Group Discussion (Integrity and reduction proofs). Take some time to think about and discuss:

(a) Informally, what would you say integrity means for a symmetric encryption scheme? Why does the IND-CPA security definition for symmetric encryption not capture integrity?

(b) What is the difference between the reduction proofs for block cipher constructions that you've seen in the exercises last week and the proof for CTR mode in the lecture?

Reading the following sections in the Boneh-Shoup book [3] might help with the problems on this exercise sheet: Sections 5.1, 5.3, and 5.4.

Problem 1 (Integrity of CBC mode). We previously saw that counter (CTR) mode encryption does not provide integrity. For example, this means that an attacker who intercepts a CTR mode encryption of the message `To:bob@gmail.com` can change the ciphertext to be an encryption of the message `To:mel@gmail.com` . In this exercise we show that the same holds for CBC mode encryption.

Suppose you intercept the following hex-encoded ciphertext, consisting of two 128-bit blocks:

`65e2654a8b52038c659360ecd8638532 b365828d548b3f742504e7203be41548`

You know that the ciphertext is a randomized CBC encryption of the plaintext `To:bob@gmail.com` using AES, where the plaintext is encoded as ASCII bytes. The first 16-byte block is the IV and the second 16-byte block carries the message.

(a) Modify the ciphertext above so that it decrypts to the message `To:mel@gmail.com`. Your answer should be the two block modified ciphertext.

(b) Does this attack work if the goal is to modify a plaintext block that is, unlike the above, *not* the first block of the encrypted message?

(c) Explain why the same attack cannot be used to purposefully modify the contents of all or multiple adjacent plaintext blocks at the same time.

Problem 2 (Padding oracle attack in SSLv3). In Exercise Sheet 3 we analysed different choices of message padding. Let us now consider one more padding scheme that is used in SSLv3, for a block cipher `E` with 16-byte (128-bit) block length. This problem is based on the POODLE attack against SSLv3 [4], however our model sidesteps some details that were necessary for the original attack.

SSLv3 Padding. Assume that the length of a message is an integer number of bytes. Let N be the number of bytes necessary to pad the message to a multiple of the block length. To ensure that $N > 0$, if the message is already a multiple of the block length, then let N be the block length (in bytes). It follows that $1 \leq N \leq 16$. Now pad the message with $N - 1$ bytes, each of an arbitrary value. Then pad the message with one last byte set to the value $N - 1$. For example, if $N = 3$, then append 2 arbitrary bytes (the values of these bytes should be ignored when decoding the padded

message) followed by the byte set to 00000010. Let us call this padding algorithm $\mathsf{SSLv3.Encode}$; we write $\tilde{m} \leftarrow \mathsf{SSLv3.Encode}(m)$ to denote that the SSLv3 padding scheme is used to pad a message m, resulting in the padded message $\tilde{m}$.

(a) Let $\mathsf{bin}_n \colon \{0, \ldots, 2^n - 1\} \to \{0,1\}^n$ be a function that on input a non-negative integer $i \leq 2^n - 1$ returns the binary n-bit representation of i, and $\mathsf{bin}_n^{-1} \colon \{0,1\}^n \to \{0, \ldots, 2^n - 1\}$ its inverse. Below we provide an example of how the algorithm $\mathsf{SSLv3.Encode}$ can be defined. Define algorithm $\mathsf{SSLv3.Decode}$ that reverses the padding, removing the padded bytes. Take care to check that the padding is valid. If the padding is not valid, then $\mathsf{SSLv3.Decode}$ should return $\perp$ (typically denoting an occurrence of error, such as an invalid input).

$\underline{\mathsf{SSLv3.Encode}(m)}$	$\underline{\mathsf{SSLv3.Decode}(\tilde{m})}$
If $\lvert m\rvert \bmod 8 \neq 0$ then return $\perp$	
$x \leftarrow \lvert m\rvert / 8$ // Number of bytes in message m	
$y \leftarrow x \bmod 16$ // Number of bytes in last block of m	
If $y > 0$ then	
$\quad N \leftarrow 16 - y$ // Number of bytes to pad	
Else	
$\quad N \leftarrow 16$ // Special case: full block of padding	
$\tilde{m} \leftarrow m \,\|\, 0^{(N-1)\cdot 8} \,\|\, \mathsf{bin}_8(N-1)$	
Return $\tilde{m}$	

(b) Let K be a block cipher key for E. Let $\mathrm{SE} = (\mathrm{KGen}, \mathrm{Enc}, \mathrm{Dec})$ be the symmetric encryption scheme implementing the CBC mode based on the block cipher E. Let m be any message such that the length of m is an integer number of bytes. Let c be a ciphertext that encrypts the SSLv3-padded message m using SE, meaning it was produced as

$$c \leftarrow\!\!\$\ \mathrm{Enc}(\mathsf{K}, \mathsf{SSLv3.Encode}(m)).$$

The implementation of SSLv3 contains a padding oracle that allows an adversary to learn whether a ciphertext encrypts a correctly padded message. We formalize it below, using oracle $\mathsf{CheckPadding}_{\mathsf{K}}(c)$ that has the key K hardcoded in it, and takes ciphertext c as input from an adversary to return a decision in $\{\mathsf{true}, \mathsf{false}\}$.

$\underline{\text{Oracle } \mathsf{CheckPadding}_{\mathsf{K}}(c)}$	$\underline{\text{Adversary } \mathcal{A}^{\mathsf{CheckPadding}}(c, i)}$
$\tilde{m} \leftarrow \mathrm{Dec}(\mathsf{K}, c)$	$c[0] \,\|\, c[1] \,\|\, \ldots \,\|\, c[t] \leftarrow c$ // s.t. $\lvert c[0]\rvert = \ldots = \lvert c[t]\rvert = 128$
$m \leftarrow \mathsf{SSLv3.Decode}(\tilde{m})$	If not $(1 \leq i \leq t-1)$ then return $\perp$
If $m \neq \perp$ then	
$\quad \mathsf{good\text{-}padding} \leftarrow \mathsf{true}$	
Else	
$\quad \mathsf{good\text{-}padding} \leftarrow \mathsf{false}$	
Return $\mathsf{good\text{-}padding}$	

Your task is to write an adversary $\mathcal{A}$ that uses this oracle to recover parts of the encrypted plaintext message. We provided a starting point for the adversary above; your job is to

complete this adversary. Adversary $\mathcal{A}$ takes ciphertext c and block index i as input. If c consists of $t+1$ blocks $c = c[0] \,\|\, c[1] \,\|\, \ldots \,\|\, c[t]$ then we require $1 \leq i \leq t-1$, meaning i denotes the index number of a *full* plaintext block (here $c[0]$ corresponds to an IV, and $c[t]$ encrypts a block that containts at least 1 byte of padding). Let $m[i]$ denote the plaintext block encrypted in the ciphertext block $c[i]$. The task of adversary $\mathcal{A}$ is to return the 4 most-significant bits of the last byte of $m[i]$.

(c) Explain the conditions under which it might be possible to extend the above attack to recover the *entire* plaintext message encrypted in c.

Problem 3 (Collisions in CTR keystream). This problem asks you to derive the collision probability for keystreams in CTR-mode. Consider the following experiment involving an adversary $\mathcal{A}$ and a block size n. For convenience, we define this experiment as a security game COLL.

```
Game COLL(A, n)
reused ← false                    // Indicates whether the same keystream block was used twice
strm[·] ← ⊥                       // Initialize an empty vector strm
A^GetKeystream()                  // Uses an oracle which on input ℓ returns ℓ n-bit keystream blocks
Return reused

Oracle GetKeystream(ℓ)            // Request 1 ≤ ℓ ≪ 2^n keystream blocks (n bits long each)
ctr ←$ {0, ..., 2^n − 1}          // Sample a random n-bit counter (defined in ℕ for simplicity)
out ← ε                           // Initialize output as empty string
For i = 0, ..., (ℓ − 1) do
  p ← (ctr + i) mod 2^n
  If strm[p] = ⊥ then             // The keystream block at position p was not used before
    strm[p] ←$ {0,1}^n            // Sample n bits uniformly at random
  Else
    reused ← true                 // The same keystream block was reused at least once
  out ← out || strm[p]            // Concatenate the keystream block at the position p to the output
Return out
```

Game COLL allows adversary $\mathcal{A}$ to call oracle $\mathsf{GetKeystream}$ in order to request multiple n-bit keystream blocks at a time, and returns a flag $\mathsf{reused} \in \{\mathsf{false}, \mathsf{true}\}$ indicating whether two keystream blocks were reused over multiple calls to this oracle. For any adversary $\mathcal{A}$ and any $n \in \mathbb{N}$, let $\Pr[\mathsf{COLL}(\mathcal{A}, n)]$ denote the probability of game $\mathsf{COLL}(\mathcal{A}, n)$ returning true over all random choices made by adversary $\mathcal{A}$ and in oracle $\mathsf{GetKeystream}$.

(a) Let $\mathcal{A}$ be an adversary that makes q queries to oracle $\mathsf{GetKeystream}$, each query made for an input $\ell = 1$. Show that $\Pr[\mathsf{COLL}(\mathcal{A}, n)] \leq \frac{q \cdot (q-1)}{2^{n+1}}$. Explain where this is used in the security proof of CTR (with uniformly random counters) on the lecture slides.

(b) Let $m \in \mathbb{N}$ be some constant. Let $\mathcal{A}$ be an adversary that makes q queries to oracle $\mathsf{GetKeystream}$, each query made for an input $\ell = m$ (meaning the adversary receives a total of $\sigma = q \cdot m$ keystream blocks). Show that $\Pr[\mathsf{COLL}(\mathcal{A}, n)] \leq \frac{q \cdot (q-1) \cdot m}{2^n}$.

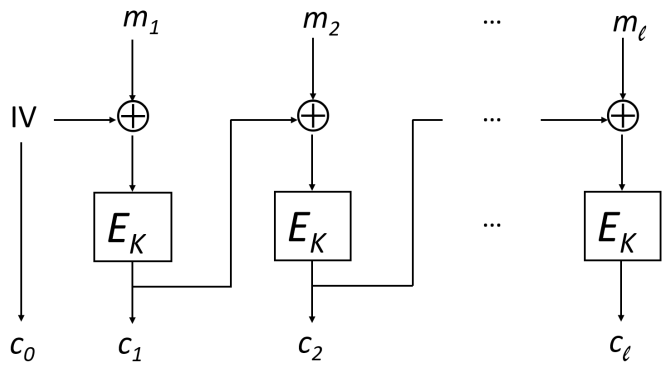

Figure 1: Illustration of CBC mode encryption.

(c) **Bonus:** Let $\mathcal{A}$ be an adversary that makes q queries to oracle $\mathsf{GetKeystream}$, and receives a total of σ keystream blocks. Each query can be made for an arbitrary $\ell \in \mathbb{N}$. Show that $\Pr[\mathsf{COLL}(\mathcal{A}, n)] \leq \frac{\sigma \cdot (q-1)}{2^n}$. Our solution replicates the result shown by Bellare et al. [1] in 1997; it is not trivial to come up with this solution on your own.

Problem 4 (Bonus: Attacks on CBC mode). Let $\mathsf{E}\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a PRP-secure block cipher. Encryption scheme $\mathrm{SE}_{\mathrm{CBC}} = (\mathrm{KGen}, \mathrm{Enc}, \mathrm{Dec})$ with keyspace $\mathcal{K} = \{0,1\}^k$ and message space[1] $\mathcal{M} = \{0,1\}^{n\ell}$ is defined as follows.

$\underline{\mathrm{KGen}()}$	$\underline{\mathrm{Enc}(\mathsf{K}, m_1 \,\|\, \ldots \,\|\, m_\ell)}$	$\underline{\mathrm{Dec}(\mathsf{K}, c_0 \,\|\, c_1 \,\|\, \ldots \,\|\, c_\ell)}$
$\mathsf{K} \leftarrow\!\!\$\ \{0,1\}^k$	$\mathsf{IV} \leftarrow\!\!\$\ \{0,1\}^n\ ;\ c_0 \leftarrow \mathsf{IV}$	For $i = 1, \ldots, \ell$ do
Return K	For $i = 1, \ldots, \ell$ do	$\quad m_i \leftarrow \mathsf{E}_{\mathsf{K}}^{-1}(c_i) \oplus c_{i-1}$
	$\quad c_i \leftarrow \mathsf{E}_{\mathsf{K}}(m_i \oplus c_{i-1})$	Return $m_1 \,\|\, \ldots \,\|\, m_\ell$
	Return $c_0 \,\|\, \ldots \,\|\, c_\ell$	

$\mathrm{SE}_{\mathrm{CBC}}$ is commonly referred to as "CBC mode" (Cipher Block Chaining mode). Figure 1 shows an illustration of the encryption algorithm of $\mathrm{SE}_{\mathrm{CBC}}$. As you have seen in the lectures, CBC mode becomes vulnerable when ciphertext blocks repeat across encryptions under a fixed key. Namely, if two ciphertext blocks c_i and c_j (for $i \neq j$ and $i, j \geq 1$) are generated with the same key K and $c_i = c_j$, this allows a collision attack: By construction,

$$c_i = c_j \text{ for } \begin{cases} c_i = \mathsf{E}_{\mathsf{K}}(m_i \oplus c_{i-1}) \\ c_j = \mathsf{E}_{\mathsf{K}}(m_j \oplus c_{j-1}) \end{cases} \iff m_i \oplus c_{i-1} = m_j \oplus c_{j-1} \iff m_i \oplus m_j = c_{i-1} \oplus c_{j-1}.$$

When such a collision occurs, an adversary that knows one (or part of one) plaintext block can use this knowledge to recover the other plaintext block. Note that this applies even if c_i and c_j are ciphertext blocks from separate encryptions of two different messages under distinct IVs.

[1] For simplicity, we restrict ourselves in this problem to messages consisting of a fixed number of full blocks. In general, CBC mode is defined for message space $\mathcal{M} = \{0,1\}^{\leq L}$, for some positive integer L, and messages may require padding before encryption.

(a) Compute the exact probability of at least one ciphertext block collision as a function of the block size n and the number of encrypted blocks $D = 2^d$ (assumed a power of 2 for simplicity). Assume that each ciphertext block of CBC is indistinguishable from a uniformly random n-bit string that is sampled independently of other blocks of the same ciphertext. This allows us to work with just the number of encrypted blocks D, regardless of how many blocks are used per ciphertext.

(b) Bound the probability in part (a) from above and below with some reasonable approximations. Using $n = 64$ (the block size of DES), plot your approximation for the lower bound as a function of d (the base-2 logarithm of the number of encrypted blocks).

(c) Compute an estimate of the expected number of collisions as a function of the number of encrypted blocks, for block size n. You may assume that collisions occur independently (so for example, a three-way collision can be counted just as three separate collisions, occurring with probability p^3 if p is the probability of one collision).

(d) This attack has been shown in practice on, for example, 3DES used in TLS for HTTPS connections. In that particular setting, it was employed to recover a secret session cookie. The attack is called "Sweet-32" [2]. Discuss the requirements for a successful collision attack in a real-world setting. What data is needed for the attack? What assumptions are useful and/or necessary?

Acknowledgements. This exercise sheet is in part inspired by (and adapted from) problems by Mark Zhandry, as well as from the book "A Graduate Course in Applied Cryptography" by Dan Boneh and Victor Shoup, and from the book "The Joy of Cryptography" by Mike Rosulek.

References

[1] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 394–403. IEEE, 1997.

[2] K. Bhargavan and G. Leurent. On the practical (in-)security of 64-bit block ciphers: Collision attacks on HTTP over TLS and OpenVPN. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 456–467, New York, NY, USA, 2016. Association for Computing Machinery.

[3] D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography.* Online, version 0.5 edition, Jan. 2020.

[4] POODLE attack. https://www.openssl.org/~bodo/ssl-poodle.pdf.