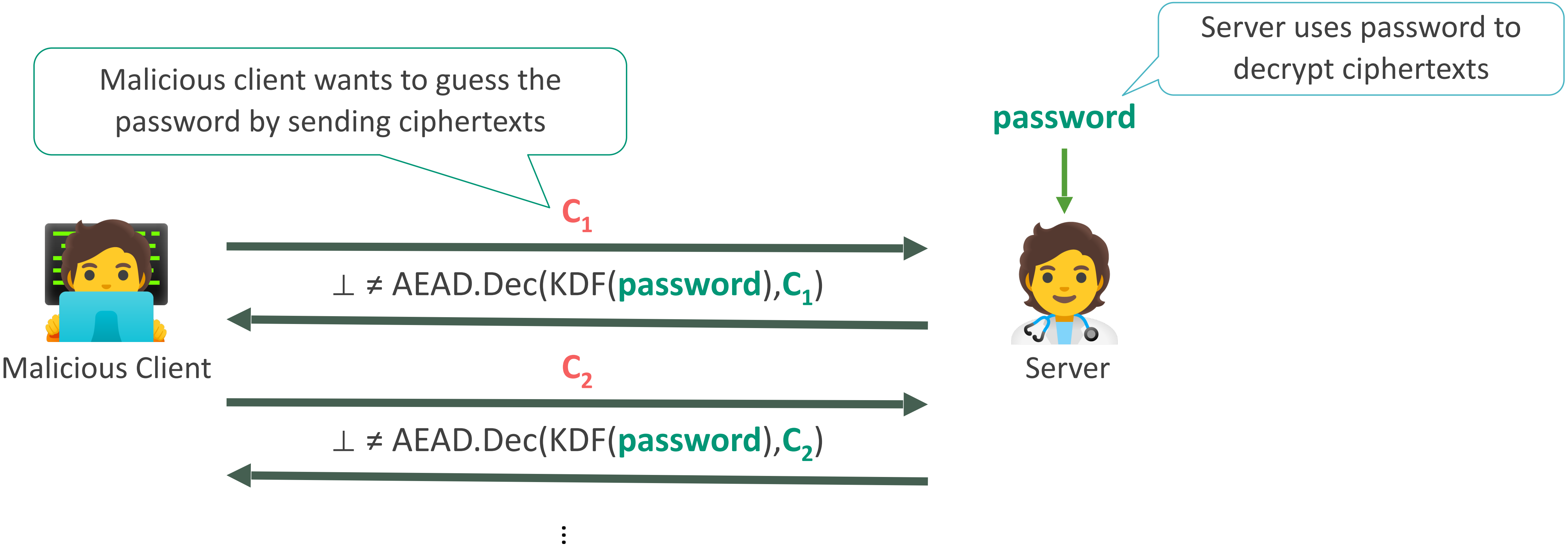
Threat #2: Partitioning oracle attacks [LGR20]



Threat #2: Partitioning oracle attacks [LGR20]

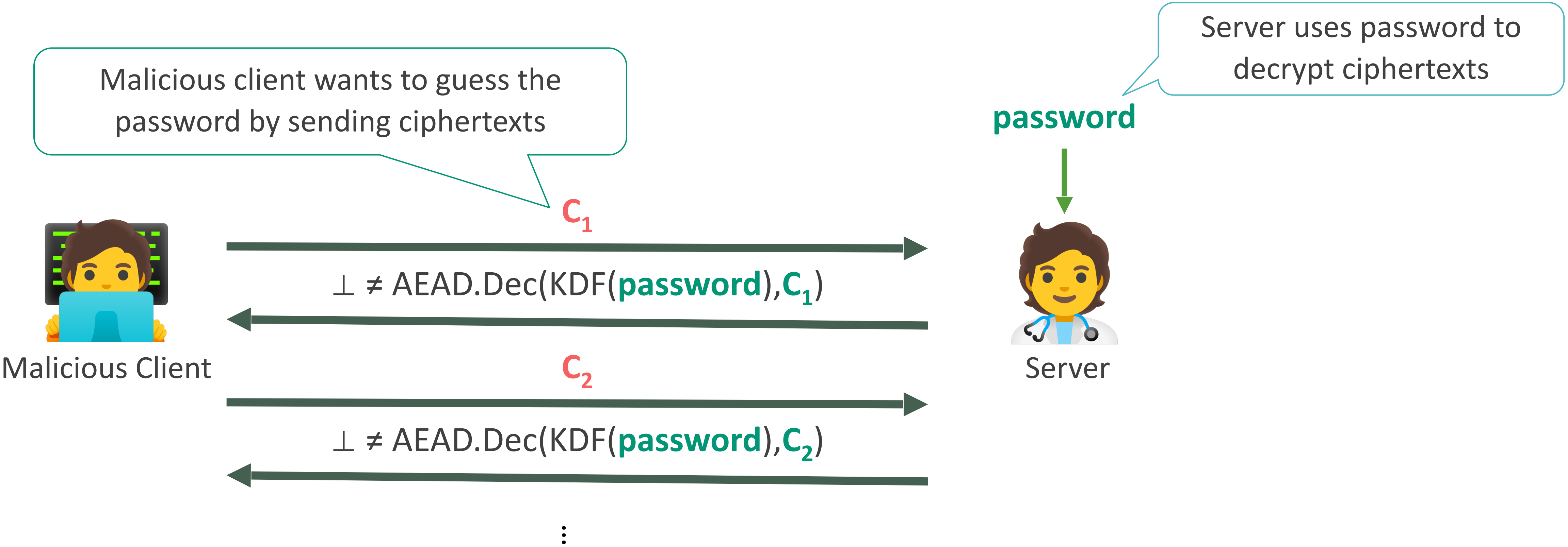


Threat #2: Partitioning oracle attacks [LGR20]



Security expected: one ciphertext = one guess

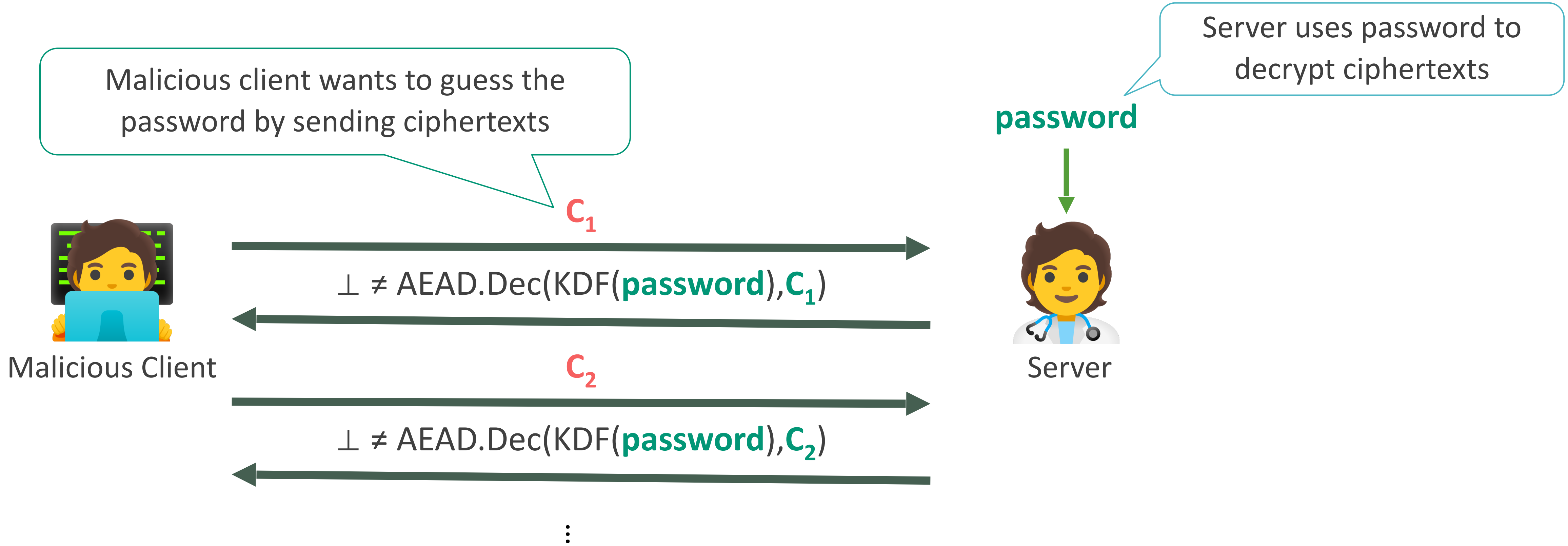
Threat #2: Partitioning oracle attacks [LGR20]



Security expected: one ciphertext = one guess

Previous slide: one ciphertext = ***two*** guesses

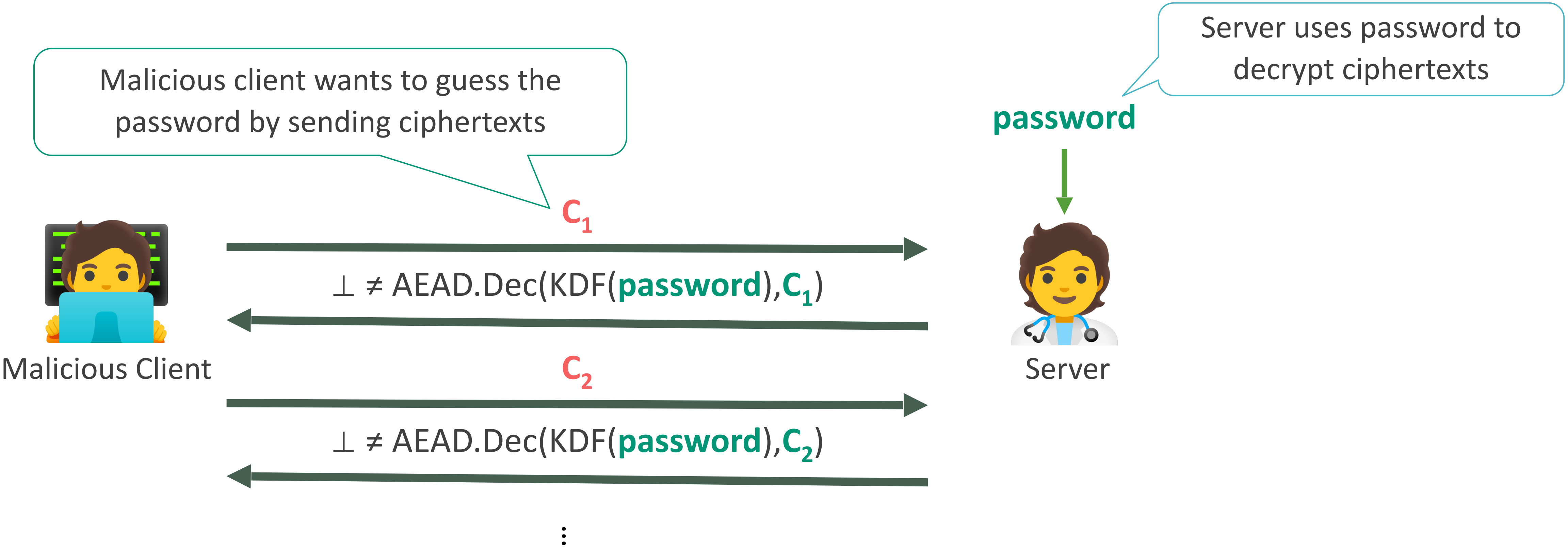
Threat #2: Partitioning oracle attacks [LGR20]



Security expected: one ciphertext = one guess

Previous slide: one ciphertext = ***two*** guesses

Threat #2: Partitioning oracle attacks [LGR20]

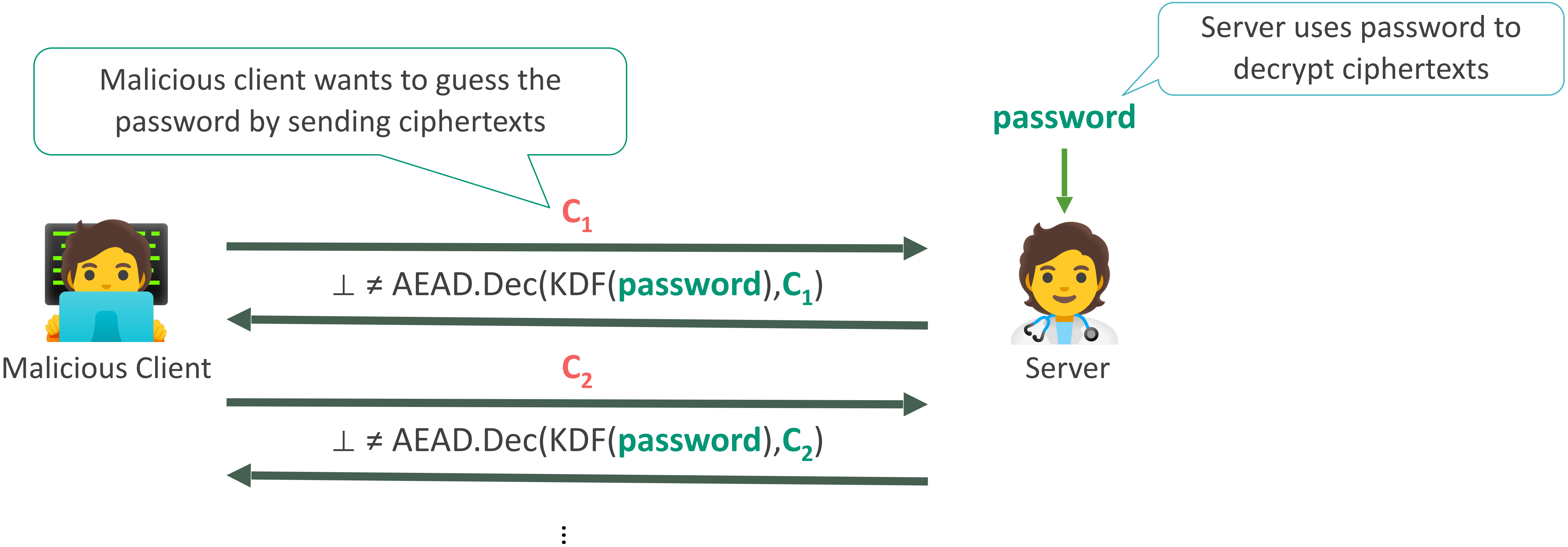


Security expected: one ciphertext = one guess

Previous slide: one ciphertext = ***two*** guesses

[LGR20]: one ciphertext = up to **4096 guesses** (for AES-GCM)

Threat #2: Partitioning oracle attacks [LGR20]

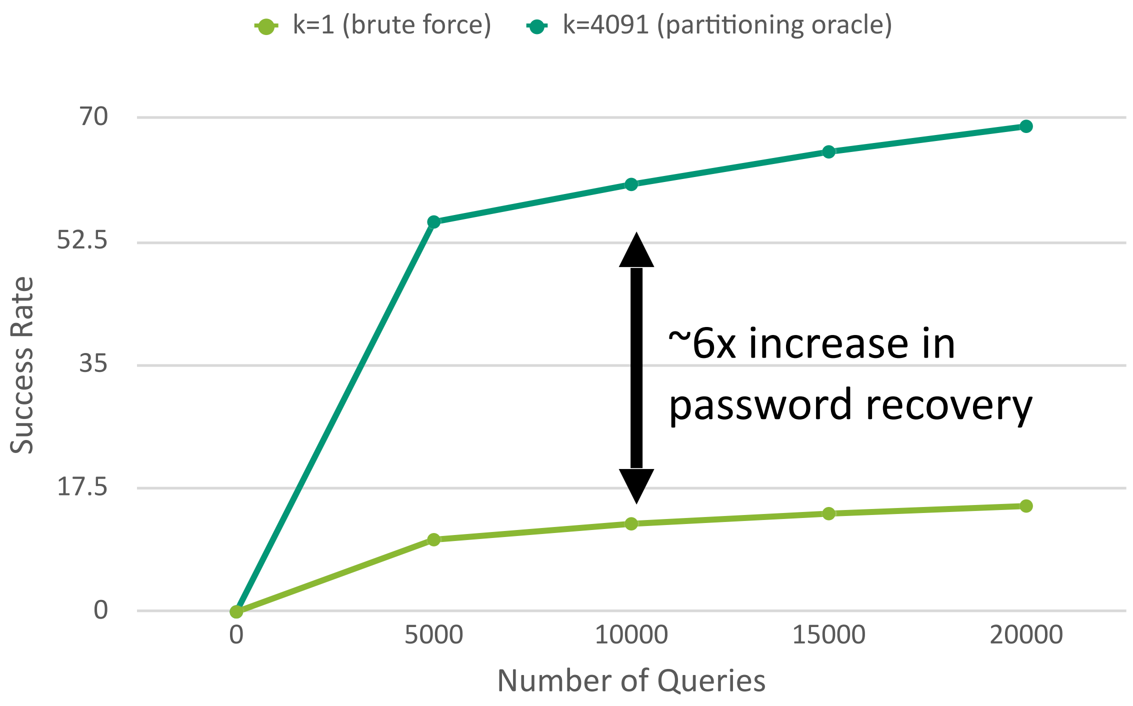


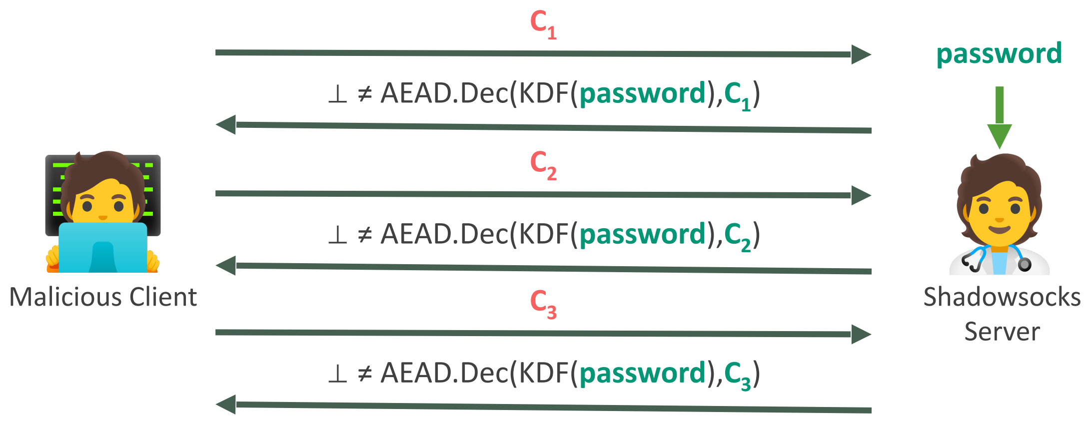
Security expected: one ciphertext = one guess

Previous slide: one ciphertext = ***two*** guesses

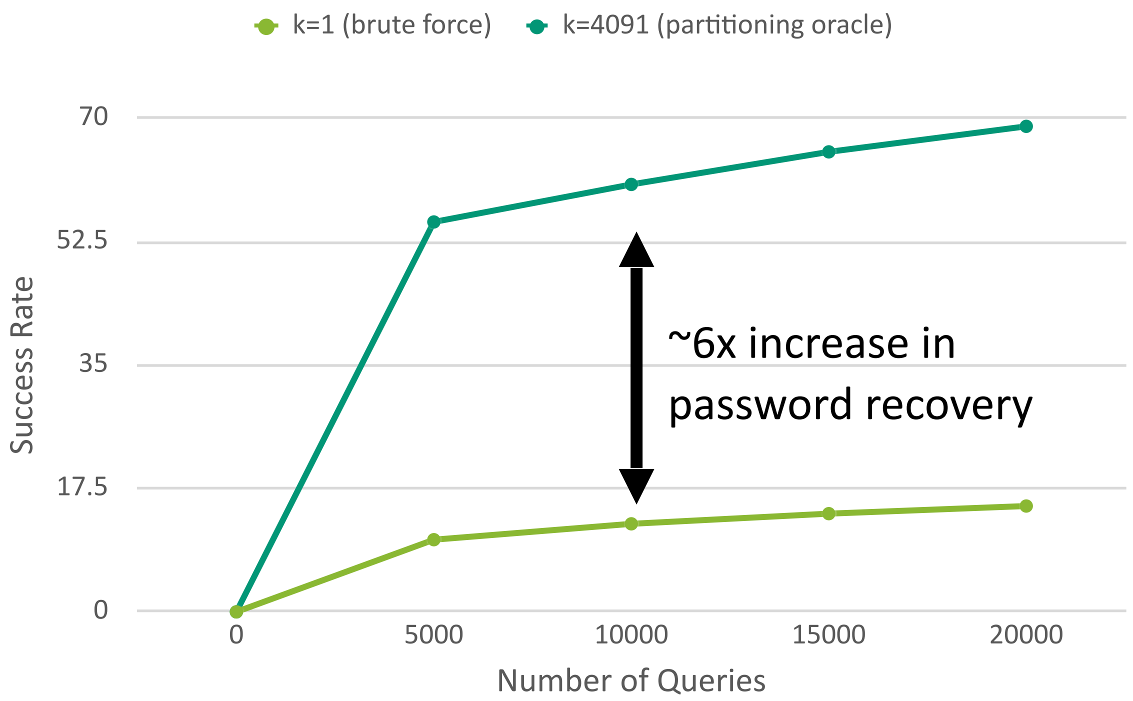
[LGR20]: one ciphertext = up to **4096 guesses** (for AES-GCM)

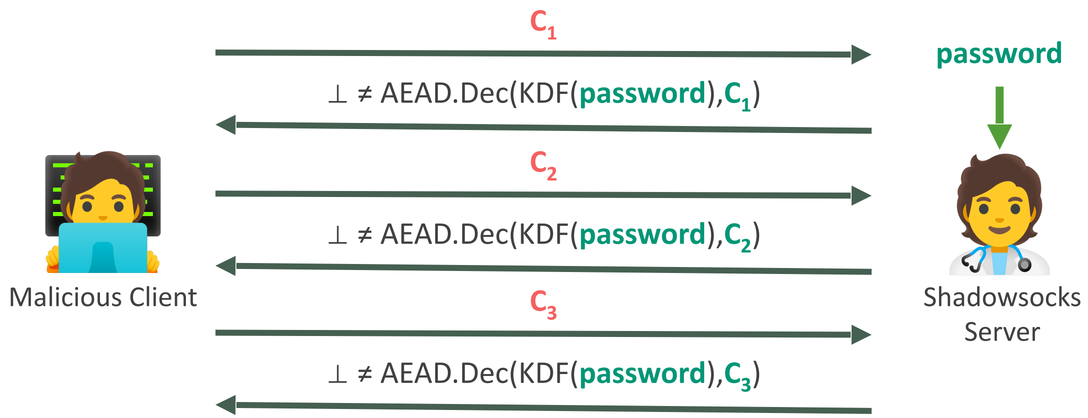
Threat #2: Partitioning oracle attacks [LGR20]





Threat #2: Partitioning oracle attacks [LGR20]





Partitioning oracle vulnerabilities also found in

- early, non-compliant OPAQUE implementations
- other open-source libraries

Summary of Vulnerabilities

Application	Attack	Impact	Citation
Facebook Messenger abuse reporting	Multi-recipient integrity	Makes it impossible to report specifically crafted images.	[GLR17] [DGRW19]
AWS Encryption SDK multi-recipient sending	Multi-recipient integrity	Can send different messages to different recipients.	[ADGKLS20]
…	…	…	…
Shadowsocks UDP	Partitioning oracle	Faster password guessing	[LGR20]
Non-compliant OPAQUE implementations	Partitioning oracle	Faster password guessing	[LGR20]
…	…	…	…

[GLR17] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message franking via committing authenticated encryption. ia.cr/2017/664
[LGR20] Julia Len, Paul Grubbs, and Thomas Ristenpart. Partitioning Oracle Attacks. ia.cr/2020/1491
[DGRW19] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. ia.cr/2019/016
[ADGKLS20] Ange Albertini, Thai Duong, Shay Gueron, Stefan Kölbl, Atul Luykx, and Sophie Schmieg. How to Abuse and Fix Authenticated Encryption Without Key Commitment. ia.cr/2020/1456

Attacks break the most widely used AEAD schemes

Attacks break the most widely used AEAD schemes

They do not invalidate prior security analyses …

Attacks break the most widely used AEAD schemes

They do not invalidate prior security analyses …

… they exploit **lack of key commitment**

Some Proposals for Key Commitment [ADGKLS20]

Key hashing [ADGKLS20]:
Append CR hash H(K)
to ciphertext

C = AEAD.Enc(K,N,A,M)
Output H(K) || C

Some Proposals for Key Commitment [ADGKLS20]

Key hashing [ADGKLS20]:
Append CR hash H(K)
to ciphertext

C = AEAD.Enc(**K**,**N**,**A**,**M**)
Output H(K) ‖ **C**

+ Simple, prevents attacks
– Longer ciphertexts
– Multiple primitives

Some Proposals for Key Commitment [ADGKLS20]

Key hashing [ADGKLS20]**:**
Append CR hash H(K)
to ciphertext

C = AEAD.Enc(**K**,**N**,**A**,**M**)
Output H(K) || **C**

+ Simple, prevents attacks
– Longer ciphertexts
– Multiple primitives

Padding zeros [ADGKLS20]**:**
Add plaintext redundancy,
check on decrypt

C = AEAD.Enc(**K**,**N**,**A**, $0^{2\lambda}$ || **M**)
Output **C**

Some Proposals for Key Commitment [ADGKLS20]

Key hashing [ADGKLS20]:
Append CR hash H(K)
to ciphertext

C = AEAD.Enc(K,N,A,M)
Output H(K) || C

+ Simple, prevents attacks
– Longer ciphertexts
– Multiple primitives

Padding zeros [ADGKLS20]:
Add plaintext redundancy,
check on decrypt

C = AEAD.Enc(K,N,A, $0^{2\lambda}$ || M)
Output C

+ Simple, fast
– Longer ciphertexts
– Only specific schemes

Some Proposals for Key Commitment [ADGKLS20]

Key hashing [ADGKLS20]:
Append CR hash H(K)
to ciphertext

C = AEAD.Enc(**K**,**N**,**A**,**M**)
Output H(K) ‖ **C**

+ Simple, prevents attacks
– Longer ciphertexts
– Multiple primitives

Padding zeros [ADGKLS20]:
Add plaintext redundancy,
check on decrypt

C = AEAD.Enc(**K**,**N**,**A**, $0^{2\lambda}$ ‖ **M**)
Output **C**

+ Simple, fast
– Longer ciphertexts
– Only specific schemes

We could standardize these or other key-committing solutions

We could standardize a key-committing solution

We could standardize a key-committing solution

4.3.3. Key commitment

Definition. An AEAD algorithm guarantees that it is difficult to find a tuple of the nonce, associated data, and ciphertext such that it can be decrypted correctly with more than one key.

Synonyms. Key-robustness, key collision resistance.

Further reading. [FOR17], [LGR21], [GLR17]

draft-irtf-cfrg-aead-properties-01

https://datatracker.ietf.org/doc/draft-irtf-cfrg-aead-properties/01/

We could standardize a key-committing solution

4.3.3. Key commitment

Definition. An AEAD algorithm guarantees that it is difficult to find a tuple of the nonce, associated data, and ciphertext such that it can be decrypted correctly with more than one key.

Synonyms. Key-robustness, key collision resistance.

Further reading. [FOR17], [LGR21], [GLR17]

draft-irtf-cfrg-aead-properties-01
https://datatracker.ietf.org/doc/draft-irtf-cfrg-aead-properties/01/

Topics for discussion include:

- The security and efficiency of current NIST modes
- Additional security features (e.g., misuse-resistance, key commitment, etc.) that would be desirable in a new encryption technique

The Third NIST Workshop on Block Cipher Modes of Operation
https://csrc.nist.gov/Events/2023/third-workshop-on-block-cipher-modes-of-operation

We could standardize a key-committing solution

4.3.3. Key commitment

Definition. An AEAD algorithm guarantees that it is difficult to find a tuple of the nonce, associated data, and ciphertext such that it can be decrypted correctly with more than one key.

Synonyms. Key-robust

Further reading. [FOR

draft-irtf-cfrg-aead-properties-01
https://datatracker.ietf.org/doc/draft-irtf-cfrg-aead-properties/01/

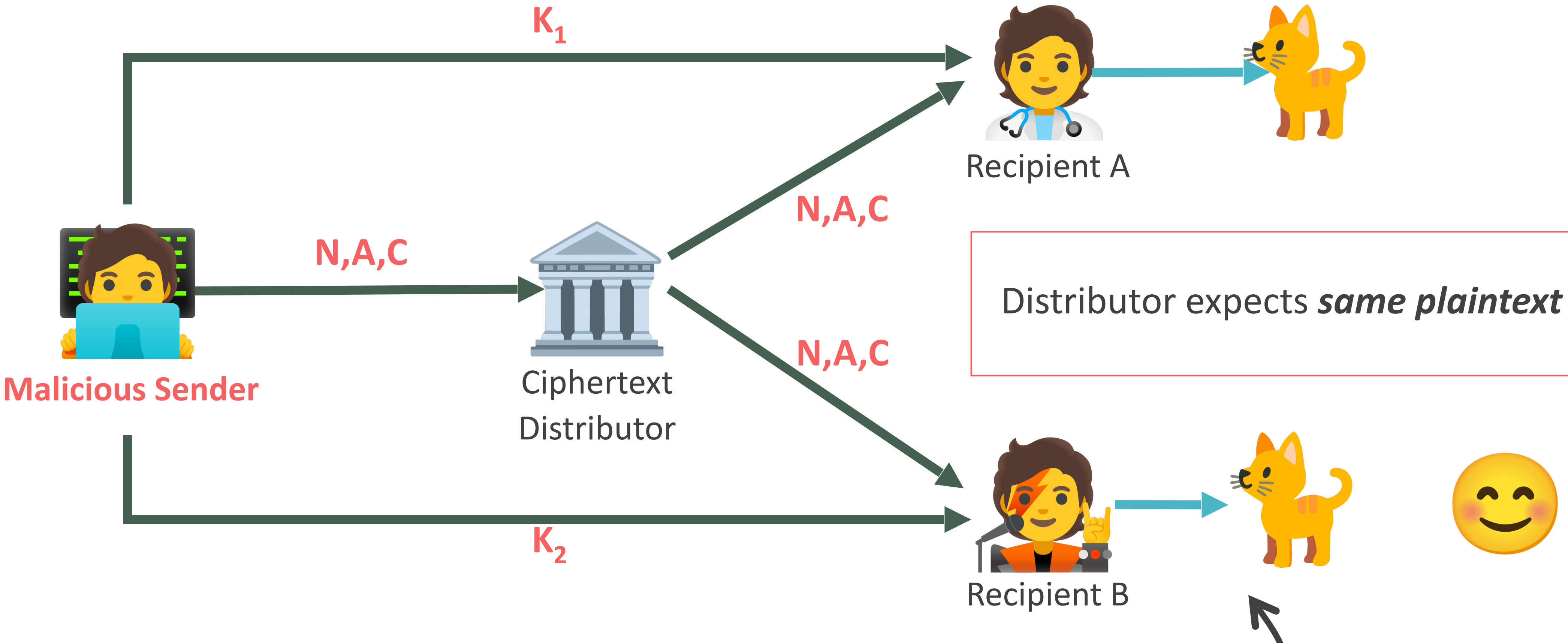
But we fear this is **short-sighted**

NIST modes

- Additional security features (e.g., misuse-resistance, key commitment, etc.) that would be desirable in a new encryption technique

The Third NIST Workshop on Block Cipher Modes of Operation
https://csrc.nist.gov/Events/2023/third-workshop-on-block-cipher-modes-of-operation

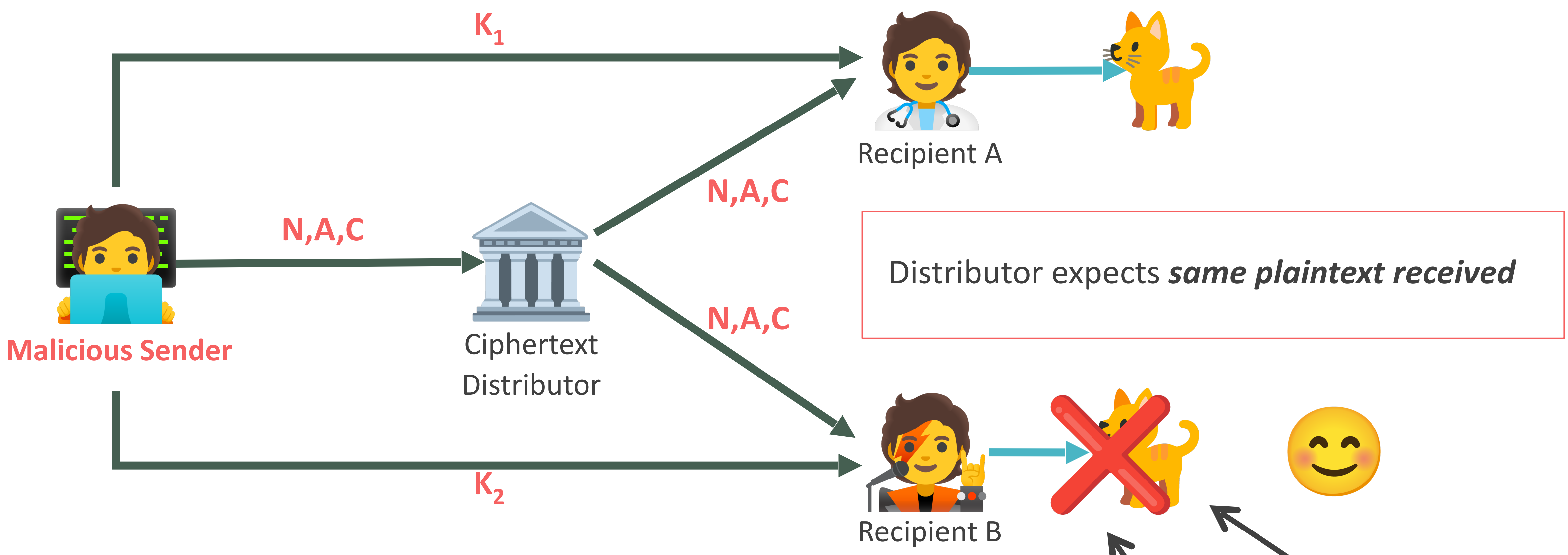
Revisiting Multi-Recipient Integrity [BT22,CR22,MLGR23]



Distributor expects ***same plaintext received***

If $K_1 = K_2$, both get same plaintext

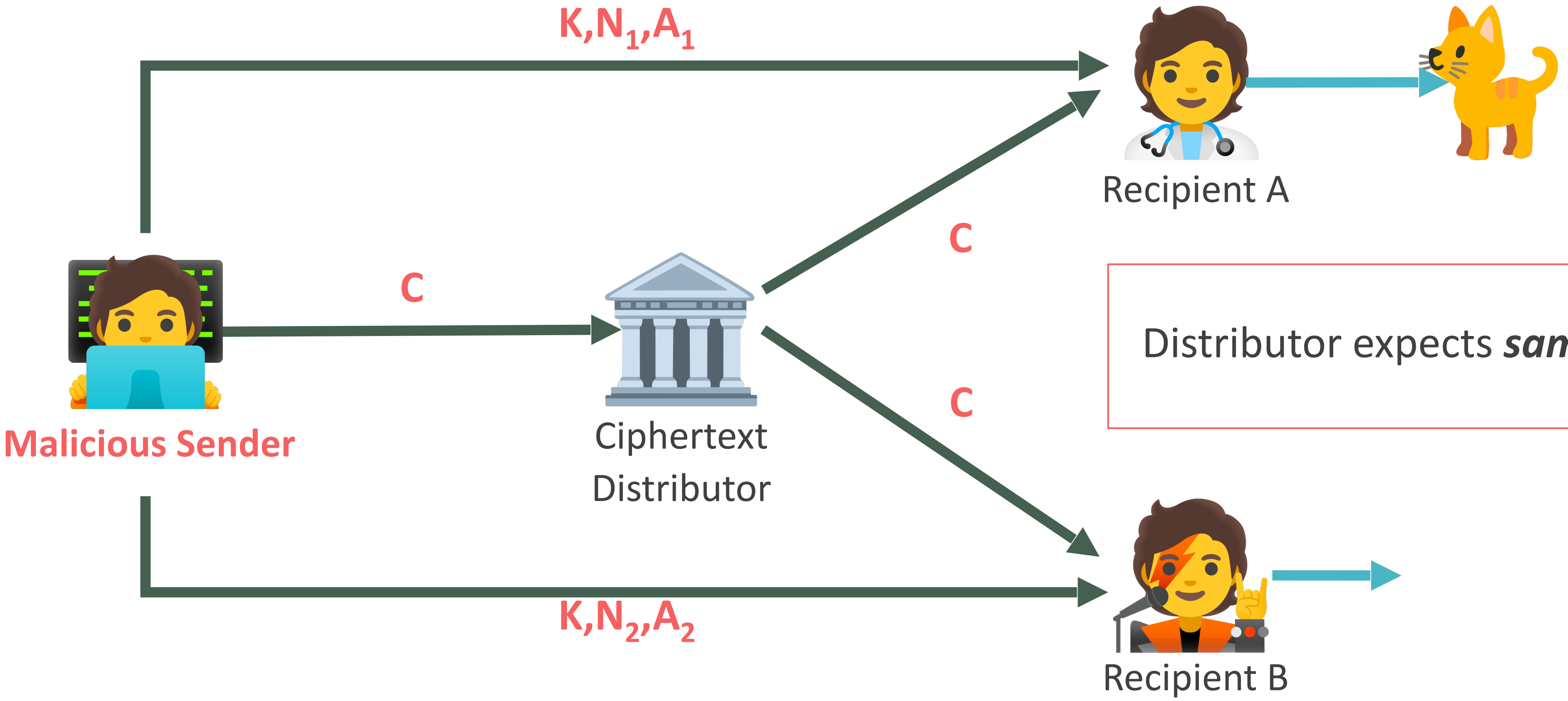
Revisiting Multi-Recipient Integrity [BT22,CR22,MLGR23]



If $K_1 = K_2$, both get same plaintext

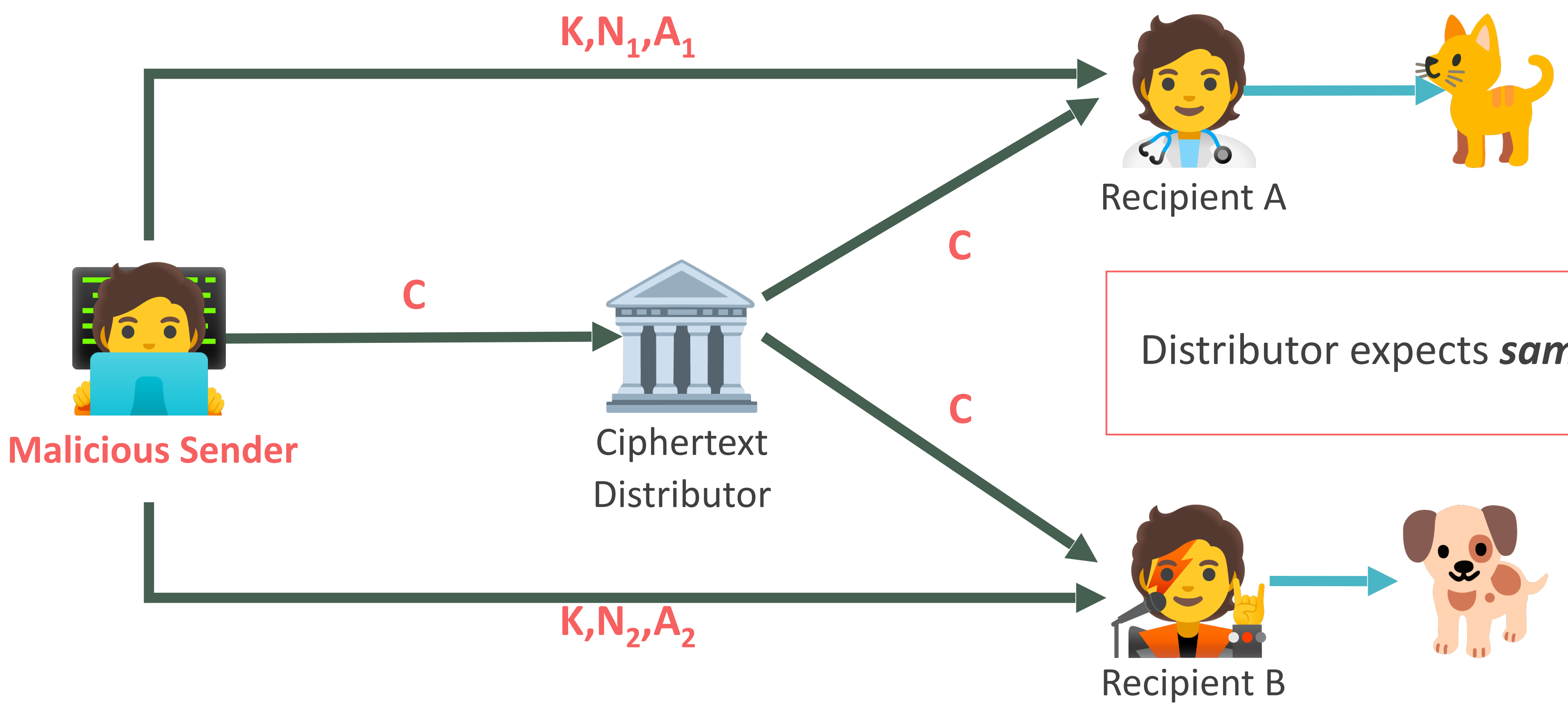
If $K_1 \neq K_2$, and key commitment, one gets error

Revisiting Multi-Recipient Integrity [BT22,CR22,MLGR23]



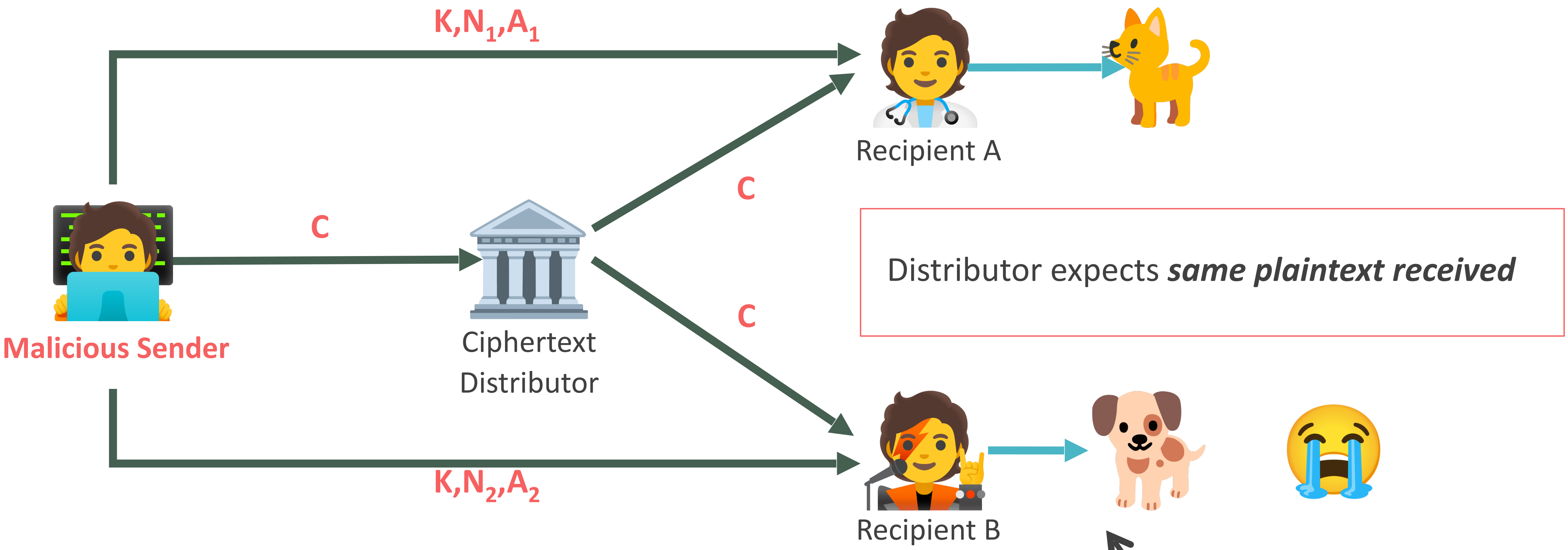
Distributor expects ***same plaintext received***

Revisiting Multi-Recipient Integrity [BT22,CR22,MLGR23]



Distributor expects ***same plaintext received***

Revisiting Multi-Recipient Integrity [BT22,CR22,MLGR23]



If **keys are same** but **N and A differ**,
even with key commitment, get different plaintexts

In-use AEAD schemes are not *context committing*

For AEAD = XXX computationally efficient to find

$$(K_1, N_1, A_1) \neq (K_2, N_2, A_2) \quad \text{and} \quad C$$

such that decryption

$$M_1 = \text{AEAD.Dec}(K_1\ N_1, A_1, C)$$
$$M_2 = \text{AEAD.Dec}(K_2, N_2, A_2, C)$$

succeeds

In-use AEAD schemes are not *context committing*

For AEAD = XXX computationally efficient to find

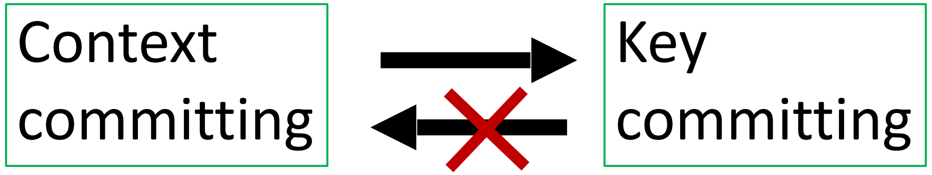
$(K_1, N_1, A_1) \neq (K_2, N_2, A_2)$ and C

such that decryption

M_1 = AEAD.Dec(K_1 N_1,A_1,C)

M_2 = AEAD.Dec(K_2,N_2,A_2,C)

succeeds



In-use AEAD schemes are not *context committing*

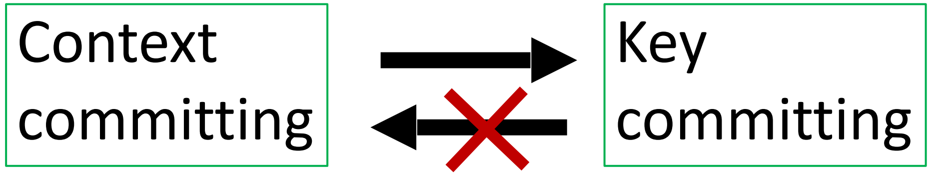
For AEAD = XXX computationally efficient to find

$$(K_1, N_1, A_1) \neq (K_2, N_2, A_2) \quad \text{and} \quad C$$

such that decryption

$$M_1 = \text{AEAD.Dec}(K_1\ N_1, A_1, C)$$
$$M_2 = \text{AEAD.Dec}(K_2, N_2, A_2, C)$$

succeeds



Large space of definitions **[BT22, CR22, MLGR23]**

Analogous to hash functions:

collision resistance ~ context commitment

preimage resistance ~ context discovery

In-use AEAD schemes are not *context committing*

For AEAD = XXX computationally efficient to find

$$(K_1, N_1, A_1) \neq (K_2, N_2, A_2) \quad \text{and} \quad C$$

such that decryption

$$M_1 = \text{AEAD.Dec}(K_1, N_1, A_1, C)$$
$$M_2 = \text{AEAD.Dec}(K_2, N_2, A_2, C)$$

succeeds

XXX	Key Committing Attack?	Context Committing Attack?
AES-GCM	[GLR17]	
ChaCha20 /Poly1305	[LGR20]	
AES-GCM-SIV	[Sch20, LGR20]	
AES-OCB3	[ADGKLS20]	
AES-SIV	[MLGR23]	
XSalsa20 /Poly1305	[LGR20]	

In-use AEAD schemes are not *context committing*

For AEAD = XXX computationally efficient to find

$$(K_1, N_1, A_1) \neq (K_2, N_2, A_2) \quad \text{and} \quad C$$

such that decryption

$$M_1 = \text{AEAD.Dec}(K_1, N_1, A_1, C)$$
$$M_2 = \text{AEAD.Dec}(K_2, N_2, A_2, C)$$

succeeds

XXX	Key Committing Attack?	Context Committing Attack?
AES-GCM	[GLR17]	[GLR17]
ChaCha20 /Poly1305	[LGR20]	[LGR20]
AES-GCM-SIV	[Sch20, LGR20]	[Sch20, LGR20]
AES-OCB3	[ADGKLS20]	[ADGKLS20]
AES-SIV	[MLGR23]	[MLGR23]
XSalsa20 /Poly1305	[LGR20]	[LGR20]

In-use AEAD schemes are not *context committing*

For AEAD = XXX computationally efficient to find

$(K_1, N_1, A_1) \neq (K_2, N_2, A_2)$ and C

such that decryption

M_1 = AEAD.Dec(K_1 N_1,A_1,C)
M_2 = AEAD.Dec(K_2,N_2,A_2,C)

succeeds

XXX	Key Committing Attack?	Context Committing Attack?
AES-GCM	[GLR17]	[GLR17]
ChaCha20 /Poly1305	[LGR20]	[LGR20]
AES-GCM-SIV	[Sch20, LGR20]	[Sch20, LGR20]
AES-OCB3	[ADGKLS20]	[ADGKLS20]
AES-SIV	[MLGR23]	[MLGR23]
XSalsa20 /Poly1305	[LGR20]	[LGR20]
Padding Zeros		[BH22]
Key hashing		[MGLR23]

Key commitment countermeasures don’t ensure context commitment (applies to: Padding Zeros, Key hashing)

Committing Encryption Timeline

Committing Encryption Timeline

Key commitment theory
[FOR17, GLR17]

2017

Committing Encryption Timeline

Key commitment theory
[FOR17, GLR17]

Invisible salamanders attack
[DGRW19]

2017

2019

Committing Encryption Timeline

Key commitment theory
[FOR17, GLR17]

2017

Invisible salamanders attack
[DGRW19]

2019

Partitioning oracle attack
and
Key commitment countermeasures
[LGR20, ADGKLS20]

2020

Committing Encryption Timeline

- **2017** — Key commitment theory [FOR17, GLR17]
- **2019** — Invisible salamanders attack [DGRW19]
- **2020** — Partitioning oracle attack *and* Key commitment countermeasures [LGR20, ADGKLS20]
- **2022** — Context commitment theory [BH22,CR22,MLGR23]

Committing Encryption Timeline

Key commitment theory
[FOR17, GLR17]

2017

Invisible salamanders attack
[DGRW19]

2019

Partitioning oracle attack
and
Key commitment countermeasures
[LGR20, ADGKLS20]

2020

Context commitment theory
[BH22,CR22,MLGR23]

2022

???

Future

Some Proposals for Context Commitment

Context hashing:
Append CR hash H(K,N,A) to ciphertext

C = AEAD.Enc(**K**,**N**,**A**,**M**)
Output H(K,N,A) || **C**

Some Proposals for Context Commitment

Context hashing:
Append CR hash H(K,N,A) to ciphertext

C = AEAD.Enc(**K**,**N**,**A**,**M**)
Output H(K,N,A) || **C**

+ Simple, prevents attacks
– Longer ciphertexts
– Slow for large A
– Multiple primitives

Some Proposals for Context Commitment

Context hashing:
Append CR hash H(K,N,A) to ciphertext

C = AEAD.Enc(K,N,A,M)
Output H(K,N,A) || C

- \+ Simple, prevents attacks
- – Longer ciphertexts
- – Slow for large A
- – Multiple primitives

CTX construction [CR22]:
Append CR hash of context and tag (saves space)

C,T = AEAD.Enc(K,N,A,M)
Output H(K,N,A,T) || C

Some Proposals for Context Commitment

Context hashing:
Append CR hash H(K,N,A) to ciphertext

C = AEAD.Enc(K,N,A,M)
Output H(K,N,A) || C

- \+ Simple, prevents attacks
- – Longer ciphertexts
- – Slow for large A
- – Multiple primitives

CTX construction [CR22]:
Append CR hash of context and tag (saves space)

C,T = AEAD.Enc(K,N,A,M)
Output H(K,N,A,T) || C

- \+ Simple, prevents attacks
- \+ Optimal length ciphertexts
- – Slow for large A
- – Multiple primitives

Some Proposals for Context Commitment

Context hashing:
Append CR hash H(K,N,A) to ciphertext

C = AEAD.Enc(**K**,**N**,**A**,**M**)
Output H(K,N,A) || **C**

- \+ Simple, prevents attacks
- – Longer ciphertexts
- – Slow for large A
- – Multiple primitives

CTX construction [CR22]:
Append CR hash of context and tag (saves space)

C,T = AEAD.Enc(**K**,**N**,**A**,**M**)
Output H(K,N,A,T) || C

- \+ Simple, prevents attacks
- \+ Optimal length ciphertexts
- – Slow for large A
- – Multiple primitives

Hash-based constructions [BDPA11, DGRW19]:
Duplex-style that use a single pass of hash function

Init(**K**)
Absorb(**N**,**A**)
C = Encrypt(**M**)
Output Squeeze() || **C**

Some Proposals for Context Commitment

Context hashing:
Append CR hash H(K,N,A) to ciphertext

C = AEAD.Enc(K,N,A,M)
Output H(K,N,A) || C

- \+ Simple, prevents attacks
- – Longer ciphertexts
- – Slow for large A
- – Multiple primitives

CTX construction [CR22]:
Append CR hash of context and tag (saves space)

C,T = AEAD.Enc(K,N,A,M)
Output H(K,N,A,T) || C

- \+ Simple, prevents attacks
- \+ Optimal length ciphertexts
- – Slow for large A
- – Multiple primitives

Hash-based constructions [BDPA11, DGRW19]:
Duplex-style that use a single pass of hash function

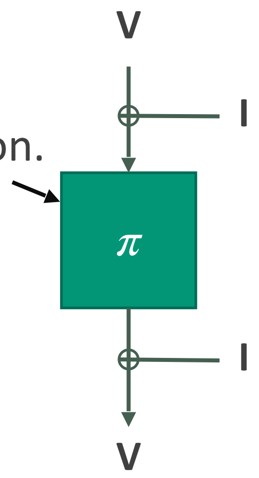
Init(K)
Absorb(N,A)
C = Encrypt(M)
Output Squeeze() || C

- \+ Simple, single primitive
- \+ Optimal length ciphertexts
- – Not parallelizable

OCH: Fast, Parallelizable Context Committing AEAD [BHLMR23]

OCH: Fast, Parallelizable Context Committing AEAD [BHLMR23]

Wide permutation like Keccak or Ascon.

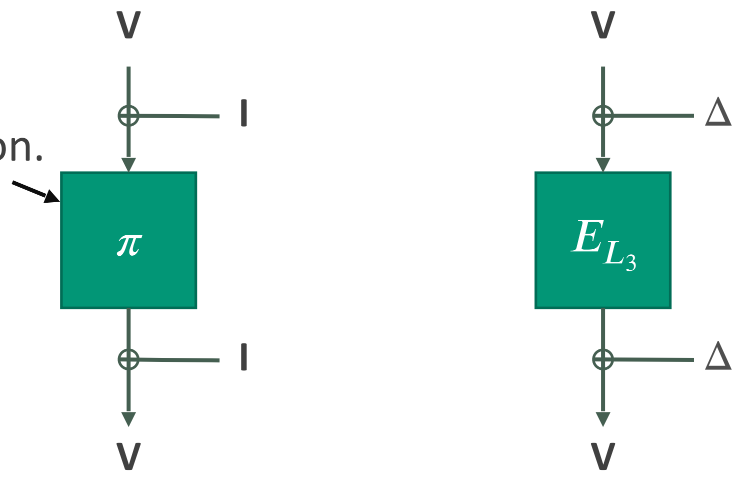


$$E(I, V) = \pi(V \oplus I) \oplus I$$

Even-Mansour block cipher [EM97]

OCH: Fast, Parallelizable Context Committing AEAD [BHLMR23]

Wide permutation like Keccak or Ascon.



$$E(I, V) = \pi(V \oplus I) \oplus I$$

$$\widetilde{E}_L^T(V) = E\left(L_3,\ V \oplus \Delta\right) \oplus \Delta$$

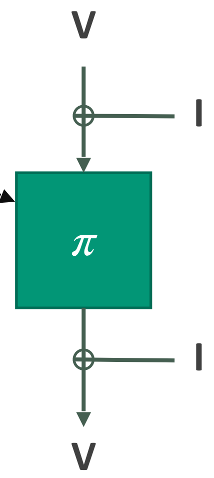
$$\Delta = \text{Offset}(L_1, L_2)$$

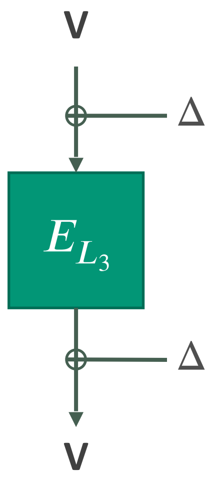
Even-Mansour block cipher [EM97]

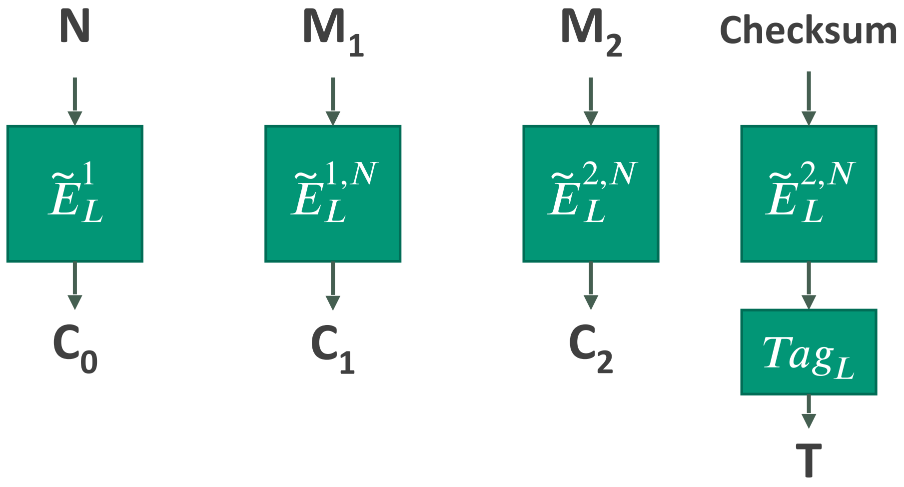
XE/XEX inspired tweakable block cipher [R04]

OCH: Fast, Parallelizable Context Committing AEAD [BHLMR23]

Wide permutation like Keccak or Ascon.







$$E(I, V) = \pi(V \oplus I) \oplus I$$

$$\widetilde{E}_L^T(V) = E\left(L_3,\ V \oplus \Delta\right) \oplus \Delta$$

$$\Delta = \mathrm{Offset}(L_1, L_2)$$

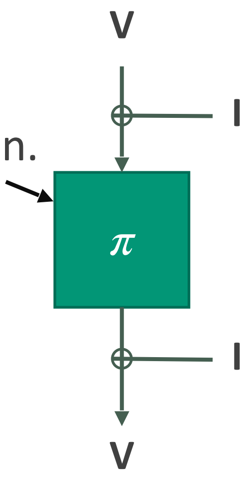
Even-Mansour block cipher [EM97]

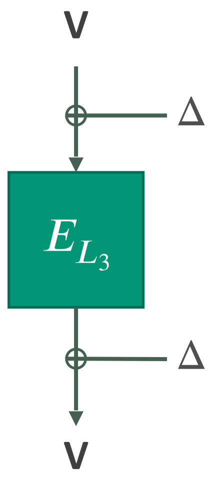
XE/XEX inspired tweakable block cipher [R04]

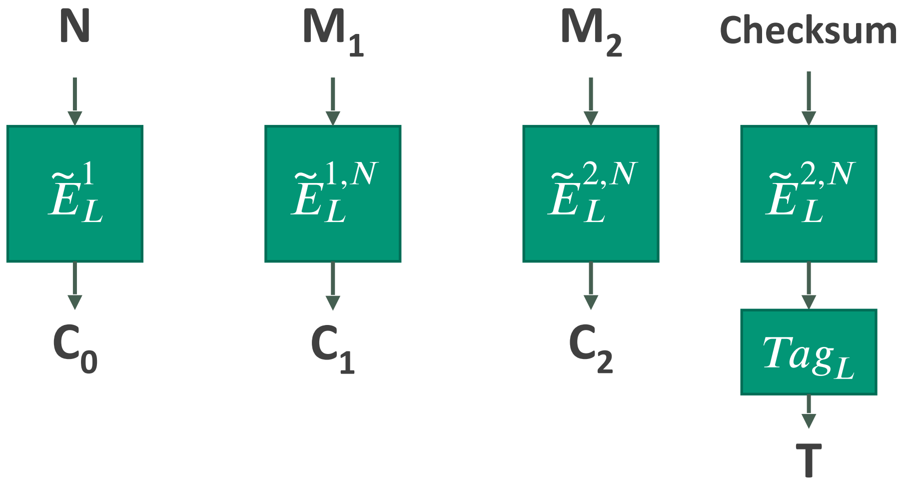
OCB inspired authenticated encryption [RBB03]

OCH: Fast, Parallelizable Context Committing AEAD [BHLMR23]

Wide permutation like Keccak or Ascon.







$$E(I,V) = \pi(V \oplus I) \oplus I$$

$$\widetilde{E}_L^T(V) = E\big(L_3,\ V \oplus \Delta\big) \oplus \Delta$$

$$\Delta = \mathrm{Offset}(L_1, L_2)$$

Even-Mansour block cipher [EM97]

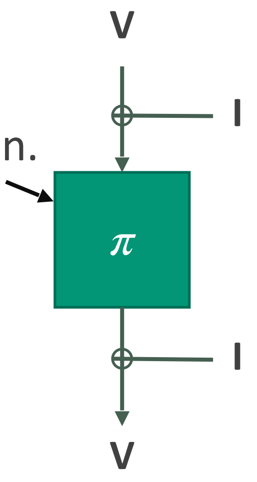
XE/XEX inspired tweakable block cipher [R04]

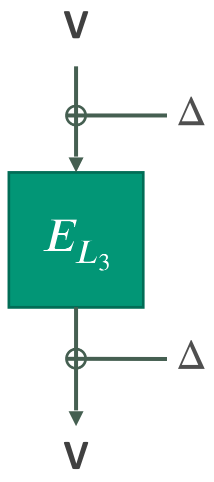
OCB inspired authenticated encryption [RBB03]

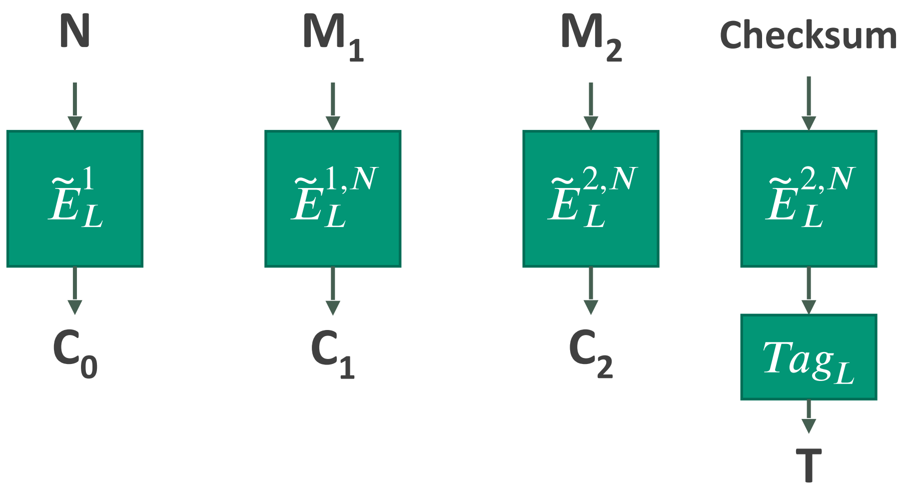
HN4 inspired nonce-hiding [BNT19]

OCH: Fast, Parallelizable Context Committing AEAD [BHLMR23]

Wide permutation like Keccak or Ascon.







$$E(I, V) = \pi(V \oplus I) \oplus I$$

$$\widetilde{E}_L^T(V) = E\big(L_3,\ V \oplus \Delta\big) \oplus \Delta$$

$$\Delta = \mathrm{Offset}(L_1, L_2)$$

Even-Mansour block cipher [EM97]

XE/XEX inspired tweakable block cipher [R04]

OCB inspired authenticated encryption [RBB03]

HN4 inspired nonce-hiding [BNT19]

+ Simple, single primitive
+ Optimal length ciphertexts
+ Maximally parallelizable

Is context committing AEAD right for you?

Is context committing AEAD right for you?

Yes

Is context committing AEAD right for you?

Yes

1. Future-proof against potential context commitment attacks.

Is context committing AEAD right for you?

Yes

1. Future-proof against potential context commitment attacks.
2. Minimal performance overhead over a key committing scheme.

Next Steps

Next Steps

Build context committing AEAD schemes.

Next Steps

Build context committing AEAD schemes.

Standardize a few canonical context committing AEAD schemes.

Next Steps

Build context committing AEAD schemes.

Standardize a few canonical context committing AEAD schemes.

Deploy these few canonical context committing AEAD schemes.

Next Steps

Build context committing AEAD schemes.

Standardize a few canonical context committing AEAD schemes.

Deploy these few canonical context committing AEAD schemes.

See papers for lots more: ia.cr/2022/268 & ia.cr/2022/1260 & preprint.link/ec23

Next Steps

Build context committing AEAD schemes.

Standardize a few canonical context committing AEAD schemes.

Deploy these few canonical context committing AEAD schemes.

See papers for lots more: ia.cr/2022/268 & ia.cr/2022/1260 & preprint.link/ec23

Thank you! 💙

Next Steps

Build context committing AEAD schemes.

Standardize a few canonical context committing AEAD schemes.

Deploy these few canonical context committing AEAD schemes.

See papers for lots more: ia.cr/2022/268 & ia.cr/2022/1260 & preprint.link/ec23

Thank you! 💙

Emoji in figures from Noto Emoji.

Thanks to my co-authors and the Cornell Security Seminar for feedback.

References

[EM97] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. https://doi.org/10.1007/s001459900025

[RBB03] Phillip Rogaway, Mihir Bellare, and John Black. OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption. https://www.cs.ucdavis.edu/~rogaway/papers/ocb-full.pdf

[R04] Phillip Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. https://www.cs.ucdavis.edu/~rogaway/papers/offsets.pdf

[BDPA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: single-pass authenticated encryption and other applications. ia.cr/2011/499

[FOR17] Pooya Farshim, Claudio Orlandi, and Răzvan Roşie. Security of Symmetric Primitives under Incorrect Usage of Keys. ia.cr/2017/288

[GLR17] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message franking via committing authenticated encryption. ia.cr/2017/664

[DGRW19] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. ia.cr/2019/016

References

[BNT19] Mihir Bellare, Ruth Ng, and Björn Tackmann. Nonces are Noticed: AEAD Revisited. ia.cr/2019/624

[LGR20] Julia Len, Paul Grubbs, and Thomas Ristenpart. Partitioning Oracle Attacks. ia.cr/2020/1491

[ADGKLS20] Ange Albertini, Thai Duong, Shay Gueron, Stefan Kölbl, Atul Luykx, and Sophie Schmieg. How to Abuse and Fix Authenticated Encryption Without Key Commitment. ia.cr/2020/1456

[BH22] Mihir Bellare and Viet Tung Hoang. Efficient schemes for committing authenticated encryption. ia.cr/2022/268

[CR22] John Chan and Phillip Rogaway. On committing authenticated-encryption. ia.cr/2022/1260

[MLGR23] Sanketh Menda, Julia Len, Paul Grubbs, and Thomas Ristenpart. Context Discovery and Commitment Attacks: How to Break CCM, EAX, SIV, and More. preprint.link/ec23

[BHLMR23] Mihir Bellare, Viet Tung Hoang, Julia Len, Sanketh Menda, Thomas Ristenpart. In preparation.