COL759 Cryptography and Computer Security Assignment 2

Anshik Sahu(2021CS10577) & Sanya Mittal(2021CS10565)

September 2023

Contents

Part A

1: Cryptosystems secure against side-channel attacks

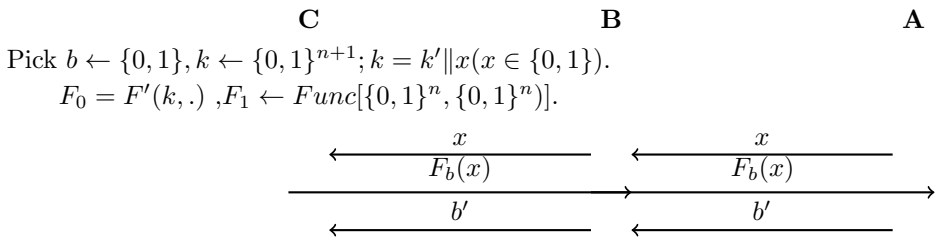
Given : F : $\{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$, a secure PRF.
Goal: To construct a secure PRF F' where leaking one bit of the key breaks PRF security.

Let the Key Space of F', K' = $\{0,1\}^{n+1}$, input space X' = $\{0,1\}^n$ and output space Y' = $\{0,1\}^n$.
Let $k \in \{0,1\}^n$ and $b \in \{0,1\}$.

$$F'(k_1 = k \| b, x) = \begin{cases} F(k,x), & \text{if } x \neq 0^n \\ b \| \ F(k,x)[1:], & \text{otherwise} \end{cases}$$

(a) Proof by Contra-positive

We prove the security of F' by showing that if there exists an adversary A that can distinguish the output of F'(k,.) for a uniformly randomly sampled key k from a truly random function f(.) with some non-negligible advantage, then there exists an adversary that can distinguish the output of F(k,.) for a uniformly randomly sampled key k from a truly random function f(.) with some non-negligible advantage.



Let Adversary A can distinguish the output of F'(k,.) for a uniformly randomly sampled key k from a truly random function f(.) with probability $\frac{1}{2} + \epsilon$.

$\Pr[B \text{ wins }] = \Pr[B \text{ wins } \wedge x = 0^n] + \Pr[B \text{ wins } \wedge x \neq 0^n]$

When $x \neq 0^n$adversary gets F(k,x),so adversary can distinguish with a probability $\frac{1}{2} + \epsilon$ as it is the same as playing against PRF Challenger of F.

And when x= 0^n, even if the adversary has no advantage in guessing b, it can still make a random guess with a probability of $\frac{1}{2}$

Thus probability of B winning in this case is atleast 1/2 (random guess).

$\Pr[B \text{ wins }] >= (1 - \frac{1}{2^n}) * (\frac{1}{2} + \epsilon) + \frac{1}{2^n} * \frac{1}{2} = \frac{1}{2} + \epsilon * (1 - \frac{1}{2^n})$

Thus the adversary has a non-negligible advantage of winning. Thus, by PRF Security of F, F' is a secure PRF.

(b) F' does not satisfy 1-leakage resilience.

Goal : To construct a PPT Adversary A that wins the Security Game for Leakage resilient PRFs with non-negligible advantage.

Challenger **Adversary**

Pick $b \leftarrow \{0,1\}, k \leftarrow \{0,1\}^{n+1}; k = k' \| x (x \in \{0,1\})$.
$F_0 = F'(k,.)$,$F_1 \leftarrow Func[\{0,1\}^n, \{0,1\}^n)]$.

Leakage Query: last bit of k ←

x →

0^n ←

$F_b(0^n)$ →

If $F_b(0^n)[0] = x$ then $b' = 0$
otherwise $b' = 1$.

b' ←

Advantage of this adversary = $\Pr[A \text{ outputs } 0 \,|\, C \text{ chooses } 0] - \Pr[A \text{ outputs } 0 \,|\, C \text{ chooses } 1]$

$\Pr[A \text{ outputs } 0 \,|\, C \text{ chooses } 0] = \Pr[0^{th} \text{ bit of y= } x |\ F_b \text{ is a PRF}] = 1$
$\Pr[A \text{ outputs } 0 \,|\, C \text{ chooses } 1] = \Pr[0^{th} \text{ bit of y= } x |\ F_b \text{ is a uniformly random function}] = \frac{1}{2}$
Advantage of A $\approx 1 - \frac{1}{2} = \frac{1}{2}$
Thus, $Pr[Awinsthe1 - leakageresiliencegame] \approx \frac{1}{2} + \frac{1}{2} * \frac{1}{2} \approx \frac{3}{4}$

2: MACs: unique queries vs non-unique queries

(a) MAC Scheme

Given a pseudorandom function $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, the Signing Algorithm samples randomness r_0 and r_1 from $\{0,1\}^{n-4}$ and a key k from $\{0,1\}^n$.

$$Sign(k,m) = a||b \text{ , where}$$

$$a = r_0||F(k, m[0:\lceil n/2\rceil]||00||r_0[0:\lfloor n/2\rfloor - 2])||F(k, m[\lceil n/2\rceil : n]||01||r_0[\lfloor n/2\rfloor - 2 : n-4])$$

$$b = r_1||F(k, m[0:\lceil n/2\rceil]||10||r_1[0:\lfloor n/2\rfloor - 2])||F(k, m[\lceil n/2\rceil : n]||11||r_1[\lfloor n/2\rfloor - 2 : n-4])$$

$$Verify(k,m,a||b) = \begin{cases} 1 & \text{if } check(k,m,a) = check(k,m,b) = 1 \\ 0 & \text{otherwise} \end{cases} \text{ , where}$$

$$check(k,m,x) = \begin{cases} 1 & \text{if} \\ & x = r||F(k, m[0:\lceil n/2\rceil]||00||r[0:\lfloor n/2\rfloor - 2])||F(k, m[\lceil n/2\rceil : n]||01||r[\lfloor n/2\rfloor - 2 : n-4]) \\ & where, r = x[0:n-4] \\ 0 & \text{otherwise} \end{cases}$$

Here, $x[a:b]$ means x from index a to b, including $x[a]$ but excluding $x[b]$.

(b) UFCMA-Unique security

To prove the security UFCMA security of the scheme, we create the following hybrid worlds: Note: In each world, the challenger also chooses a key.

- World 0:
  - Challenger receives new m.
  - Challenger chooses r_0 and r_1 from $\{0,1\}^{n-4}$.
  - Challenger sends $a||b$, where

    $a = r_0||F(k, m[0:\lceil n/2\rceil]||00||r_0[0:\lfloor n/2\rfloor - 2])||F(k, m[\lceil n/2\rceil : n]||01||r_0[\lfloor n/2\rfloor - 2 : n-4])$

    $b = r_1||F(k, m[0:\lceil n/2\rceil]||10||r_1[0:\lfloor n/2\rfloor - 2])||F(k, m[\lceil n/2\rceil : n]||11||r_1[\lfloor n/2\rfloor - 2 : n-4])$
  - A outputs b'.

    $$Pr[A \text{ outputs } 0] = p_0$$
- Hybrid World 1:
  - Challenger receives new m.
  - Challenger chooses r_0 and r_1 from $\{0,1\}^{n-4}$ and p from $\{0,1\}^n$.
  - Challenger sends $a||b$, where

    $$a = r_0||p||F(k, m[\lceil n/2\rceil : n]||01||r_0[\lfloor n/2\rfloor - 2 : n-4])$$

    $b = r_1||F(k, m[0:\lceil n/2\rceil]||10||r_1[0:\lfloor n/2\rfloor - 2])||F(k, m[\lceil n/2\rceil : n]||11||r_1[\lfloor n/2\rfloor - 2 : n-4])$
  - A outputs b'.

    $$Pr[A \text{ outputs } 0] = p_{H1}$$
- Hybrid World 2:
  - Challenger receives new m.

- Challenger chooses r_0 and r_1 from $\{0,1\}^{n-4}$ and p and q from $\{0,1\}^n$.
- Challenger sends $a||b$, where

$$a = r_0||p||q$$

$$b = r_1||F(k, m[0 : \lceil n/2 \rceil]||10||r_1[0 : \lfloor n/2 \rfloor - 2])||F(k, m[\lceil n/2 \rceil : n]||11||r_1[\lfloor n/2 \rfloor - 2 : n-4])$$

- A outputs b'.

$$Pr[A \text{ outputs } 0] = p_{H2}$$

- Hybrid World 3:
  - Challenger receives new m.
  - Challenger chooses r_0 and r_1 from $\{0,1\}^{n-4}$ and p, q and r from $\{0,1\}^n$.
  - Challenger sends $a||b$, where

$$a = r_0||p||q$$

$$b = r_1||r||F(k, m[\lceil n/2 \rceil : n]||11||r_1[\lfloor n/2 \rfloor - 2 : n-4])$$

  - A outputs b'.

$$Pr[A \text{ outputs } 0] = p_{H3}$$

- World 1:
  - Challenger receives new m.
  - Challenger chooses r_0 and r_1 from $\{0,1\}^{n-4}$ and p, q, r and s from $\{0,1\}^n$.
  - Challenger sends $a||b$, where

$$a = r_0||p||q$$

$$b = r_1||r||s$$

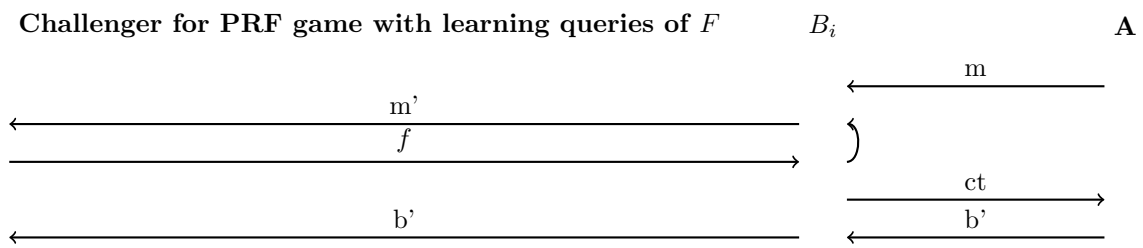
  - A outputs b'.

$$Pr[A \text{ outputs } 0] = p_1$$

Proof by contrapositive
Let there be a ppt. adversary A, which breaks the UFCMA-Unique security of the scheme. Thus,

$$|p_0 - p_1| \text{ is non-negligible.}$$

From this, we can imply at least one from $|p_0 - p_{H1}|$, $|p_{H1} - p_{H2}|$, $|p_{H2} - p_{H3}|$, $|p_{H3} - p_1|$ is non-negligible.
We can define a set of reductions B_i each of the following form:



- B_1
  - Receives m from A.
  - chooses r_0 and r_1 from $\{0,1\}^{n-4}$

- Sends learning queries for a,b,c to the challenger, where

$$a = m[\lceil n/2 \rceil : n]||01||r_0[\lfloor n/2 \rfloor - 2 : n-4]$$

$$b = m[0 : \lceil n/2 \rceil]||10||r_1[0 : \lfloor n/2 \rfloor - 2]$$

$$c = m[\lceil n/2 \rceil : n]||11||r_1[\lfloor n/2 \rfloor - 2 : n-4]$$

- Receives ct_1, ct_2 and ct_3 respectively.
- Sends query for m'=$m[0 : \lceil n/2 \rceil]||00||r_0[0 : \lfloor n/2 \rfloor - 2]$ to the challenger and receives ct_0.
- Sends $r_0||ct_0||ct_1||r_1||ct_2||ct_3$ to A and receives b'.
- Sends b' as its guess to challenger.

If $|p_0 - p_{H1}|$ is non-negligible, A can distinguish between World 0 and Hybrid World 1. If Challenger chooses $b = 0$, this is equivalent to World 0 and if challenger chooses $b = 1$, this is equivalent to Hybrid world 1. Thus by construction, B_1 breaks the security of F

- B_2
  - Receives m from A.
  - chooses r_0 and r_1 from $\{0,1\}^{n-4}$
  - chooses ct_0 from $\{0,1\}^n$
  - Sends learning queries for b,c to the challenger, where

$$b = m[0 : \lceil n/2 \rceil]||10||r_1[0 : \lfloor n/2 \rfloor - 2]$$

$$c = m[\lceil n/2 \rceil : n]||11||r_1[\lfloor n/2 \rfloor - 2 : n-4]$$

  - Receives ct_2 and ct_3 respectively.
  - Sends query for m'=$m[\lceil n/2 \rceil : n]||01||r_0[\lfloor n/2 \rfloor - 2 : n-4]$ to the challenger and receives ct_1.
  - Sends $r_0||ct_0||ct_1||r_1||ct_2||ct_3$ to A and receives b'.
  - Sends b' as its guess to challenger.

If $|p_{H1} - p_{H2}|$ is non-negligible, A can distinguish between Hybrid World 1 and Hybrid World 2. If Challenger chooses $b = 0$, this is equivalent to Hybrid World 1 and if challenger chooses $b = 1$, this is equivalent to Hybrid world 2. Thus by construction, B_2 breaks the security of F

- B_3
  - Receives m from A.
  - chooses r_0 and r_1 from $\{0,1\}^{n-4}$
  - chooses ct_0 and ct_1 from $\{0,1\}^n$
  - Sends learning queries for c to the challenger, where

$$c = m[\lceil n/2 \rceil : n]||11||r_1[\lfloor n/2 \rfloor - 2 : n-4]$$

  - Receives ct_3.
  - Sends query for m'=$m[0 : \lceil n/2 \rceil]||10||r_1[0 : \lfloor n/2 \rfloor - 2]$ to the challenger and receives ct_2.
  - Sends $r_0||ct_0||ct_1||r_1||ct_2||ct_3$ to A and receives b'.
  - Sends b' as its guess to challenger.

If $|p_{H2} - p_{H3}|$ is non-negligible, A can distinguish between Hybrid World 2 and Hybrid World 3. If Challenger chooses $b = 0$, this is equivalent to Hybrid World 2 and if challenger chooses $b = 1$, this is equivalent to Hybrid world 3. Thus by construction, B_3 breaks the security of F

- B_4
  - Receives m from A.
  - chooses r_0 and r_1 from $\{0,1\}^{n-4}$
  - chooses ct_0, ct_1 and ct_2 from $\{0,1\}^n$
  - Sends query for m'=$m[\lceil n/2 \rceil : n]||11||r_1[\lfloor n/2 \rfloor - 2 : n-4]$ to the challenger and receives ct_3.
  - Sends $r_0||ct_0||ct_1||r_1||ct_2||ct_3$ to A and receives b'.
  - Sends b' as its guess to challenger.

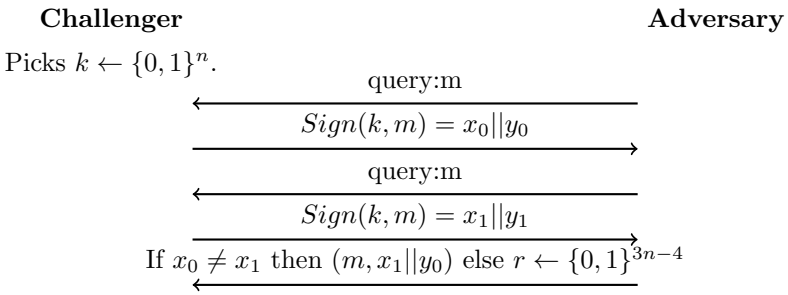
  If $|p_{H3} - p_1|$ is non-negligible, A can distinguish between Hybrid World 3 and World 1. If Challenger chooses $b = 0$, this is equivalent to Hybrid World 3 and if challenger chooses $b = 1$, this is equivalent to World 1. Thus by construction, B_4 breaks the security of F

Thus, for an adversary A which breaks the scheme, using B_1, B_2, B_2, B_3 and A, we can define a reduction B, which randomly chooses between B_1, B_2, B_2 and B_3. Thus, it has $\frac{1}{4}$ chance of choosing the correct reduction which breaks F. Hence, the probability that B breaks the security of F is non-negligible.

Thus, if this scheme is not secure, F is also not secure. Thus, from F being secure PRF, the scheme is UFCMA-unique secure.

(c) UFCMA security

To show that the scheme is not UFCMA secure, we define the following adversary A:



By the construction of the MAC scheme, if x_0 is not equal to x_1, we can create a new signature for the message m by concatenating either x_0 with y_1 or x_1 with y_0. As x_0 is not equal to x_1, these are different from $x_0||y_0$ and $x_1||y_1$. As the check function for each of x_0,x_1,y_0 and y_1 equals 1, $Verify(x_0||y_1)$ and $Verify(x_1||y_0)$ are both 1. Thus,

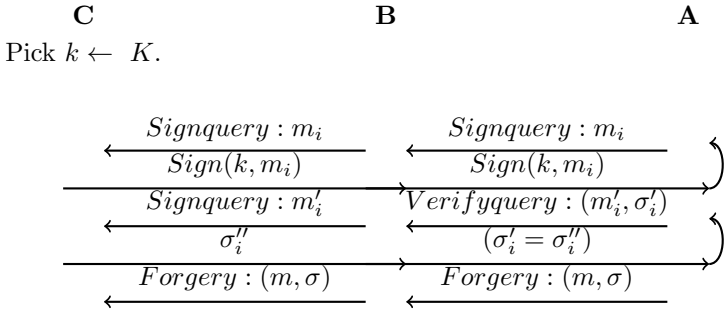
$$Pr[Awins] = Pr[\text{A wins } |x_0 \neq x_1] \times Pr[x_0 \neq x_1] + Pr[\text{A wins } |x_0 = x_1] \times Pr[x_0 = x_1]$$

$$= 1 \times (1 - \frac{1}{2^{n-4}}) + (\frac{1}{2^{3n-4}}) \times (\frac{1}{2^{n-4}})$$

Thus, $Pr[\text{A wins}] \approx 1$, given n is large.

3: A mistake in the lecture notes

Let A be an adversary that runs in time t, makes qs signing queries, qv verification queries, and wins the security game - Unforgeability under Chosen Message Attack with Verification Queries with probability 1.
We need to show that the reduction B (which plays the Unforgeability under Chosen Message Attack game without verification queries and upon receiving a verification query (m,σ) from the adversary, it sends m to the challenger, and receives a signature σ', if $\sigma = \sigma$', then it sends 1, else it sends 0) does not win with probability 1.



Key Observation: Forgery - adversary sends a message m together with a signature σ*, and wins if (m*, σ*) $\neq$ (m_i, σ_i) for all i, and Verify(m*, σ*, k) = 1.

The intuition for such an adversary A is that the adversary can send a verified (m, σ) pair but not a signed pair. Thus A can get queries verified and send them as forgery while reduction B will not be able to send that (m, σ) pair as forgery.

When Adversary A plays the game with Verification queries, it makes $|K|$ queries to challenger, for a message m_i' and its sign σ_i' computed for each key in the Key Space. Since each message has a unique valid signature there will only be one such sigma for which the Verify output is 1, A sends that (m,σ) pair as forgery and wins with probability 1. Thus $q_v = |K|$, A runs in time t and makes no sign queries.

The reduction B, for each verification query, will send m to the challenger and it will send a sign and when σ sent by challneger matches that send by adversary it outputs 1, else 0. But for each sign query made B will not be able to send it as forgery. Thus B will not be able to win this game with a probability 1. B makes q_v sign queries, runs in time t + poly(λ)$\cdot$(qv), the only extra time would be to find message in the (m,σ) pair and after receiving the sign from challenger comparing it with σ, both of which can be done in poly(security parameter) for each query.

4: Even-Mansour instantiated with a bad permutation

Given : A permutation $\pi(.)$ such that $\pi(x) = (x^{-1}\%p)$ and a PRP such that

$$PRP(x) = (\pi((k_1 + x)\%p) + k2)\%p$$

where p is a very large prime and k_1 and k_2 are random integers between 0 and 1, all sampled by challenger.
Goal: To break the security of the Even-Mansour cipher with this permutation, which is provably secure in the idealised permutation model.

$$PRP(x) = (\pi((k_1 + x)\%p) + k2)\%p$$

$$PRP(x) * ((k_1 + x)\%p) = ((\pi((k_1 + x)\%p) + k2)\%p) * ((k_1 + x)\%p)$$

$$(PRP(x) * ((k_1 + x)\%p))\%p = (((\pi((k_1 + x)\%p) + k2)\%p) * ((k_1 + x)\%p))\%p$$

As, $PRP(x) = PRP(x)\%p$,

$$(((k_1 + x)\%p) * PRP(x))\%p = ((k_1 \ + x) * PRP(x))\%p$$

Similarly,

$$(((\pi((k_1 + x)\%p) + k2)\%p) * ((k_1 + x)\%p))\%p = ((\pi((k_1 + x)\%p) + k2) * (k_1 + x))\%p$$

[using $[(a\%p * b\%p)\%p = (a * b)\%p]$]

We get,

$$(PRP(x) \times k_1 + PRP(x) \times x)\%p = ((\pi((k_1 + x)\%p) + k2) * (k_1 + x))\%p$$

$$(PRP(x) \times k_1 + PRP(x) \times x)\%p = (\pi((k_1 + x)\%p) * (k_1 + x) + k2 * (k_1 + x))\%p$$

$$(PRP(x) \times k_1 + PRP(x) \times x)\%p = ((\pi((k_1 + x)\%p) * (k_1 + x))\%p + (k2 * (k_1 + x))\%p)\%p$$

$$(PRP(x) \times k_1 + PRP(x) \times x)\%p = ((\pi((k_1 + x)\%p) * ((k_1 + x)\%p)) + (k2 * (k_1 + x))\%p)\%p$$

[using $[\pi((k_1 + x)\%p)\%p = \pi((k_1 + x)\%p)]$]

Since $\pi(x)$ is greater than or equal to 0 and strictly less than p,

$$(PRP(x) \times k_1 + PRP(x) \times x)\%p = ((\pi((k_1 + x)\%p) * ((k_1 + x)\%p))\%p + (k2 * (k_1 + x))\%p)\%p$$

[using $((k2 * (k_1 + x))\%p)\%p = (k2 * (k_1 + x))\%p)$]

$$(\pi((k_1 + x)\%p) * ((k_1 + x)\%p))\%p = 1$$

$$(PRP(x) \times k_1 + PRP(x) \times x)\%p = (1 + ((k_1 + x) * k_2)\%p)\%p$$

$$(PRP(x) \times k_1 + PRP(x) \times x)\%p = (1 + (k_1 + x) * k_2)\%p$$

- Query on x=0, $(PRP(0) \times k_1)\%p = (1 + k_1 * k2)\%p$
- Query on x=1, $(PRP(1) \times k_1 + PRP(1))\%p = (1 + k_1 * k2 + k2)\%p$
- Query on x=p-1, $(PRP(p-1) \times k_1 - PRP(p-1))\%p = (1 + k_1 * k2 - k2)\%p$
  [Since (p*PRP(p-1))%p=0]

$$(k_1 * (PRP(1) + PRP(p-1)) + PRP(1) - PRP(p-1))\%p = 2 * (1 + k_1 * k2)\%p$$

$$(k_1 * (PRP(1) + PRP(p-1) - 2 * PRP(0)) + PRP(1) - PRP(p-1))\%p = 0$$

$$(k_1 * (PRP(1) + PRP(p-1) - 2 * PRP(0)))\%p = (PRP(p-1) - PRP(1))\%p$$

Multiplying by $(PRP(1) + PRP(p-1) - 2 * PRP(0))^{-1}$ (modular inverse with respect to p) on both sides we get,

$$k_1\%p = ((PRP(p-1) - PRP(1)) * (PRP(1) + PRP(p-1) - 2 * PRP(0))^{-1})\%p$$

As k_1 is between 0 and p,

$$k1 = ((PRP(p-1) - PRP(1)) * (PRP(1) + PRP(p-1) - 2 * PRP(0))^{-1})\%p$$

Therefore, using three oracle queries, we can find k_1

Thus, if $b = 0$,

$$PRP(1) - PRP(0) = \pi(k_1 + 1) - \pi(k_1)$$

As the probability that this happens when $b = 1$ is negligible, we can use this as a check to guess b correctly.

```
def attack(oracle, pi_func, p):
    zero=oracle(0)
    one=oracle(1)
    minusone=oracle(p-1)
    denom=(one+minusone-2*zero)%p
    inv=pi_func(denom)
    l=(minusone-one)%p
    k1=(l*inv)%p
    check=(pi_func(k1+1)-pi_func(k1))%p
    if(check==(one-zero)%p):
        return 0
    else:
        return 1
```

5: 3-round Luby-Rackoff with inversion queries

```
def predict(permute, inverse_permute):
    zero=inverse_permute(b'\x00'*32)
    one=permute(zero[:16]+b'\x00'*16)
    d=one[16:]
    bxord=bytes([zero[i+16]^one[i+16] for i in range(16)])
    two=inverse_permute(one[:16]+bxord)
    cxora=bytes([one[i]^zero[i] for i in range(16)])
    if all([two[i]==cxora[i] for i in range(16)]):
        return 0
    else:
        return 1
```

If k_1, k_2 and k_3 respectively are the keys used, the attack can be described as follows:

$$\text{let } zero = F_b^{-1}(0||0) \text{ be parsed as } z_0 \text{ and } z_1$$

$$one = F_b(z_0||0) \text{ be parsed as } o_0 \text{ and } o_1$$

$$two = F_b^{-1}(o_0||z_1 \oplus o_1) \text{ be parsed as } t_0 \text{ and } t_1$$

Thus, P being the function used in each step, if $b = 0$:
Using

$$F_0(a||b) = (b \oplus P_{k_2}(a \oplus P_{k_1}(b)))||(a \oplus P_{k_1}(b) \oplus P_{k_3}(b \oplus P_{k_2}(a \oplus P_{k_1}(b))))$$

and

$$F_0^{-1}(a||b) = (b \oplus P_{k_2}(a \oplus P_{k_3}(b)))||(a \oplus P_{k_3}(b) \oplus P_{k_1}(b \oplus P_{k_2}(a \oplus P_{k_3}(b))))$$

We get,

$$z_0 = P_{k_2}(P_{k_2}(0))$$

$$z_1 = P_{k_1}(z_0) \oplus P_{k_3}(0)$$

$$o_0 = z_0 \oplus P_{k_2}(P_{k_1}(z_0))$$

$$o_1 = P_{k_1}(z_0) \oplus P_{k_3}(o_0)$$

$$t_0 = o_0 \oplus P_{k_2}(P_{k_3}(o_0) \oplus z_1 \oplus o_1)$$

From this we get $t_0 = o_0 \oplus z_0$.
As the probability of this happening with b=1 is negligible, we can use this to differentiate between F_0 anf F_1.

6: CBC mode with bad initialization

To define an attack on the CBC with key as the initialisation vector, we construct a cipher-text whose first two blocks (16 bytes each) are both ct_1. When this ciphertext is decrypted the first two message block obtained, say m_1 and m_2, would be

$$m_1 = AES^{-1}(k, ct_1) \oplus IV \text{ and } m_2 = AES^{-1}(k, ct_1) \oplus ct_1$$

where k is the key of the AES and IV is the initialization vector

Thus, $m_1 \oplus m_2 = IV \oplus ct_1$ and xorring this with ct_1 we get the Initialisation vector IV, which is the key.

Here, m[0:16] from the code is represented by m_1 and m[16:32] as m_2 and n[0:16] as ct_1.

```
def attack(ciphertext, decrypt):

    n=ciphertext[:16]*2+ciphertext[16:32]
    m=decrypt(n)
    key=bytes([m[i]^m[i+16]^n[i] for i in range(16)])
    return key
```

Failed try: We also modified the cipher-text as $(ct_1, ct_2 = 0^n, ct_3 = ct_1)$, the decryption of this would be $(AES^{-1}(k, ct_1) \oplus IV, AES^{-1}(k, ct_2) \oplus ct_1, AES^{-1}(k, ct_3) \oplus ct_2)$ which is $(AES^{-1}(k, ct_1) \oplus IV, AES^{-1}(k, 0^n) \oplus ct_1, AES^{-1}(k, ct_1) \oplus 0^n) = (AES^{-1}(k, ct_1) \oplus IV, AES^{-1}(k, 0^n) \oplus ct_1, AES^{-1}(k, ct_1))$.

Thus xor-ing the first and third component gives me the Initialisation vector IV. The issue with this design is that it is not necessary that $AES^{-1}(k, 0^n)$ would exist. And in such a situation, we would get an error. To ensure that this does not occur in our design, the new cipher-text only uses blocks from the original cipher-text.

Part B

1: Coding Problem: Padding Oracle Attack

```
from decrypt import check_padding

def attack(cipher_text):
    l=len(cipher_text)//16-1
    if(l==0):
        return []
    padding=16
    # figure out padding
    while(padding>0):
        cipher_text[padding-1]=255-cipher_text[padding-1]
        valid=check_padding(cipher_text)
        cipher_text[padding-1]=255-cipher_text[padding-1]
        if(valid==2):
            break
        padding-=1
    m=[padding]*padding
    # figure out the message
    temp=padding
    for i in range(0,l):
        new_cypher=cipher_text[i*16:(i+2)*16].copy()
        for j in range(temp,16):
            for k in range(0,j):
                new_cypher[k]=(j+1)^m[i*16+k]^cipher_text[i*16+k]
            for k in range(0,256):
                new_cypher[j]=k^(j+1)^cipher_text[i*16+j]
                valid=check_padding(new_cypher)
                if(valid==0):
                    m.append(k)
                    break
        temp=0
    return m[padding:]
```

This attack works in the following steps:

- figure out l, the length of the message.
- figure out the padding.
  This is done by changing the first block of cipher-text which denotes the randomness iteratively from right until we get a padding error as this equates changing the first block of the message and the byte at which the error first occurs is the last padded byte.
- figure out the message.
  We repeat this step l times for each block of message. For the i^{th} message block we know that $ct^i||ct^{i+1}$ is a valid encryption. Thus, for figuring out the i^{th} message block, we consider only this as the cipher-text. Let this be parsed as $ct_0||ct_1$. Therefore,

  $$ct_1 = AES(k, ct_0 \oplus m_i)$$

  If we change the first j bytes of ct_0 to $x_0, x_1, ..., x_j$, it will only be a valid cipher-text if the all first j bytes of m decryption become equal to j+1. Given that we know the first j-1 bytes of m, we can change the first j-1 bytes of decryption by setting x_k to $ct_{0_k} \oplus m_k \oplus (j+1)$ For the jth bit of decryption to become $j+1$ we must set x_j to $k \oplus (j+1) \oplus ct_{0_j}$ where $k = m_j$. As

we do not know m_j we iterate over all possible k until the decryption is valid. Using this we can find m_j. Thus by using the same method for successive values of j we can find the entire block and by successive values of i, the entire message.