Partitioning Oracle Attacks

Julia Len Paul Grubbs Thomas Ristenpart

Cornell Tech

USENIX Security 2021

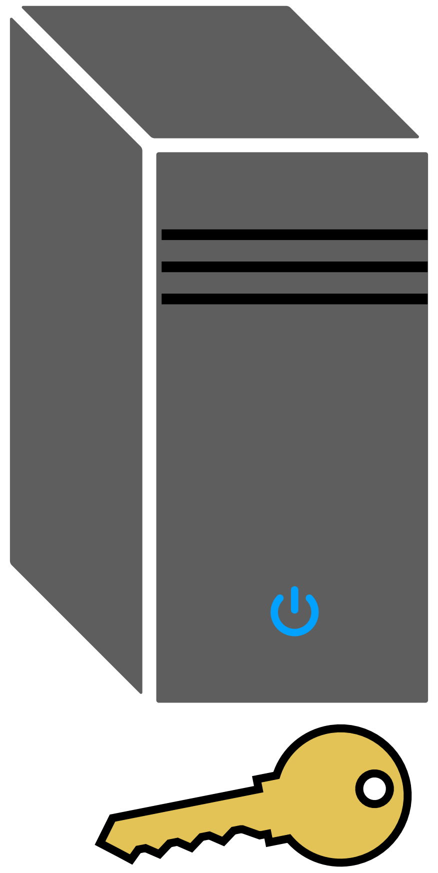
Authenticated Encryption

For simplicity, we ignore associated data in this presentation

Nonce N

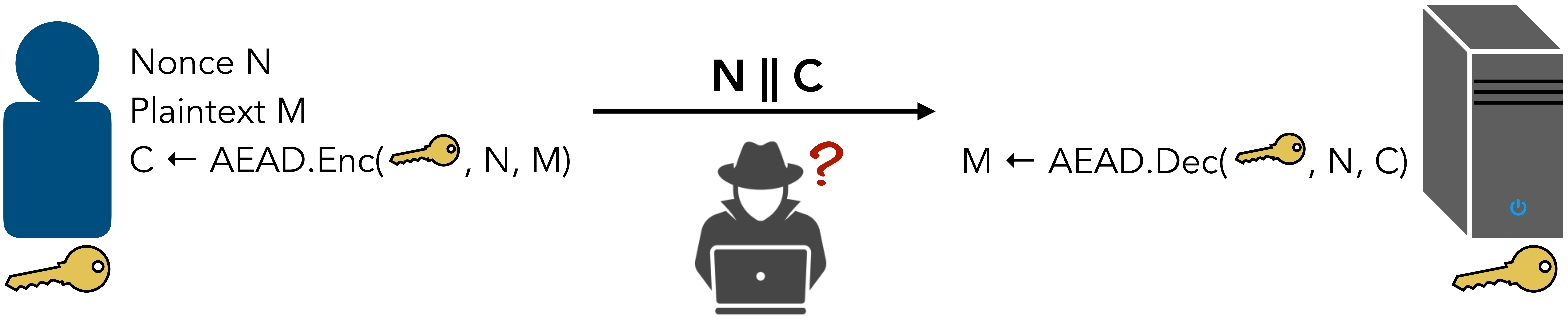
Plaintext M

C ← AEAD.Enc(, N, M)



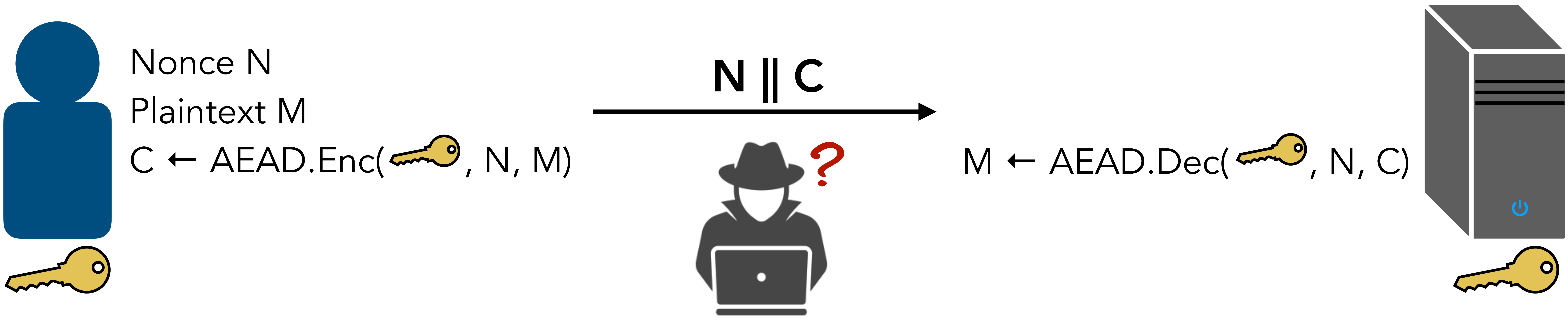
Authenticated Encryption

For simplicity, we ignore associated data in this presentation



Authenticated Encryption

For simplicity, we ignore associated data in this presentation



Popular

- AES-GCM
- XSalsa20/Poly1305
- ChaCha20/Poly1305
- AES-GCM-SIV

Easy to use

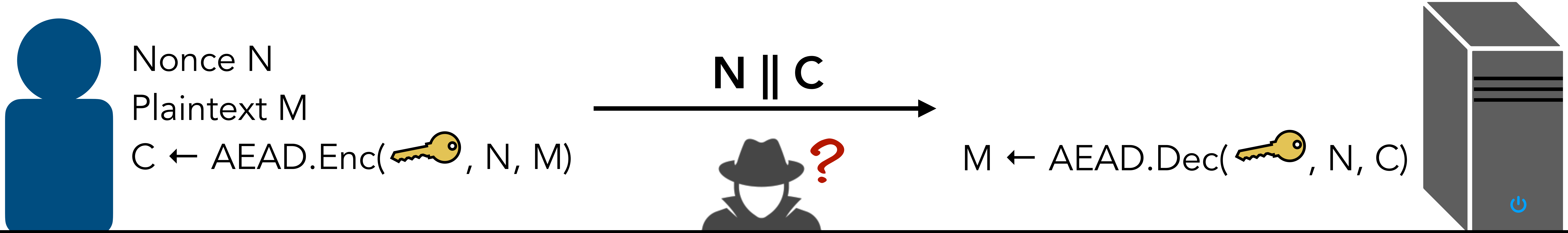
- Efficient
- Standardized
- Widely supported

Secure

- Proven CCA-secure
- Confidentiality
- Integrity

Authenticated Encryption

For simplicity, we ignore associated data in this presentation



But don't target robustness, also called **committing AEAD**, as a security goal

[ABN TCC'10], [FLPQ PKC'13] for PKE, [FOR FSE'17] for AEAD

- AES-GCM
- XSalsa20/Poly1305
- ChaCha20/Poly1305
- AES-GCM-SIV

- Efficient
- Standardized
- Widely supported

- Proven CCA-secure
- Confidentiality
- Integrity

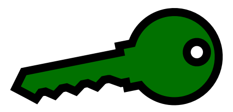
(Non-) Committing AEAD

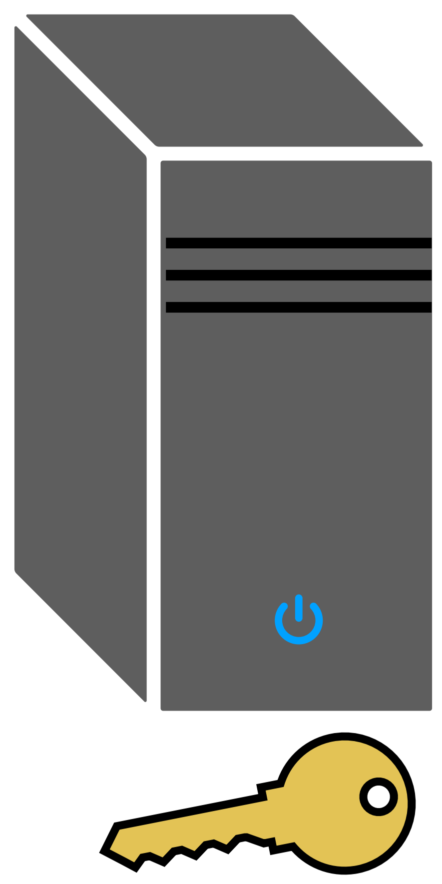
For simplicity, we ignore associated data in this presentation

Nonce N′

Ciphertext C′

M ← AEAD.Dec(, N′, C′)

M* ← AEAD.Dec(, N′, C′)



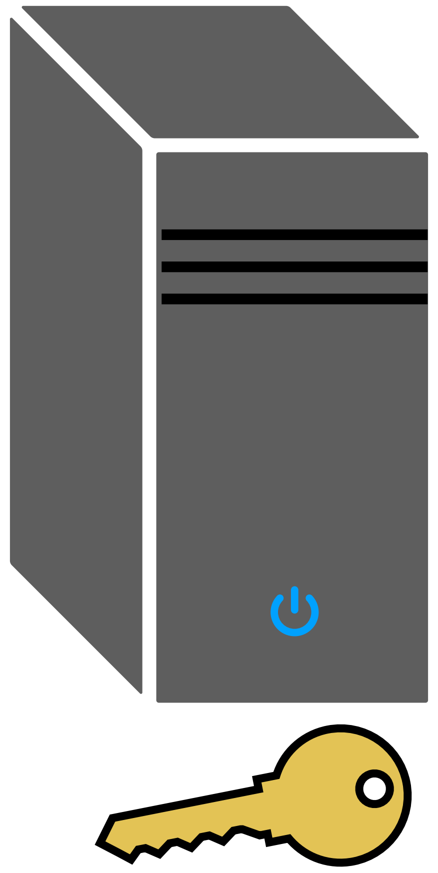
(Non-) Committing AEAD

For simplicity, we ignore associated data in this presentation

Nonce N′
Ciphertext C′

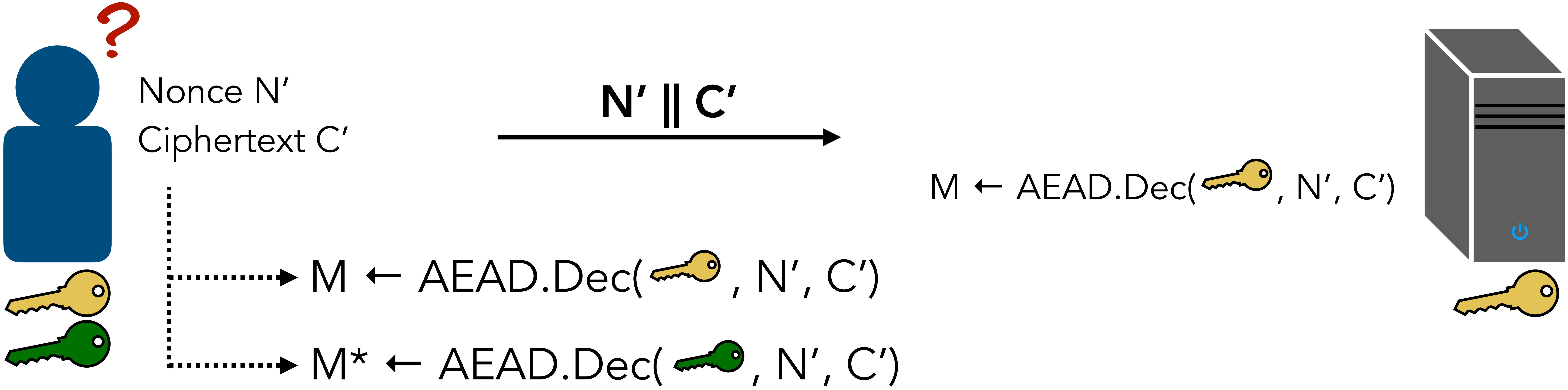
M ← AEAD.Dec((yellow key) , N′, C′)

M* ← AEAD.Dec((green key) , N′, C′)



(Non-) Committing AEAD

For simplicity, we ignore associated data in this presentation



(Non-) Committing AEAD

For simplicity, we ignore associated data in this presentation

Nonce N′

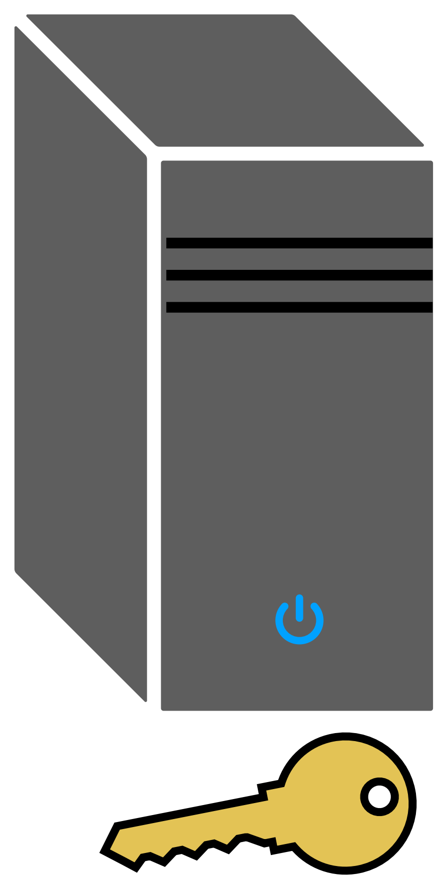
Ciphertext C′

N′ || C′

M ← AEAD.Dec(, N′, C′)

M ← AEAD.Dec(, N′, C′)

M* ← AEAD.Dec(, N′, C′)



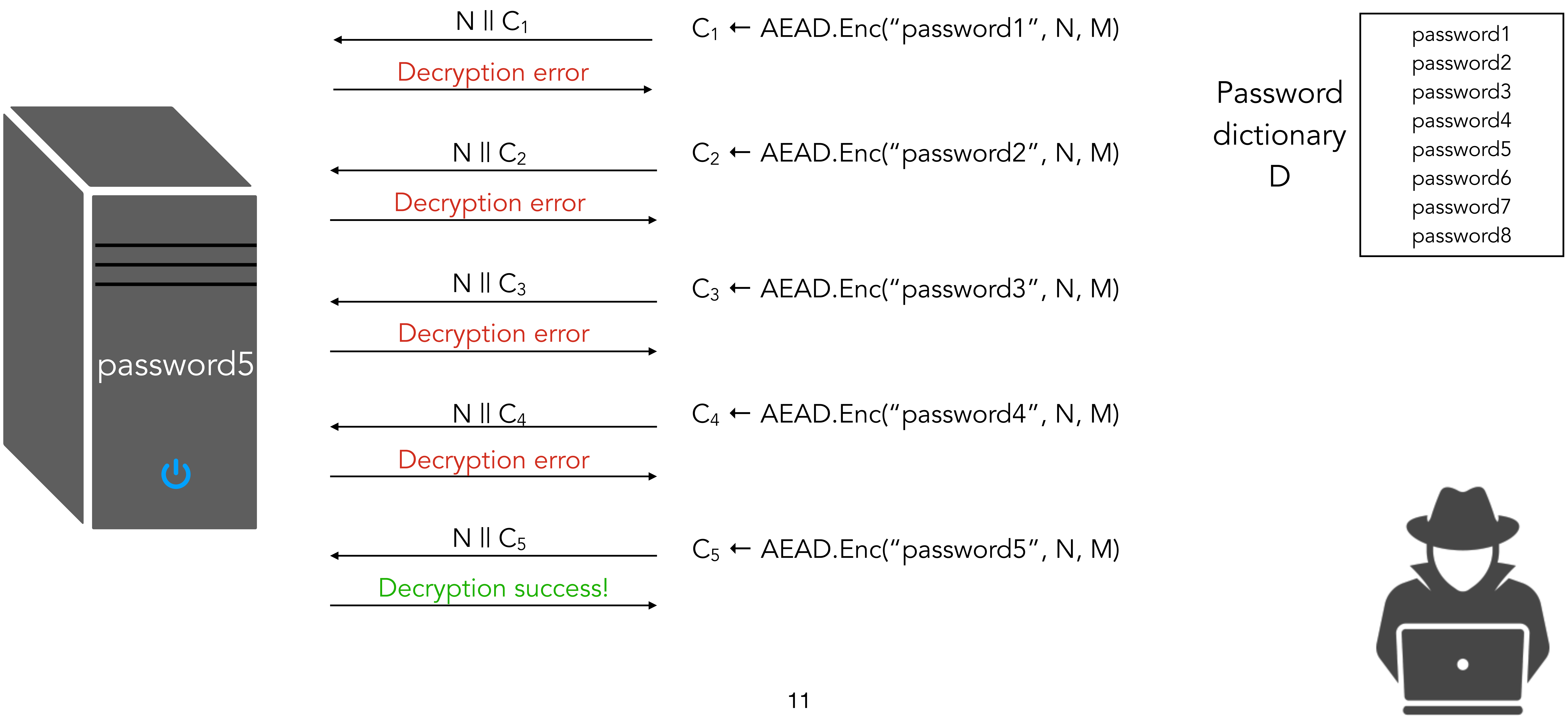
No guarantee the sender actually knows the *exact* key the recipient will use to decrypt!

Not considered an essential security goal, except in moderation settings [GLR CRYPTO’17], [DGRW CRYPTO’18]

Password dictionary D

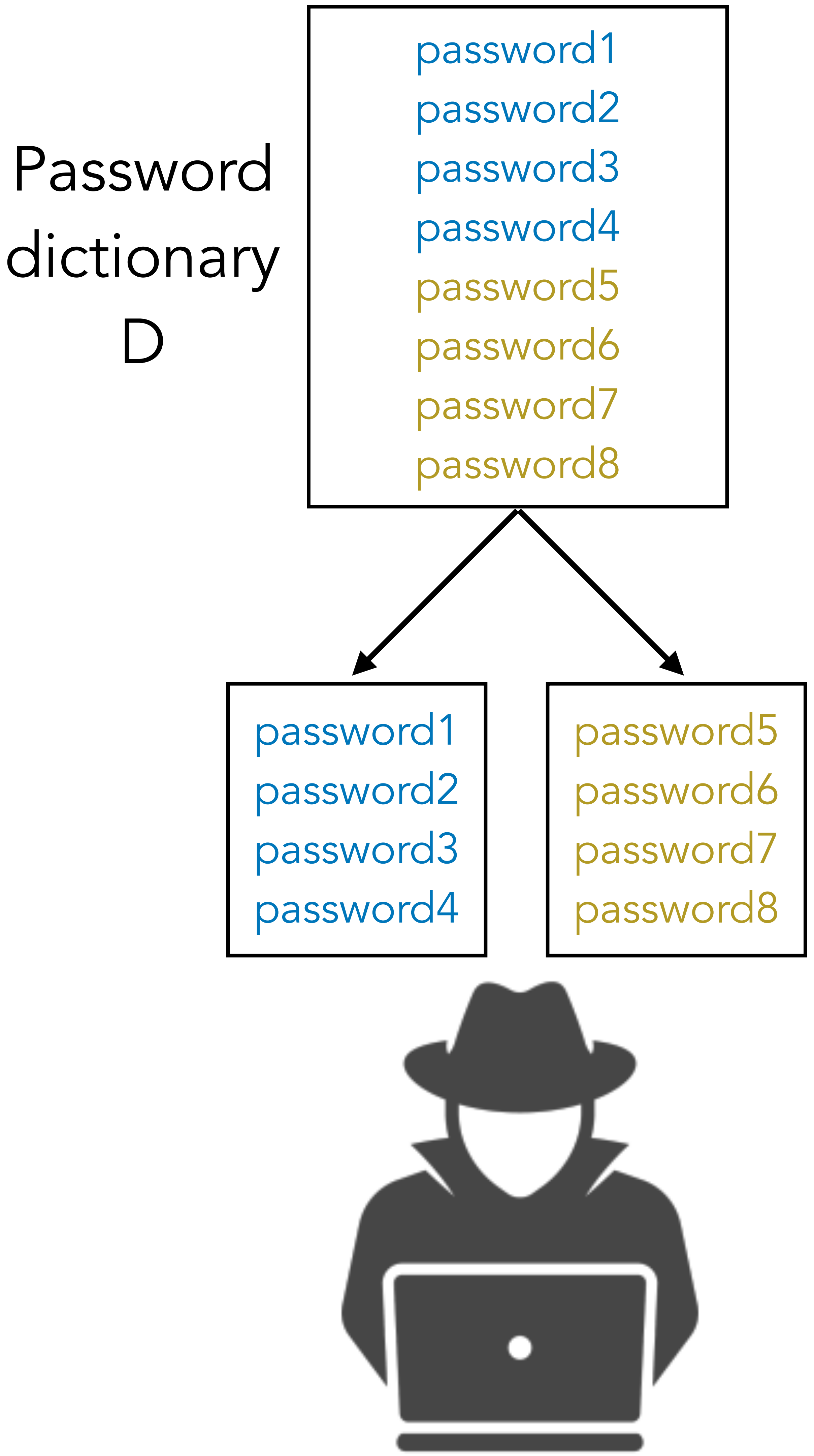
password1
password2
password3
password4
password5
password6
password7
password8

Brute-force Dictionary Attack



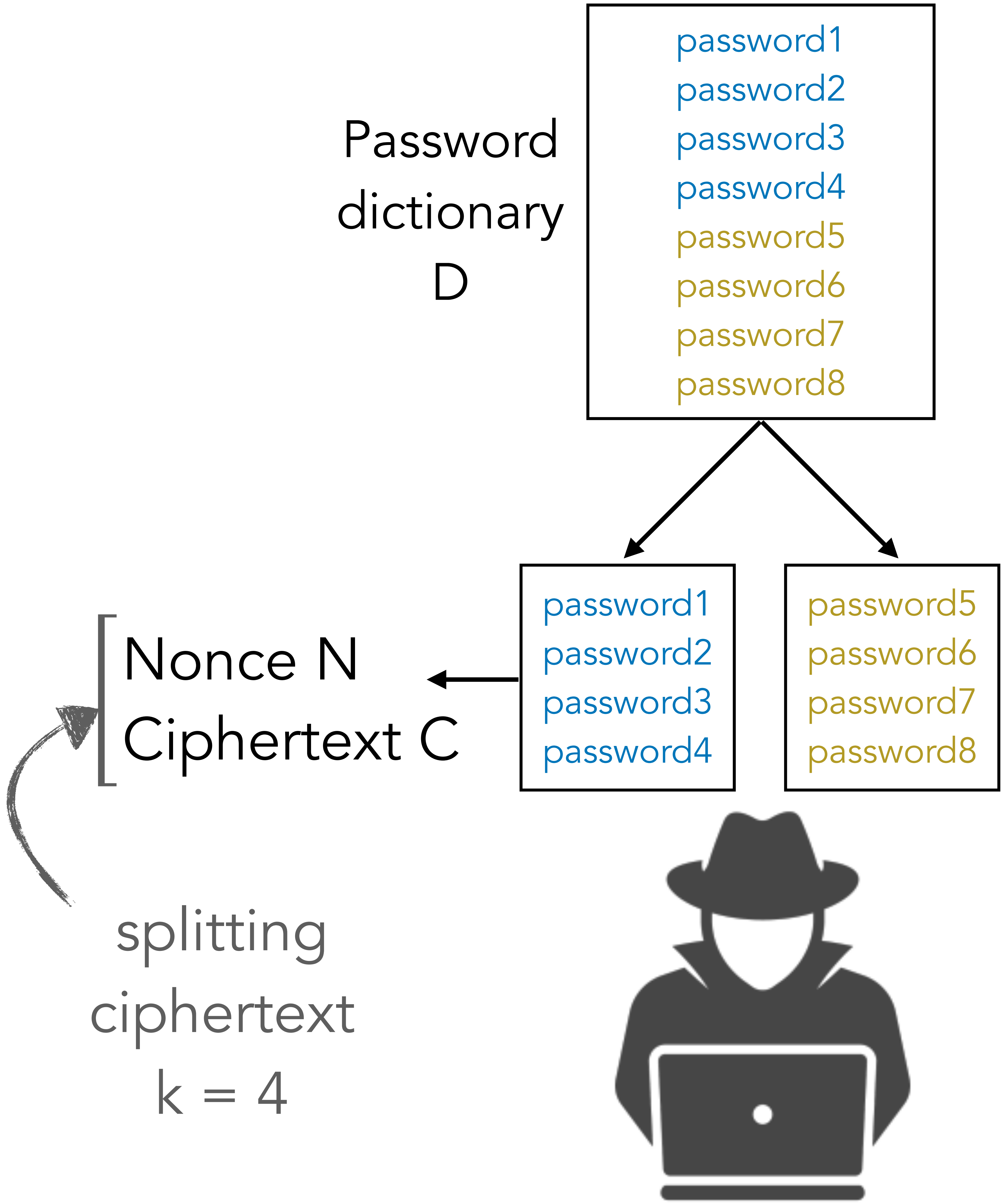
Partitioning Oracle Attack

A high level overview of our attack



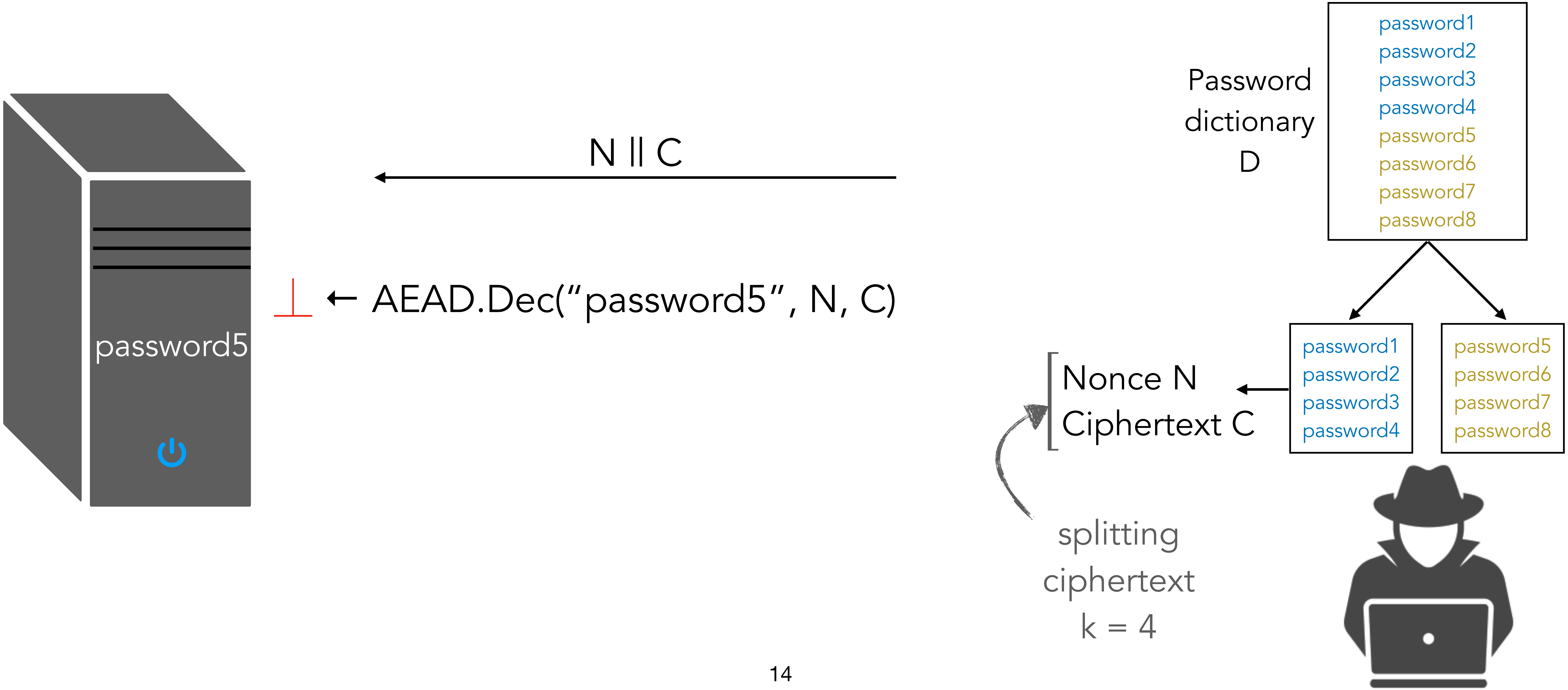
Partitioning Oracle Attack

A high level overview of our attack



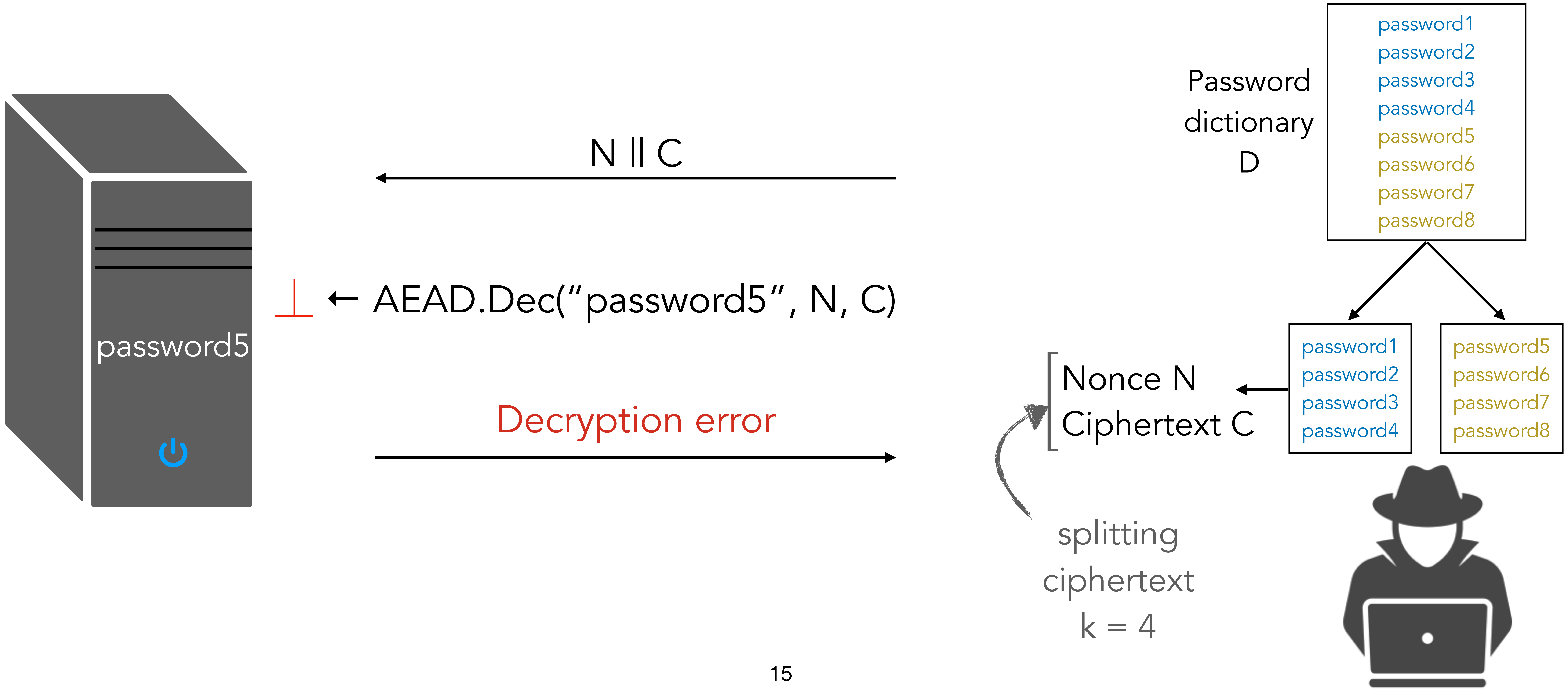
Partitioning Oracle Attack

A high level overview of our attack



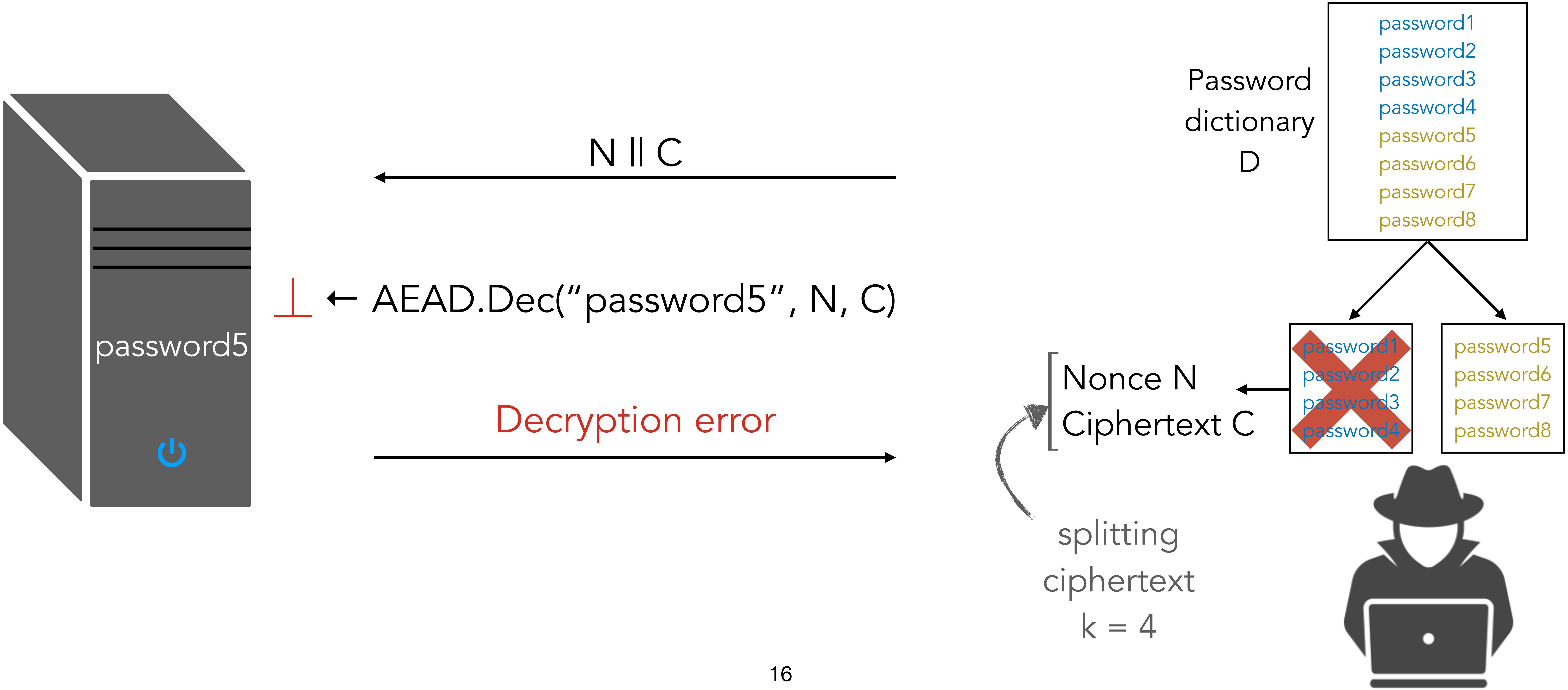
Partitioning Oracle Attack

A high level overview of our attack



Partitioning Oracle Attack

A high level overview of our attack



Partitioning Oracle Attack

D $k = 1$

↓

D_1 $k = 1$

↓

D_2 $k = 1$

↓

D_3 $k = 1$

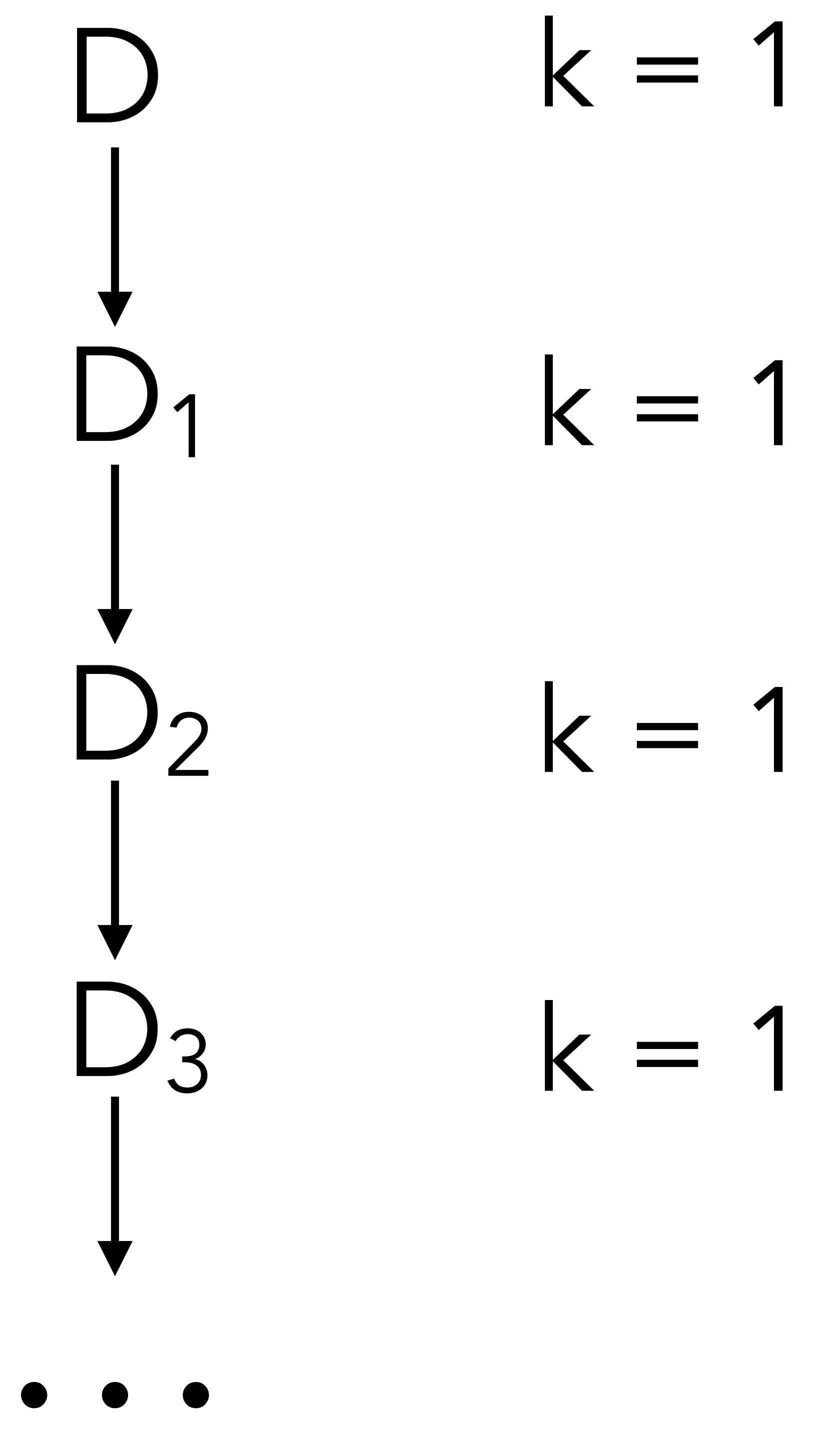
↓

…

Brute-force dictionary attack

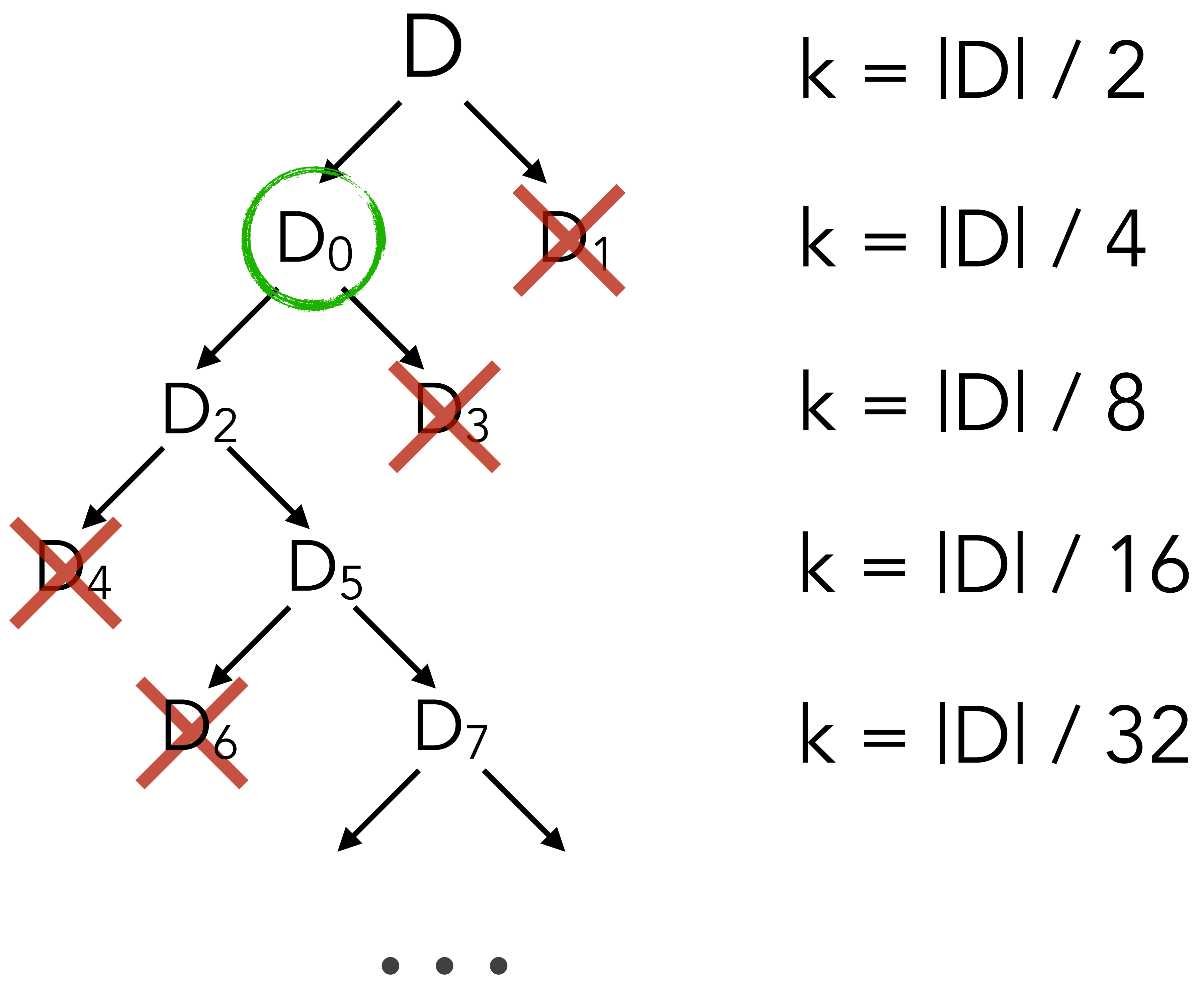
Requires $\mathcal{O}(|D|)$ queries to learn the password

Partitioning Oracle Attack



Brute-force dictionary attack

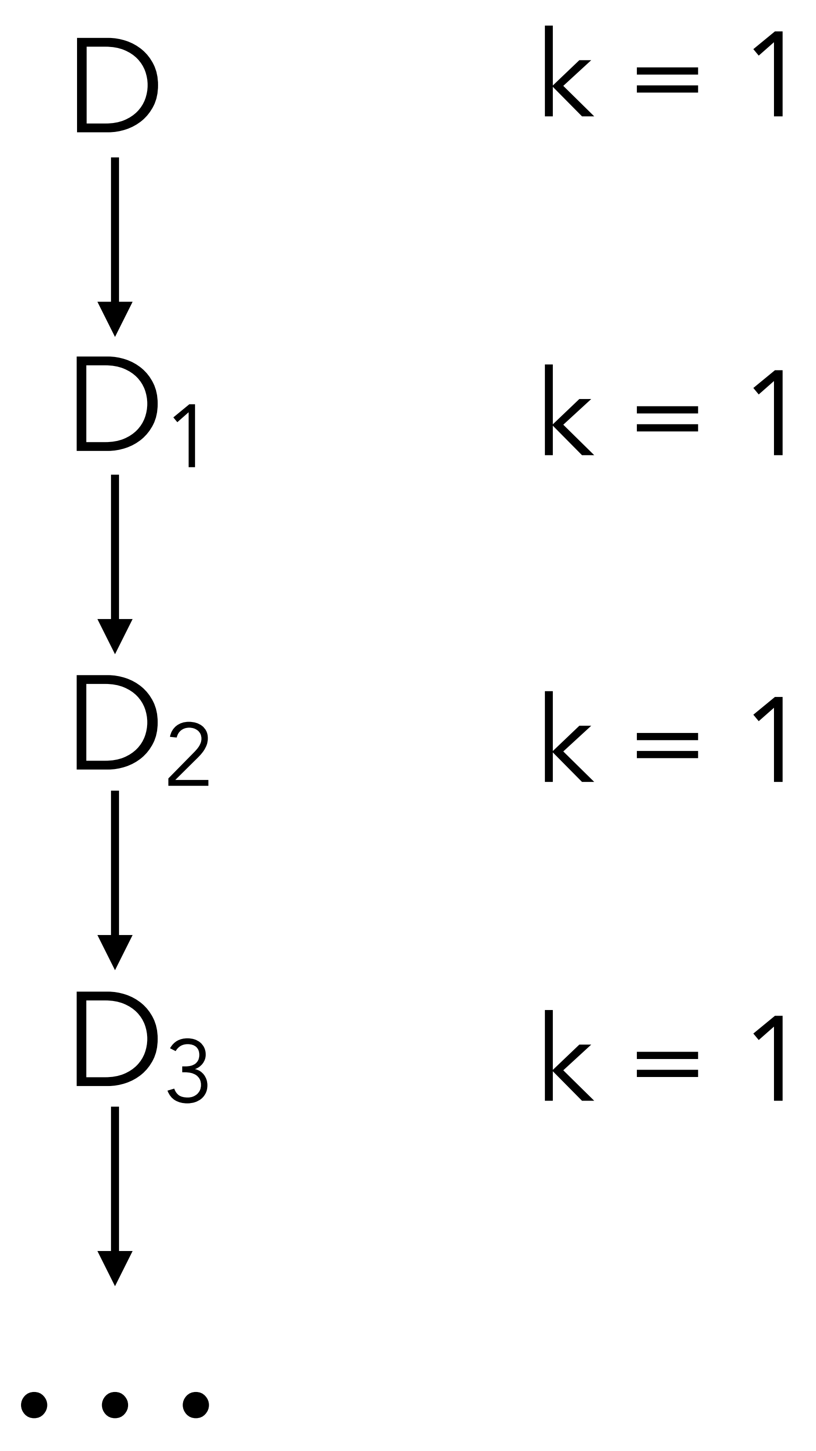
Requires $\mathcal{O}(|D|)$ queries to learn the password



Requires $\mathcal{O}(\log |D|)$ queries to learn the password

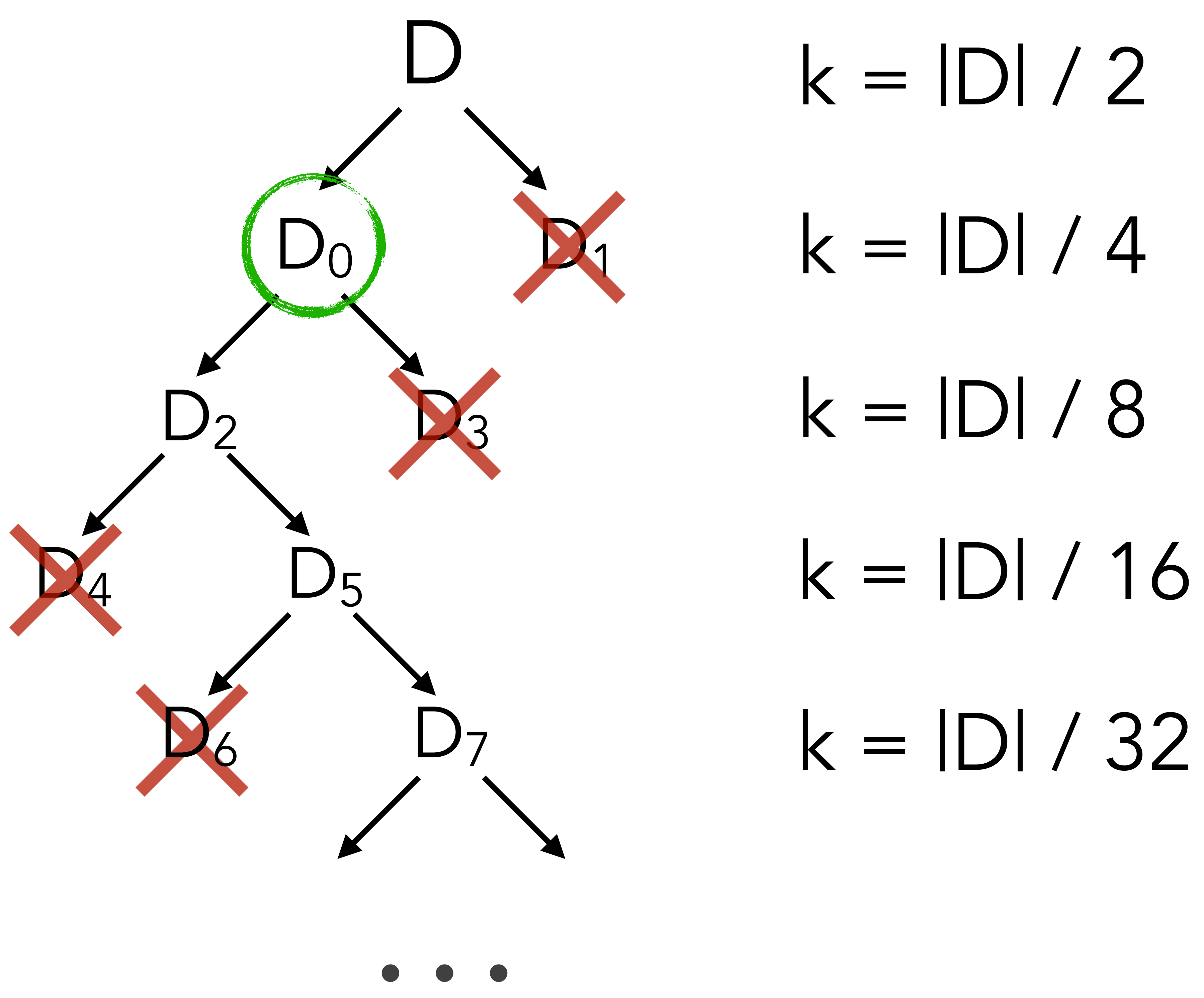
Exponential speedup over brute-force dictionary attack!

Partitioning Oracle Attack



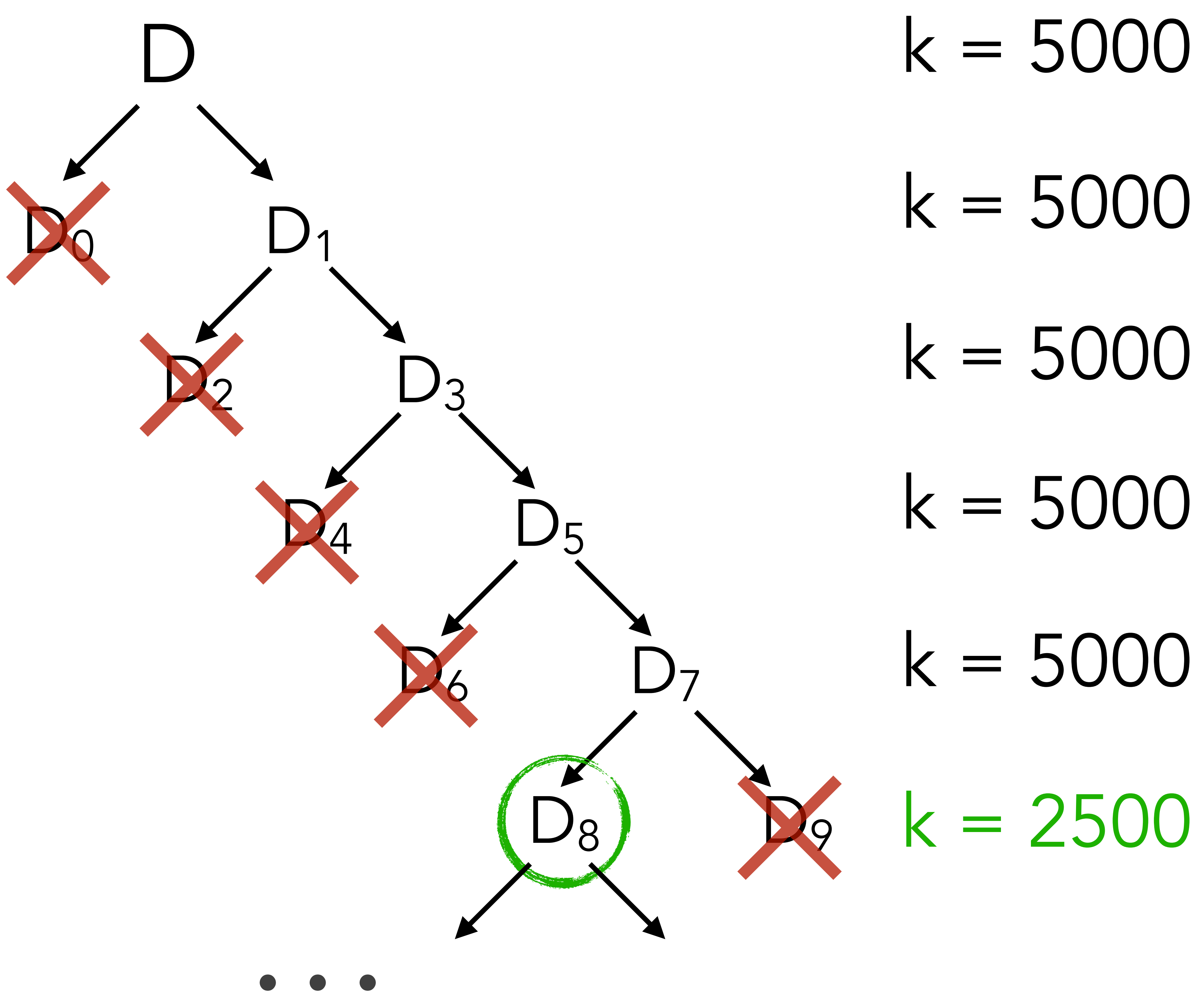
Brute-force dictionary attack

Requires $\mathcal{O}(|D|)$ queries to learn the password



Requires $\mathcal{O}(\log |D|)$ queries to learn the password

Exponential speedup over brute-force dictionary attack!



|D| is large so a more realistic case is k = 5000

This still offers a *good speedup* over brute-force

Partitioning oracle attacks rely on:

1. Building splitting ciphertexts that can decrypt under k > 1 different keys

2. Access to a partitioning oracle

Partitioning oracle attacks rely on:

1. Building splitting ciphertexts that can decrypt under k > 1 different keys

    Key Multi-collision Attacks

    [GLR CRYPTO'17] first showed an attack against AES-GCM for k = 2

2. Access to a partitioning oracle

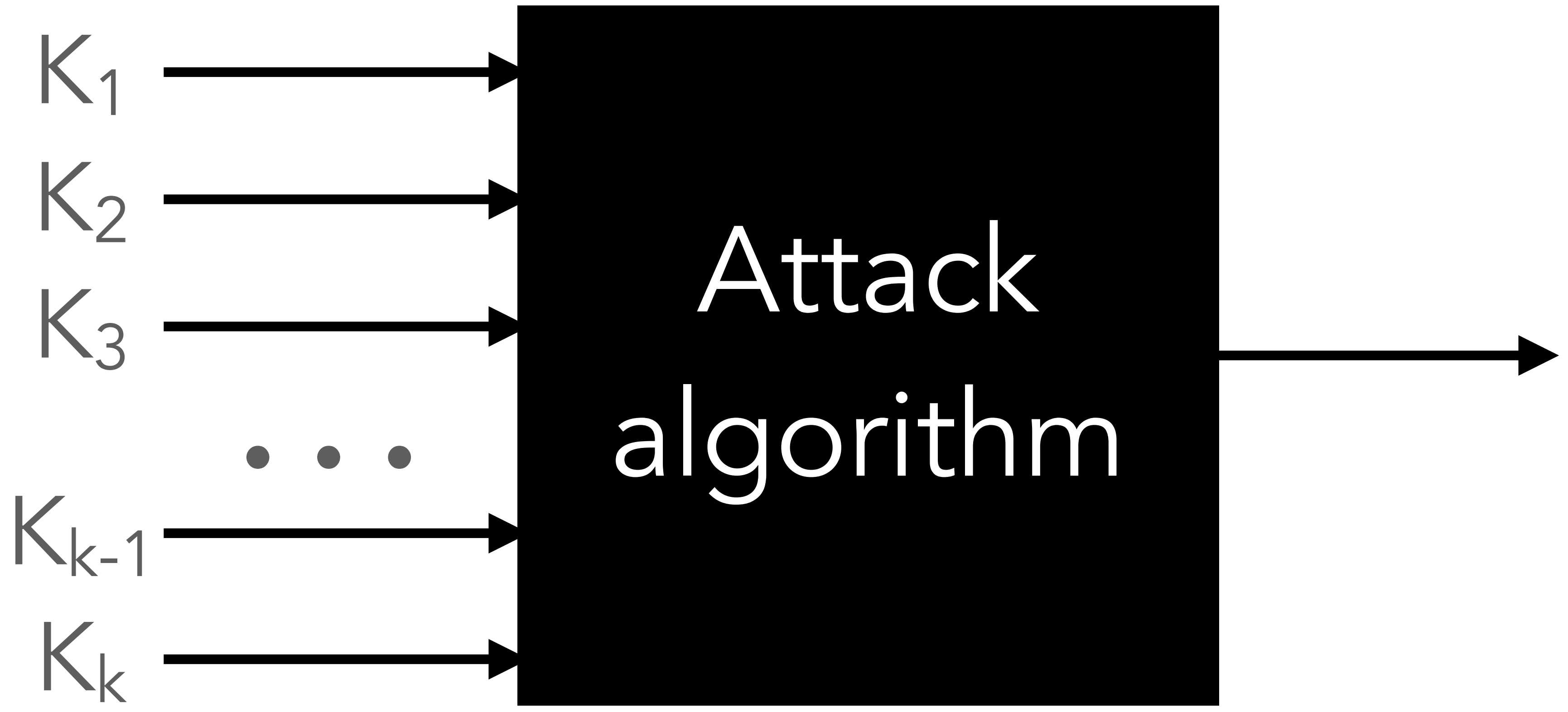
Computing Key Multi-Collisions: AES-GCM

Encrypt - then - MAC

Encrypt → Counter mode encryption of AES

MAC → GHASH: polynomial MAC

Run time: $\mathcal{O}(k^2)$



Ciphertext that decrypts under all k keys

Length: k 16-byte blocks

Computing Key Multi-Collisions: AES-GCM

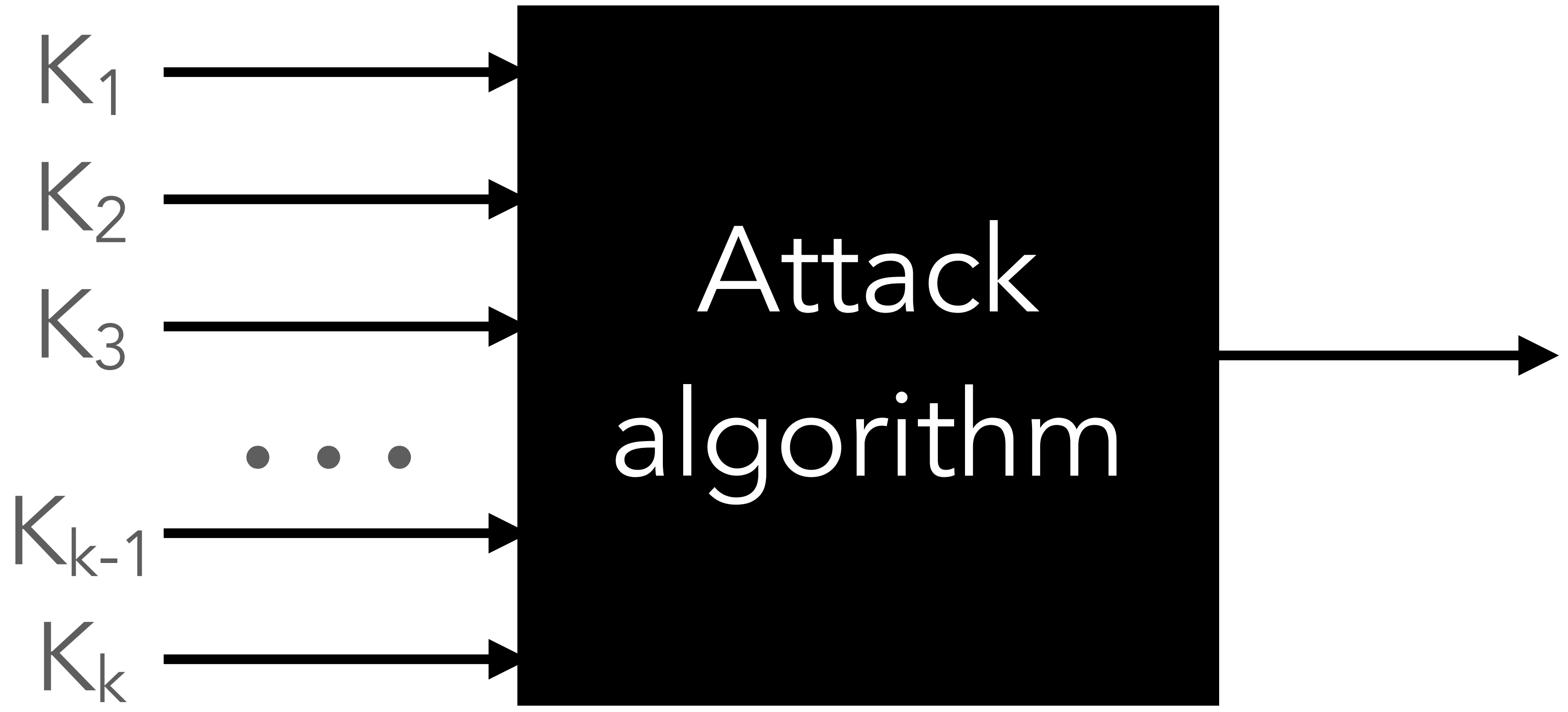
Encrypt - then - MAC

Encrypt → Counter mode encryption of AES

MAC → GHASH: polynomial MAC

Run time: $\mathcal{O}(k^2)$

Reduces finding ciphertext to solving set of linear equations



Ciphertext that decrypts under all k keys

Length: k 16-byte blocks

Computing Key Multi-Collisions: AES-GCM

Input: Let nonce N, authentication tag T, and keys K_1, K_2, K_3 be arbitrary

Goal: Compute ciphertext C that decrypts under all 3 keys

Pre-compute: $H_i = AES_{K_i}(0^{128})$, $P_i = AES_{K_i}(N \,||\, 0^{31}1)$, $L = |C|$

$$H_1^4 \cdot C_1 \oplus H_1^3 \cdot C_2 \oplus H_1^2 \cdot C_3 \oplus H_1 \cdot L \oplus P_1 = T$$

$$H_2^4 \cdot C_1 \oplus H_2^3 \cdot C_2 \oplus H_2^2 \cdot C_3 \oplus H_2 \cdot L \oplus P_2 = T$$

$$H_3^4 \cdot C_1 \oplus H_3^3 \cdot C_2 \oplus H_3^2 \cdot C_3 \oplus H_3 \cdot L \oplus P_3 = T$$

Computing Key Multi-Collisions: AES-GCM

Input: Let nonce N, authentication tag T, and keys K_1, K_2, K_3 be arbitrary

Goal: Compute ciphertext C that decrypts under all 3 keys

Pre-compute: $H_i = AES_{K_i}(0^{128})$, $P_i = AES_{K_i}(N \,||\, 0^{31}1)$, $L = |C|$

$$\begin{bmatrix} H_1^2 & H_1 & 1 \\ H_2^2 & H_2 & 1 \\ H_3^2 & H_3 & 1 \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} (T \oplus H_1 \cdot L \oplus P_1) \cdot H_1^{-2} \\ (T \oplus H_2 \cdot L \oplus P_2) \cdot H_2^{-2} \\ (T \oplus H_3 \cdot L \oplus P_3) \cdot H_3^{-2} \end{bmatrix}$$

Vandermonde matrix: we can use polynomial interpolation!

Computing Key Multi-Collisions: AES-GCM

- Implemented Multi-Collide-GCM using SageMath and Magma computational algebra system
- Timing experiments performed on desktop with Intel Core i9 processor and 128 GB RAM, running Linux x86-64

We make a ciphertext that decrypts under > 4000 keys in < 30 seconds!

k	Time (s)	Size (B)
2	0.18	48
2^{10}	6.6	16,400
2^{12}	29	65,552
2^{16}	1,820	1,048,592

Computing Key Multi-Collisions: AES-GCM

- Implemented Multi-Collide-GCM using SageMath and Magma computational algebra system
- Timing experiments performed on desktop with Intel Core i9 processor and 128 GB RAM, running Linux x86-64

We make a ciphertext that decrypts under > 4000 keys in < 30 seconds!

k	Time (s)	Size (B)
2	0.18	48
2^{10}	6.6	16,400
2^{12}	29	65,552
2^{16}	1,820	1,048,592

There exists an algorithm that does polynomial interpolation in $\mathcal{O}(k \log^2 k)$ using FFTs, so it's possible to create multi-collisions much faster [BM '74]

Computing Key Multi-Collisions

XSalsa20/Poly1305

ChaCha20/Poly1305

AES-GCM-SIV

⇒

Also vulnerable to key multi-collision attacks!

Attacks are more complex and less scalable than those for AES-GCM

Partitioning oracle attacks rely on:

1. Building splitting ciphertexts that can decrypt under k > 1 different keys

Key Multi-collision Attacks

[GLR CRYPTO'17] first showed an attack against AES-GCM for k = 2

2. Access to a partitioning oracle

Partitioning oracle attacks rely on:

1. Building splitting ciphertexts that can decrypt under k > 1 different keys

    Key Multi-collision Attacks

    [GLR CRYPTO'17] first showed an attack against AES-GCM for k = 2

2. Access to a partitioning oracle

    Where do partitioning oracles arise?

Partitioning Oracles

Schemes we looked at in depth

- Shadowsocks proxy servers for UDP
  - Popular Internet censorship evasion tool
  - Partitioning oracle attacks enable an attacker to efficiently recover a password from a Shadowsocks server

Partitioning Oracles

Schemes we looked at in depth

- Shadowsocks proxy servers for UDP
  - Popular Internet censorship evasion tool
  - Partitioning oracle attacks enable an attacker to efficiently recover a password from a Shadowsocks server
- Early implementations of the OPAQUE asymmetric PAKE protocol
  - Selected by the IETF CFRG for standardization
  - Many early implementations went against protocol specification to use a non-committing AEAD scheme
  - These schemes are vulnerable to partitioning oracle attacks

Partitioning Oracles

Schemes we looked at in depth

- Shadowsocks proxy servers for UDP
  - Popular Internet censorship evasion tool
  - Partitioning oracle attacks enable an attacker to efficiently recover a password from a Shadowsocks server
- Early implementations of the OPAQUE asymmetric PAKE protocol
  - Selected by the IETF CFRG for standardization
  - Many early implementations went against protocol specification to use a non-committing AEAD scheme
  - These schemes are vulnerable to partitioning oracle attacks

Possible partitioning oracles

- Hybrid encryption: Hybrid Public-Key Encryption (HPKE)
- Age file encryption tool
- Kerberos drafts (not adopted)
- JavaScript Object Signing and Encryption (JOSE)
- Anonymity systems: use partitioning oracles to learn which public key a recipient is using from a set of public keys

What do we do?

- Our paper is the latest in a growing body of evidence that non-committing AEAD can lead to vulnerabilities*
- So which committing AEAD scheme do we use?
  - None currently standardized!

We need a committing AEAD standard, and it should be the default choice for AEAD

* After we published our results, [ADGKLS '20] also discussed the importance of committing AEAD

Conclusion

Contact: jlen@cs.cornell.edu

Full version: https://eprint.iacr.org/2020/1491.pdf

- Described partitioning oracle attacks, which exploit non-committing AEAD to recover secrets
- Widely-used AEAD schemes, such as AES-GCM, XSalsa20/Poly1305, ChaCha20/Poly1305, and AES-GCM-SIV, are *not* committing
- Partitioning oracle attacks can be used to recover passwords from Shadowsocks proxy servers and incorrect implementations of OPAQUE
- **Recommendation**: Design and standardize committing AEAD for deployment

Thank you to my co-authors and Hugo Krawczyk, Mihir Bellare, Scott Fluhrer, David McGrew, Kenny Patterson, Chris Wood, Steven Bellovin, and Samuel Neves!

References

[ABN TCC'10] Michel Abdalla, Mihir Bellare, Gregory Neven. Robust Encryption. TCC, 2010.

[FLPQ PKC'13] Pooya Farshim, Benoît Libert, Kenneth Paterson, Elizabeth Quaglia. Robust encryption, revisited. PKC, 2013.

[FOR FSE'17] Pooya Farshim, Claudio Orlandi, Răzvan Roşie. Security of symmetric primitives under incorrect usage of keys. FSE, 2017.

[GLR CRYPTO'17] Paul Grubbs, Jiahui Lu, Thomas Ristenpart. Message franking via committing authenticated encryption. CRYPTO, 2017.

[DGRW CRYPTO'18] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. CRYPTO, 2018.

[BM '74] A. Borodin and R. Moenck. Fast modular transforms. Journal of Computer and System Sciences, 1974.

[ADGKLS '20] Ange Albertini, Thai Duong, Shay Gueron, Stefan Kölbl, Atul Luykx, Sophie Schmieg. How to abuse and fix authenticated encryption without key commitment. ePrint 2020/1456.