CMSC414 Computer and Network Security

Midterm 2 Recap

Yizheng Chen | University of Maryland
surrealyz.github.io

Apr 7, 2026

Credits: original slides from instructors and staff from CS161 at UC Berkeley, and Dave Levin

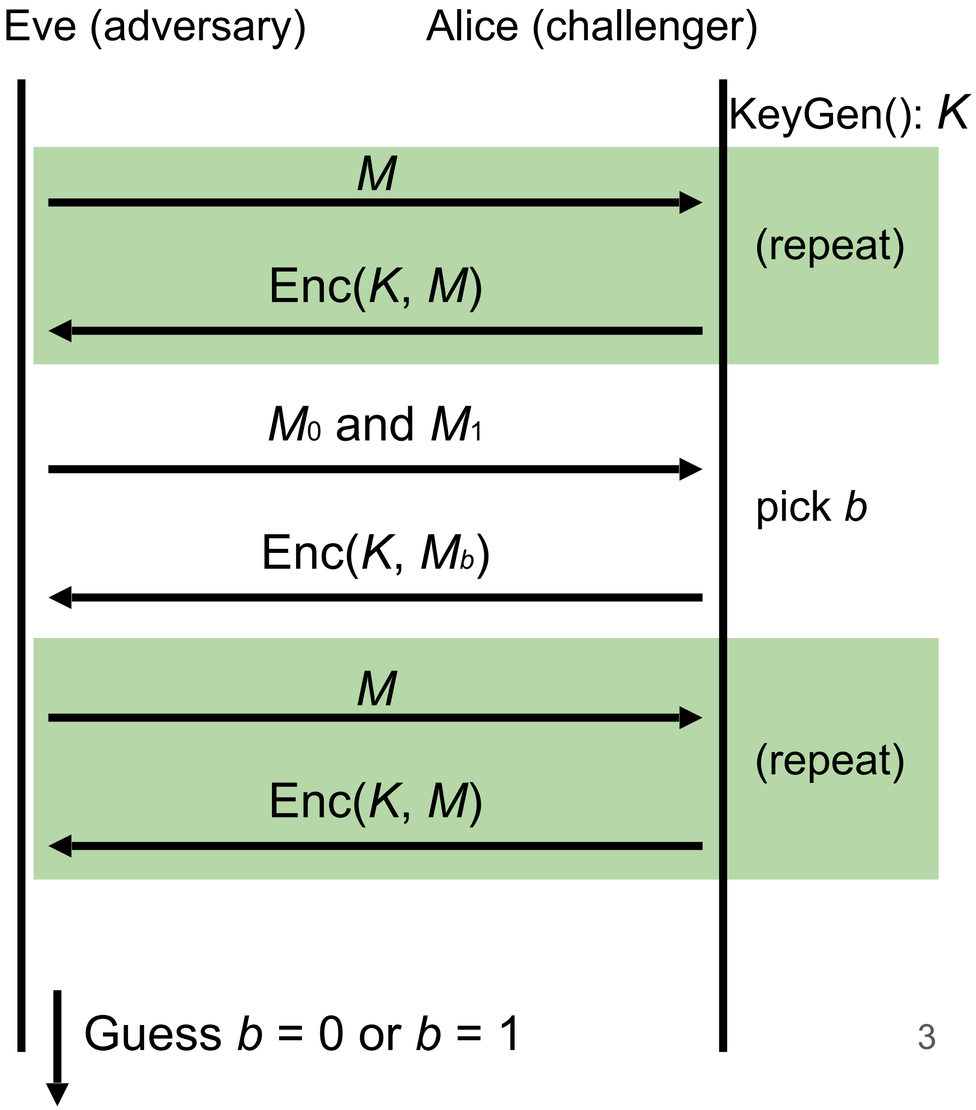
Announcement

- Sign up groups on ELMS
- Change your group name to letters, no space
- Project 4 will be released by Friday
- Midterm 2 will cover 3/3 (Intro Crypto) and 3/12 to 4/7 lectures
  - i.e., everything cryptography

IND-CPA (indistinguishability under chosen plaintext attack)

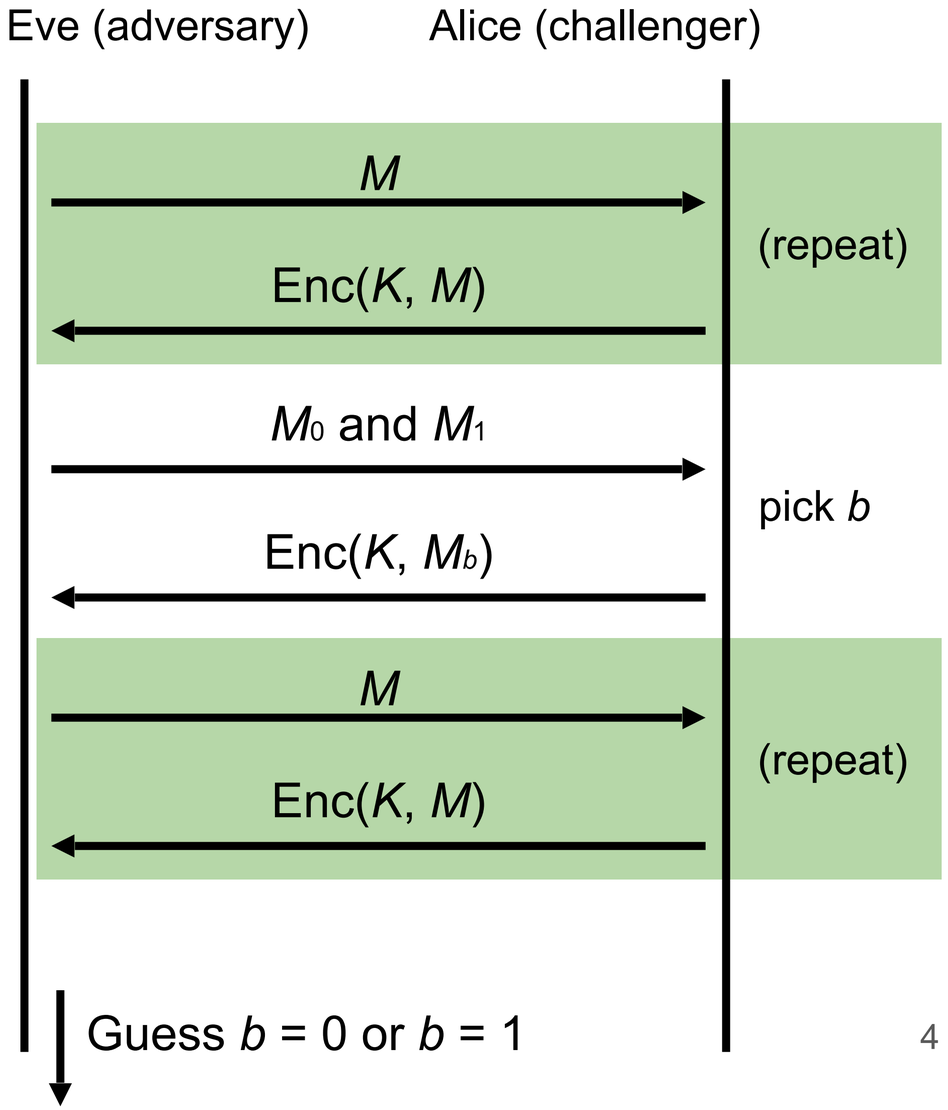
1. Eve may choose plaintexts to send to Alice and receives their ciphertexts
2. Eve issues a pair of plaintexts M_0 and M_1 to Alice
3. Alice randomly chooses either M_0 or M_1 to encrypt and sends the encryption back
   - Alice does not tell Eve which one was encrypted!
4. Eve may again choose plaintexts to send to Alice and receives their ciphertexts
5. Eventually, Eve outputs a guess as to whether encrypted M_0 or M_1

- An encryption scheme is IND-CPA secure if for all polynomial time attackers Eve:
  - Eve can win with probability ≤ 1/2 + Ɛ, where Ɛ is *negligible.*



Are Block Ciphers IND-CPA Secure?

- Consider the following adversary:
  - Eve sends two different messages M_0 and M_1
  - Eve receives either $E_K(M_0)$ or $E_K(M_1)$
  - Eve requests the encryption of M_0 again
  - Strategy: If the encryption of M_0 matches what she received, guess b = 0. Else, guess b = 1.
- Eve can win the IND-CPA game with probability 1!
  - Block ciphers are not IND-CPA secure because they are deterministic

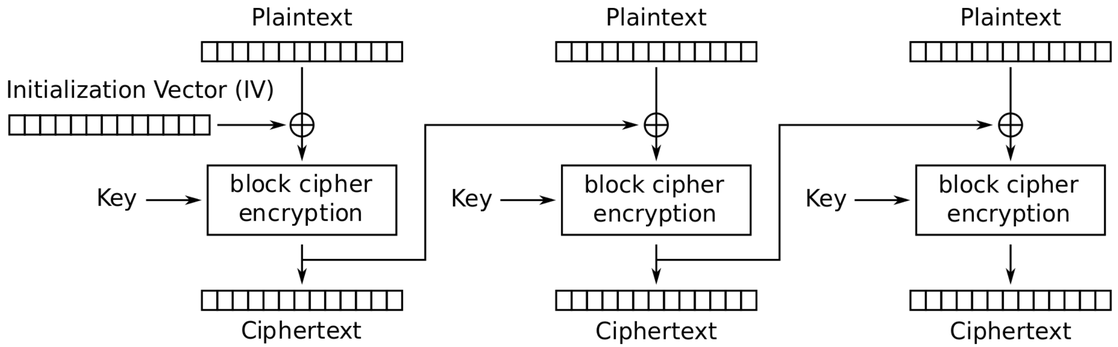


Issues with Block Ciphers

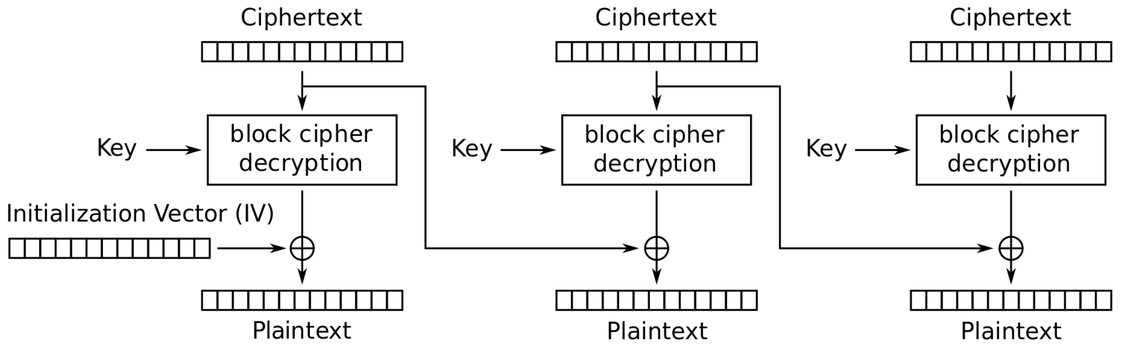
- Block ciphers are not IND-CPA secure, because they're deterministic
  - A scheme is **deterministic** if the same input always produces the same output
  - No deterministic scheme can be IND-CPA secure because the adversary can always tell if the same message was encrypted twice
- Block ciphers can only encrypt messages of a fixed size
  - For example, AES can only encrypt-decrypt 128-bit messages
  - What if we want to encrypt something longer than 128 bits?
- To address these problems, we'll add **modes of operation** that use block ciphers as a building block!

Cipher Block Chaining (CBC) Mode

- IV: Initialization Vector
- Can encryption be parallelized?
- Can decryption be parallelized?
- IND-CPA secure, under what assumption?



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

FAILED IDEA #4: STORE SALTED HASHED PASSWORDS

username : H(salt | password), salt

- Remember: *small changes to the input leads to large, unpredictable changes in the output*

Good news: Rainbow tables don't work anymore

Bad news: Can still try a dictionary attack against a given user because **hash functions are *very efficient to compute***

*Ideally we would have a very **slow** hash function*

How can we create a slow hash function out of a fast hash function?

HOW PASSWORDS ARE STORED

username : H^k(salt | password), salt

- $H^k = H(H(H(...H(x)...)))$
- Compute the hash of the hash of the hash of the...

H is a fast hash function; H^k is a slow one!

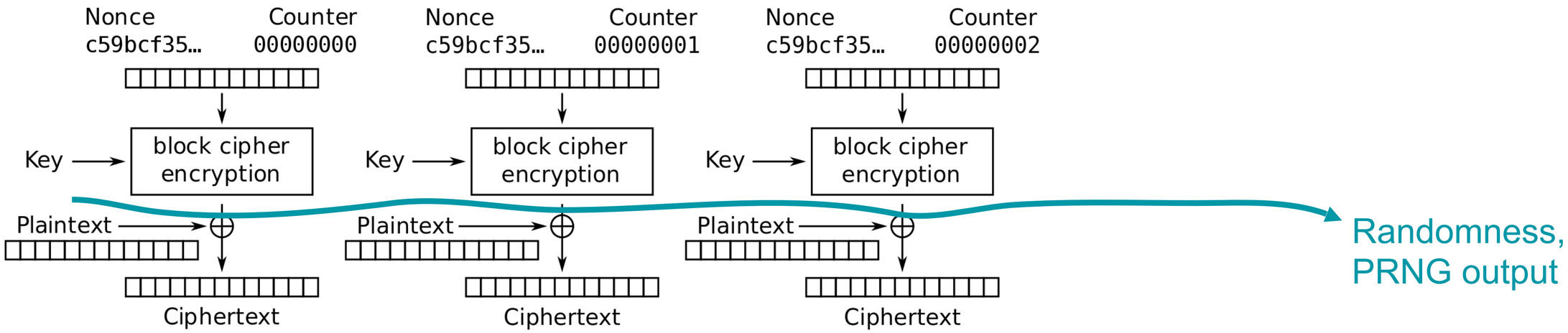
This is how passwords are stored in Linux today

Recall: Given H(password), it is infeasible to recover password,

So what does it mean if a website can email you your password?

Example construction of PRNG

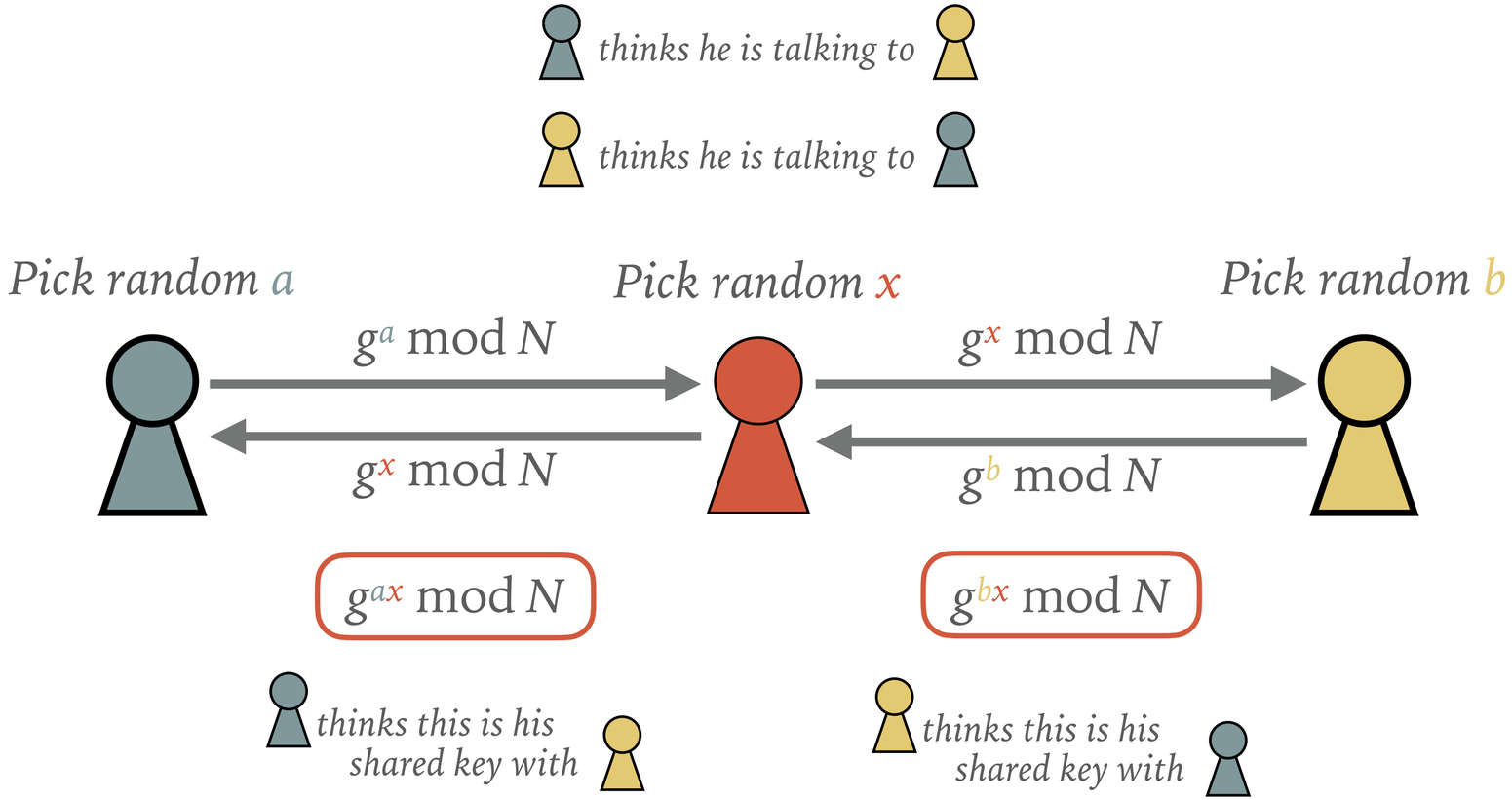
- Using block cipher in CTR mode:
- If you want m random bits, and a block cipher with E_k has n bits, apply the block cipher m/n times and concatenate the result:
- PRNG.Seed(K | IV);
- Generate(m) = E_k(IV|1) | E_k(IV| 2) | E_k(IV|3) … E_k(IV| ceil(m/n)),
  - | is concatenation



Counter (CTR) mode encryption

MAN-IN-THE-MIDDLE (MITM) ATTACKS

The attacker can interpose between the two communicating parties and insert, delete, and modify messages.



The attacker can now eavesdrop on the conversation.

Key property: Diffie-Hellman is *not* resilient to a MITM attack

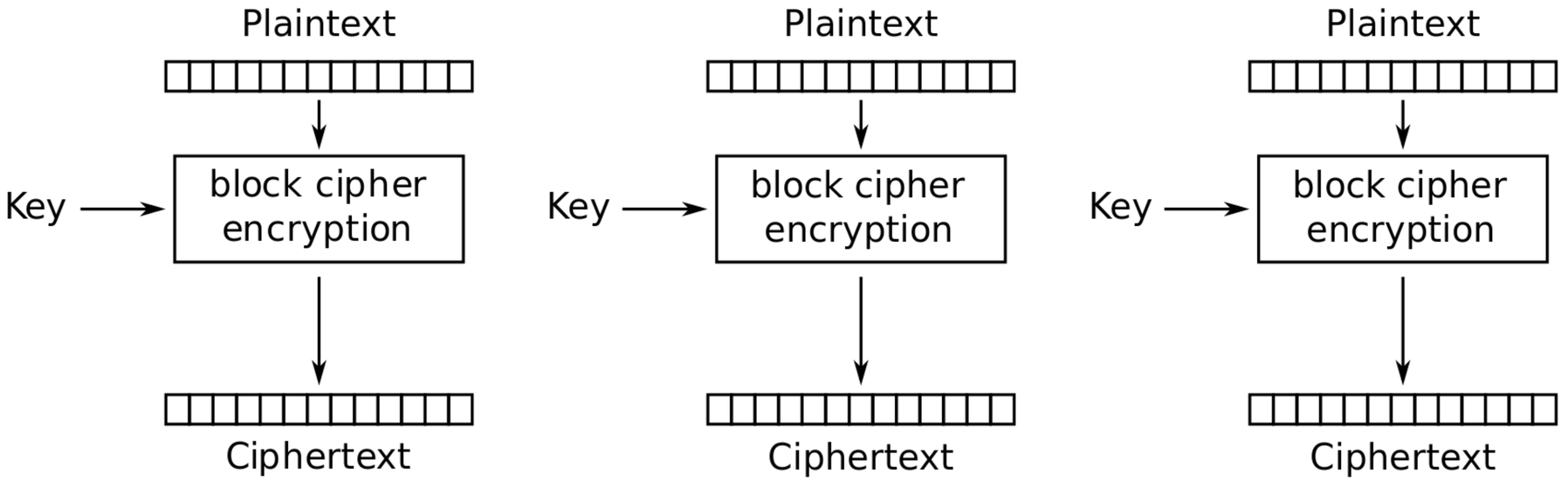
Do MACs provide integrity?

- Do MACs provide integrity?
  - Yes. An attacker cannot tamper with the message without being detected

- Do MACs provide authenticity?
  - It depends on your threat model
  - If a message has a valid MAC, you can be sure it came from *someone with the secret key*, but you can't narrow it down to one person
  - If only two people have the secret key, MACs provide authenticity: it has a valid MAC, and it's not from me, so it must be from the other person

- Do MACs provide confidentiality?
  - MACs are deterministic ⇒ No IND-CPA security
  - MACs in general have no confidentiality guarantees; they can leak information about the message

ECB Mode



Electronic Codebook (ECB) mode encryption

Stream Ciphers

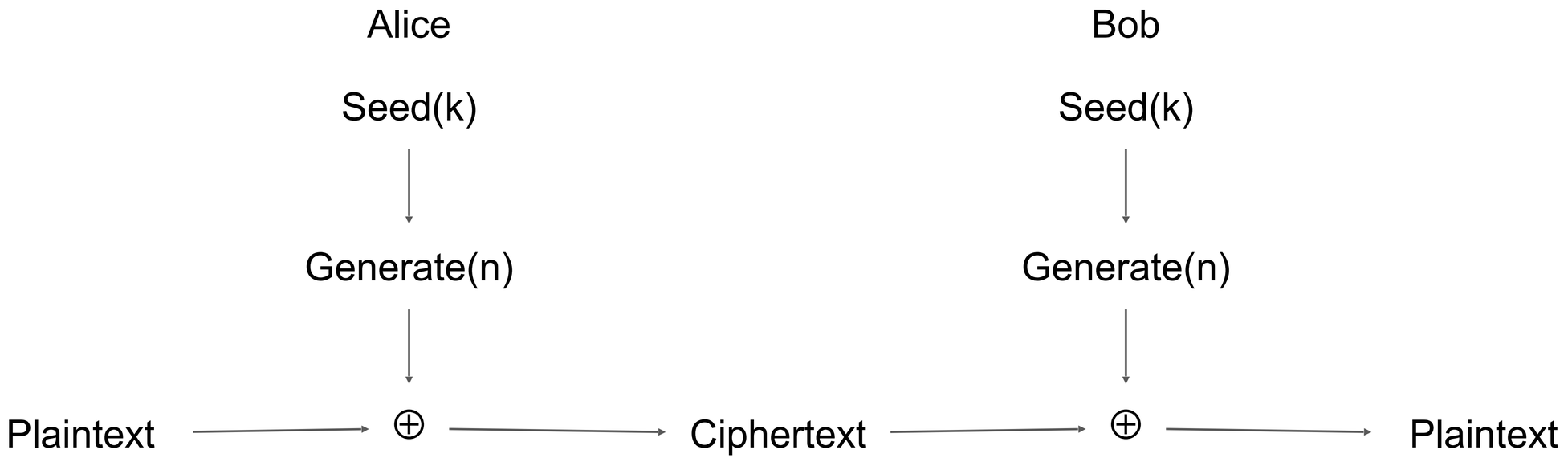
- Another way to construct symmetric key encryption schemes

- Idea
  - A secure PRNG produces output that looks indistinguishable from random
  - An attacker who can’t see the internal PRNG state can’t learn any output
  - What if we used PRNG output as the key to a one-time pad?

- **Stream cipher**: A symmetric encryption algorithm that uses pseudorandom bits as the key to a one-time pad

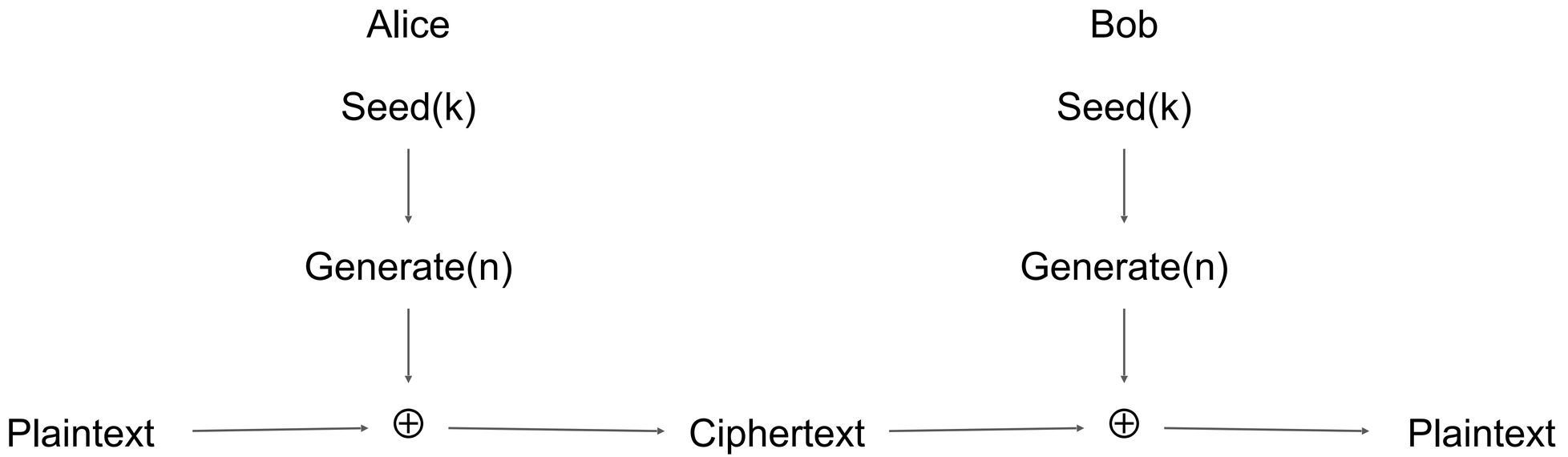
Stream Ciphers

- Protocol: Alice and Bob both seed a secure PRNG with their symmetric secret key, and then use the output as the key for a one-time pad



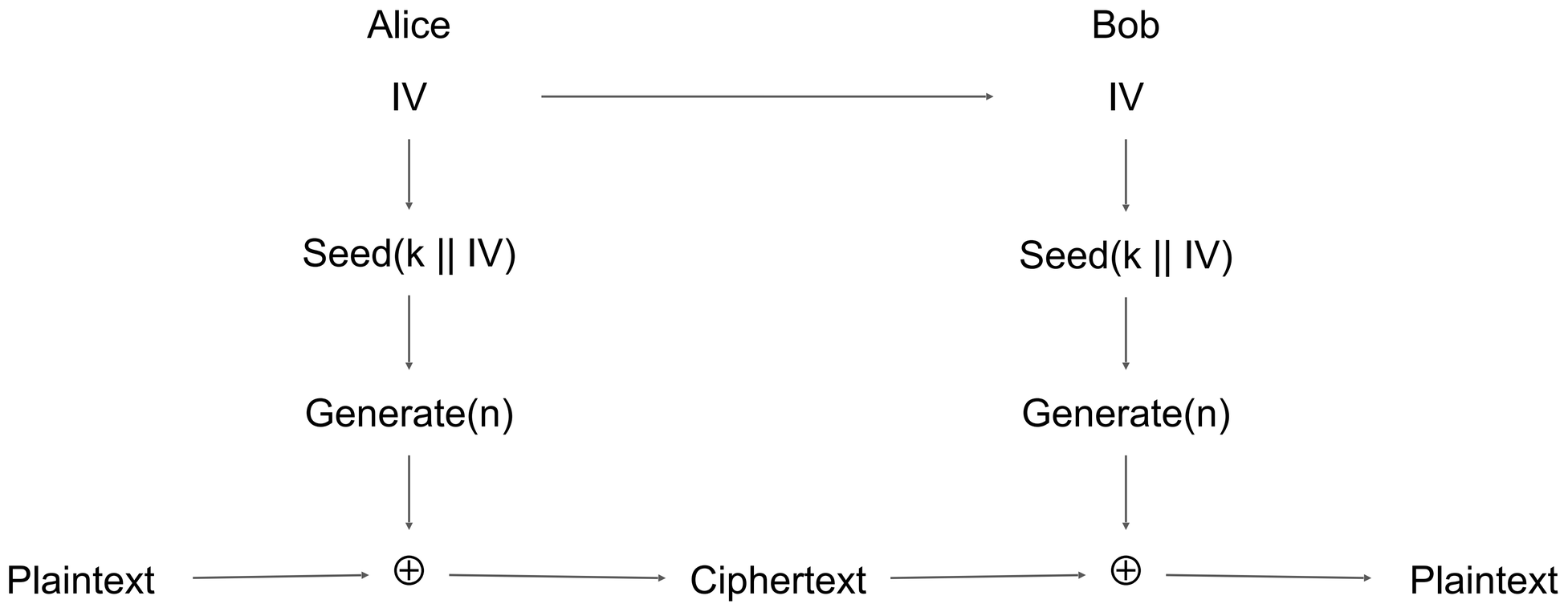
Stream Ciphers: Encrypting Multiple Messages

- Recall: One-time pads are insecure when the key is reused. How do we encrypt multiple messages without key reuse?



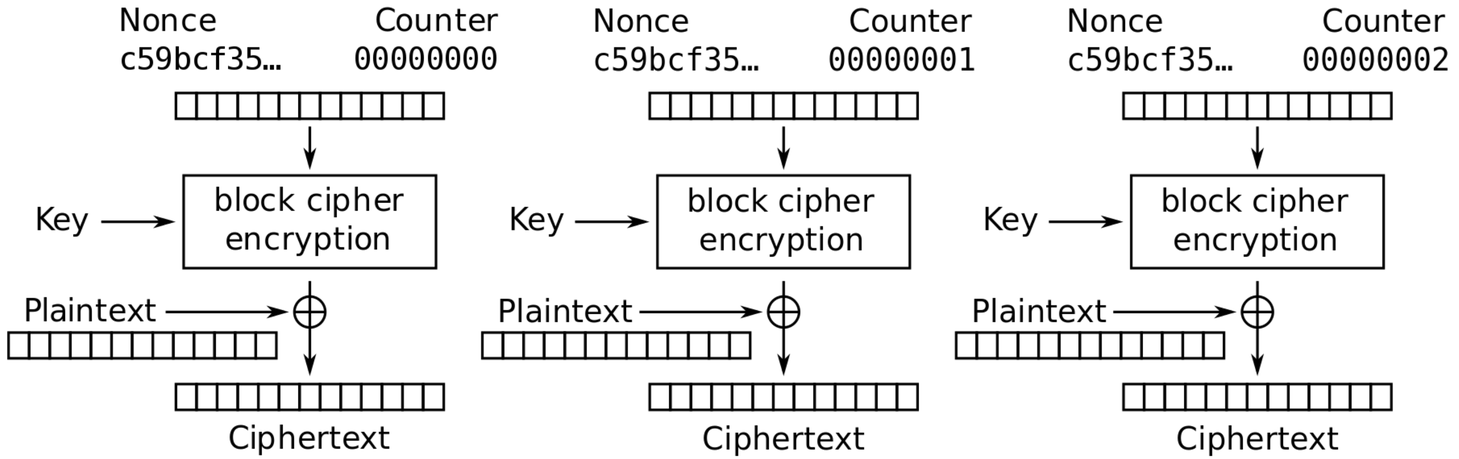
Stream Ciphers: Encrypting Multiple Messages

- Solution: For each message, seed the PRNG with the key and a random IV, concatenated. Send the IV with the ciphertext



Stream Ciphers: AES-CTR

- If you squint carefully, AES-CTR is a type of stream cipher
- Output of the block ciphers is pseudorandom and used as a one-time pad



Counter (CTR) mode encryption

Pseudorandom Number Generators (PRNGs)

- True randomness is expensive and biased
- **Pseudorandom number generator** (**PRNGs**): An algorithm that uses a little bit of true randomness to generate a lot of random-looking output
  - Also called **deterministic random bit generators** (**DRBGs**)
- Usage
  - Generate some expensive true randomness (e.g. noisy circuit on your CPU)
  - Use the true randomness as input to the PRNG
  - Generate random-looking numbers quickly and cheaply with the PRNG
- PRNGs are deterministic: Output is generated according to a set algorithm
  - However, for an attacker who can't see the internal state, the output is *computationally indistinguishable* from true randomness

PRNG: Definition

- A PRNG has two functions:
  - PRNG.Seed(randomness): Initializes the internal state using the entropy
    - Input: Some truly random bits
  - PRNG.Generate(*m*): Generate *m* pseudorandom bits
    - Input: A number *m*
    - Output: *m* pseudorandom bits
    - Updates the internal state as needed

Properties

- **Correctness**: Deterministic
- **Efficiency**: Efficient to generate pseudorandom bits
- **Security**: Indistinguishability from random

PRNG: Security

- Can we design a PRNG that is truly random?
- A PRNG cannot be truly random
  - The output is deterministic given the initial seed
  - If the initial seed is *s* bits long, there are only 2^s possible output sequences
- A secure PRNG is computationally indistinguishable from random to an attacker
  - Game: Present an attacker with a truly random sequence and a sequence outputted from a secure PRNG
  - An attacker should not be able to determine which is which with probability > ½+negl
- Equivalence: An attacker cannot predict future output of the PRNG

Example: NMAC

- Can we use secure cryptographic hashes to build a secure MAC?
  - Intuition: Hash output is unpredictable and looks random, so let's hash the key and the message together
- KeyGen():
  - Output two random, n-bit keys K_1 and K_2, where n is the length of the hash output
- NMAC(K_1, K_2, M):
  - Output $H(K_1 \,||\, H(K_2 \,||\, M))$
- NMAC is EU-CPA secure if the two keys are different
  - Provably secure if the underlying hash function is secure
- Intuition: Using two hashes prevents a length extension attack
  - Otherwise, an attacker who sees a tag for M could generate a tag for $M \,||\, M'$

Example: HMAC

- Issues with NMAC:
  - Recall: NMAC(K_1, K_2, M) = $H(K_1 \,||\, H\,(K_2 \,||\, M))$
  - We need two different keys
  - NMAC requires the keys to be the same length as the hash output (n bits)

- HMAC(K, M):
  - Compute K' as a version of K that is the length of the hash output
    - If K is too short, pad K with 0's to make it n bits (be careful with keys that are too short and lack randomness)
    - If K is too long, hash it so it's n bits
  - Output $H((K' \oplus opad) \,||\, H((K' \oplus ipad) \,||\, M))$

Example: HMAC

- HMAC(*K*, *M*):
  - Compute *K'* as a version of *K* that is the length of the hash output
    - If *K* is too short, pad *K* with 0's to make it *n* bits (be careful with keys that are too short and lack randomness)
    - If *K* is too long, hash it so it's *n* bits
  - Output $H((K' \oplus opad) \,||\, H((K' \oplus ipad) \,||\, M))$

- Use *K'* to derive two different keys
  - *opad* (outer pad) is the hard-coded byte `0x5c` repeated until it's the same length as *K'*
  - *ipad* (inner pad) is the hard-coded byte `0x36` repeated until it's the same length as *K'*
  - As long as *opad* and *ipad* are different, you'll get two different keys
  - For paranoia, the designers chose two very different bit patterns, even though they theoretically need to only differ in one bit

HMAC Properties

- HMAC(K, M) = $H((K' \oplus opad) \,||\, H((K' \oplus ipad) \,||\, M))$

- HMAC is a hash function, so it has the properties of the underlying hash too
  - It is collision resistant
  - Given HMAC(K, M), an attacker can't learn M
  - If the underlying hash is secure, HMAC doesn't reveal M, but it is still deterministic

- You can't verify a tag T if you don't have K
  - The attacker can't brute-force the message M without knowing K

Do MACs provide integrity?

- Do MACs provide integrity?
  - Yes. An attacker cannot tamper with the message without being detected
- Do MACs provide authenticity?
  - It depends on your threat model
  - If a message has a valid MAC, you can be sure it came from *someone with the secret key*, but you can't narrow it down to one person
  - If only two people have the secret key, MACs provide authenticity: it has a valid MAC, and it's not from me, so it must be from the other person
- Do MACs provide confidentiality?
  - MACs are deterministic ⇒ No IND-CPA security
  - MACs in general have no confidentiality guarantees; they can leak information about the message

MACs: Summary

- Inputs: a secret key and a message
- Output: a tag on the message
- A secure MAC is unforgeable: Even if Mallory can trick Alice into creating MACs for messages that Mallory chooses, Mallory cannot create a valid MAC on a message that she hasn't seen before
  - Example: HMAC(K, M) = $H((K' \oplus opad) \,||\, H((K' \oplus ipad) \,||\, M))$
- MACs do not provide confidentiality

Three Main Goals of Cryptography

- In cryptography, there are three common properties that we want on our data
- **Confidentiality**: An adversary cannot *read* our messages.
- **Integrity**: An adversary cannot *change* our messages without being detected.
- **Authenticity**: I can prove that this message came from the person who claims to have written it.