DD2448 Foundations of cryptography

Homework krypto24

Persnr	Name	Email
20010410-T316	Yohan Pellerin	yohanpel@kth.se

Problem 1 (Ciphers).

Task 1.1 (1T). SOLVED
This arrangement is flawed because it relies on permutations occurring only in groups of 4 bits. However, since the substitution process involves sub-blocks of a 4x4 S-box, performing a permutation after the substitution within the same bits is essentially equivalent to just one substitution.

Task 1.2 (5T). SOLVED
The key mixing process involves modifying individual bits independently and without relation to all subkeys. Consequently, we can argue that the architecture is tantamount to having several substitutions in succession, which is essentially equivalent to just one substitution because the bits act independently unless they are within the same S-boxes. Furthermore, since the four S-boxes are independent, we can focus on each one separately. Taking the first S-box as an example, to fully compromise it, we only require the 16th possible input and output. Once these are obtained, we can predict the ciphertext for these 4 bits. This process can then be repeated for the remaining three S-boxes to completely break the cipher.

Problem 2 (Ciphers, 3T). SOLVED
Let's prove this is a symmetric cryptosystem:

$$\begin{aligned} D_k(E_k(m)) &= D_k(m \oplus k) \\ &= (m \oplus k) \oplus k^{-1} \\ &= m \oplus (k \oplus k^{-1}) \quad \text{(because G is an abelian group)} \\ &= m \oplus e \quad \text{(where } e \text{ is the identity element)} \\ &= m \end{aligned}$$

To prove that the cryptosystem achieves perfect secrecy for the encryption of a single message m $\in G$, we need to show that given an encrypted message c, an attacker gains no information about the plaintext m. As k is ramdomly chosen in G and each element has exactly one inverse and the probability of finding the message is the same of finding the inverse of k, we can say it is uniformly distributed over G just like if we don't know the cyphertext.

Problem 3 (Negligible Functions).

Task 3.1 (1T). SOLVED
We want to show that : for every constant $c > 0$, there exists a constant n_0 such that $l(n)\varepsilon(n) < \frac{1}{n^c}$ for all $n \geq n_0$.

As l(n) is polynomial, we know that, there exists a constant n_1 such as $l(n) < (|a_k| + 1)n^k$ for all $n \geq n_1$, where a_k is the coefficient of the highest degree and k the degree. $(l(n) \sim a_k * n^k)$
Moreover, as $\frac{1}{n}$ leads to 0 when n leads to inf, there exists a constant n_2, such that $\frac{1}{n} < \frac{1}{|a_k|+1}$ for all $n \geq n_2$.
Now, let's choose a $c > 0$: As $\varepsilon(n)$ is a negligible function, there exists a constant n_3 such that $\varepsilon(n) < \frac{1}{n^{c+1+k}}$ for all $n \geq n_3$. Then, if we choose $n_0 = max(n_1, n_2, n_3)$, we have $\varepsilon(n) < \frac{1}{n^{c+1+k}} < \frac{1}{n^{c+k}*(|a_k|+1)} < \frac{1}{n^c*(l(n))}$, so $l(n)\varepsilon(n) < \frac{1}{n^c}$. We just prove that $l(n)\varepsilon(n)$ is negligible in the parameter n.

Task 3.2 (1T). SOLVED
We have $\sum_{i=1}^{l(n)} X_{n,i} > 0 = \bigcup_{i=1}^{i=l(n)} X_{n,i} = 1$ as $X_{n,i}$ are binary random variables. Then, we can write $Pr(\sum_{i=1}^{l(n)} X_{n,i} > 0) \leq \sum_{i=1}^{l(n)} Pr(X_{n,i} = 1) \leq \sum_{i=1}^{l(n)} \varepsilon(n) \leq l(n)\varepsilon(n)$
However, according to the previous question $l(n)\varepsilon(n)$ is negligible in the parameter n. Then $Pr(\sum_{i=1}^{l(n)} X_{n,i} > 0)$ is also negligible in the parameter n.

Problem 4 (Non-negligible Functions).

Task 4.1 (1T). SOLVED
As l(n) is polynomial, we know that, there exists a constant n_1 such as $l(n) < (|a_k| + 1)n^k$ for all $n \geq n_1$, where a_k is the coefficient of the highest degree, and k the degree. $(l(n) \sim a_k * n^k)$
Moreover, on $[0, n_1], l(n)$ is continuous, thus bounded (named K). Then, we can write, for every n $|l(n)| < max(K, (|a_k| + 1)n^k)$
Let's set $c_1 = \frac{ln(K)}{ln(2)}$ and $c_2 = \frac{ln(K)}{ln(2)} + k$ and $c_3 = max(c_1, c_2)$.
Then, we have for every $n \geq 2, K \leq n^{c_1}$ and $(|a_k| + 1)n^k \leq n^{c_2}$, thus $|l(n)| \leq n^{c_3}$
Finally, as Δ is non-negligeable, there exists $c_4 > 0$, such as for every n_0, there exists $n \geq n_0$, such as $\Delta(n) \geq \frac{1}{n^{c_4}}$
Let's set $c = c_3 + c_4$ and chose n_0, there exists $n \geq n_0$, such as $\Delta(n) \geq \frac{1}{n^{c_4}}$. The existence of n is for every n_0 so also for 2 then, we can find a n such as $n \geq max(n_0, 2)$.
Now, we can use the property we prove on l(n) which say $|l(n)| \leq n^{c_3}$ then $\frac{1}{|l(n)|} \geq \frac{1}{n^{c_3}}$, so $\frac{\Delta(n)}{|l(n)|} \geq \frac{1}{n^{c_3+c_4}} = \frac{1}{n^c}$. Then, we have c such as for every n_0, there exists $n \geq n_0$, such as $\Delta(n) \geq \frac{1}{n^c}$, thus $\Delta(n)/l(n)$ is non-negligible in the parameter n.

Task 4.2 (1T). SOLVED
We have $\sum_{j=1}^{l(n)} X_{n,j} > 0 = \bigcup_{j=1}^{j=l(n)} X_{n,j} = 1$ as $X_{n,i}$ are binary random variables. Then, we can write $Pr(\sum_{j=1}^{l(n)} X_{n,j} > 0) \leq \sum_{j=1}^{l(n)} Pr(X_{n,j} = 1)$
Let's choose i(n) such as $Pr(X_{n,i(n)} = 1) = max(Pr(X_{n,j} = 1)$, then we have $Pr(\sum_{j=1}^{l(n)} X_{n,j} > 0) \leq Pr(X_{n,i(n)} = 1)l(n)$
Thus $Pr(X_{n,i(n)} = 1) \geq \frac{Pr(\sum_{j=1}^{l(n)} X_{n,j} > 0)}{l(n)} \geq \frac{\Delta(n)}{l(n)}$
However, according to the previous question $\frac{\Delta(n)}{l(n)}$ is non-negligible in the parameter n. Then $Pr(X_{n,i(n)} = 1)$ is also non-negligible in the parameter n.

Problem 5 (Pseudo-random Generators).

Task 5.1 (2T). SOLVED
Let's set $f_1 = g$ and $f_2 = g \oplus 1$ where g is a pseudo random function. Here, we have both functions f_1 and f_2 that are pseudo random but $f(x) = f_1(x) \oplus f_2(x) = 1$ isn't

Task 5.2 (5T). SOLVED
Let's define A_{f1}(an adversary), A_{f1} takes some samples compute them with function being tested and combines using XOR with the outputs of some new samples computed with f_2.Then, gives the new samples to the adversary A and print the same output, where A is an adversary that violates the definition of a PRG for your function f. Then, if the function being tested is f_1, the adversary A tests the function f.
Let's set $s \in \{0,1\}^n$ a seed and $u \in \{0,1\}^{4n}$ a a random string both chosen randomly, then
$\Pr[A_{f1}(f_1(s)) = 1] = \Pr[A(f(s_2)) = 1]$ where s_2 is the concatenation of s and s_1, randomly chosen in $\{0,1\}^n$, so s_2 is like a seed $\in \{0,1\}^{2n}$ randomly chosen.
However, for the other part we have, $\Pr[A_{f1}(u) = 1] = \Pr[A(u \oplus f_2(s_1)) = 1]$. If f_2 is a PRG, we can consider that $u \oplus f_2(s_1)$ is like random string $\in \{0,1\}^{4n}$ randomly chosen. But, if not we can not really say it.
If f_2 is a PRG, since A violates the PRG definition for f, then

$$|\Pr[A(f(s_2)) = 1] - \Pr[A(u \oplus f_2(s_1)) = 1]| = |\Pr[A_{f1}(f_1(s)) = 1] - \Pr[A_{f1}(u) = 1]|$$

is not negligible, then there exists an adversary that violates the PRG for f1. If f_2 is a not PRG, as one of f_1 or f_2 is, it means it is f_1 then the same construction for A_{f2} will show there exists an adversary that violates the PRG for f2.

Problem 6 (Hash Functions).

Task 6.1 (2T). SOLVED
Let's take f such as $f(x,s) = (H_{n,\alpha}(x)||H_{n,\alpha}(s))$ where || correspond to the concatenation. Such function is a polynomial time computable function.

Task 6.2 (3T). SOLVED
As H is a collision-resistant family of hash functions to prove that our construction is well collision-resistant we just need to prove that f is collision resistant. Moreover, if there isn't $(x,s) \neq (x_1,s_1)$ such that $f(x,s)) = f(x_1,s_1)$ it means that $H_{n,\alpha}(x) = H_{n,\alpha}(x_1)$ and $H_{n,\alpha}(s) = H_{n,\alpha}(s_1)$. Which is absurd as H is collision resistant. Then, f is collision resistant, thus our construction is collision resistant.

Problem 7 (Signature Schemes).

Task 7.1 (1T). SOLVED
To explain we will suppose Alice wants to digitally sign her message to Bob, the process can be explained in 3 steps.
Key Generation Process:
To begin, Alice initiates the key generation process by generating a Lamport key pair, comprising a private key and its corresponding public key.
For the private key creation, Alice employs a secure random number generator to produce 256 pairs of random numbers, each comprising 256 bits. It's crucial to note that Alice securely stores this private key, as it is not intended for sharing with anyone.
Subsequently, for the public key generation, Alice hashes each of the 512 numbers from her private key. This action results in another set of 512 numbers, each composed of 256 bits. This resultant set constitutes the public key, which Alice can freely share with others.
Signature Generation Process:

1. Alice begins by hashing her message utilizing a 256-bit cryptographic hash function, such as SHA-256, resulting in a 256-bit digest.

2. Subsequently, for each bit in the digest:
    - If the bit value is 0, Alice selects the first number from the pair of numbers in her private key.
    - If the bit value is 1, Alice selects the second number from the pair. This selection process is repeated for all 256 bits, generating a sequence of 256 numbers, which constitutes Alice's signature.

Signature Verification Process:

1. Bob begins by hashing the message using the same 256-bit cryptographic hash function, resulting in a 256-bit digest.

2. For each bit in the digest, Bob do the same thing as Alice but with the public key, it will also generate sequence of 256 numbers.

3. Bob then hashes each of the numbers in Alice's signature to obtain a 256-bit digest. If this digest matches the sequence of 256 numbers previously selected by Bob from Alice's public key, the signature is deemed valid.

Task 7.2 (3T). SOLVED
Let's set k the number of signatures computed using distinct messages, $m_0, ..., m_k$ the messages and $s_1, ..., s_k$ the corresponding 256-digest, for every $i \in [|0, k|]$, $s_i \in \{0,1\}^{256}$. The secret key (pk) is a table of dimension (256,2,256).
The attack is the following:

- For the first message, we don't know anything, we compute the hash function to get the s corresponding to the message and we fill a bit of the table pk, for every $i \in [|0, 255|]$, we can get the element corresponding, $pk[i][s_i]$ in the signature.
- Then for every message, we compute the hash function to get the s_j corresponding and for every $i \in [|0, 255|]$, if $pk[i][s_{j,i}]$ is not known we can get him in the signature and fill the table.

Now, let's prove that this attack works with probability at least 1/2 for a certain k. We will assume that for every bit in the digest the probability of having 0 is the same as having 1 (1/2) and that every bit is independent one from an other.
On the first message, we learn half of the table. Then for the second one, for every $i \in [|0, 255|]$, we have 1/2 to learn the missing part. As there are k messages after the first one (number 0) for every $i \in [|0, 255|]$, we set A_i = "$\exists j \in [|1, k|]$], such as $s_{j,i}$ be the complementary of $s_{0,i}$". $Pr(A_i) = 1 - \frac{1}{2^k}$, Then the probability we are looking for is $Pr(\bigcap_{i=0}^{n-1} A_i) = (1-\frac{1}{2^k})^n$, where n = 256. Finally, $Pr(\bigcap_{i=0}^{n-1} A_i) \geq 1/2$ equivalent to $k \geq -\frac{ln(1-\exp^{-ln(2)/n}}{ln(2)} \approx 8.5$.
Thus, with only 10 signatures computed using distinct messages, but the same secret key, suffices to recover the complete secret key with probability of at least 1/2.

Problem 8 (Definitions).

Task 8.1 (2T). SOLVED
If the cryptosystem have a deterministic encryption function, then $E_{pk}(m_0) = c_0$ and $E_{pk}(m_1) = c_1$. To know this value, the adversary just have to send (m_0, m_0) and would get as answer c_0 same for m_1. Finally, if $c = E_{pk}(m_b) = c_0$ it means b = 0 and else b = 1. Thus, a CPA secure cryptosystem cannot have a deterministic encryption function.

Task 8.2 (1T). SOLVED
As the cryptosystem is only used to encrypt randomly chosen messages, one plaintext has several encryption possible. This randomness makes it difficult for attackers to exploit deterministic patterns in the encryption scheme, thereby enhancing the security of the cryptosystem. If n is little the possibility are little, but as n grown the possibility also grown exponentially. Thus, the previous result doesn't imply that this cryptosystem is necessarily insecure.

Problem 9 (Discrete Logarithms).

Task 9.1 (1T). SOLVED
To compute the discrete logarithm in Z_q of y in the basis g, we can use the Shank's algorithm which is :

1. Set $z = g^m$
2. Compute z_i for $0 \leq i \leq q/m$
3. Find $0 \leq j \leq m$ and $0 \leq i \leq q/m$ such that $yg^j = z^i$ and output x = mi - j

Task 9.2 (2T). SOLVED
To compute the discrete logarithm problem in G_q, one possibility is to use the isomorphism f. First, find the corresponding generator and element in Z_q using f, then we can apply the Shank's algorithm mention before to find the discreet log in Z_q,(x) and then inverse the isomorphism and apply it and the element x. As the isomorphism can be computed and inverted in polynomial time, and that the shank's algorithm can also be executed in polynomial time, the previous method is also doable in polynomial time.

Task 9.3 (3T). PARTLY SOLVED
Let's prove that τ isn't a morphism. $1 \in G_q$, but $\tau(1 * 1) = \tau(1) = 1 \neq 2 = \tau(1) + \tau(1)$

Task 9.4 (1T). NOT SOLVED

Task 9.5 (Optional, 0T). NOT SOLVED

Problem 10 (Hybrid Argument, 7T). SOLVED
Let's show by recurrence on $d \in N^*$, that t_d is a PRG.
Initialisation For d =1 we have $f = t_d$, then t_d is a PRG.
Hypothesis Let's fix $d \in N^*$, we suppose that t_d is a PRG.
Now, we are going to show that if there is an adversary that violates the definition of a PRG for t_{d+1} (named A), then there exists an adversary that violates it for f.
Let's define A_f, the adversary that is going to violates the definition of a PRG for f. A_f takes an input s $\in \{0,1\}^{2n}$ and assign s_1 and s_2 such that $s_1 \parallel s_2 = s$, then, output $A(t_d(s_1) \parallel t_d(s_2))$.
Let's prove that this adversary violates the definition of a PRG for f. Let's set $s \in \{0,1\}^n$ a seed and $u \in \{0,1\}^{2n}$ a a random string both chosen randomly, then
$\Pr[A_f(f(s)) = 1] = \Pr[A(t_{d+1}(s)) = 1]$ by construction of the adversary A_f
For the other part we have, $\Pr[A_f(u) = 1] = \Pr[A(t_d(u_1) \parallel t_d(u_2)) = 1]$, where u_1 and u_2 are assigned such that $u_1 \parallel u_2 = u$. By hypothesis, we know t_d is a PRG, then, $t_d(u_1)$ and $t_d(u_2)$ are like random string in $\{0,1\}^{2^d * n}$, then $t_d(u_1) \parallel t_d(u_2)$ is like a random string in $\{0,1\}^{2^{d+1} * n}$, then $\Pr[A_f(u) = 1] = \Pr[A(t_d(u_1) \parallel t_d(u_2)) = 1] = \Pr[A(r) = 1]$ where $r \in \{0,1\}^{2^d * n}$ chosen randomly. Finally, as A is an adversary that violates the definition of a PRG for t_{d+1} , we have $|\Pr[A_f(f(s)) = 1] - \Pr[A_f(u) = 1]| = |\Pr[A(t_{d+1}(s)) = 1] - \Pr[A(u) = 1]|$ is not negligible, then $a_f$is an adversary that violates the definition of a PRG for f. Absurd as f is a PRG. Then, t_{d+1} is also a PRG.
Conclusion, for every $d \in N^*$, t_d is a PRG.

Problem 11 (Broken Cryptography).

Task 11.1 (1T). SOLVED
The ROCA vulnerability (Nemec et al. 2017), discovered in 2017, affects the implementation of RSA key generation in certain cryptographic libraries. The vulnerability stems from the predictable structure of prime numbers generated by the vulnerable libraries, allowing an attacker to factorize the RSA keys efficiently.
Heartbleed Durumeric, Paxsonand, and Kasten 2014 is a severe vulnerability in the OpenSSL cryptographic software library. Discovered in 2014, it allowed attackers to read sensitive data from the memory of the affected systems, including private keys and passwords. The flaw was due to improper bounds checking in the implementation of the TLS/DTLS heartbeat extension.
Discovered in 2015, the Logjam attack Adrian et al. 2015 exploits weaknesses in the Diffie-Hellman key exchange protocol. It allows an attacker to downgrade the security of a connection by forcing the use of weak 512-bit keys, which can be broken with contemporary computing resources.

Task 11.2 (3T). NOT SOLVED

Task 11.3 (Optional, 0T). NOT SOLVED

References

Adrian, David et al. (2015). *Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice*. URL: https://weakdh.org/imperfect-forward-secrecy.pdf.

Durumeric, Zakir, Mathias Payer Vern Paxsonand, and James Kasten (2014). *The Matter of Heartbleed*. URL: https://www.researchgate.net/publication/284575562_The_Matter_of_Heartbleed.

Nemec, Matus et al. (2017). *The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli*. URL: https://acmccs.github.io/papers/p1631-nemecA.pdf.