Solutions for Chapter 2 – Introduction: Confidentiality

June 15, 2023

Problems

Problem 2.1 ***Caesar Cipher***
Why is the Caesar cipher a symmetric encryption algorithm? Please describe the algorithm and the key, and give the cipher's block length.

The Caesar cipher is symmetric because sender and recipient do need to know the same information – the number of positions the characters of the alphabet must be shifted. The algorithm is replacing a character m_i by another character c_i which is located k positions after m_i according to the order of the alphabet, with 'a' following 'z' to make it cyclic. If the characters are presented by one byte, the block length ist 8 bits. The effective key length is less than 5 bits, since the key k can only have 26 different values.

Problem 2.2 ***Block Ciphers***
Is there a block cipher where the key length equals the block length?

Yes, AES-128.

Problem 2.3 ***DES***
A friend of yours comes up with a brilliant idea to increase the key length of 3DES – he proposes 5DES, where he uses five DES keys $(k_1, k_2, k_3, k_4, k_5)$ to encrypt a message block m_i as follows:

$$c_i \leftarrow \mathsf{Enc}_{k_5}(\mathsf{Dec}_{k_4}(\mathsf{Enc}_{k_3}(\mathsf{Dec}_{k_2}(\mathsf{Enc}_{k_1}(m_i)))))$$

This construction uses $5 \cdot 56 = 280$ random bytes, so he claims that the security of 5DES is better than AES-256. Can you describe a more efficient attack on the 5DES key in a known-plaintext scenario?

Similar to 3DES, there is a meet-in-the-middle attack to retrieve the 280 bits of the key with complexity 2^{168}: Given a plaintext/ciphertext pair (m_i, c_i),

- encrypt m_i three-fifths of the way using all 2^{168} possible values of $K = (k_1, k_2, k_3)$: $x_K \leftarrow Enc_{k_3}(Dec_{k_2}(Enc_{k_1}(m_i)))$

- decrypt c_i two-fifths of the way with all 2^{112} possible values of $K' = (k_4, k_5)$: $x_{K'} \leftarrow Enc_{k_4}(Dec_{k_5}(c_i))$
- store the pairs (x_K, K) and $(x_{K'}, K')$ and sort them based on the first entry.

The pairs (K, K') where $x_K = x_{K'}$ are potential candidates for the 5DES key, and can be tested with other plaintext/ciphertext blocks.

Problem 2.4 ***Block Cipher Modes***
Please have a closer look at the CFB mode in Figure 2.5. In which aspects is this mode comparable to a stream cipher, and what is different? (Hint: What happens to the plaintext if you invert a single bit in c_1?)

In CFB mode, a keystream is generated blockwise by encrypting the previous ciphertext block. Encryption of plaintext is done, as with stream ciphers, by computing the bitwise XOR of the plaintext and the keystream. Thus CFB mode is malleable: By flipping a bit in c_i, the same bit in m_i is flipped. However, this will 'destroy' the complete plaintext block m_{i+1}.

Another similarity is that you do not need any padding: The last ciphertext block may by smaller than the block length.

Problem 2.5 ***Stream Ciphers***
Suppose you want to encrypt UDP traffic with a stream cipher. What additional information will you have to transmit in an unencrypted header? (Hint: Consider the effect of packet loss on the keystream.)

A stream cipher, once initialized with the secret key, will produce a continuous keystream. To compensate the possible packet loss in UDP, each packet would have to contain the starting position in this keystream which was used for XORing the contents of the packet. As an alternative approach, the stream cipher could be initialized afresh for each UDP packet. In this case a counter or a nonce must be used for initialization to modify the contents of the keystream, and this value must be transmitted with the UDP packet. For decryption, the receiver would have to store all 'unused' bits of the keystream along with their positions, or he would have to re-initialized the stream cipher and fast forward' it to the position needed. Since this is difficult to implement, stream ciphers are typically disallowed for UDP traffic, with the exception of AEAD ciphers like ChaCha20-Poly1305.

Problem 2.6 ***RSA-PKCS#1***
Your plaintext is 30 kB in size, i.e. $30 \cdot 1024$ byte. How long is your ciphertext if you encrypt the whole plaintext with RSA-PKCS#1 and a 2048-bit RSA modulus?

2048 bit are 256 byte. So there will be many RSA-PKCS#1 ciphertexts of length 256 byte. PKCS#1 encryption encoding requires 2 byte for the prefix, 8 randomly chose byte for the padding, and 1 zero byte as a delimiter. So

each RSA-ciphertext can contain at most $256 - 11 = 245$ byte. There are $30 \cdot 1024 = 30,720$ byte plaintext to encrypt. $125 \cdot 245 = 30,625$, so we can use 125 PKCS#1 encoded packets with the maximum plaintext payload, and an additional PKCS#1 packet containing the remaining 95 byte – this last packet will then contain $256 - 3 - 95 = 158$ random padding byte. Altogether this purely RSA-PKCS#1 encrypted ciphertext will be $126 \cdot 256 = 32,256$ byte long, or 31.5 kB.

Encrypting and decrypting a whole plaintext with RSA-PKCS#1 is very slow, therefore this method is not used in practice. Instead, hybrid encryption is used.

Problem 2.7 ***CPA Attacks 1***
A Bitcoin trading platform uses a "military grade" textbook RSA encryption scheme, with an RSA modulus of length 65,536 bits, to accept trading requests where only the Bitcoin amount is encrypted. Trading requests are encoded in the most significant bytes, with a leading zero byte to guarantee correct decryption. How can you break the confidentiality of the most common trading requests, which range from BTC 0.0002 to BTC 0.02?

You can use a *chosen plaintext attack* on this encryption scheme. There are only 199 different values in this range. You can encode all of them with the given simple deterministic encoding, and encrypt all 199 encodings. You then store each value together with the ciphertext in a decryption dictionary. By looking up an intercepted ciphertext in this dictionary, you get the plaintext.

Problem 2.8 ***CPA Attacks 2***
Suppose you have a black-box CPA oracle for 3DES in ECB mode. This oracle accepts single plaintext blocks of 64 bits and outputs the ciphertexts of these blocks encrypted under an unknown key. How many requests would it take to compute a complete decryption dictionary for an unknown ciphertext c of arbitrary length, assuming that the plaintext of c only contains alphanumerical ASCII characters plus the single whitespace character 0x20? (Hint: How many different combinations of ASCII characters fit into 8 bytes?)

There are $2 \cdot 26$ alphabetical ASCII characters, 10 digits, and a white space character. With these 63 different byte values, $63^8 = 248,155,780,267,521 < (2^6)^8 = 2^{48}$ different plaintext blocks can be formed. So it will take 63^8 decryption requests to build a complete decryption dictionary.

Problem 2.9 ***Discrete Logarithms***
Figure out how the Baby-step-Giant-step algorithm to compute discrete logarithms works. Which role does the birthday paradox from statistics play here?

Let $a = g^x$ be an element of a cyclic group G of order q. For any integer $1 < m < q$ we may write $x = im + j$ for the unknown discrete logarithm x. So for some yet unknown pair (i, j), we have $g^j = ag^{-im}$. If we now choose $m = \lceil \sqrt{q} \rceil$, i.e., if we take the square root of q and choose m as the next bigger

integer, there will roughly be equally many possible choice for i and j. We now compute two lists of roughly $m/2$ elements each, and order them according to their second entry. The first list contains all values (j, g^j), and the second list all values (i, ag^{-im}). We then compare these lists for a matching second entry. If we have a match, and the first entries are j and i, we can compute the discrete logarithm as $x = im + j$.

The birthday paradox from statistics roughly states that if you take a random sample of size $m = \lceil \sqrt{q} \rceil$ from a set with cardinality q, then the probability that this sample will contain two identical values is about $1/2$.

Now if we consider the computations of g^j and ag^{-im} as random walks in G, then the joint sample has size $m/2 + m/2 = m$, and the probability that at least two elements in the sample are identical is roughly $1/2$.

Problem 2.10 ***CDH and DDH***
Why is the DDH assumption stronger *than the CDH assumption? Why is a successful attack on CDH* stronger *than a successful attack on DDH?*

The DDH assumption is stronger since it assumes more: We do not only assume that g^{ab} cannot be computer from g^a and g^b, but also that we cannot distinguish g^{ab} from any other value g^r in the given cyclic group. So the DDH assumption is easier to break, since we may break any of the two assumptions: If we can break the CDH assumption in some group G, then we also can break the DDH assumption in this group, by just computing $g^{ab} = CDH(g^a, g^b)$ and then comparing it to the third DDH value. In certain elliptic curve groups we can break only the second assumption by evaluating a bilinear map $e(g^a, g^b) = e(g, g^{ab}) = e(g, g)^{ab}$ which maps $G \times G$ into another group G' (but such maps only are known for some special groups).

A successful attack on CDH is stronger because it not only breaks CDH, but also DDH (see above).

Problem 2.11 ***ElGamal KEM***
Suppose you want to send an encrypted e-mail to five recipients, who all have public keys of the form $X_i = h^{x_i}$. Can you use the ElGamal KEM in a hybrid encryption scheme to encrypt the message for these five recipients?

No, you can't because for a hybrid encryption scheme, the symmetric encryption key must be identical for all recipients. This cannot be achieved with a KEM, since the resulting keys typically are different for each recipient.

Problem 2.12 ***Hybrid Encryption***
Your plaintext is 30 kB in size, i.e. $30 \cdot 1024$ byte. You want to send this message confidentially to five recipients, all of which have RSA public keys with a 2048-bit modulus, in a single e-mail.

1. *How long is the ciphertext in this e-mail if you encrypt the whole plaintext with RSA-PKCS#1 for each recipient?*

2. *How long is the ciphertext when you use hybrid encryption and AES-128 in CBC mode?*

Please assume that besides PKCS#1, no additional encoding or padding is necessary.

E-mail systems may not be able to handle non-ASCII data, so binary data (e.g., ciphertexts) is usually encoded. We omit this encoding here, which increases the size of the message by roughly 1/3. We also omit the minimal overhead caused by adding RFC 822 and MIME headers.

From Problem 2.6 we already know that encrypting a 30 kB plaintext directly with RSA-PKCS#1 and a 2048 bit modulus results in a ciphertext of length 31.5 kB. Sending the same plaintext message to 5 different recipient requires 5 different RSA-PKCS#1 ciphertexts, so the size of the e-mail body would be $5 \cdot 31.5 = 157.5$ kB. If the sender keeps an encrypted copy of the e-mail plaintext (which is always done in secure e-mail communication), there will be 6 different ciphertexts and the size of the e-mail will be $6 \cdot 31.5 = 189kB$.

For hybrid encryption, we only need to encrypt the 128 bit AES key separately for each of the 5 recipients plus the sender, which results in $6 \cdot 256 = 1.5 \times 1024$ byte. Encryption with AES-CBC only slightly changes the size: We only need to add 16 byte for the IV and, since we need padding for a block cipher, another 16 byte of padding. So altogether we get a hybrid ciphertext of size 31 kB plus 544 byte: 30kB AES ciphertext + 1.5 kB RSA ciphertext + 16 byte IV + 16 byte AES padding.