ISE Cryptography – Lecture 03

Block Ciphers

Housekeeping

- The continuous assessment results from Week 1 and Week 2 are excellent so far
  - Keep up the good work!
- Some very “interesting” keys used in Challenge 5...
  - Everything from Taylor Swift to Moby Dick!
- Rubric allows you to skip Challenge 6 and still get an A1
  - What’s the key to solving it without too much trouble?
- Grades will be released on Brightspace later
- Quiz on stream ciphers in class today
- Open question on the other two labs for Week 5 and Week 7
  - Would it make sense to release both in advance?
  - Don’t want to clash with study periods or overlap with the EPIC!
  - Separately? At the same time?
  - Not finalised yet, but might link into some of the tutorial workbooks...

Block Ciphers

The basics.

A Cryptographic Workhorse

- A block cipher encrypts a fixed-size block of data using a secret key
  - Deterministic: the same key and input block always produce the same output block
- Block ciphers take a different approach to encryption than stream ciphers!
  - Dealing with fixed-size blocks instead of streams
- Block ciphers are very powerful and versatile! You can use them to build...
  - Various block cipher modes of operation
  - CSPRNGs, and therefore...
  - Stream ciphers
  - Hash functions
  - Ciphers with stronger security properties than just semantic security
- AES alone encrypts over 70% of SSL/TLS traffic!

Defining Block Ciphers

- A block cipher is a deterministic cipher $\mathcal{E} = (E, D)$
  - Whose message space and ciphertext space are the same set $\mathcal{X}$
  - $\mathcal{X}$ is the **data block space** of $\mathcal{E}$
  - As per usual, it's a finite set of fixed-length bitstrings
- This means we can take a shortcut with our notation!
  - Instead of $\mathcal{E}$ being defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$...
  - A block cipher $\mathcal{E}$ is defined over $(\mathcal{K}, \mathcal{X})$

Permutations

- It's helpful to think of "binding" a fixed key $k \in \mathcal{K}$ to the encryption function
  - We can define $f_k = E(k, \cdot)$, where $f_k : \mathcal{X} \to \mathcal{X}$
- By the ***correctness property***, f_k must have an inverse function f_k^{-1}
  - So f_k must be injective (one-to-one)
  - $\mathcal{X}$ is finite, so f_k must also be surjective (onto)
  - This means that f_k is a ***permutation*** on $\mathcal{X}$
- Encryption and decryption with a fixed key is just mapping between blocks!
  - Think of this like a more complex substitution cipher
  - Each plaintext block maps to a corresponding ciphertext block
  - Just with (typically) 64-bit or 128-bit blocks instead of 8-bit characters
- Can you think of any potential pitfalls here?
  - How might you attack a substitution cipher?

Block Cipher Security

- Remember semantic security and the attack games we played last time?
  - We're going to ask for a much stronger security property with block ciphers
- For a randomly chosen key $k \in \mathcal{K}$...
  - The permutation $f_k = E(k, \cdot)$ should appear random!
  - It should be ***computationally indistinguishable*** from a random permutation
- Let's add another bit of notation to make this simpler!
  - How can we talk about all possible permutations on the data block space?
  - Luckily, this is the same idea as $\mathrm{Sym}(\mathcal{X})$, the ***symmetric group*** on $\mathcal{X}$
- $\mathrm{Sym}(\mathcal{X})$ is the ***set of all permutations*** on $\mathcal{X}$
  - The set of all possible bijections from $\mathcal{X}$ to itself
  - Also notated as $S(\mathcal{X})$ or $S_{\mathcal{X}}$
  - One of the textbooks uses $\mathrm{Perms}[\mathcal{X}]$ as custom notation

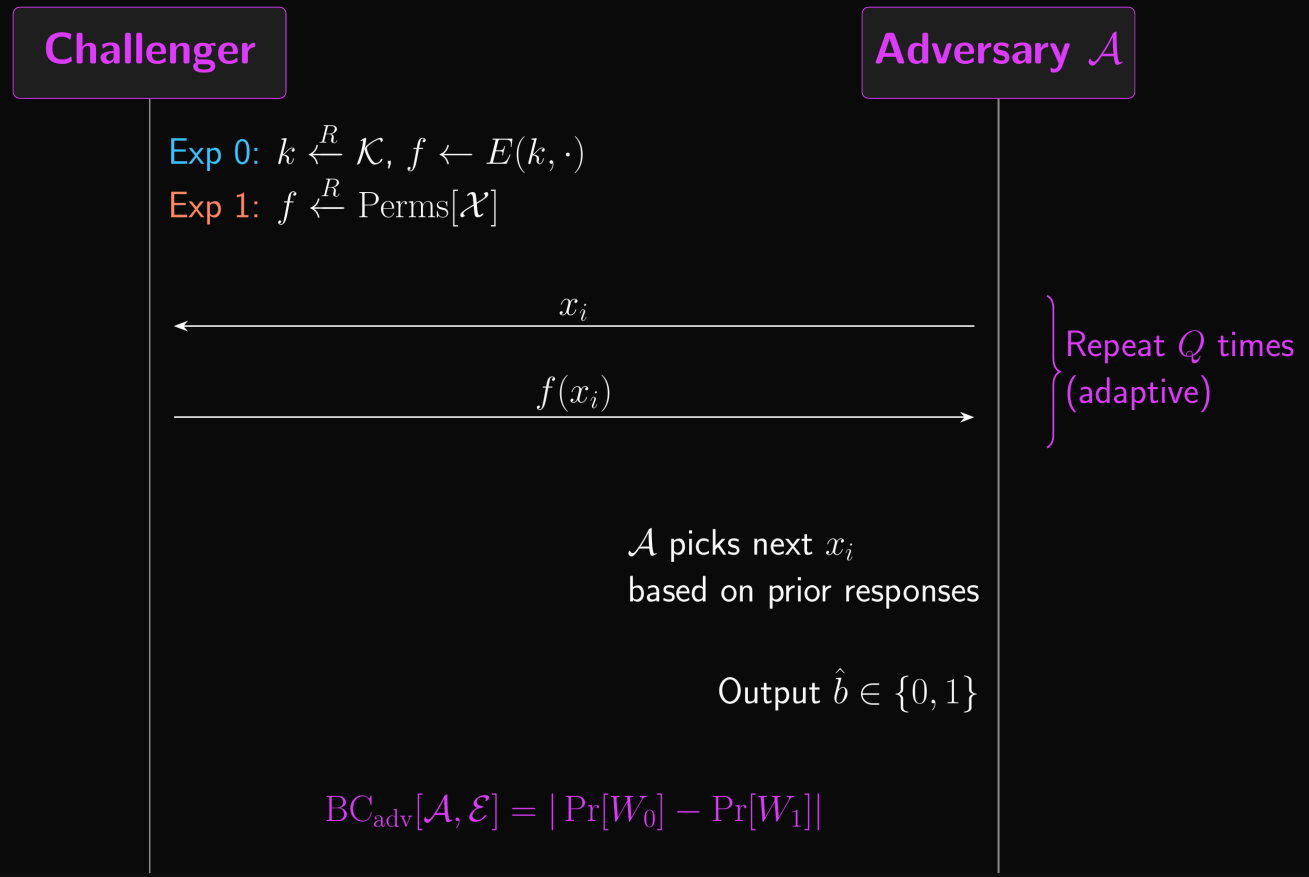
Attack Games

- We're going to allow the adversary $\mathcal{A}$ in the attack game to be much more powerful
  - They can ask as many "questions" as they want to
  - They can even decide what questions to ask based on previous answers
  - Ultimately, the adversary has to guess which experiment they're playing!
- Experiment 0:
  - The challenger selects $k \xleftarrow{R} \mathcal{K}$, $f \leftarrow E(k, \cdot)$
  - ...the block cipher encryption function with a randomly-selected fixed key
- Experiment 1:
  - The challenger selects $f \xleftarrow{R} \mathrm{Perms}[\mathcal{X}]$
  - ...a randomly-selected permutation from the symmetric group on $\mathcal{X}$
- $\mathcal{A}$ sends a sequence of queries $x_1, x_2, \ldots, x_Q$ one by one, based on the responses
  - The challenger responds with $f(x_1), f(x_2), \ldots, f(x_Q)$
- After sending Q queries, $\mathcal{A}$ guesses if it's in experiment 0 or 1

Attack Games

- Similarly to semantic security and PRG security, we can define the ***advantage***
  - $\mathrm{BC}_{\mathrm{adv}}[\mathcal{A}, \mathcal{E}] = |\Pr[W_0] - \Pr[W_1]|$
- $\mathcal{A}$ is a ***Q-query BC adversary*** if it issues at most Q queries
  - The more queries, the stronger the adversary!
  - Some block ciphers take large values of Q to break
- Note that $\mathcal{A}$ is ***adaptive***
  - It doesn't have to send all of its queries up front
  - It can wait for each response and then decide what to query next
  - Remember, ***the enemy knows the system*** - it can exploit any weaknesses
  - It can do everything a non-adaptive adversary can, plus more!
- A block cipher $\mathcal{E}$ is secure if for all efficient adversaries $\mathcal{A}$, $\mathrm{BC}_{\mathrm{adv}}[\mathcal{A}, \mathcal{E}]$ is negligible
  - A block cipher achieving this is called a ***Pseudo-Random Permutation (PRP)***
  - This implies that $\mathrm{SS}_{\mathrm{adv}}[\mathcal{A}, \mathcal{E}]$ is also negligible
  - This is a ***stronger*** notion of security! We'll see why shortly.

Block Cipher Security: Attack Game



Implications

- We don't want our block cipher to be brute-forced by an efficient adversary
  - So the size of the data block space should be super-poly
- A secure block cipher should be unpredictable
  - Knowing a subset of block-to-block mappings shouldn't let you predict others...
  - ...with anything better than random chance
  - Even if you know all but two mappings, it should be a 50-50 guess!
- This implies security against key recovery too!
  - If an adversary could guess the key with non-negligible probability...
  - ...then the block cipher would be predictable and wouldn't be secure
- And that implies that the key space must be large enough!
  - $1/|\mathcal{K}|$ probability of guessing the key correctly
  - $1/|\mathcal{K}|$ must be negligible, so $|\mathcal{K}|$ must be super-poly!

Exhaustive Search Attacks

- To illustrate why that's so important, let's see how it could be exploited.
- If $|\mathcal{K}|$ is poly-bounded, we lose to an ***exhaustive search attack***
- The adversary starts by running Q queries as usual
  - This gives them a partial permutation of blocks that map to other blocks
- Next, the adversary loops over all possible keys in $\mathcal{K}$
  - Once they find a key that produces the partial permutation, they can stop
  - Even for small values of Q, it turns out that key is almost always correct!
  - Time to find the correct key is $\Theta(|\mathcal{K}|)$, i.e. linear in $|\mathcal{K}|$
- To prevent this kind of attack, we need to make it ***prohibitively expensive***
  - We do this by making $|\mathcal{K}|$ super-poly
  - This gives a negligible probability of finding the correct key in polynomial time
- Two preconditions for block cipher security should be pretty clear by now!
  - It needs both a large key space and a large data block space

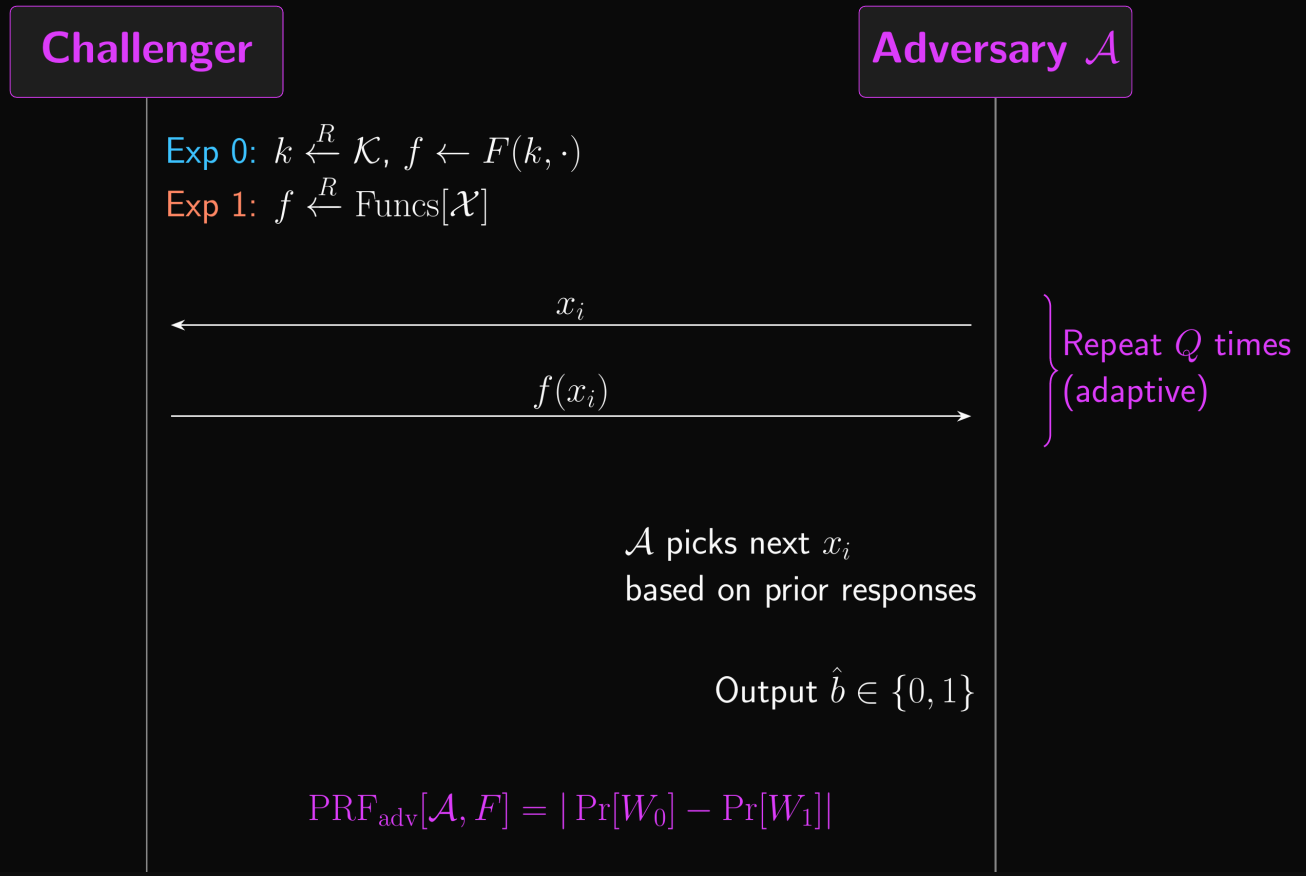
Secure?

- We can also define strongly secure block ciphers with an adjustment to the game!
- We allow forward queries already
  - $\mathcal{A}$ sends a block to the challenger, who responds with its encryption
- We can also allow inverse queries
  - $\mathcal{A}$ sends a block to the challenger, who responds with its decryption
- A strongly secure block cipher should only allow a negligible advantage here
- So, is a secure block cipher also semantically secure?
  - Yes! BC security is ***strictly stronger*** than SS for single-block messages
  - The practical limitation is that block ciphers encrypt exactly one block
    - AES uses 128-bit blocks: works for a key, not for a file
- What if we need to send longer messages?
  - How can we do it?
  - Can we maintain this level of security?

Pseudo-Random Functions (PRFs)

- We briefly mentioned pseudo-random functions (PRFs) last week!
  - A deterministic function $F(k, x)$ that takes a secret key k and input x
  - Returning a value y
  - Defined over finite sets $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$
- Intuitively, a PRF "looks like" a truly random function to any efficient adversary
  - Provided that the key remains secret, of course!
- What makes a ***secure PRF***? An adversary queries either:
  - A real PRF $F(k, \cdot)$ (Experiment 0)
  - A truly random function from $\mathcal{X} \to \mathcal{Y}$ (Experiment 1)
    - We write $\mathbf{Funcs}[\mathcal{X}]$ for the set of all such functions; unlike $\mathbf{Perms}[\mathcal{X}]$, functions can repeat output values
- If $\mathbf{PRF}_{\mathrm{adv}}[\mathcal{A}, F]$ is negligible for all efficient adversaries, the PRF is secure
  - Adversaries are adaptive and can make as many queries as they want to
- Weakly-secure PRFs are similar...
  - ...but the adversary can only query random inputs

PRF Security: Attack Game



Building PRFs from Block Ciphers

- Can a secure block cipher be a PRF?
  - Yes, if the input space is large enough ($|\mathcal{X}|$ is super-poly)
  - Small domains are vulnerable to birthday attacks
- 64-bit domain $\rightarrow$ 2^{64} possible inputs
  - Due to the birthday paradox, only ~2^{32} queries are needed to find a collision with high probability.
- In the PRF attack game:
  - Real block ciphers are permutations – so they cannot produce collisions
  - Random functions can have collisions
  - An adversary can exploit this difference to easily distinguish a block cipher from a random function!
- Block ciphers like DES (with 64-bit blocks) are not secure PRFs!
- The formal statement is the **PRF/PRP switching lemma** (B&S Theorem 4.4); we'll walk through the argument after the birthday paradox

The Cake is a Lie

- Anyone heard of the ***birthday paradox***?
  - There's a 50% chance of a shared birthday among just 23 people
  - It's a ***veridical paradox***: true, but counterintuitive
- In a domain of size N, the probability of at least one collision among Q samples is:
  - $\Pr[\text{collision}] \approx Q^2 / 2N$
  - $N = 365$ possible birthdays, Q = number of people
- Collisions become 50% likely when $Q \approx \sqrt{N}$
- If you're running the numbers, remember that these are approximations!
  - There's an exact formula, but this is good enough for our purposes
- Why does this matter for block ciphers?
  - A 64-bit block cipher doesn't have a large enough domain to make a secure PRF
  - Remember that ChaCha20 uses a 512-bit state!
- Birthday bounds (and attacks) will pop up in more detail when we discuss secure hash functions in an upcoming lecture!

PRF/PRP Switching Lemma

- We want to show $E(k, \cdot)$ is a secure PRF, given that it is a secure PRP
- Three steps:
  1. By BC security, $E(k, \cdot)$ is indistinguishable from $f \xleftarrow{R} \mathrm{Perms}[\mathcal{X}]$
  2. A random permutation differs from a truly random function only by never repeating an output
     - After Q queries, the probability a random function ever repeats an output is $\leq Q^2 / (2|\mathcal{X}|)$
     - This is the birthday bound: the gap between $\mathrm{Perms}[\mathcal{X}]$ and $\mathrm{Funcs}[\mathcal{X}]$
  3. Triangle inequality: $\mathrm{PRF}_{\mathrm{adv}}[\mathcal{A}, E] \leq \mathrm{BC}_{\mathrm{adv}}[\mathcal{B}, E] + Q^2 / (2|\mathcal{X}|)$
- An adversary's best chance of breaking the PRF is bounded by
  - Their chance of breaking the block cipher, plus...
  - Their chance of spotting a repeated output!
- Both terms are negligible when $|\mathcal{X}|$ is large:
  - AES ($|\mathcal{X}| = 2^{128}$): $Q^2 / (2^{129})$ is negligible for any polynomial Q
  - DES ($|\mathcal{X}| = 2^{64}$): $Q \approx 2^{32}$ queries suffice – **not negligible**
- This bridge is why CTR and CBC mode security can reduce directly to block cipher security

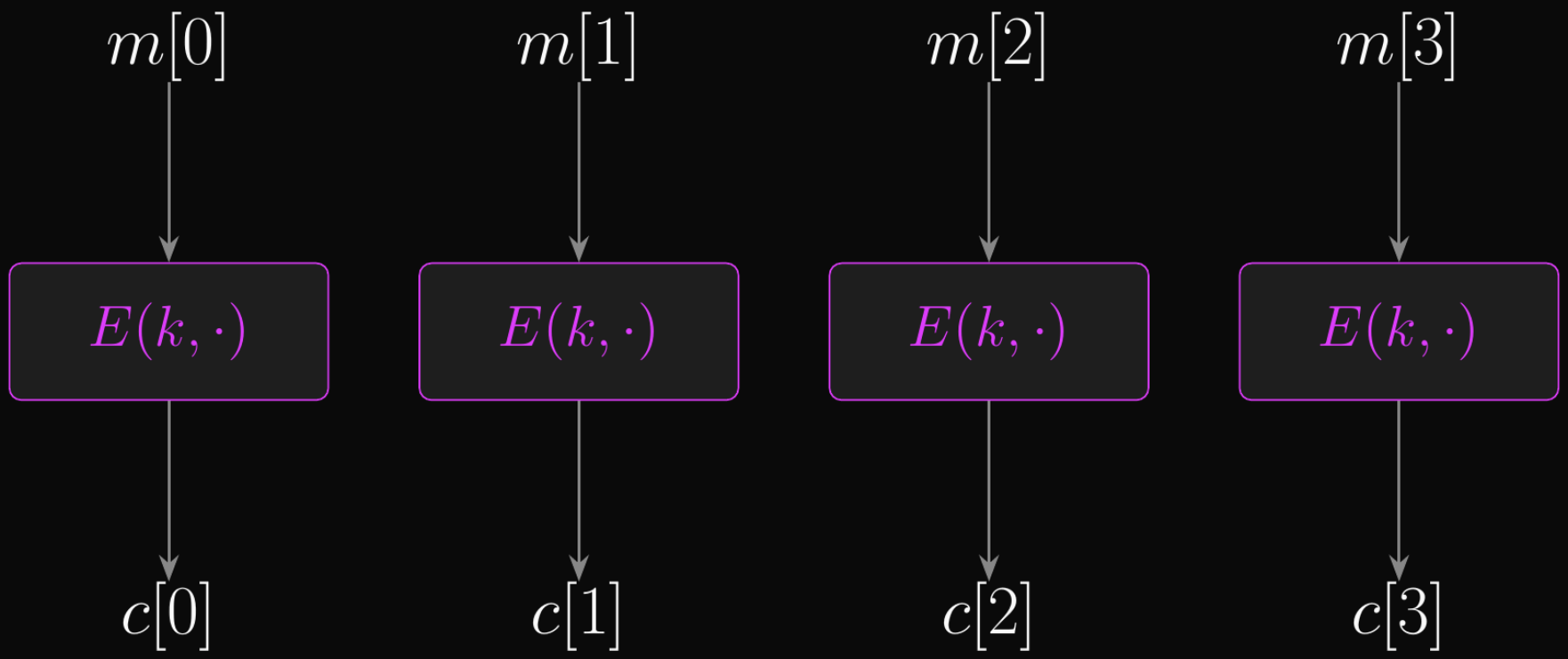
ECB Mode

What if we just encrypt all the blocks?

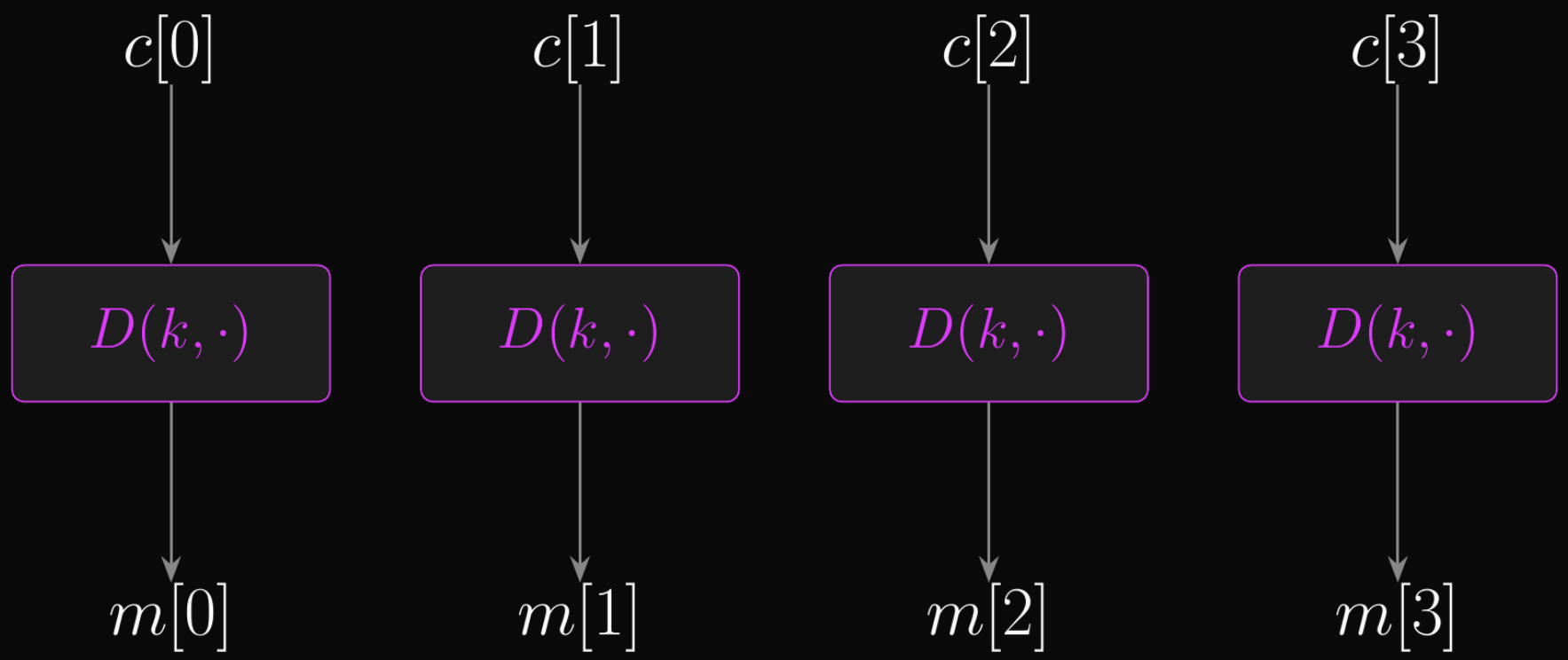
Electronic Codebook (ECB) Mode

- Our goal: encrypt multi-block messages with IND-CPA security, reusing the same key
  - A bare block cipher encrypts one block; a mode of operation handles longer messages
- We wrap the block cipher in a ***mode of operation*** to do this
- To encrypt arbitrary-length messages, we use block ciphers as the core of a larger construct called a block cipher mode of operation
- The most straightforward option is ***electronic codebook mode***
  - Usually abbreviated to ***ECB mode***
- Start by slicing the message into a sequence of data blocks
  - What if the message length isn't an integer multiple of the block size?
- Padding is an important (and surprisingly thorny) issue for block ciphers
  - For now, let's just pad with null bytes (`0x00`) until the message length is a multiple of the block size
- Encrypt each block and concatenate the results to make the ciphertext
- Decrypting is just as straightforward
  - Slice the ciphertext into blocks
  - Pass each block to the decryption function

Electronic Codebook (ECB) Mode: Encryption



Electronic Codebook (ECB) Mode: Decryption



Electronic Codebook (ECB) Mode

- This is a surprisingly flexible construct!
  - Encryption is trivial to parallelise
  - Decryption is trivial to parallelise
  - Random access is feasible
    - No need to do any extra work to read a specific block
- Not malleable in the way the one-time pad is
  - But integrity is still an unsolved problem for us
- Apart from that, why even bother talking about other modes of operation?
  - ECB mode has a tiny problem...
- Remember how we said a block cipher is kind of like a substitution cipher?

The Penguin of Doom

- ECB mode applied to a bitmap image of Tux the Linux penguin
- The penguin's outline is perfectly visible in the "encrypted" output
  - Each block encrypts independently – identical input blocks produce identical output blocks
  - The block boundaries follow the pixel grid, preserving all large-scale structure
- See the classic ECB penguin example (Wikipedia)

What Could Possibly Go Wrong?

- What went wrong there? Is it a problem with the block cipher?
  - Is it a problem with the mode of operation?
- Looking at the ciphertext, can you recover the plaintext of the blocks?
  - But it's clearly leaking information about the plaintext...
- ECB mode leaks information about the structure of the plaintext
  - Same input block will always produce the same output block!
- Not so devastating if we're using ASCII or UTF-8 text, as repetition isn't so common
  - But it absolutely breaks semantic security!
  - ECB mode isn't semantically secure, let alone IND-CPA secure
  - It should never be used in practice
- ECB mode fails to provide ***diffusion*** – patterns in plaintext survive to ciphertext

Diffusion and Confusion

- ***Diffusion*** refers to a cipher's ability to obscure the relationship between the ciphertext and plaintext
  - Patterns in the plaintext should not be apparent in the ciphertext
  - The penguin's ciphertext should look random (necessary but not sufficient)
  - Usually mentioned alongside another, related term...
- ***Confusion*** obscures the relationship between the ciphertext and the key
  - The ciphertext should not leak any information about the key
  - A single-bit change in the key should change most of the bits in the ciphertext
- Both of these ideas date back to Claude Shannon
  - As does the general idea of block ciphers via product ciphers

IND-CPA Security

From semantic security to chosen-plaintext attacks.

From Semantic Security to IND-CPA

- Recall: semantic security guarantees that a single ciphertext leaks nothing
  - This is good enough if we only encrypt one message per key
- But we reuse keys in practice! What if an adversary sees multiple ciphertexts?
- **IND-CPA** (indistinguishability under chosen-plaintext attack):
  - The adversary has access to an **encryption oracle** $\mathrm{Enc}(k, \cdot)$
    - Query it with any single message m; receive $c \leftarrow E(k, m)$
    - Can query **before and after** receiving the challenge ciphertext
  - The adversary submits a **challenge pair** (m_0, m_1) with $|m_0| = |m_1|$
    - Receives $c^* \leftarrow E(k, m_b)$ for a uniformly random secret bit b
  - The adversary outputs $\hat{b}$; $\mathrm{CPA}_{\mathrm{adv}}[\mathcal{A}, \mathcal{E}]$ must be negligible for all efficient $\mathcal{A}$
- This is the standard we need for any practical encryption scheme
  - Keys are expensive to establish, so we reuse them!

No Deterministic Cipher is IND-CPA Secure

- **Claim**: if $E(k, m)$ is deterministic, it *can't* be IND-CPA secure
  - This is true for *any* deterministic cipher, not just block ciphers!
- **Proof**: construct an adversary $\mathcal{A}$ that always wins
  - Pick any two distinct messages m_0, m_1
  - Submit the **challenge pair** (m_0, m_1); receive $c^* \leftarrow E(k, m_b)$
  - **After** the challenge: query the **encryption oracle** with the single message m_0
  - The oracle returns $c' \leftarrow E(k, m_0)$
  - Since encryption is deterministic: $E(k, m_0)$ always gives the same result
  - If $c^* = c'$, output $\hat{b} = 0$. Otherwise, output $\hat{b} = 1$.
- $\mathcal{A}$ is correct with probability 1, so $\mathrm{CPA}_{\mathrm{adv}}[\mathcal{A}, \mathcal{E}] = 1$
  - Not negligible! The cipher is broken under IND-CPA. □

What This Tells Us

- A bare block cipher $E(k, \cdot)$ **is** semantically secure (for a single encryption)
  - But it is **not** IND-CPA secure, because it's deterministic!
- The proof above applies to *any* deterministic cipher
  - Bare AES or DES? Deterministic. Not IND-CPA secure.
  - ECB mode? Deterministic. Not IND-CPA secure.
  - Stream cipher without a nonce? Deterministic. Not IND-CPA secure.
- **To achieve IND-CPA, encryption must be non-deterministic**
  - Either randomised (e.g. random IV) or nonce-based (e.g. counter)
  - This is exactly why we need ***modes of operation***

So are SS and IND-CPA equivalent?

- For a **deterministic** cipher: no! We just saw the bare block cipher is SS but not IND-CPA
- For a **probabilistic** cipher (one that uses randomness or a nonce): yes!
  - SS $\Rightarrow$ IND-CPA and IND-CPA $\Rightarrow$ SS
- The second direction (IND-CPA $\Rightarrow$ SS) is easy:
  - If you can't distinguish ciphertexts even with an encryption oracle, you certainly can't without one!
- The first direction (SS $\Rightarrow$ IND-CPA) is the interesting one
  - How do we go from "one ciphertext is safe" to "Q ciphertexts are safe"?
  - We need a proof technique: the **hybrid argument**

The Hybrid Argument: Intuition

- Imagine two photos of a room that differ in 10 small ways
  - Spotting *all* the differences at once is hard
  - But comparing two photos that differ in *one* way? Much easier
- A hybrid argument works the same way:
  - You want to show that World A and World B are indistinguishable
  - Instead of comparing them directly, build a chain of intermediate “hybrid” worlds
  - Each neighbouring pair differs in exactly **one** small change
  - If nobody can spot any single change, nobody can spot all the changes together

Hybrid Proof: SS $\Rightarrow$ IND-CPA

- Suppose the adversary makes Q encryption-oracle queries in the IND-CPA game
  - Each oracle response is either $E(k, m_0)$ or $E(k, m_1)$; this varies across hybrid worlds
- Build a chain of **hybrid worlds** $H_0, H_1, \ldots, H_Q$:
  - H_0: all Q ciphertexts encrypt m_0 (this is Experiment 0)
  - H_1: the **first** ciphertext encrypts m_1, the rest encrypt m_0
  - H_2: the first **two** encrypt m_1, the rest encrypt m_0
  - $\ldots$
  - H_Q: all Q ciphertexts encrypt m_1 (this is Experiment 1)
- Each step $H_i \rightarrow H_{i+1}$ changes **exactly one** ciphertext from $E(k, m_0)$ to $E(k, m_1)$
  - But the cipher is probabilistic, so the ciphertext looks fresh each time!

Hybrid Proof: The Punchline

- If the adversary could distinguish H_0 from H_Q...
  - ...they must be able to distinguish at least one neighbouring pair H_i from H_{i+1}
  - But H_i and H_{i+1} differ in only one ciphertext
  - Distinguishing one ciphertext of m_0 from one of m_1?
    - That's just the SS game!
- So the IND-CPA advantage is bounded by the sum of Q SS advantages:

$$\mathrm{CPA}_{\mathrm{adv}}[\mathcal{A}, \mathcal{E}] \leq Q \cdot \mathrm{SS}_{\mathrm{adv}}[\mathcal{B}, \mathcal{E}]$$

- If $\mathrm{SS}_{\mathrm{adv}}$ is negligible and Q is polynomial, the product is still negligible
- For a **probabilistic** cipher, SS and IND-CPA are equivalent!
  - For a **deterministic** cipher, this breaks down: fresh encryptions of m_0 are identical, so the adversary *can* spot the change

Hybrid Proofs: The Bigger Picture

- The hybrid technique shows up throughout cryptography
  - PRF switching, CCA reductions, key exchange proofs...
  - You'll see it again!
- Note: this shows SS $\Rightarrow$ IND-CPA for any probabilistic cipher where each fresh ciphertext is independently random
  - For CTR and CBC specifically, B&S uses a tighter proof via PRF security directly
  - The principle is the same: replace real randomness with ideal randomness, one step at a time

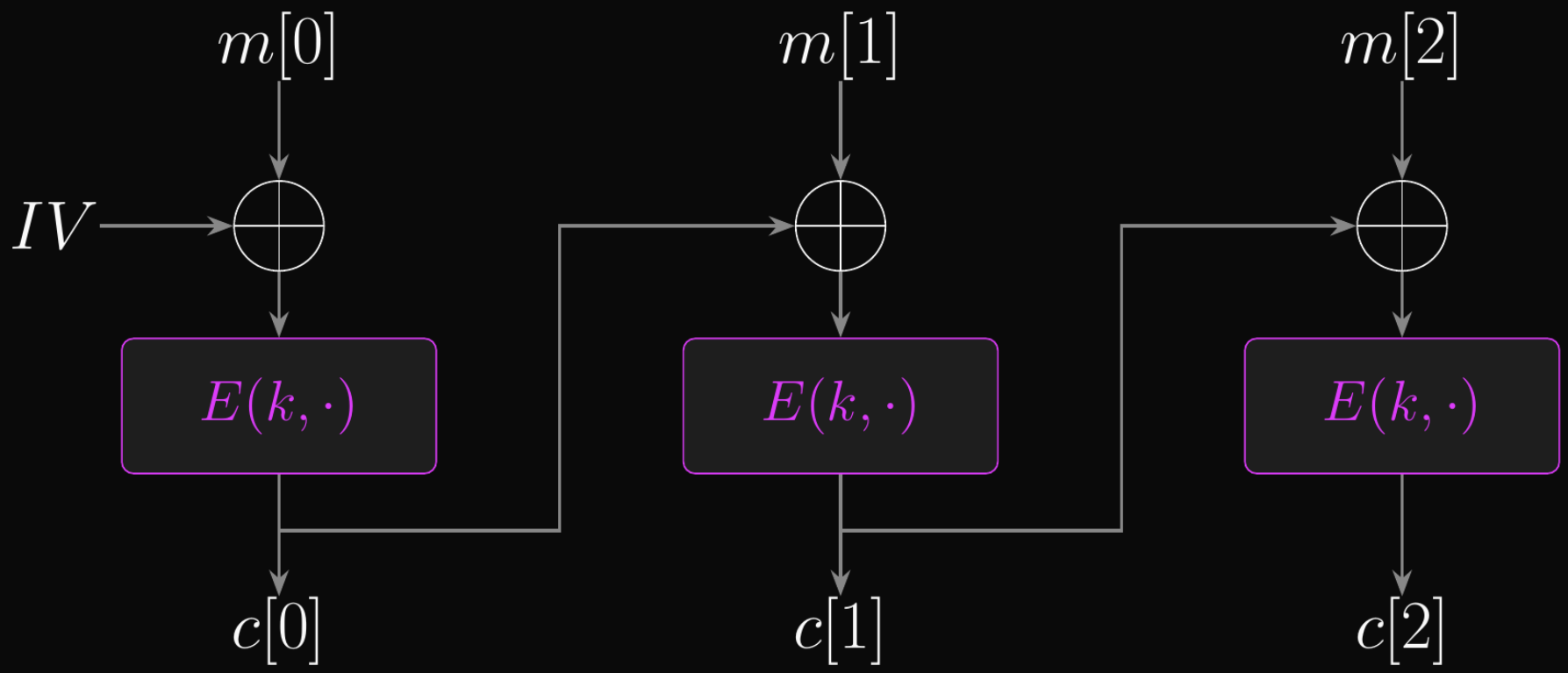
CBC Mode

What if we just XOR all the things?

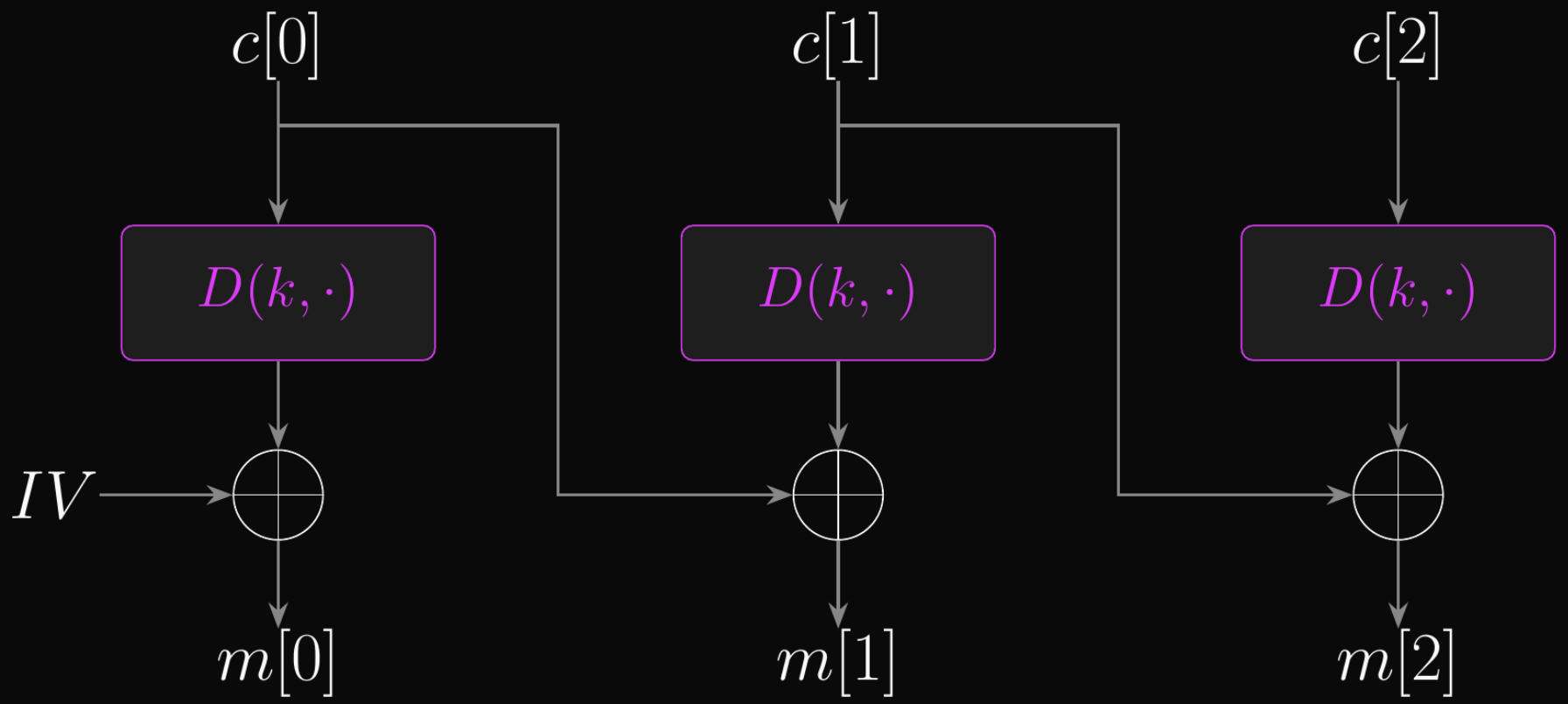
Cipher Block Chaining (CBC) Mode

- ECB mode is critically flawed, so we need another approach
- We need a strategy to obscure the output of the block cipher function
  - Identical plaintext blocks should produce different output blocks
- What if we just throw XOR at the problem? This has worked so far!
  - Anything XOR a random number produces a random output
  - Or at least, an output indistinguishable from a truly random output...
- What kind of pseudo-random values do we have available?
  - Block cipher output should be indistinguishable from random blocks
- What if we XOR the plaintext block with the previous ciphertext block?
  - Then we're encrypting essentially random input!
- What about the first plaintext block?
  - We'll need to use some kind of ***initialisation vector (IV)*** as a starting point
  - It turns out that doing this incorrectly can have consequences...

Cipher Block Chaining (CBC) Mode: Encryption



Cipher Block Chaining (CBC) Mode: Decryption



Cipher Block Chaining (CBC) Mode

- Can CBC mode encryption be parallelised?
  - No! Each block encryption has a dependency on the previous ciphertext block
- Can CBC mode decryption be parallelised?
  - Yes! All ciphertext blocks are already known before decryption begins
  - Decrypting block i only requires $c[i]$ and $c[i-1]$ (no forward dependencies)
- Does CBC mode allow random access?
  - Yes, for the same reason!

CBC Mode: IV and Security

- Note that the IV must be ***unpredictable***
  - But doesn't have to be secret!
  - In fact, it's sent in the clear most of the time!
    - Sometimes it's prepended to the ciphertext as $c[0]$, sometimes it's sent separately
  - If the IV is predictable, all bets are off!
    - We'll see some concrete attacks on this later...
- CBC with a random, unpredictable IV **is IND-CPA secure** (B&S Theorem 5.2)
  - Security reduces to PRF security of the block cipher via the switching lemma
  - The unpredictable IV ensures each message is encrypted with a fresh "effective key"
  - Even encrypting the same plaintext twice produces different ciphertexts

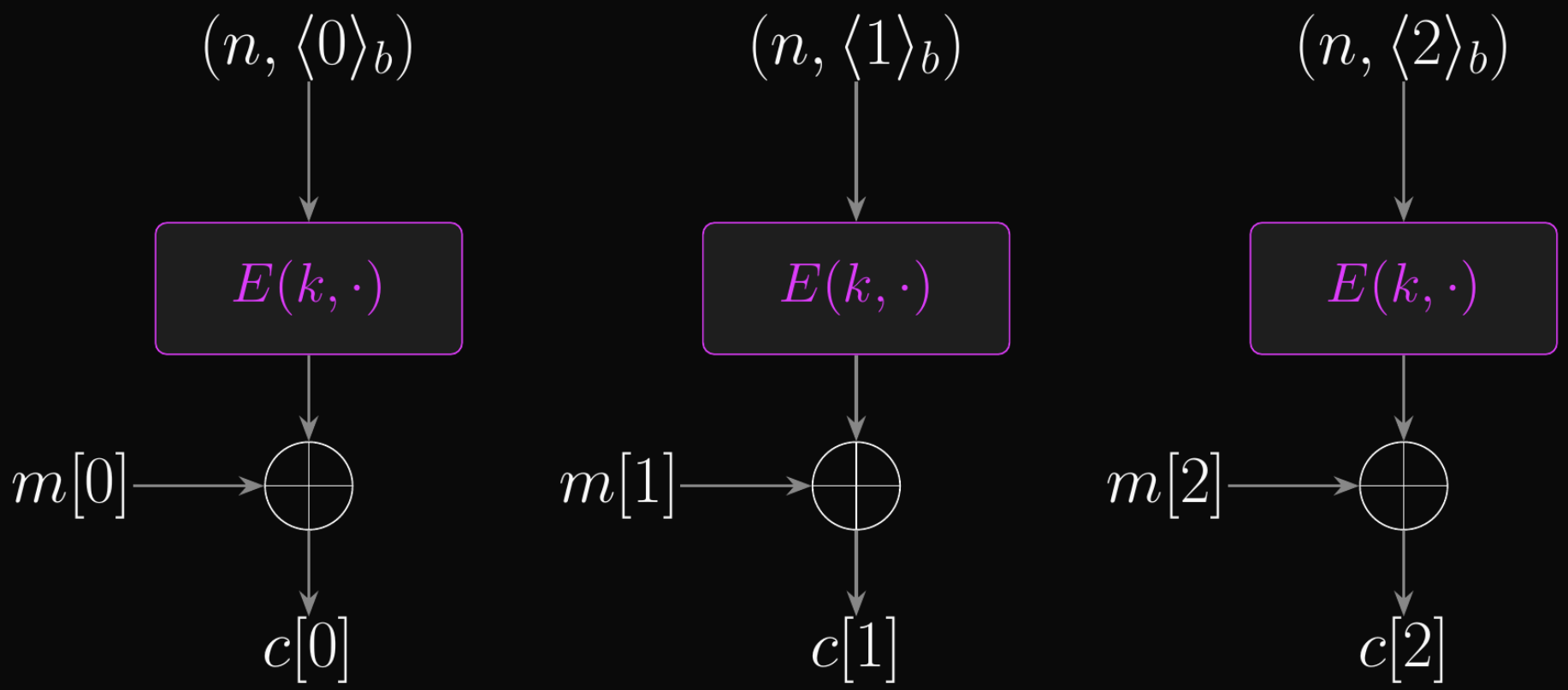
CTR Mode

What if we just build a stream cipher instead?

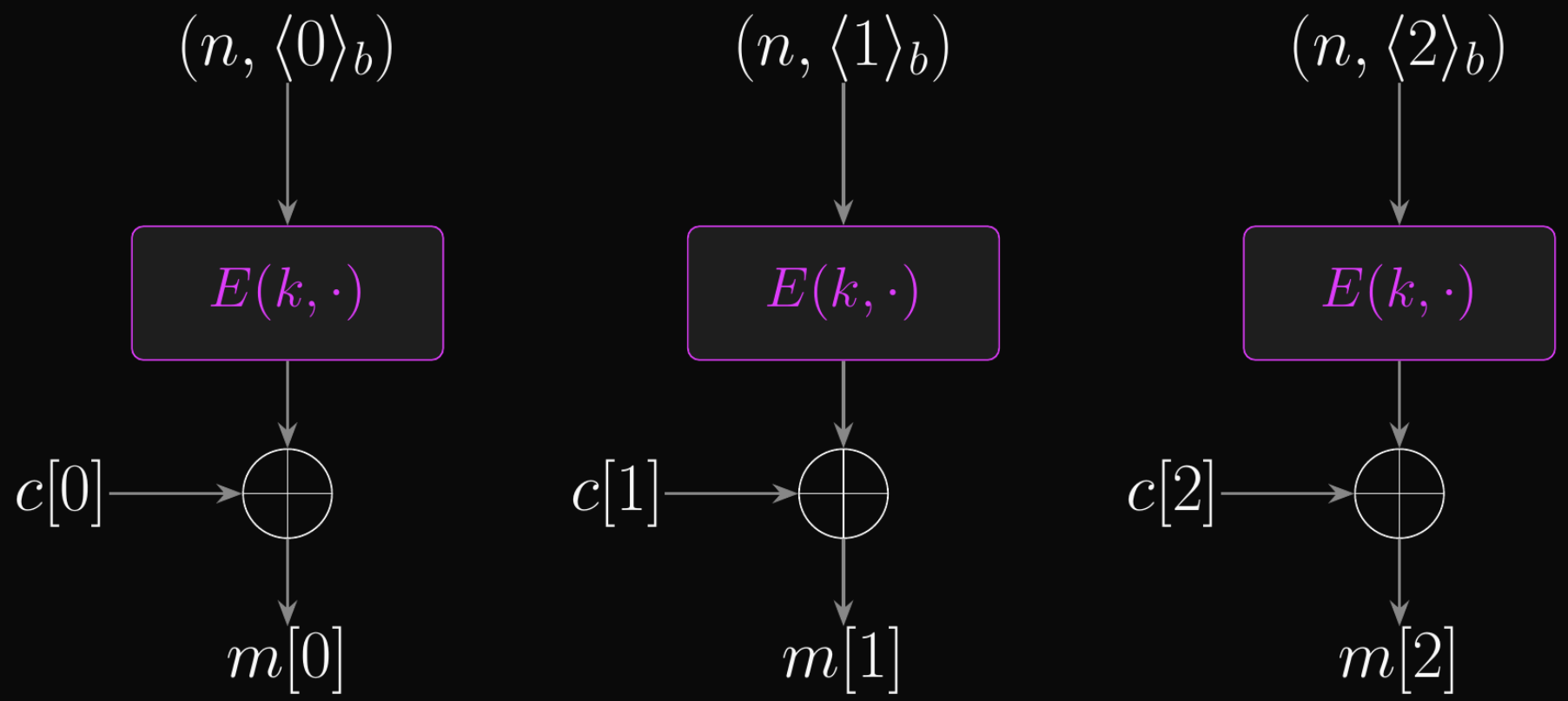
Counter (CTR) Mode

- CBC mode is great (or not so great, as we'll discuss), but not being able to parallelise encryption is a serious disadvantage compared to ECB mode
  - Can we have the best of both worlds?
- ***Counter mode*** is an interesting (and powerful) construct
  - Instead of encrypting the plaintext blocks using the block cipher...
  - ...we use the block cipher as a CSPRNG to create a keystream...
  - ...then we XOR the plaintext with the keystream...
  - ...to yield the ciphertext!
- Basically, CTR mode turns a block cipher into a stream cipher!
- But wait! Won't this make key reuse a huge problem, just like the many-time pad?
  - Not if we use a ***nonce***!
  - CTR mode incorporates a nonce, so instead of counting 0, 1, 2, 3, ...
  - CTR mode counts $(n, \langle 0 \rangle_b), (n, \langle 1 \rangle_b), (n, \langle 2 \rangle_b), \ldots$
    - Where $\langle x \rangle_b$ is the b-bit binary representation of x

Counter (CTR) Mode: Encryption



Counter (CTR) Mode: Decryption



Counter (CTR) Mode

- Can encryption be parallelised?
  - Yes! Each keystream block $E(k, (n, \langle i \rangle_b))$ is independent of all others
- Can decryption be parallelised?
  - Yes, for the same reason: decryption is identical to encryption in CTR mode
- Is random access feasible?
  - Yes! Generate any keystream block directly from its counter value, with no chain dependency
- As an interesting historical note, CTR mode was quite controversial when it was originally proposed!
  - The idea of systematic, predictable block cipher input was viewed negatively
  - But now it's popular and well-regarded
  - And is used as the basis for Galois/counter mode (GCM), which we'll meet later on during our discussion of authenticated encryption and AEAD
  - Originally proposed by Whitfield Diffie and Martin Hellman in 1979

Counter (CTR) Mode

- Notice that encryption and decryption are the same operation!
  - CTR mode is, to all intents and purposes, a stream cipher
  - This is a good example of how block ciphers can build CSPRNGs too
- The combination of key, nonce and counter must be unique!
  - No two blocks should be encrypted with identical values
  - This applies both within the same message and across many messages
  - Much like with ChaCha20, the split between nonce and counter isn't significant
- You could just start the count from a random number to make it unpredictable
  - But this allows for potential overlaps across many messages!
  - Using a nonce in the construction avoids that risk by design

CTR Mode: Security

- Note that concatenating the nonce and counter is the safest option!
  - You could XOR the nonce and counter, but this requires a truly random nonce
  - With a non-random nonce, XOR can produce collisions:
    - $(n_1 \oplus \langle 0 \rangle_b) = (n_2 \oplus \langle 1 \rangle_b)$ if $n_1 \oplus n_2 = 1$, defeating uniqueness!
- CTR with a unique nonce **is IND-CPA secure** (B&S Theorem 5.1)
  - Security reduces to PRF security of the block cipher via the switching lemma
  - An adversary breaking CTR-mode IND-CPA can be used to break the underlying PRF, which is a contradiction

Block Cipher Modes: Comparison

	ECB	CBC	CTR
Encryption	$c[i] = E(k, m[i])$	$c[i] = E(k, m[i] \oplus c[i-1])$	$c[i] = m[i] \oplus E(k, (n, \langle i \rangle_b))$
Decryption	$m[i] = D(k, c[i])$	$m[i] = D(k, c[i]) \oplus c[i-1]$	$m[i] = c[i] \oplus E(k, (n, \langle i \rangle_b))$
`Enc` parallelisable	Yes	No	Yes
`Dec` parallelisable	Yes	Yes	Yes
Random read	Yes	Decryption only	Yes
Partial last block	Padding required	Padding required	Yes
IV / Nonce	None	Unpredictable IV	Unique nonce
Semantically secure	No	Yes	Yes
Notes	Leaks block equality	IV sent unencrypted	`Enc` = `Dec` (stream cipher)

PKCS#5 and PKCS#7

- Zero-padding (appending null bytes) is the simplest approach but fails at decryption time
  - A message that legitimately ends with the padding byte is indistinguishable from a padded message!
- Both ***PKCS#5*** and ***PKCS#7*** solve this elegantly; they are basically the same thing!
  - PKCS#5 is technically only for 64-bit (8-byte) blocks.
  - PKCS#7 generalises to any block size up to 255 bytes.
- First, figure out how many bytes of padding you need.
  - If the message is already a multiple of the block size, you need to add an entire block of padding!
    - Why?
  - Let's say we need n bytes of padding
- Add that n bytes of padding, using n as the byte value.
  - E.g. if $n = 3$, then add `03 03 03` (base 16)
  - E.g. if $n = 6$, then add `06 06 06 06 06 06` (base 16)
- Finally, go ahead and encrypt the padded message as usual!

Removing PKCS Padding

- After decryption, read the last byte of the message.
  - This gives you n, the number of padding bytes
  - Trim n bytes from the end of the message
  - Now we're left with the original plaintext
  - Unlike zero-padding, this is unambiguous
- There are lots of other padding schemes out there too...

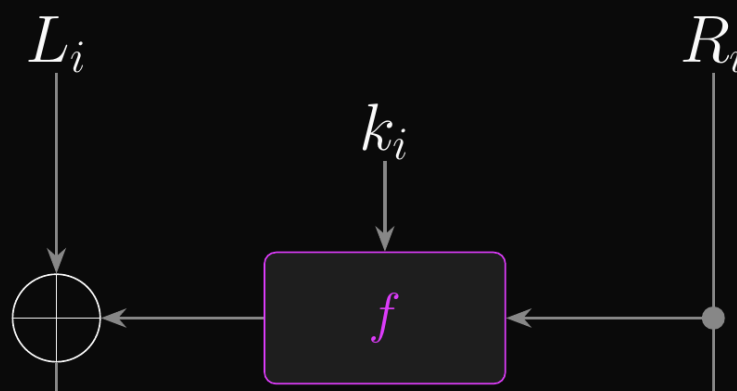
Feistel Networks

From round functions to full block ciphers.

Feistel Networks

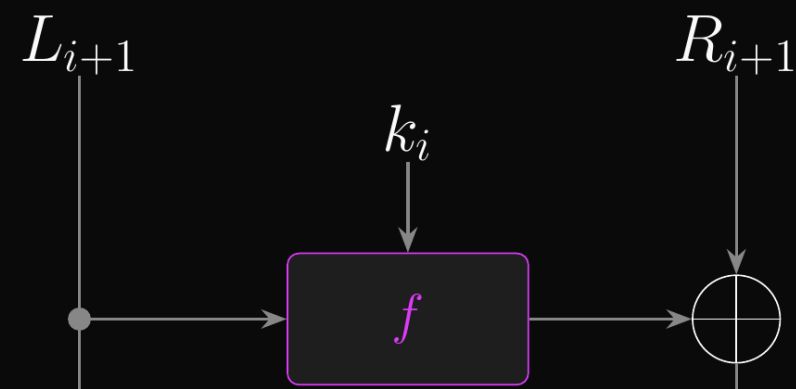
- A ***Feistel network*** builds a block cipher from a simple ***round function*** f
- Split the input block into two halves: L_0 and R_0
- Each round i applies:
  - $L_{i+1} \leftarrow R_i$
  - $R_{i+1} \leftarrow L_i \oplus f(k_i, R_i)$
- After all rounds, concatenate the halves to produce the output
- Key property: ***always invertible***, even if f itself is not
  - To invert, just recompute $f(k_i, R_i)$ and XOR again
  - This means the round function can be *any* function – it doesn't need an inverse
- Used by DES, Blowfish, Camellia, and many other block ciphers

Feistel Round: $\pi(L_i, R_i)$



$$L_{i+1} = R_i \qquad R_{i+1} = L_i \oplus f(k_i, R_i)$$

Feistel Round Inverse: $\pi^{-1}(L_{i+1}, R_{i+1})$



$$L_i = R_{i+1} \oplus f(k_i, L_{i+1}) \qquad R_i = L_{i+1}$$

Constructing Iterated Block Ciphers

- Most practical block ciphers use the ***iterated cipher*** paradigm
- Building an iterated block cipher requires two design choices
  - Pick a simple round function to use as the per-round cipher
  - Pick a simple PRG to expand the key into a number of **subkeys** or **round keys**
    - This is called the ***key expansion function***, and doesn't have to be secure
    - The subkeys it produces are collectively called the ***key schedule***
- Neither the round function nor the key expansion needs to be individually secure
  - Security emerges from sufficient iteration of the two together

Mathematical Rigour?

- Does iterating possibly-insecure round ciphers and PRGs like this give security?
  - Nobody knows, but we think so!
- Can any block cipher function work as a round cipher?
  - No! Linear or affinefunctions shouldn't be used, e.g. $f(k, x) = k \oplus x$
  - We need non-linearity, e.g. via S-boxes, to achieve security
- Is there an easy way to tell if a specific function will work as a round cipher?
  - Unfortunately, no...
- How many times does a round cipher need to be iterated to give security?
  - No general solution for this one either...
- What conclusion should we draw from this?
  - **DON'T ROLL YOUR OWN BLOCK CIPHER**
  - Block cipher design is *highly non-trivial* and very easy to get wrong
  - It takes years of analysis to gain confidence in a block cipher's security

Feistel Networks vs SPNs

- Two dominant paradigms for building iterated block ciphers
- ***Feistel network***: splits block into halves, applies a round function to one half and XORs with the other
  - Round function does not need to be invertible – Feistel structure guarantees invertibility
  - Examples: DES, Blowfish, Camellia
- ***Substitution-permutation network (SPN)***: applies substitution (S-box, non-linear) and permutation (linear diffusion) layers to the entire block
  - Each layer must be individually invertible
  - Simpler structure; rounds can be parallelised more easily
  - Examples: AES, Serpent, PRESENT

DES

A block cipher from the bad old days.

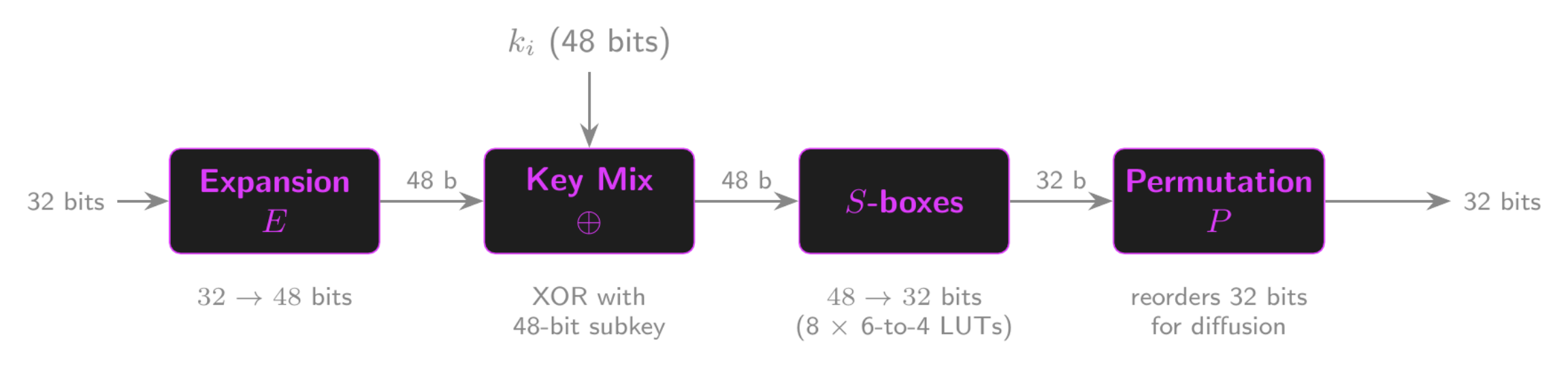
The Origin of DES

- The ***Data Encryption Standard (DES)*** was built by IBM for NIST in the 1970s
  - Adopted as a federal standard for unclassified data in 1977
- NBS (the future NIST) constrained the block to 64 bits and key to 56 bits
  - Very controversial! Allegations of deliberate sabotage by intel agencies
  - Small block size is also a problem
- Fantastic moment for cryptanalysis: a gold standard cipher to attack!
  - Lots of theoretical breakthroughs
  - Eventually cracked in the late 1990s
- Please don't use DES for anything these days!
  - Exhaustive key search is feasible to the point where there's SaaS for it
  - But it's not as gone as you might think...

The DES Round Function

- DES uses a 16-round ***Feistel network***
  - Split 64-bit input $\rightarrow$ 32-bit x and 32-bit y
  - Apply round function f to x and subkey, output XORed with y
  - Swap x and y
- The ***DES round function*** takes a 48-bit subkey and 32-bit input
  - **Expansion**: the function E expands 32-bit input to 48 bits
    - This is done with a fixed set of one-to-many bit mappings
  - **Key mixing**: XOR with 48-bit subkey
  - **Substitution**: S-boxes reduce 48 bits to 32 bits using 6-bit to 4-bit LUTs
    - The design criteria for the 8 S-boxes were initially kept ***secret*** by the NSA, raising ***backdoor suspicions***
    - Later analysis found them to be hardened against ***differential cryptanalysis*** (which wasn't public knowledge at the time)
    - This controversy helped establish the ***nothing-up-my-sleeve*** principle: derive constants from natural mathematical values (e.g. SHA uses fractional parts of square roots of primes) so no hidden structure can be engineered in
  - **Permutation**: the function P reorders bits for diffusion
    - This is a fixed ***mixing permutation***

DES Round Function



The DES Algorithm

- The key expansion function G takes the 56-bit key k as input
  - Outputs 16 48-bit round keys
  - Each round key's 48 bits are a specific subset of bits from k
- DES runs a 16-round Feistel network using the key expansion function (KEF) and round function
  - An initial permutation IP runs at the start
  - A final permutation $\mathrm{FP} = \mathrm{IP}^{-1}$ runs at the end
  - We don't really know why...
- The permutations have no cryptographic significance
  - DES isn't any more or less secure with them present
  - One theory is deliberate performance degradation for DES software!

The Flaw in the Plan

- The 56-bit key size should be ringing alarm bells!
  - Minimum safe key size today is 128 bits
  - Controversial even at the time!
- DES is vulnerable to ***exhaustive key search*** (demonstrated in 1998)
  - DeepCrack cost $250k and managed it in 56 hours to win a $10k prize
    - No one said cryptographers were good at economics...
  - Takes about a day on modern hardware
- One of the S-boxes is too linear, and can be attacked with linear cryptanalysis
  - The key can be recovered after 2^{41} operations
- The 64-bit block size is also too small!
  - Birthday attacks mean that 2^{32} queries would be enough to get a solid advantage against DES in CBC mode
  - 128-bit blocks should be considered the standard

Triple-DES

- So, your federal encryption standard is rapidly falling apart. What next?
  - Make a new standard, right? But...
  - Lots of companies have bought dedicated DES hardware
  - They're not going to be happy if it's made obsolete...
- As a stopgap measure, we got 3DES in 1998
  - What if, instead of doing DES once, we just did it three times?
  - That means triple the key size! A 168-bit key is much more secure
- Triple-DES encrypts, decrypts, and then encrypts again:
  - $E_{\text{3DES}}((k_1, k_2, k_3), x) = E_{\text{DES}}(k_3, D_{\text{DES}}(k_2, E_{\text{DES}}(k_1, x)))$
  - Why? For backwards-compatibility with DES if all keys are the same!
  - Double-DES is no more secure than DES due to ***meet-in-the-middle attacks***
    - Encrypt forward under k_1, decrypt backward under k_2, find a match: cost is $2|\mathcal{K}|$, not $|\mathcal{K}|^2$
  - 3DES is much more expensive to run than DES
  - It's obvious that a better standard is needed...

AES

Rijndael to the rescue!

The Foot-Shooting Prevention Agreement

- The AES pledge (paraphrased from Jeff Moser):
  - “I promise that once I see how simple AES really is, I will not implement it in production code even though it would be really fun.”
  - “This agreement will remain in effect until I learn all about side-channel attacks and countermeasures to the point where I lose all interest in implementing AES myself.”
- The Stick Figure Guide to AES

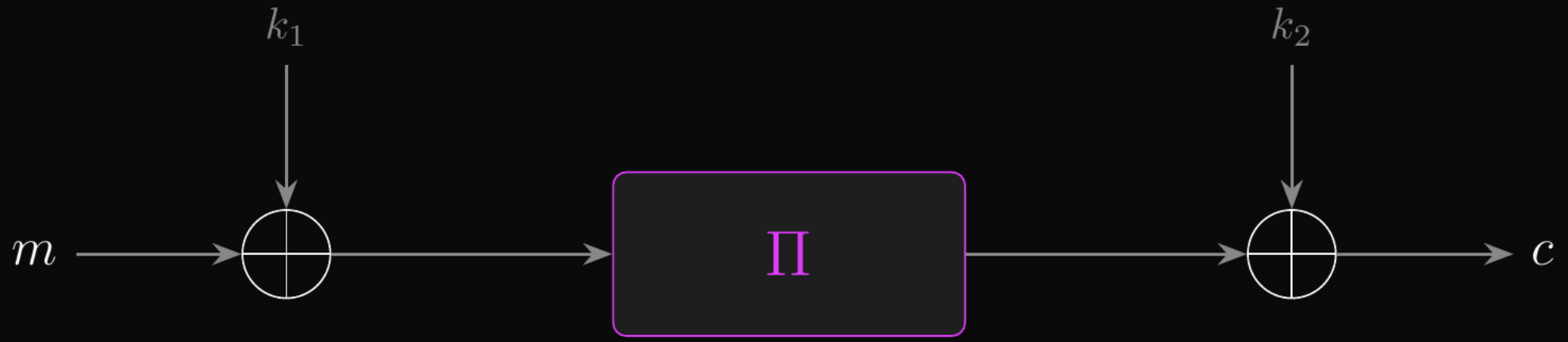
The AES Process

- The AES process was started by NIST in 1997 to replace DES
  - The new cipher would be the Advanced Encryption Standard (AES)
  - Requirements: 128-bit blocks, key sizes of 128, 192 and 256 bits
- Global submissions were accepted, discussed and subjected to cryptanalysis
  - A final five candidates presented at an open conference in April 2000
  - In October 2000, Rijndael was selected as the AES cipher
  - Designed by Joan Daemen and Vincent Rijmen (Belgium).
- AES became official in 2001 (FIPS 197).
  - Unlike DES, AES is recommended by the NSA for classified information
  - Some of Rijndael's features, like variable block sizes, were removed
- AES has hardware-level support from all major CPU manufacturers
  - For example, AES-NI in the x86 ISA
  - Sometimes provided by an AES co-processor on a separate chip
  - Extremely fast, usually optimised and pipelined

The AES Algorithm

- AES is an alternating key cipher, or an iterated Even-Mansour cipher
  - The input is XORed with the zeroth round key
  - Each round starts with a fixed permutation that doesn't depend on the key
  - At the end of each round, the current round key is XORed with the output
  - AES-128 has 10 rounds, AES-192 has 12 rounds, and AES-256 has 14 rounds
- Each round permutation Π_{AES} is an ***SPN*** – a substitution layer followed by a permutation layer:
  - `SubBytes`: the ***substitution*** layer – applies a fixed S-box to each byte for non-linearity
  - `ShiftRows` + `MixColumns`: the ***permutation*** layer – linear diffusion across the block
    - `ShiftRows` cyclically shifts rows in a 4×4 byte matrix
    - `MixColumns` mixes columns using matrix multiplication in GF(2^8)

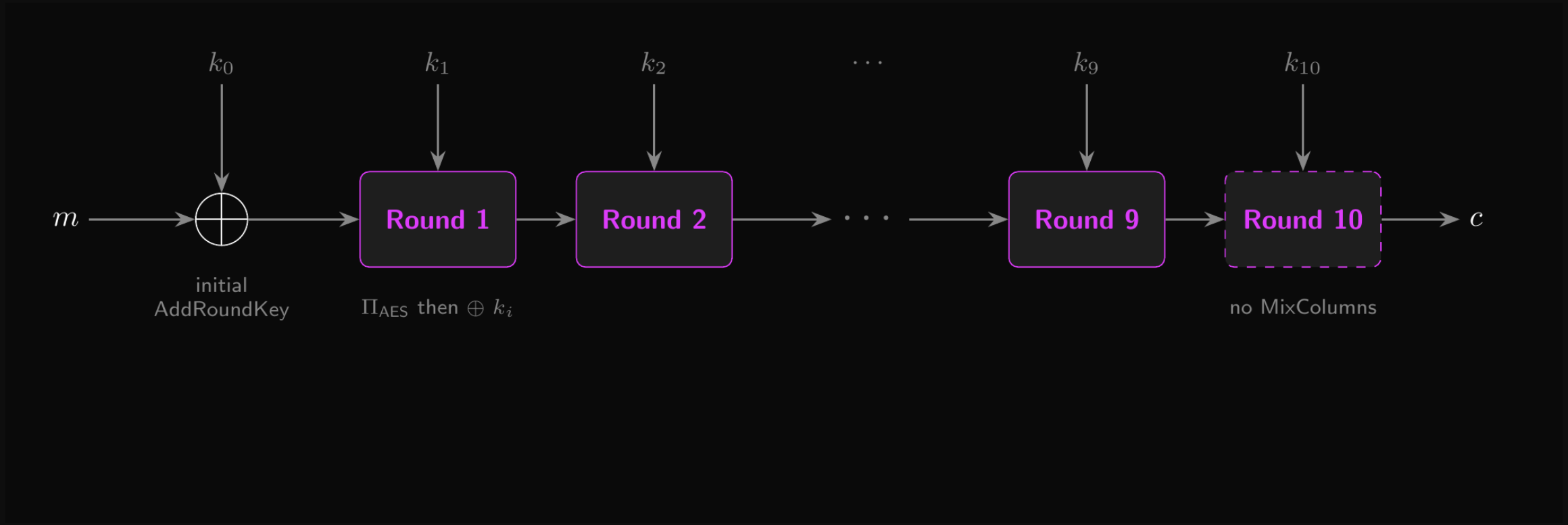
Even-Mansour Construction



fixed permutation
(key-independent)

Secure in the ideal cipher model when Π is a random permutation

AES-128: Cipher Structure

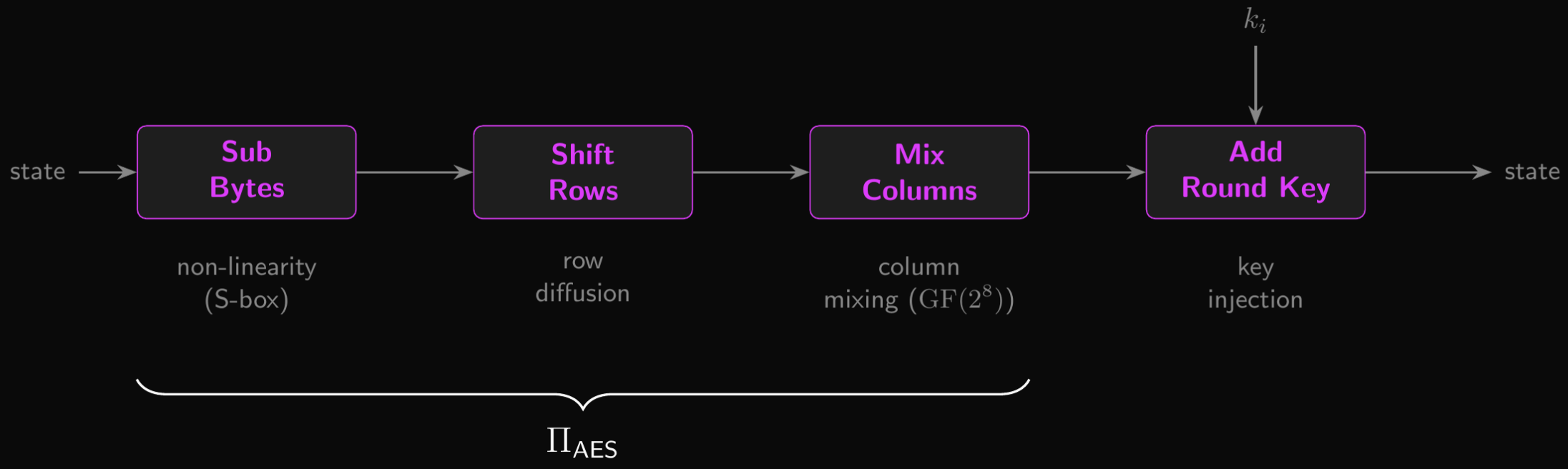


AES: Security and Implementation

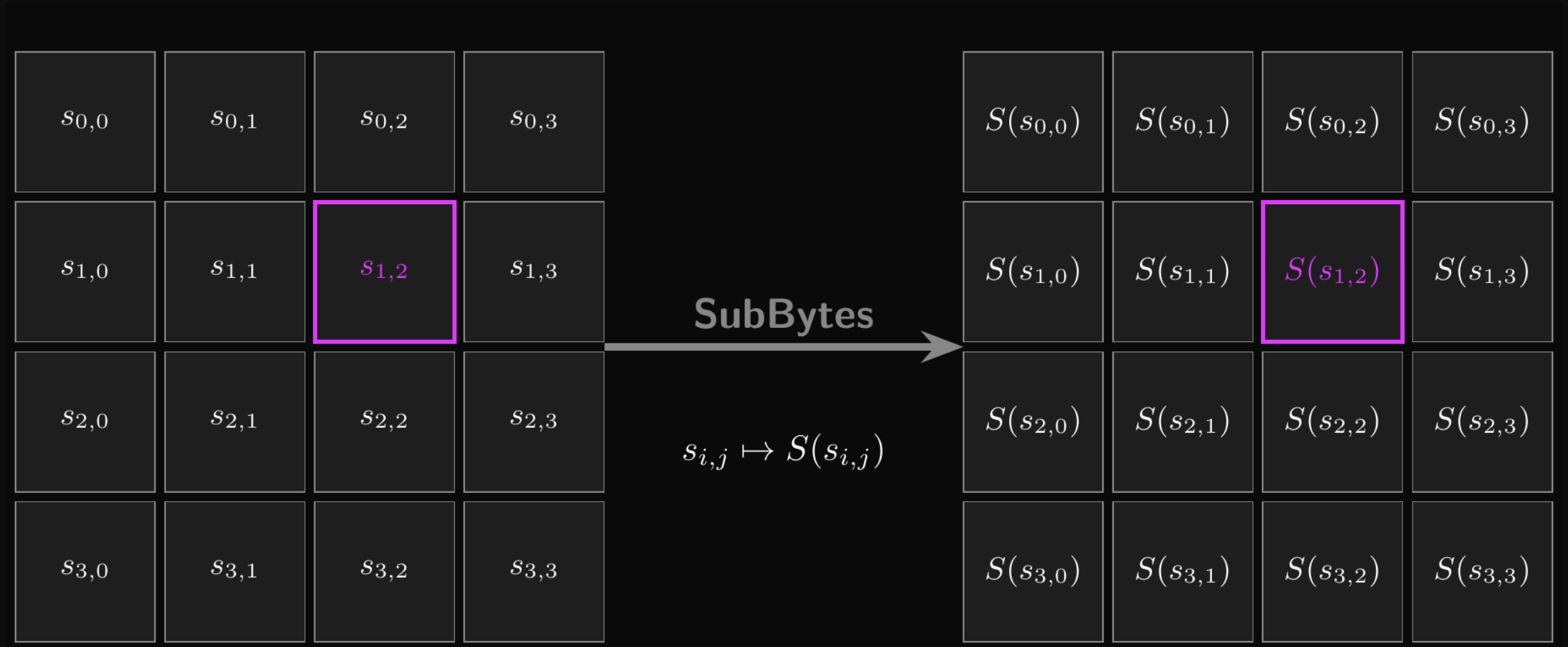
- This structure has a formal security proof in the **ideal cipher model**
  - If Π_{AES} is modelled as a random permutation, the iterated Even-Mansour construction is a secure block cipher in the ideal cipher model (B&S §4.3.4)
  - AES inherits its security argument from this framework
- The final round omits `MixColumns` to simplify decryption logic
- Each step is designed to be easily invertible!
- AES can be implemented with pre-computed lookup tables (LUTs) for efficiency
  - This is a nice idea that usually ends in disaster...

AES Round Permutation Π_{AES}

Final round omits **MixColumns** to simplify decryption

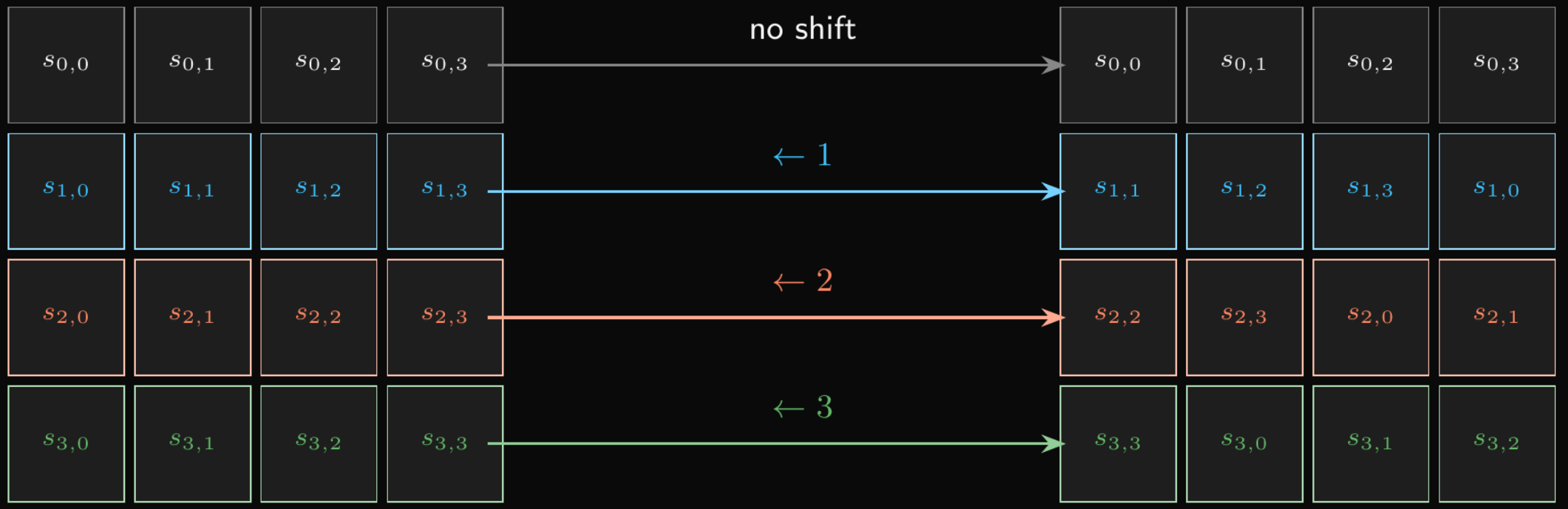


SubBytes: Byte Substitution



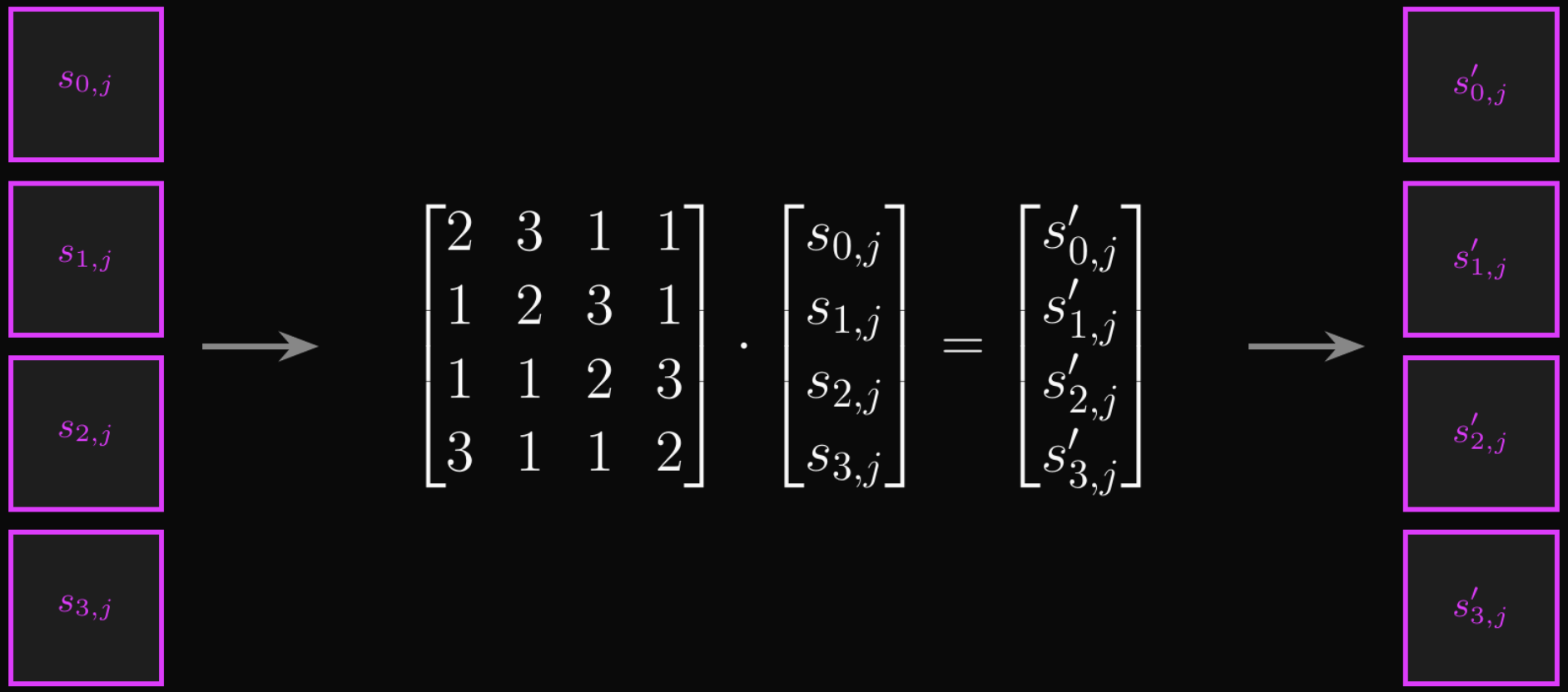
Fixed 256-entry lookup table applied independently to each byte

ShiftRows: Row Diffusion



Cyclic left shift of each row spreads bytes across columns

MixColumns: Column Mixing

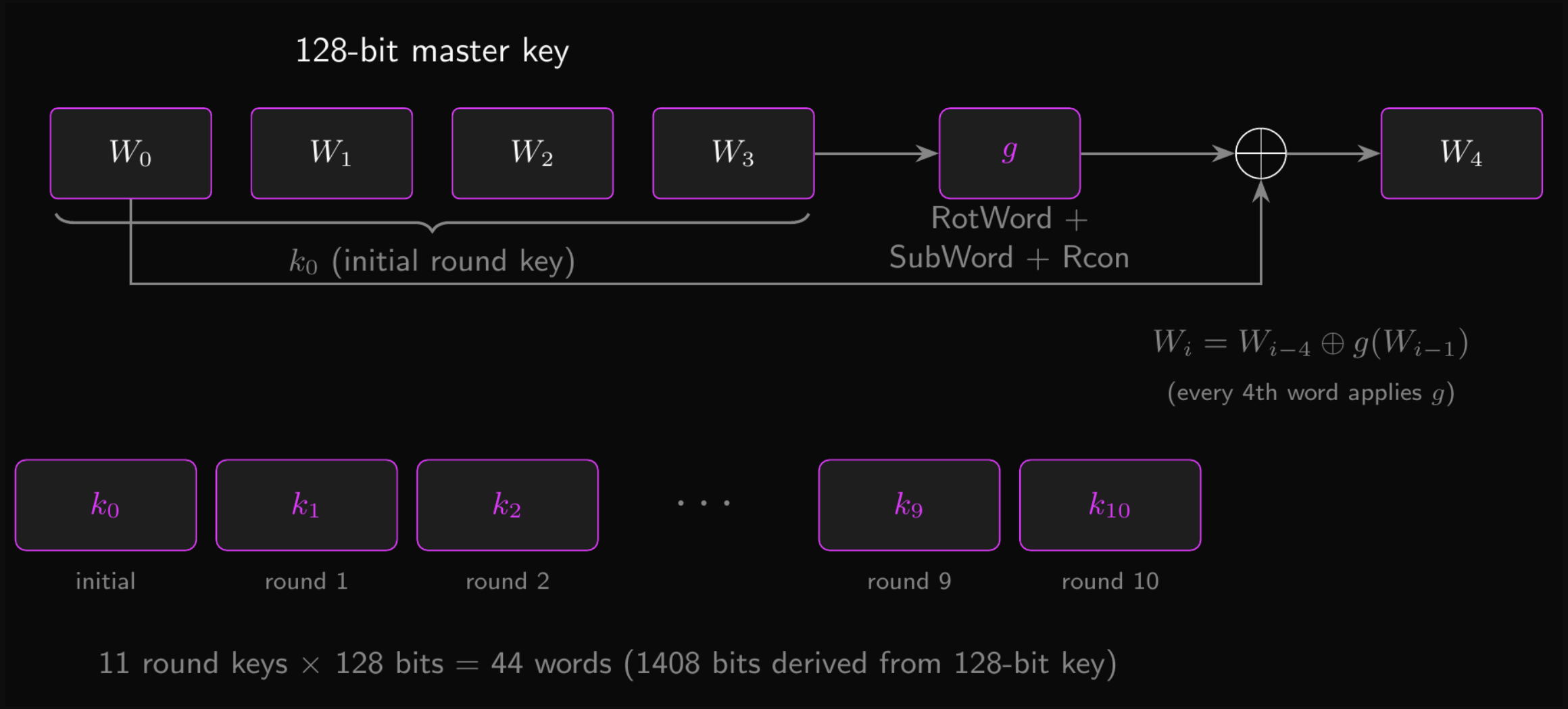


Each column independently mixed via fixed MDS matrix in GF(2^8)

AES Key Schedule

- AES-128 needs 11 round keys (one initial + ten rounds) from a single 128-bit key
  - The key schedule expands 128 bits into $11 \times 128 = 1408$ bits
- The master key is split into four 32-bit words: W_0, W_1, W_2, W_3
- Subsequent words are derived recursively:
  - Most words: $W_i = W_{i-4} \oplus W_{i-1}$
  - Every 4th word applies a function g: $W_i = W_{i-4} \oplus g(W_{i-1})$
- The function g provides non-linearity:
  - `RotWord`: cyclic byte rotation
  - `SubWord`: apply the SubBytes S-box to each byte
  - XOR with a round constant $\mathrm{Rcon}[i]$
- Each round key k_i is the next group of four words: $k_i = (W_{4i}, W_{4i+1}, W_{4i+2}, W_{4i+3})$

AES-128: Key Schedule



Security of AES

- AES has withstood extensive cryptanalysis since its standardization.
  - A much more impressive history than DES!
- Best key recovery attacks use ***biclique*** techniques
  - A refined form of meet-in-the-middle
  - Slightly faster than exhaustive search, but still impractical
    - Best-known attack on AES-128 takes $2^{126.1}$ AES evaluations
    - Best-known attack on AES-256 takes $2^{254.4}$ AES evaluations
  - No real-world security impact
- AES-256 shows theoretical vulnerability to ***related key attacks***
  - With four carefully chosen related keys satisfying a specific XOR equation...
    - ...an attacker could recover all of the keys in ~$2^{99.5}$ AES evaluations
  - Not relevant in practice, where keys are random and unrelated
- Attacks are mostly of academic interest under unrealistic assumptions (for now)

Side-Channel Attacks

- Side-channel attacks are a bit different from mathematical attacks
  - They attack the implementation rather than the design of the cipher
- ***Timing attacks*** exploit a relationship between the time it takes to encrypt a block and the value of the secret key
  - This might be caused by branching or caching at the CPU level
  - With vulnerable implementations, key recovery attacks have been demonstrated in practice within a few minutes!
  - Hardware implementations are a better option when available
- ***Power attacks*** exploit a relationship between the power consumption of a device and the instructions it executes
  - AES is secure against simple power analysis
  - But not all implementations are secure against ***differential power analysis***
    - Power traces over thousands of encryptions can leak the key
  - Mitigations at a hardware level use capacitors for constant power consumption

The Quantum Apocalypse?

- We've only discussed classical computers thus far
  - What about quantum computers? Is there an apocalypse coming?
- Remember exhaustively searching the key space in our attack game?
  - On a classical computer, this takes $O(|\mathcal{K}|)$
  - For AES-128, $|\mathcal{K}|$ means $2^{128} \approx 3.4 \times 10^{38}$
- Let's introduce Grover's algorithm for quantum exhaustive search
  - Given $f : \mathcal{K} \to \{0, 1\}$, where $f(k) = 1$ if $k = k_0$ and 0 otherwise...
  - ...a quantum computer can find $k_0 \in \mathcal{K}$ in $O(\sqrt{|\mathcal{K}|} \cdot \mathrm{time}(f))$
- So if we define $f_{\mathrm{AES}}(k) = 1$ if $\mathrm{AES}(k, m) = c$ and 0 otherwise...
  - ...we can find the key in $O(\sqrt{|\mathcal{K}|})$
  - For AES-128, $\sqrt{|\mathcal{K}|}$ means $2^{64} \approx 1.84 \times 10^{19}$
  - That's alarmingly feasible, and a massive reduction in security!

Are We Doomed?

- So, time to panic yet? Or is there still some hope for AES?
  - First of all, no one's got a quantum computer big enough, fast enough or reliable enough to do this!
- 128-bit keys are our benchmark for block cipher security
- AES-128 under quantum exhaustive search essentially has a 64-bit key
- AES-192 likewise gets reduced to the effective security of a 96-bit key
- But AES-256 is reduced to the security level of AES-128, more or less
  - So if you're worried about quantum attacks, just use AES-256
  - This is part of the reason why AES supports a 256-bit key size!
- Why act now? Adversaries may follow a ***harvest now, decrypt later*** strategy
  - Intercept and store encrypted traffic today, even if it is currently undecryptable
  - Decrypt it in the future once a sufficiently powerful quantum computer exists
  - Medical records, financial data, state secrets: some information stays valuable for decades
- So you can already do post-quantum symmetric cryptography - crisis averted!
  - You should really be panicking about asymmetric cryptography instead...
  - More on that in an upcoming lecture!

Attack the Block

Cracking block ciphers (for fun and profit).

Encryption Oracle Attack

- ECB mode can be defeated by simple visual inspection, e.g. Tux the penguin
  - You might argue that we haven't actually decrypted anything...
  - But we've successfully extracted information from the ciphertext!
  - You might also point out that text doesn't seem so vulnerable...
- But with just a few assumptions, ECB mode can blow encryption wide open
  - No matter how strong the block cipher we use is!
- Let's see how this works with AES

Encryption Oracle Attack

- All we need is a setup where we can:
  - inject some chosen plaintext
  - to be prepended to some secret information being encrypted
  - E.g. an input to a server.
- What's AES's block size?
  - How many possible values can we store in 128 bits / 16 bytes?
  - How feasible is a brute-force attack?
- Can we be smarter about this using those assumptions?

Encryption Oracle Attack

- We can't feasibly brute-force AES, even in ECB mode
  - But we can prepend bytes to the message! Let's exploit this...
- Let's take the first block...
  - ???????? ???????? ???????? ????????
  - Can we narrow down the possibilities?
- We can inject 15 bytes of known plaintext at the start!
- The first block now looks like this:
  - 00000000 00000000 00000000 000000??
- Only a single byte (the first byte of the secret message) is unknown!
  - So the ciphertext for our new block...
  - ...only has 256 corresponding plaintexts
- That's a huge improvement for our chances!

Encryption Oracle: Isolating One Byte

Inject 15 known bytes: the first unknown secret byte is isolated

only 256 possible plaintexts for this block

Try all 256 candidates — match to find the secret byte

`xx` $\in \{$`00`, `01`, $\ldots$, `FF`$\}$ — when ciphertext matches, the secret byte is found

Encryption Oracle Attack

- Hold on a second! We still don't have the key.
  - How are we supposed to brute force those 256 possibilities?
- Easy! We've basically got an encryption oracle we can exploit.
  - Keep the ciphertext we got for our block on the last slide in mind
  - Let's inject 16 bytes (a full block) instead of 15 this time...
- Try it 256 times, once for each possible plaintext byte:
  - 00000000 00000000 00000000 00000000
  - 00000000 00000000 00000000 00000001
  - ...and so on, all the way to...
  - 00000000 00000000 00000000 000000FF
- When the ciphertext matches the one we got on the last slide...
  - We've cracked the first byte of the secret message!

Encryption Oracle Attack

- From here, it's easy!
- How can we get the second character of the secret message?
  - We now know the first character, remember!
  - Inject 14 bytes + 1 cracked byte (15 in total)
  - Now the second byte of the message is isolated
  - Cycle through 256 possibilities and find a match
- And so on!
  - Keep cycling the length of the prefix, crack one byte at a time
  - After 16 bytes, target the second block instead
  - Always aim to isolate a single unknown byte
- Quick, easy and incredibly dangerous!
- This is a textbook chosen-plaintext attack
  - The adversary chooses plaintexts, observes ciphertexts, and exploits ECB's determinism
  - This is exactly what IND-CPA security is designed to prevent
  - Only feasible under certain conditions

Predictable IVs

- If we can predict ahead of time what IV is going to be used to encrypt a message, we can exploit it!
- How might this happen?
  - Using an all-zero IV
  - Using the same IV for a user all the time
  - Using the last ciphertext block from the previous message as the IV (BEAST attack on TLS 1.0)
  - Using a misconfigured or low-quality RNG.
- If we've already intercepted an IV and ciphertext...
  - We can potentially crack it!
- Let's assume that we can send messages to the server...
  - ...knowing that they'll be encrypted with the same key...
  - ...and knowing that we can predict the IV

Predictable IVs

- First, let's guess what the intercepted ciphertext block's plaintext might be
  - Let's call the guess G
- We predict the IV that will be used for us, and ask the server to encrypt:
  - $M = IV_{\text{ours}} \oplus IV_{\text{theirs}} \oplus G$
- The server, using CBC, will try to encrypt:
  - $IV_{\text{ours}} \oplus M$
  - $= IV_{\text{ours}} \oplus (IV_{\text{ours}} \oplus IV_{\text{theirs}} \oplus G)$
  - $= (IV_{\text{ours}} \oplus IV_{\text{ours}}) \oplus IV_{\text{theirs}} \oplus G$
  - $= IV_{\text{theirs}} \oplus G$
- Using that result, we can verify whether our guess was correct or not!
  - With structured plaintexts like HTTP, HTML and JSON...
  - ...guessing is easier than random chance!
- Predictable IVs effectively make CBC deterministic
  - This breaks the non-determinism that CBC needs for IND-CPA security

Key-as-IV

- Using the key as the IV is really tempting...
  - It's right there and saves us some effort!
  - We don't even have to bother generating, storing or sending the IV.
- Sounds too good to be true...
  - Because it is too good to be true!
  - A chosen ciphertext attack can recover the key.
- The key is meant to be secret.
  - The IV is not meant to be secret, just unpredictable.
  - Mixing up secret and non-secret data can have dire consequences!
- Sometimes it's okay to use secrets where they're not required...
  - ...but not as a general rule. Always pay attention to the context!

Chosen Ciphertext Attack on Key-as-IV

- In a chosen-ciphertext attack (CCA), we get to choose ciphertexts and view their corresponding plaintexts.
- Alice and Bob have, in their infinite wisdom, chosen to use the key as the IV in CBC.
- Alice encrypts her message (P_1, P_2, P_3) with the key and sends the ciphertext to Bob.
- Alice and Bob don't have to transmit the IV
  - It's the key, they know it already!
- Alice sends the encrypted blocks (C_1, C_2, C_3)
  - Mallory is feeling chaotic, so they intercept the ciphertext and modify it!
  - Mallory also happens to have access to Bob's decryption software
  - Mallory can view plaintexts for chosen ciphertexts
- Mallory asks for the decryption of (C_1, Z, C_1)
  - Z is a block of null bytes (all zero in binary)

Chosen Ciphertext Attack on Key-as-IV

- We'll use P for Alice's original plaintext and P' for Mallory's fake plaintext
- $P'_1 = D(k, C_1) \oplus IV = D(k, C_1) \oplus k = P_1$
- $P'_2 = D(k, Z) \oplus C_1 = R$ for some $R \in \mathcal{X}$
- $P'_3 = D(k, C_1) \oplus Z = P_1 \oplus IV = P_1 \oplus k$
- Mallory receives the decryption (P'_1, P'_2, P'_3) and computes $P'_1 \oplus P'_3$:
  - $P'_1 \oplus P'_3 = P_1 \oplus P_1 \oplus k = k$
  - The key falls directly out of a simple XOR!
- CBC with the key as the IV is not semantically secure under CCA
  - Even worse, it's susceptible to full key recovery!
  - This goes beyond IND-CPA; we'll formalise CCA security later in the module

Conclusion

What did we learn?

So, what did we learn?

- A block cipher $E : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ is a keyed permutation
  - Security: $E(k, \cdot)$ is computationally indistinguishable from a random permutation
- No deterministic cipher can be IND-CPA secure – modes of operation are required
  - ECB is broken: identical plaintext blocks always produce identical ciphertext blocks
  - CBC (unpredictable random IV) and CTR (unique nonce) are IND-CPA secure
- Block ciphers are built from iterated round ciphers and key schedules (Feistel networks, SPNs)
  - DES is broken: 56-bit key, 64-bit block; use AES-128 or AES-256
- 64-bit block ciphers cannot be secure PRFs due to the birthday bound
- AES-256 provides quantum-safe symmetric encryption
  - Harvest-now, decrypt-later is a real and present threat
- Misusing modes (predictable IVs, key-as-IV) breaks IND-CPA security or enables key recovery

Where do we go from here?

- We've covered a massive amount of content so far!
  - Started with classical cryptography
  - Covered stream ciphers
  - Covered block ciphers
  - Ready to do real-world (post-quantum) symmetric encryption
- What problems do we still need to solve?
  - How can we achieve message integrity?
  - Can we sign a piece of data to show that it's authentic?
  - How can we share a symmetric key over an insecure channel?
  - Can we derive keys from passwords?
  - Can we encrypt and decrypt with different keys?

For next time...

- Easter break is next week! Good opportunity to review and understand symmetric cryptography for the midterm in Week 5
  - “Knowledge cutoff” for the midterm is Week 4
  - If you can explain the material, then you understand it!
- Bloom’s taxonomy is a good guide for learning and exam prep
  - Remember, Understand, Apply, Analyze, Evaluate, and Create
- Complete some assigned reading for Week 4:
- Chapter 6 of Crypto 101
- Chapter 4 of A Graduate Course in Applied Cryptography
  - Sections 4.1 and 4.2
  - Proofs are for understanding, not for rote learning!

Questions?

Ask now, catch me after class, or email eoin@eoin.ai