CS255: Cryptography and Computer Security **Winter 2007**

Assignment #1: Solutions

Problem 1 Secret sharing.

a. Suppose Alice shares a secret block cipher key, K_{AB} with Bob, and a different secret block cipher key, K_{AC} with Charlie. Describe a method for Alice to encrypt an m-block message such that it can only be decrypted with the cooperation of both Bob and Charlie. The ciphertext should only be a constant size greater than m blocks. You may assume that Bob and Charlie have a pre-established secret channel on which to communicate.

Alice can first pick a random key K to encrypt the message M, and then she encrypts the key K with both K_{AB} and K_{AC}. In other words, Alice does the following:

$$\underbrace{E_{K_{AB}}[E_{K_{AC}}(K)]}_{header} \| E_K(M)$$

b. Now, suppose Alice shares a block cipher key, K_{AB} with Bob, a block cipher key K_{AC} with Charlie, and a block cipher key K_{AD} with David. Describe a method for Alice to encrypt an m-block message such that any two of Bob, Charlie, and David can decrypt (for example, Bob and Charlie can decrypt), but none of them can decrypt the message themselves. Again, the ciphertext should only be a constant size greater than m blocks. **Hint:** Pick a random message encryption key to encrypt the message with. Then add three ciphertext blocks to the ciphertext header.

Similar to Part a, Alice does the following:

$$\underbrace{E_{K_{AB}}[E_{K_{AC}}(K)] \| E_{K_{AB}}[E_{K_{AD}}(K)] \| E_{K_{AC}}[E_{K_{AD}}(K)]}_{header} \| E_K(M)$$

c. How does your solution from part (b) scale as we increase the number of recipients? In other words, suppose Alice has a secret key with each of n recipients and wants to encrypt so that any k out of n recipients can decrypt, but any $k-1$ cannot. What would be the length of the header as a function of n and k?
Your answer shows that this solution scales poorly. We will discuss a far more efficient solution later on in the class.

It is straightforward to generalize the above scheme to the case where any k out of the n recipients can decrypt, but any $k-1$ cannot. The length of the header would be $\binom{n}{k} = n!/k!(n-k)!$. Note that we have the inequalities:

$$\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{ne}{k}\right)^k$$

This shows that the proposed solution scales poorly.

Problem 2 The movie industry wants to protect digital content distributed on DVD's. We study one possible approach. Suppose there are at most a total of n DVD players in the world (e.g. $n = 2^{32}$). We view these n players as the leaves of a binary tree of height $\log_2 n$. Each node v_i in this binary tree contains an AES key K_i. These keys are kept secret from consumers and are fixed for all time. At manufacturing time each DVD player is assigned a serial number $i \in [0, n-1]$. Consider the set S_i of $\log_2 n$ nodes along the path from the root to leaf number i in the binary tree. The manufacturer of the DVD player embeds in player number i the $\log_2 n$ keys associated with the nodes in S_i. In this way each DVD player ships with $\log_2 n$ keys embedded in it (these keys are supposedly inaccessible to consumers). A DVD movie M is encrypted as

$$DVD = \underbrace{E_{K_{root}}(K)}_{\text{header}} \Big\| \underbrace{E_K(M)}_{\text{body}}$$

where K is some random AES key called a content-key. Since all DVD players have the key K_{root} all players can decrypt the movie M. We refer to $E_{K_{root}}(K)$ as the header and $E_K(M)$ as the body. In what follows the DVD header may contain multiple ciphertexts where each ciphertext is the encryption of the content-key K under some key K_i in the binary tree.

a. Suppose the $\log_2 n$ keys embedded in DVD player number r are exposed by hackers and published on the Internet (say in a program like DeCSS). Show that when the movie industry is about to distribute a new DVD movie they can encrypt the contents of the DVD using a header of size $\log_2 n$ so that all DVD players can decrypt the movie except for player number r. In effect, the movie industry disables player number r.
Hint: the header will contain $\log_2 n$ ciphertexts where each ciphertext is the encryption of the content-key K under certain $\log_2 n$ keys from the binary tree.

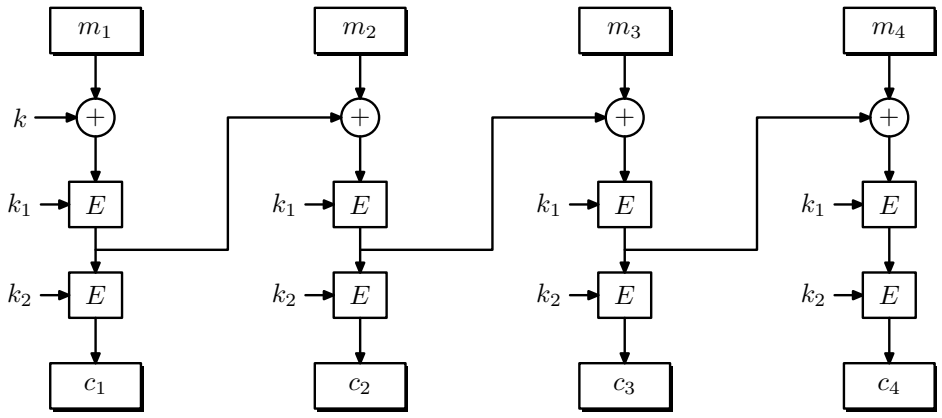
Let nodes $v_1, \ldots, v_{\log_2 n}$ be the nodes along the path from the root of the tree to leaf node number r. Since the tree is binary every node v_i for $i = 2, \ldots, \log_2 n$ has exactly one sibling. Let $u_2, u_3, \ldots, u_{\log_2 n}$ be the siblings of nodes $v_2, \ldots, v_{\log_2 n}$. Let $K_2, \ldots, K_{\log_2 n}$ be the AES keys associated with the nodes $u_2, \ldots, u_{\log_2 n}$. The header will contain the encryption of the content-key K under all keys $\mathcal{K} = \{K_2, \ldots, K_{\log_2 n}\}$. Clearly player number r cannot decrypt the movie since it does not have any of the keys in $\mathcal{K}$. However, any other player $t \neq r$ can decrypt the movie. To see this, consider the path from the root of the binary tree to leaf node t. Let u be the highest node along this path that does not appear on the path from the root to leaf node r. Then the key K_u associated with node u is in the set $\mathcal{K}$ (since the sibling of u is a on on the path to r). Furthermore, player t contains the key K_u and it can therefore obtain the content-key and then the movie. Hence, player r cannot play the movie, but all other players can.

b. Suppose the keys embedded in k DVD players $R = \{r_1, \ldots, r_k\}$ are exposed by hackers. Show that the movie industry can encrypt the contents of a new DVD using a header of size $O(k \log n)$ so that all players can decrypt the movie except for the players in R. You have just shown that all hacked players can be disabled without affecting other consumers.

For $i = 1, \ldots, k$ let U_i be the set of consecutive leaves in the range $[r_i, r_{i+1}]$ (we are assuming $r_0 = 0$ and $r_{k+1} = n$). For an internal node v in the binary tree let S_v be the set of leaves in the subtree rooted at v. We say that a set of leaves U_i is exactly covered by internal nodes $v_1, \ldots, v_b$ if $U_i = \cup_{j=1}^{b} S_{v_j}$. We show below that each set U_i can be exactly covered by a set of at most $2\log_2 n$ internal nodes. This means that to exactly cover all sets $U_0, \ldots, U_r$ we need at most $c = 2(r+1)\log_2 n$ internal nodes. Let $v_1, \ldots, v_c$ be the internal nodes needed to cover all of $U_0, \ldots, U_r$. If we encrypt the content-key using the keys K_v associated with each of these nodes then all players other than those in R can recover the content-key while the players in R cannot. This shows that using header containing at most $2(r+1)\log_2 n$ ciphertexts we can revoke all players in R without affective any of the other players.

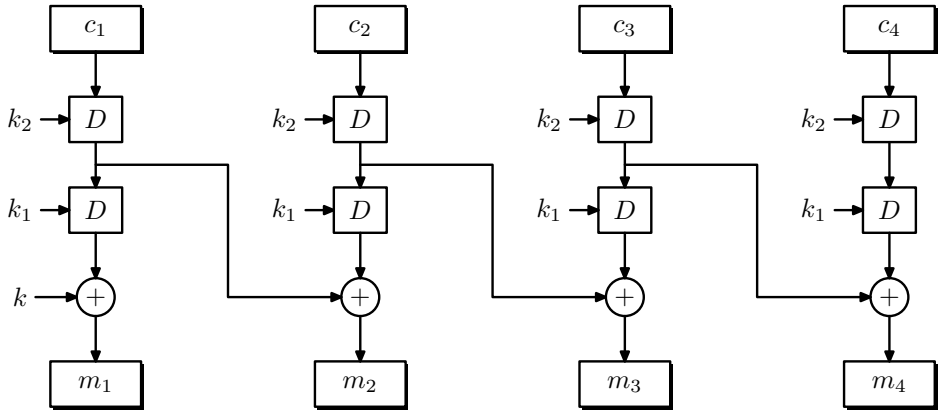
It remains to show that given a set of consecutive leaves U in some range $[a, b]$ it is possible to exactly cover U using at most $2\log_2 n$ internal nodes. Let u_1 be the highest node in the tree so that the subtree rooted at u_1 has leaf a as its left most leaf. Let v_1 be the highest node in the tree so that the subtree rooted at v_1 has leaf b as its right most leaf. Now, for $i = 2, \ldots, \log_2 n$ define u_i to be the right sibling of the parent of u_{i-1} (the right sibling of a node w is the node at the same height as w which is immediately on the right of w). Similarly, define v_i to be the left sibling of the parent of v_{i-1}. Let j be the smallest value so that $u_j = v_j$ or that u_j, v_j are adjacent siblings in the tree. Then it is easy to see that $u_1, \ldots, u_j, v_j, \ldots, v_1$ is an exact cover of $[a, b]$. This covering set contains at most $2\log_2 n$ nodes as required.

Problem 3 Let E, D be the encryption/decryption algorithms of a certain block cipher. Consider the following chaining method for double DES like encryption:



The secret key is a triple (k, k_1, k_2) where k is as long as E's block size (64 bits for DES) and k_1, k_2 are as long as E's key size (56 bits for DES). For example, when E is DES the total key size is 64+56+56 = 176 bits.

a. Describe the decryption circuit for this system.



b. Show that using two short chosen ciphertext decryption queries an attacker can recover the full key (k, k_1, k_2) in approximately the time it takes to run algorithm D 2^ℓ times (i.e. the attack running time should be $O(2^\ell \text{time}(D))$. Here ℓ is the block cipher's key-length (56 bits for DES). Your attack shows that this system can be broken much faster than exhaustive search.
Hint: Consider the two decryption queries $\langle C_1, C_2, C_3, C_4\rangle$ and $\langle C_1', C_2, C_3', C_4\rangle$ where $C_1, \ldots, C_4$ and C_1', C_3' are random ciphertext blocks.

Following the hint, query the chosen ciphertext oracle on randomly-chosen inputs $\langle C_1, C_2, C_3, C_4\rangle$ *and* $\langle C_1', C_2, C_3', C_4\rangle$ *to get* $\langle M_1, M_2, M_3, M_4\rangle$ *and* $\langle M_1', M_2', M_3', M_4'\rangle$. *¿From the decryption circuit we see*

$$\begin{aligned} M_2 &= D_{k_2}(C_1) \oplus D_{k_1}(D_{k_2}(C_2)), \\ M_2' &= D_{k_2}(C_1') \oplus D_{k_1}(D_{k_2}(C_2)), \\ M_4 &= D_{k_2}(C_3) \oplus D_{k_1}(D_{k_2}(C_4)), \\ M_4' &= D_{k_2}(C_3') \oplus D_{k_1}(D_{k_2}(C_4)). \end{aligned}$$

If we pair these equations and take xors, we find

$$\begin{aligned} M_2 \oplus M_2' &= D_{k_2}(C_1) \oplus D_{k_2}(C_1'), \\ M_4 \oplus M_4' &= D_{k_2}(C_3) \oplus D_{k_2}(C_3'). \end{aligned}$$

We iterate over possible candidate keys $K_2 \in \{0,1\}^\ell$ *for* k_2, *and check the following equations:*

$$M_2 \oplus M_2' \stackrel{?}{=} D_{K_2}(C_1) \oplus D_{K_2}(C_1'), \quad (1)$$

$$M_4 \oplus M_4' \stackrel{?}{=} D_{K_2}(C_3) \oplus D_{K_2}(C_3'). \quad (2)$$

If we find a unique K_2 *satisfying both the above, we conclude* $K_2 = k_2$ *and proceed to the next step of the attack. (We come back to the probability analysis for uniqueness below.) Otherwise, choose new random* C_i *and* C_i' *and restart.*
Now notice

$$\begin{aligned} M_2 &= D_{k_2}(C_1) \oplus D_{k_1}(D_{k_2}(C_2)), \\ M_3 &= D_{k_2}(C_2) \oplus D_{k_1}(D_{k_2}(C_3)). \end{aligned}$$

Cache $X_i := D_{k_2}(C_i)$ for $i = 1, 2, 3$. We can rewrite these equations as

$$M_i \oplus X_i = D_{k_1}(X_{i+1})$$

for $i = 1, 2$. We iterate over possible candidates K_1 for k_1, and check if the following equations hold.

$$M_i \oplus X_1 \stackrel{?}{=} D_{K_1}(X_2) \quad (3)$$

$$M_i \oplus X_2 \stackrel{?}{=} D_{K_1}(X_3) \quad (4)$$

If we find a unique K_1 satisfying equations (3) and (4), we conclude $K_1 = k_1$ and proceed to next step of the attack.

Lastly, once k_1 and k_2 have been found, we can compute $k = M_1 \oplus D_{k_1}(D_{k_2}(C_1))$ directly.

Probability Analysis and Running Time: *If we model each D_{K_2}, $K_2 \neq k_2$ as a random permutation, then $D_{K_2}(C_i)$ and $D_{K_2}(C_i')$ are independent, uniform random variables in $\{0,1\}^B$, where B is the block size of D. So $D_{K_2}(C_i) \oplus D_{K_2}(C_i')$ is uniformly random in $\{0,1\}^B$. Consequently, the probability that a particular $K_2 \neq k_2$ satisfies equation (1) is 2^{-B}. Similarly for equation (2).*

Since $\langle C_1, C_1' \rangle$ and $\langle C_3, C_3' \rangle$ are independent, the probability that a particular $K_2 \neq k_2$ satisfies both equations (1) and (2) is 2^{-2B}. So

$$\begin{aligned} &\Pr[\exists K_2 \neq k_2 \quad s.t. \quad D_{K_2}(C_i) \oplus D_{K_2}(C_i') = M_{i+1} \oplus M_{i+1}', i = 1, 3] \\ &\leq \sum_{K_2 \neq k_2} \Pr[D_{K_2}(C_i) \oplus D_{K_2}(C_i') = M_{i+1} \oplus M_{i+1}', i = 1, 3] \\ &\leq (2^\ell)(2^{-2B}) = 2^{-72} \quad \text{for } \ell = 56,\ B = 64. \end{aligned}$$

So the success probability is better than $1 - 2^{-72}$, which is awfully close to 1.

A nearly identical argument holds for the uniqueness of $K_1 = k_1$.

Since there is probability very close to 1 that k_1 and k_2 are both uniquely determined and hence found by the above algorithm, an upper bound on the expected running time of this algorithm is given by approximately one iteration of the main attack. The number of decryptions executed in the first stage (finding k_2) is $4 \cdot 2^\ell$, and the number of decryptions executed in the second stage is also $4 \cdot 2^\ell$. So the total running time is bounded above by $8 \cdot 2^\ell \cdot \text{time}(D)$, which is $O(2^\ell \cdot \text{time}(D))$.

Alternatively, we could only check equations (1) and (3) (referring to equations (2) and (4) if we have nonuniqueness from those equations) which would in practice shave off a constant factor...

Problem 4 Traitor tracing. Satellite content providers (such as satellite radio) often use hardware players to enforce specific usage policy (e.g. the content cannot be saved after it is played). Player i contains an encryption key K_i that it uses to decrypt and play the broadcast content. Now suppose some user j breaks open his player, recovers key K_j, and builds a pirate player P that decrypts and saves all broadcast content in the clear. When this pirate player P is somehow found, the content provider would like to tell whose key K_j was used to construct P (supposedly, this user j will have to answer some tough questions). Finding the key K_j that was used to build P is called *tracing* and the key K_j is called the *traitor key*.

Let $n = 32$ and suppose there are at most 2^n players in existence. Consider the following encryption system:

Setup: generate $2n$ keys:

$k_{0,0}$	$k_{1,0}$	$k_{2,0}$	$\cdots$	$k_{n-1,0}$
$k_{0,1}$	$k_{1,1}$	$k_{2,1}$	$\cdots$	$k_{n-1,1}$

Player number ℓ (for $\ell = 0, 1, \ldots, 2^n - 1$) is given key K_ℓ defined as follows. Let $b_{n-1}b_{n-2}\ldots b_0 \in \{0,1\}^n$ be the binary representation of ℓ (so that $\ell = \sum_{i=0}^{n-1} b_i 2^i$). Then key K_ℓ is

$$K_\ell = (\ k_{0,b_0},\ k_{1,b_1},\ \ldots,\ k_{n-1,b_{n-1}}\ )$$

Encrypt: to transmit content m, the content provider picks a random $i \in \{0, 1, \ldots, n-1\}$ and broadcasts via satellite the ciphertext:

$$C = (\ i,\ E(k_{i,0}, m),\ E(k_{i,1}, m)\ )$$

a. Show that all players $\ell = 0, 1, \ldots, 2^n - 1$ can decrypt the broadcast and obtain m.

For every $i \in \{1, \ldots, 32\}$, a particular player ℓ has either key $k_{i,0}$ or key $k_{i,1}$, depending on b_i – the i-th bit in the binary representation of ℓ. Given a ciphertext (i, c_0, c_1), player ℓ can obtain the message m by decrypting c_{b_i}.

b. Suppose key K_j is used to create a pirate decoder P. Show that the content provider can use P as a *black-box* and recover the index j. The content owner need not reverse engineer player P — it only uses P as a black box feeding it ciphertexts and observing the result. We are assuming that users do not collude so that P is created using knowledge of a single secret key K_j.
Hint: try to recover one bit of j at a time by feeding P a total of n carefully crafted ciphertexts $C_0, C_1, \ldots, C_{n-1}$.

Assume that the pirate decoder has the key K_j embedded. Suppose we feed the ciphertext $C_i = (\ i,\ E(k_{i,0}, m_0),\ E(k_{i,1}, m_1)\ )$ into the player, with m_0 and m_1 corresponding to observably different content. If the player plays content m_0 (resp. m_1) then it must posses the key $k_{i,0}$ (resp. $k_{i,1}$). Hence, by observing the result, we learn the i-th bit of index j. By feeding n different ciphertext we can retrieve the complete index j bit by bit.

c. Suppose a pirate is able to obtain two player keys K_i and K_j for some i, j (where $i \oplus j$ is not a power of 2). Show how the pirate can build a player P that will evade detection by your tracing algorithm from part (b). That is, your tracing algorithm will fail to output either i or j.

Side note: traitor tracing is a somewhat evil use of cryptography. Fortunately, traitor tracing systems have a number of applications unrelated to content protection.

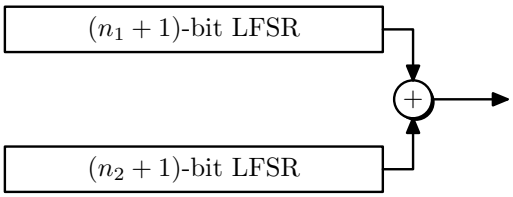
Let $i_{n-1}i_{n-2}\ldots i_0$ and $j_{n-1}j_{n-2}\ldots j_0$ be the binary representations of i and j respectively. Let a be an index where the two binary representation differ – i.e. $a \in \{0, 1, \ldots, n-1\}$ such

that $i_a \neq j_a$. We build the pirate player P by replacing the a-th key in player i with the a-th key from player j. In other words, player P has the following key embedded:

$$K_P = (k_{i_0}, k_{i_1}, \ldots, k_{i_{a-1}}, k_{j_a}, k_{i_{a+1}}, \ldots, k_{i_{n-1}})$$

Tracing algorithm from part (b) produces the index $l = i_{n-1} \ldots i_{a+1} j_a i_{a-1} \ldots i_0$. Since $i_a \neq j_a$ index l is obviously different from i, since $i \oplus j$ is not a power of 2, index l has to be different from j as well.

Problem 5 Consider the following CSS-like pseudo random generator. Assume the generator is used as a stream cipher to encrypt the contents of a DVD.



The secret key is $n = n_1 + n_2$ bits (recall that $n_1 = 16$ and $n_2 = 24$ for CSS). The top LFSR is initialized with $1||k_1$ where k_1 is the leftmost n_1 bits of the key. The bottom LFSR is initialized with $1||k_2$ where k_2 is the rightmost n_2 bits of the key. The output of the two LFSR's is xored and the resulting bitstream is the pseudo random sequence used to encrypt the plaintext. Show that an attacker who is only given the initial $2n$ bits of output of this generator can produce the rest of the output sequence in time approximately $2^{min(n_1,n_2)}$. **Hint:** Do an exhaustive search on all possible states of one LFSR and try to deduce the state of the other LFSR.

An exhaustive search attack takes time 2^n to produce the rest of the output sequence. Your attack is much faster. For CSS, your attack takes approximately 2^{16} steps which only takes a few milliseconds on a modern processor. Consequently, the resulting stream cipher is completely insecure once a few bits of a plaintext/ciphertext pair are known.

Let $K|_{2n}$ be the 2n-bit output of the keystream generator. And assume that $n_1 \leq n_2$. We proceed with the following algorithm:

Iterate through the 2^{n_1} possible states of the unknown n_1 bits of the top LFSR. For each possible state $T|_{n_1+1}$ we:

1. *Generate an additional $n_2 - n_1$ bits from the top LFSR using $T|_{n_1+1}$ as the initial state of the top LFSR. This yields the first $n_2 + 1$ bits, $T|_{n_2+1}$, of the top LFSR's output.*
2. *Compute $B|_{n_2+1} = K|_{n_2+1} \oplus T$. This yields the first $n_2 + 1$ bits, $B|_{n_2+1}$ output by the bottom LFSR.*

3. *Using $T|_{n_1+1}$ and $B|_{n_2+1}$ as the initial states of the top and bottom LFSR's respectively, generate $K'|_{2n}$ (the proposed first $2n$ bits of the keystream generator's output).*
4. *If $K'|_{2n} = K|_{2n}$ then we have a match and, with high probability, can predict the rest of the keystream. If $K'|_{2n} \neq K|_{2n}$, then we try another initial state of the top LFSR.*

This algorithm requires only an exhaustive search over the 2^{n_1} initial states of the top LFSR (the first bit of both LFSR's is always 1) and so our attack runs in time $O(2^{n_1})$.

Probability Analysis

Let us model both T_{s_1} (the top LFSR with initial state s_1) and B_{s_2} (the bottom LFSR with initial state s_2) as random permutations. Then, the first $2n$ bits produced by T_{s_1} and B_{s_2} are independent, uniform random variables in $\{0,1\}^{2n}$. Thus $T_{s_1}|_{2n} \oplus B_{s_2}|_{2n}$ is an independent, uniform random variable in $\{0,1\}^{2n}$.

Let s be the actual n bit initial state of the two LFSRs and let s' represent some n-bit string. We say that the 4th step of the algorithm fails if it succeeds for some $s' \neq s$. The probability that some $s' \neq s$ satisfies the test in step 4 of the algorithm is

$$\begin{aligned} & \Pr[\exists (s' = s'_1 || s'_2) \neq s \quad s.t. \quad T_{s'_1}|_{2n} \oplus B_{s'_2}|_{2n} = K|_{2n}] \\ \leq & \sum_{s'_1 \neq s_1} \Pr[T_{s'_1}|_{2n} \oplus B_{s'_2}|_{2n} = K|_{2n}] \\ \leq & (2^{n_1+1})(2^{-2n}) \end{aligned}$$

So, in the case of CSS, the probability of a unique solution is better that $1 - 2^{-63}$ which is very close to 1.

Problem 6 Given a cryptosystem E_k, define the randomized cryptosystem F_k by

$$F_k(M) = (E_k(R), R \oplus M),$$

where R is a random bit string of the same size as the message. That is, the output of $F_k(M)$ is the encryption of a random one-time pad along with the original message XORed with the random pad. A new independent random pad R is chosen for every encryption.

We consider two attack models. The goal of both models is to reconstruct the actual secret key k.[1]

- In the key-reconstruction chosen plaintext attack (KR-CPA), the adversary is allowed to generate strings $M_1, M_2, \ldots$ and for each M_i learn a corresponding ciphertext.
- In the key-reconstruction random plaintext attack (KR-RPA), the adversary is given random plaintext/ciphertext pairs.

Note that for the case of F_k the opponent has no control over the random pad R used in the creation of the given plaintext/ciphertext pairs. Clearly a KR-CPA attack gives the attacker more power than a KR-RPA attack. Consequently, it is harder to build cryptosystems that are secure against KR-CPA.

Prove that if E_k is secure against KR-RPA attacks then F_k is secure against $\mathsf{KR} - \mathsf{CPA}$ attacks.

[1]This is a very strong goal - one might be able to decrypt messages without ever learning k.

Hint: It is easiest to show the contrapositive. Given an algorithm A that executes a successful $\mathsf{KR-CPA}$ attack against F_k, construct an algorithm B (using A as a "subroutine") that executes a successful $\mathsf{KR-RPA}$ attack against E_k. First, define precisely what algorithm A takes as input, what queries it makes, and what it produced as output. Do the same for B. Then construct an algorithm B that runs A on a certain input and properly answers all of A's queries. Show that the output produced by A enables B to complete the $\mathsf{KR-RPA}$ attack against E_k.

Suppose we have an algorithm A which can successfully execute a KR-CPA attack against F_k, and we wish to break E_k with a KR-RPA attack. That is, we design an algorithm B which can query to obtain a random plaintext/ciphertext pairs $(R_i, E_k(R_i))$, and B wishes to find k.

The basic problem is that in order to use A as a subroutine, we must simulate F_k on arbitrary inputs M.

The algorithm B begins the execution of A, and to each query M_i which A requests to see encrypted, B obtains a new random plaintext/ciphertext pair $(R_i, E_k(R_i))$ and responds with $\tilde{F}_k(M) := (E_k(R_i), R_i \oplus M)$. Since each R_i is independent and uniformly random, the simulated $\tilde{F}_k(M)$ have exactly the same distribution as the genuine $F_k(M)$. So after some number of queries, A completes its attack successfully and returns the key k, which is subsequently returned by B.