1

1. a) The standard components that are needed to define an encryption scheme are the message space, the key space, the encryption function, and the decryption function. The message space is the set of all possible messages that can be encrypted. The key space is the set of all possible keys that can be used to encrypt or decrypt a message. The encryption function is a function that takes a message and a key and produces an encrypted message. The decryption function is a function that takes an encrypted message and a key and produces the original message. In a perfectly secret private key encryption scheme, the message space and the key space are both finite sets. The encryption function is a one-to-one mapping from the message space to the key space. The decryption function is the inverse of the encryption function.

   b)

   - **Shift Cipher**: Gen outputs a random number k in the set {0, . . . , 25}; algorithm Enc takes a key k and a plaintext written using English letters and shifts each letter of the plaintext forward k positions (wrapping around from z to a); and algorithm Dec takes a key k and a ciphertext written using English letters and shifts every letter of the ciphertext backward k positions (this time wrapping around from a to z). The plaintext message space M is defined to be all finite strings of characters from the English alphabet (note that numbers, punctuation, or other characters are not allowed in this scheme).
   - **Mono-alphabetic substitution**: Gen outputs a random permutation of alphabet permutation a a key k. Enc takes the key k and plaintext message m writing using English letters and substituting each letter in the message corresponding to the letter in in the key. Dec takes a cipher text and decrypts it by doing the inverse mapping. Key space is all the permutations of the key i.e 26!. Message space is all possible messages in English.

2

a) Shift Cipher: A shift cipher involves replacing each letter in the message by a letter that is some fixed number of positions further along in the alphabet.We'll call this number the encryption key. It is just the length of the shift we are using. For example, upon encrypting the message "cookie" using a shift cipher with encryption key 3, we obtain the encoded message (or ciphertext): FRRNLH.

Enc(x) = (x+k) mod 26
Dec(x) = (x-k) mod 26
K = key

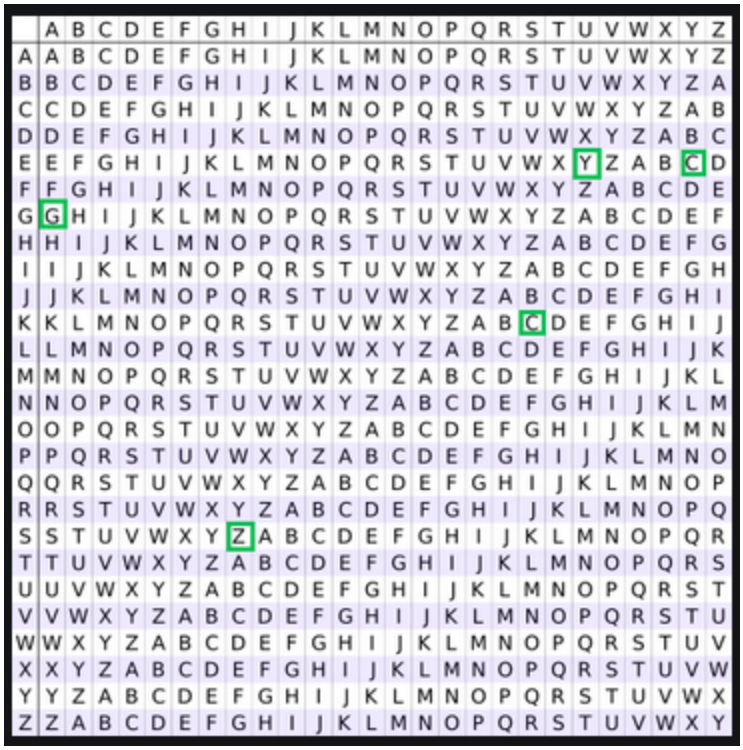
(Also try to extend the definition by adding the definition of mod)

Vignere Cipher: Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of polyalphabetic substitution. A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets. The encryption of the original text is done using the *Vigenère square or Vigenère table*.

- The table consists of the alphabets written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar Ciphers.
- At different points in the encryption process, the cipher uses a different alphabet from one of the rows.
- The alphabet used at each point depends on a repeating keyword.

Example
Input : Plaintext : GEEKSFORGEEKS
Keyword : AYUSH
Output : Ciphertext : GCYCZFMLYLEIM
For generating key, the given keyword is repeated
in a circular manner until it matches the length of
the plain text.
The keyword "AYUSH" generates the key "AYUSHAYUSHAYU"
The plain text is then encrypted using the process
explained below.

Encryption:
The first letter of the plaintext, G is paired with A, the first letter of the key. So use row G and column A of the Vigenère square, namely G. Similarly, for the second letter of the plaintext, the second letter of the key is used, the letter at row E, and column Y is C. The rest of the plaintext is enciphered in a similar fashion.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Decryption is performed by going to the row in the table corresponding to the key, finding the position of the ciphertext letter in this row, and then using the column's label as the plaintext. For example, in row A (from AYUSH), the ciphertext G appears in column G, which is the first plaintext letter. Next, we go to row Y (from AYUSH), locate the ciphertext C which is found in column E, thus E is the second plaintext letter.

To be done(Find and prove a perfectly secure answer)

A more **easy implementation** could be to visualize Vigenère algebraically by converting [A-Z] into numbers [0–25].

Encryption
The plaintext(P) and key(K) are added modulo 26.
$E_i = (P_i + K_i) \bmod 26$

Decryption
$D_i = (E_i - K_i + 26) \bmod 26$

To obtain perfect secrecy with a Vigenère cipher:

- The key must be at least as long as the message.
- Each of the key letters must be selected independently and uniformly at random from the entire ciphertext alphabet.
- The same key must not be used for more than one message.

Since in the above example key is not as long as the message the above encryption scheme is not secure.

Proof (To be done)

Breaking Shift Ciphers

For shift ciphers, performing this simple step will be sufficient to crack the cipher. Using the following steps, the plaintext can be retrieved:

Identify the most common letter in the ciphertext
Determine the shift used to make this letter decrypt to an E. For example, E is the fifth letter in the alphabet and, if J is the most common letter and is the tenth letter in the alphabet, the shift used is five
Decrypt the plaintext using the calculated shift value
If the result makes sense, terminate. Otherwise, repeat with the next most common letter in the ciphertext

This process is the most efficient method for cracking shift ciphers. However, due to the small number of possible shifts (twenty-six in English), it is also possible to try every possibility and look for a result that makes sense (maybe by checking to see if a reasonable majority of the message consists of words in the English dictionary).

Without the keyword the primary method of breaking the Vigenère cipher is known as the Kasiski test, after the Prussian major who first published it. The first stage is determining the length of the keyword.

Determining the key length

Given an enciphered message such as:

Plaintext: TOBEORNOTTOBE
Keyword: KEYKEYKEYKEYK
Ciphertext: DSZOSPXSRDSZO

Upon inspection of the ciphertext, we see that there are a few digraphs repeated, namely DS, SZ, and ZO. It is statistically unlikely that all of these would arise by random chance; the odds are that repeated digraphs in the ciphertext correspond to repetitions in the plaintext. If that is the case, the digraphs must be encoded by the same section of the key both times. Therefore, the length of the key is a factor of the distance in the text between the repetitions.

Digraph	First Position	Second Position	Distance	Factors
DS	1	10	9	3
SZ	1	10	9	3
ZO	1	10	3	3

The common factors (indeed, the only factors in this simple example) are 3 and 9. This narrows down the possibilities significantly, and the effect is even more pronounced with longer texts and keys.

Frequency analysis

Once the length of the key is known, a slightly modified frequency analysis technique can be applied. Suppose the length of the key is known to be three. Then every third letter will be encrypted with the same letter of the key. The ciphertext can be split into three segments - one for each key letter—and the procedure described for the Caesar cipher can be used.

2 b) The size of the keyspace will vary between 26^(8) to 26^(12) [To be confirmed]

2 c) Answer Link:
https://nanopdf.com/download/cis-5371-cryptography-home-assignment-1-wt-answers_pdf#

3) Answer link :
https://pub.ist.ac.at/crypto/MC18/ModernCrypto18Homework1Solution.pdf

4 a) Answer link: https://cse.iitkgp.ac.in/~somindu/focry/tutorial1-solutions.pdf

4 b) Answer link: https://crypto.stackexchange.com/questions/18956/perfectly-secure-shift-cipher

4 c) Answer link:
https://nanopdf.com/download/cis-5371-cryptography-home-assignment-1-wt-answers_pdf

4 d) http://math.umd.edu/~lcw/OneTimePad.pdf

Other solution

Perfect secrecy of one-time pad for n-bit messages

Fix arbitrary distribution over $\mathcal{M} = \{0,1\}^n$, and arbitrary $m, c \in \{0,1\}^n$

Want: $\Pr[M = m | C = c] = \Pr[M = m]$

By Bayes's Theorem:
$\Pr[M = m | C = c] = \Pr[C = c | M = m] \cdot \frac{\Pr[M=m]}{\Pr[C=c]}$
So need

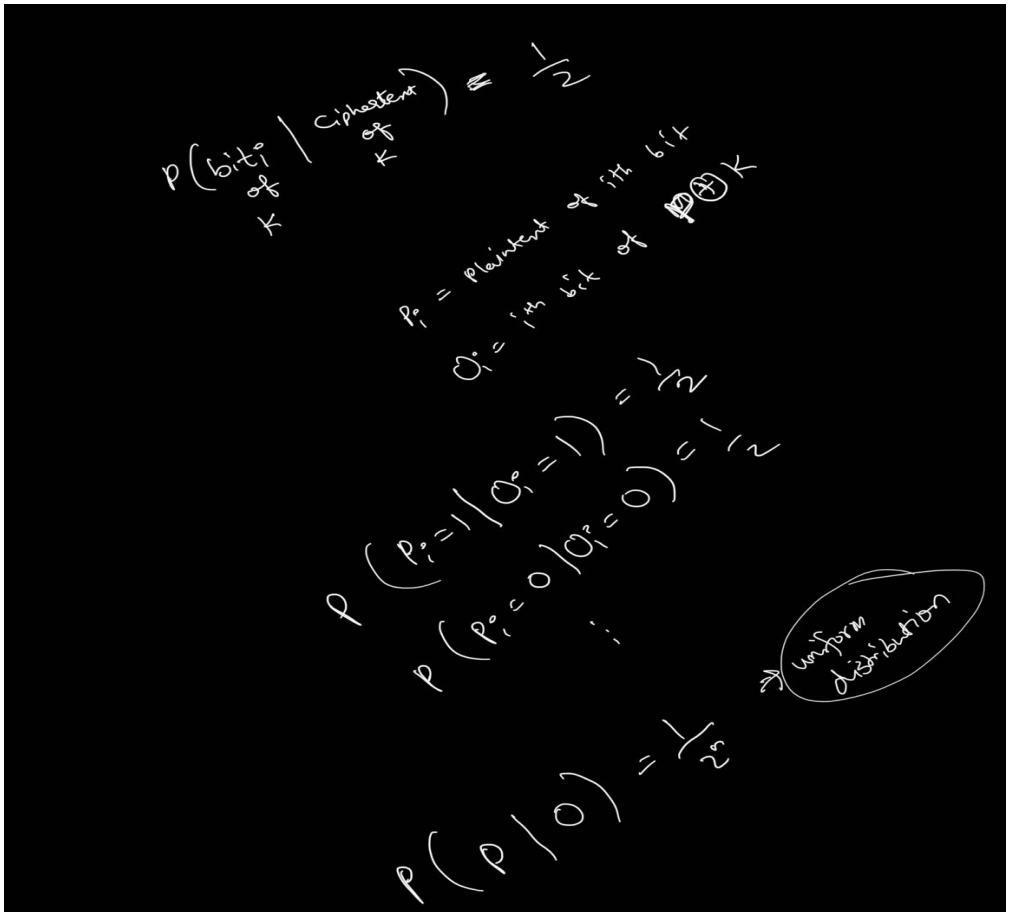
1. $\Pr[C = c | M = m] = \Pr[K = m \oplus c] = 2^{-n}$
2. $\Pr[M = m]$. DO NOT KNOW. Arbitrary Distribution!
3. $\Pr[C = c] = \Pr[c = K \oplus m] = \Pr[K = m \oplus c] = 2^{-n}]$

Hence: $\Pr[M = m | C = c] = 2^{-n} \cdot \frac{\Pr[M=m]}{2^{-n}} = \Pr[M = m]$.

4 e) 26! (key size) because key space should be greater than message space. If more than that then frequency analysis can broke down the mono-alphabetic substitution.

5 a) http://mytechclassnotes.blogspot.com/2017/01/cryptography-week-1-quiz.html
5 b) http://mytechclassnotes.blogspot.com/2017/01/cryptography-week-1-quiz.html

6)
Perfect Indistinguishability: P[M=m1 | C=c] = P[M=m0 | C=c] where m1 != m0, for all possible pairs m0,m1 belongs to M.



Without the key the receiver will also have the same things as the adversary and this scheme is perfectly secret, so generating plaintext from a ciphertext is the same as a random guess with a 2^-n probability.

7 a) https://en.wikipedia.org/wiki/Negligible_function or https://mathwiki.cs.ut.ee/asymptotics/06_the_negligible_the_noticeable_and_the_overwhelming (Different lemmas are defined) (Use these lemmas to check if they are negligible or not)
7 b)

Lemma. A positive function $\mu(n)$ is negligible if and only if for any natural number i the product $n^i\mu(n)$ converges to zero.

1. Negligible https://people.eecs.berkeley.edu/~sanjamg/classes/cs276-fall14/scribe/lec02.pdf

5.1 Proof: 2^{-n} is negligible

Main Idea: Just use the definition of a negligible function. As long as we choose a large enough n_0 given a specific c, 2^{-n} will be sufficiently small since exponentials drop off faster than polynomials.

Proof. Consider the function $\mu(n) = 2^{-n}$ and an arbitrary $c \in N$.

Then we can choose $n_0 = c^2$.

Then, for any $n \geq n_0$, we have $2^{-n} = (2^{\log_2 n})^{-\frac{n}{\log_2 n}} = n^{-\frac{n}{\log_2 n}}$

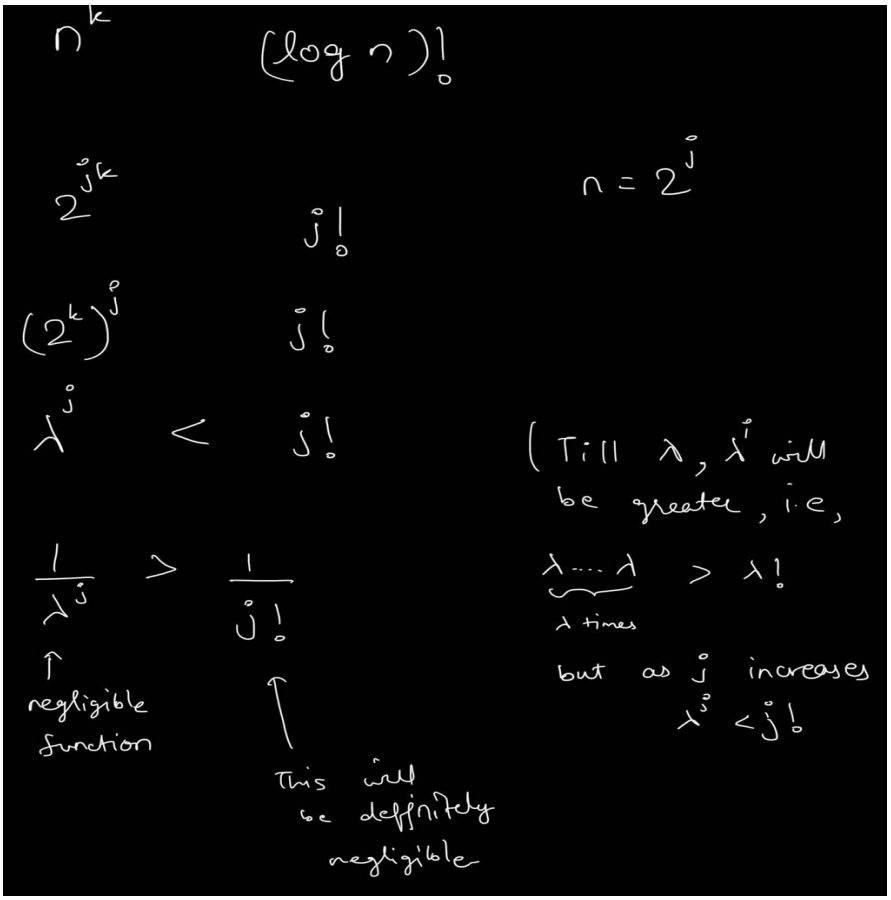
Since $n \geq n_0$, we know $\frac{n}{\log_2 n} \geq \frac{n_0}{\log_2 n_0} \geq \frac{n_0}{\sqrt{n_0}} = \sqrt{n_0} = c$ (since $n_0 = c^2$).

Thus $\mu(n) = 2^{-n} = n^{-\frac{n}{\log_2 n}} \leq n^{-c}$.

So, we have proved that for any $c \in N$, there exists $n_0 \in N$ such that for any $n \geq n_0$, $\mu(n) \leq n^{-c}$.

Hence, $\mu(n) = 2^{-n}$ is negligible. ■

2. Negligible



3. To be done
4. Not Negligible

5. Not Negligible (Confirm Please)
6. Not Negligible
7. Negligible Function
8. Not negligible

7 c)

Lemma (Some properties of negligible functions).

1. If f_1 and f_2 are negligible, then $f_1 + f_2$ is negligible. Proof ▾If, for any positive polynomial p, $p(n)f_1(n) \to 0$ and $p(n)f_2(n) \to 0$, then $\lim p(n)(f_1(n) + f_2(n)) = \lim p(n)f_1(n) + \lim p(n)f_2(n) = 0 + 0 = 0$ as well.
2. If f is negligible and $c > 0$ is a constant, then $c \cdot f$ is negligible. Proof ▾|If, for any positive polynomial p, $\lim p(n)f(n) = 0$, then $\lim cp(n)f(n) = c \lim p(n)f(n) = c \cdot 0 = 0$ as well.
3. If f is negligible and p is polynomially bounded (i.e., there is a positive polynomial r that is bigger than p for sufficiently large n), then $p \cdot f$ is negligible. Proof ▾Let f be negligible. We have to show that for any positive polynomial q, the product $q(n)p(n)f(n)$ converges to 0. Since $q(n)r(n)$ is a positive polynomial and f is negligible, we know that $r(n)p(n)f(n) \to 0$; hence also $q(n)p(n)f(n) \to 0$ since $q(n)p(n)f(n) < r(n)p(n)f(n)$ starting for some n.
4. If f is negligible, then for any $c > 0$, f^c is negligible. (This also includes cases like $f^{1/2} = \sqrt{f}$.) Proof ▾Let f be negligible. Then $f(n) \leq 1$ starting from some point (this is obvious from the definition of negligible function if you take the polynomial there to be the constant polynomial 1). So if $c \geq 1$ then the validity of the claim is rather obvious, since $f(n)^c \leq f(n)$ if $f(n) \leq 1$. It remains to prove that f^c is negligible if $c < 1$. Let p be a positive polynomial. We have to show that there exists n_0 such that $f(n)^c \leq 1/p(n)$ for all $n \geq n_0$. Let d be a natural number such that $cd \geq 1$. Then f^{cd} is negligible, as we just proved. Since f^{cd} is negligible and $p(n)^d$ is a positive polynomial, there exists m_0 such that $f(n)^{cd} \leq 1/p(n)^d$ for all $n \geq m_0$. Taking the d-th root of the inequality reveals that we can let $n_0 = m_0$. Q.E.D.

1) https://mathwiki.cs.ut.ee/asymptotics/06_the_negligible_the_noticeable_and_the_overwhelming

Proof. We need to show that for any $c \in N$, we can find n_0 such that $\forall n \geq n_0$, $h(n) \leq n^{-c}$. So, consider an arbitrary $c \in N$.

Then, since $c + 1 \in N$, and since f and g are negligible, there exists n_f and n_g such that: $\forall n \geq n_f$, $f(n) \leq n^{-(c+1)}$ and $\forall n \geq n_g$, $g(n) \leq n^{-(c+1)}$.

2) Choose $n_0 = \max(n_f, n_g, 2)$. Then, for any $n \geq n_0$, we have

Now consider h(n) = f(n)*g(n)
Hence f(n)*g(n) <= n^-(c+1)*n^-(c+1)
h(n) <= n^-2*(c+1)
Hence it is a negligible function
https://people.eecs.berkeley.edu/~sanjamg/classes/cs276-fall14/scribe/lec02.pdf (follow similar steps)

3) It is not a negligible function, one counter example is let f(n) = n/2^(n) and g(n) = 1/(2^n), then h(n) = n which is not a negligible function.

8 a) A pseudorandom generator G is an efficient, deterministic algorithm for transforming a short, uniform string (called a seed) into a longer, “uniform-looking" (or “pseudorandom") output string. Stated differently, a pseudorandom generator uses a small amount of

true randomness in order to generate a large amount of pseudorandomness.. This is useful whenever a large number of random(-looking) bits are needed, since generating true random bits is often difficult and slow.

Let G be an efficiently computable function that maps strings of length n to outputs of length l(n)> n, and define Dist(n) to be the distribution on l(n)-bit strings obtained by choosing a uniform s belongs to {0,1}^(n) and outputting G(s).
Then G is a pseudorandom generator if and only if the distribution Dist(n) is pseudorandom.
G is a pseudorandom generator if no efficient distinguisher can detect whether it is given a string output by G or a string chosen uniformly at random. This is formalized by requiring that every efficient algorithm outputs 1 with almost the same probability when given G(s) (for uniform seed s) or a uniform string. We obtain a definition in the asymptotic setting by letting the security parameter n determine the length of the seed, and insisting that G be computable by an efficient algorithm. As a technicality, we also require that G's output be longer than its input; otherwise, G is not very useful or interesting.

DEFINITION 3.14 *Let G be a deterministic polynomial-time algorithm such that for any n and any input $s \in \{0,1\}^n$, the result $G(s)$ is a string of length $\ell(n)$. G is a* pseudorandom generator *if the following conditions hold:*

1. **(Expansion.)** *For every n it holds that $\ell(n) > n$.*
2. **(Pseudorandomness.)** *For any* PPT *algorithm D, there is a negligible function* negl *such that*

$$\left| \Pr[D(G(s)) = 1] - \Pr[D(r) = 1] \right| \leq \mathsf{negl}(n),$$

where the first probability is taken over uniform choice of $s \in \{0,1\}^n$ and the randomness of D, and the second probability is taken over uniform choice of $r \in \{0,1\}^{\ell(n)}$ and the randomness of D.

We call $\ell(n)$ the expansion factor *of G.*

8 b) https://thisismyclassnotes.blogspot.com/2015/07/cryptography-computational-secrecy.html

9) http://www.crypto-it.net/eng/theory/one-way-function.html
https://www.csa.iisc.ac.in/~arpita/Cryptography16/Tutorial3.pdf

a) Not a one-way function
b) One way function (Please confirm)
c) One way function
d) One way function

10) Private key encryption scheme (Book pg-67)
   a) True
   b) False (Proof is in the book (reduction method)) (Pg-70)

11) To be done
We can conclude perfect PRGS do not exist https://www.cs.umd.edu/~jkatz/gradcrypto/s07/hw1.pdf

12) Answer link:
https://thisismyclassnotes.blogspot.com/2015/07/cryptography-computational-secrecy.html

13) https://thisismyclassnotes.blogspot.com/2015/07/cryptography-computational-secrecy.html
   https://niyander.blogspot.com/2021/10/cryptography-week-3-quiz-answer.html
(Reasoning is not clear)
a,c,d
B would not be an answer as we know the key (0..0) and therefore anyone can generate the random number.

14) For a pseudorandom generator certain properties should hold true:
   1) l(n) > n for any input of length n
   2) The pseudorandom string should be equivalent to the string chosen at uniform (use the probability notation)
   In this way show that the above two relations are equivalent

15) https://thisismyclassnotes.blogspot.com/2015/07/cryptography-key-encription.html
a) True
If a private-key encryption scheme is CPA-secure then no PPT adversary can distinguish betweeen different messages that are encrypted and thus we can say that this scheme remains indistinguishable (Try to add mathematical notations and other shits in real answer)

In CPA secure,
Enc(m) = < r, $F_k(r)$ ^ m >
Dec(c) = c ^ $F_k(r)$
Here $F_k(.)$ is a pseudo random function due to which it cannot be distinguished from a truly random function computationally. r is also a random no, therefore $F_k(r)$ is also random and hence when we xor it with m it is computationally truly random and indistinguishable.

b) False
c) True

16) d)

17) One way permutation: https://www.cs.princeton.edu/courses/archive/fall07/cos433/lec9.pdf

Invertible pseudorandom generator: https://eprint.iacr.org/2017/477.pdf

To be done!!

18) a)

19) a)

Probabilistic encryption is the use of randomness in an encryption algorithm, so that when encrypting the same message several times it will, in general, yield different ciphertexts. The term "probabilistic encryption" is typically used in reference to public key encryption algorithms; however various symmetric key encryption algorithms achieve a similar property

Probabilistic encryption is particularly important when using public key cryptography. Suppose that the adversary observes a ciphertext, and suspects that the plaintext is either "YES" or "NO", or has a hunch that the plaintext might be "ATTACK AT CALAIS". When a deterministic encryption algorithm is used, the adversary can simply try encrypting each of his guesses under the recipient's public key, and compare each result to the target ciphertext. To combat this attack, public key encryption schemes must incorporate an element of randomness, ensuring that each plaintext maps into one of a large number of possible ciphertexts.

DEFINITION 3.22 *Private-key encryption scheme* Π *has* indistinguishable multiple encryptions under a chosen-plaintext attack, *or is* CPA-secure for multiple encryptions, *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *there is a negligible function* negl *such that*

$$\Pr\left[\mathsf{PrivK}^{\mathsf{LR\text{-}cpa}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n),$$

---

where the probability is taken over the randomness used by $\mathcal{A}$ *and the randomness used in the experiment.*

Note that an attacker given access to $\mathsf{LR}_{k,b}$ can simulate access to an encryption oracle: to obtain the encryption of a message m, the attacker simply queries $\mathsf{LR}_{k,b}(m, m)$. Given this observation, it is immediate that if Π is CPA-secure for multiple encryptions then it is also CPA-secure. It should also be clear that if Π is CPA-secure for multiple encryptions then it has indistinguishable multiple encryptions in the presence of an eavesdropper. In other words, Definition 3.22 is at least as strong as Definitions 3.18 and 3.21.

It turns out that CPA-security is *equivalent* to CPA-security for multiple encryptions. (This stands in contrast to the case of eavesdropping adversaries; cf. Proposition 3.19.) We state the following without proof; an analogous result in the public-key setting is proved in Section 12.2.2.

19 b)

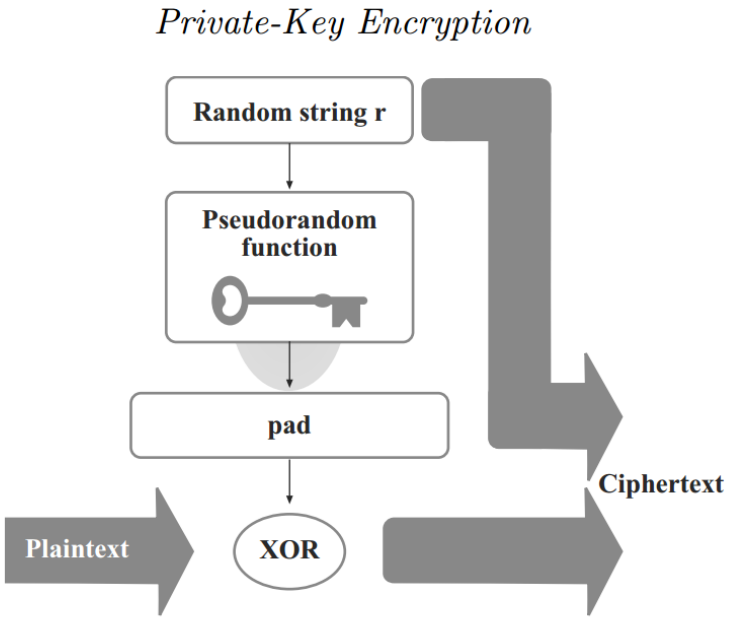
Pseudorandom functions (PRFs) generalize the notion of pseudorandom generators. Now, instead of considering "random-looking" *strings* we consider "random-looking" *functions*. As in our earlier discussion of pseudorandomness, it does not make much sense to say that any *fixed* function $f : \{0,1\}^* \to \{0,1\}^*$ is pseudorandom (in the same way it makes little sense to say that any fixed function is random). Instead, we must consider the pseudorandomness of a *distribution* on functions. Such a distribution is induced naturally by considering *keyed functions*, defined next.

Formal Definition:

DEFINITION 3.24 *An efficient, length preserving, keyed function* $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *is a* pseudorandom function *if for all probabilistic polynomial-time distinguishers* D*, there is a negligible function* negl *such that:*

$$\left| \Pr[D^{F_k(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1] \right| \leq \mathsf{negl}(n),$$

where the first probability is taken over uniform choice of $k \in \{0,1\}^n$ *and the randomness of* D*, and the second probability is taken over uniform choice of* $f \in \mathsf{Func}_n$ *and the randomness of* D*.*



GURE 3.3: Encryption with a pseudorandom function.

Our CPA-secure construction uses *randomized* encryption. Specifically, we encrypt by applying a pseudorandom function to a *random value* $r \in \{0,1\}^n$ and XORing the output with the plaintext; the ciphertext includes both the result as well as r (to enable the receiver to decrypt). See Figure 3.3 and Construction 3.28. Encryption can again be viewed as XORing a pseudorandom pad with the plaintext (just like in the "pseudo-"one-time pad), with the major difference being the fact that here a *fresh* pseudorandom pad—that depends on r—is used each time a message is encrypted. (The pseudorandom pad is only "fresh" if the pseudorandom function is applied to a "fresh" value r on which it has never been evaluated before. The proof below shows that with overwhelming probability this is always the case.)

Proof is long!!!(Refer page 86 of the book)

CONSTRUCTION 3.28

Let F be a pseudorandom function. Define a fixed-length, private-key encryption scheme for messages of length n as follows:

- Gen: on input 1^n, choose uniform $k \in \{0,1\}^n$ and output it.
- Enc: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^n$, choose uniform $r \in \{0,1\}^n$ and output the ciphertext

$$c := \langle r,\ F_k(r) \oplus m \rangle.$$

- Dec: on input a key $k \in \{0,1\}^n$ and a ciphertext $c = \langle r, s \rangle$, output the message

$$m := F_k(r) \oplus s.$$

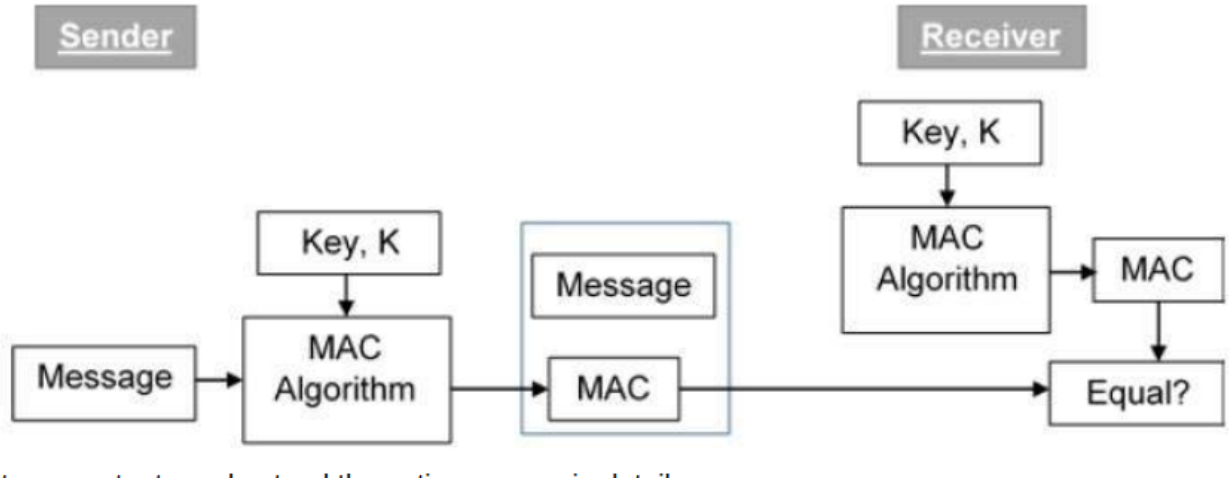
20 a) Message Integrity: The validity of a transmitted message. Message integrity means that a message has not been tampered with or altered.
Message Authentication Code:

Message Authentication Code (MAC)

MAC algorithm is a symmetric key cryptographic technique to provide message authentication. For establishing MAC process, the sender and receiver share a symmetric key K.

Essentially, a MAC is an encrypted checksum generated on the underlying message that is sent along with a message to ensure message authentication.

The process of using MAC for authentication is depicted in the following illustration −



Fixed Length MAC construction:

CONSTRUCTION 4.5

Let F be a (length preserving) pseudorandom function. Define a fixed-length MAC for messages of length n as follows:

- Mac: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^n$, output the tag $t := F_k(m)$.
- Vrfy: on input a key $k \in \{0,1\}^n$, a message $m \in \{0,1\}^n$, and a tag $t \in \{0,1\}^n$, output 1 if and only if $t \stackrel{?}{=} F_k(m)$.

Pseudorandom functions are a natural tool for constructing secure message authentication codes. Intuitively, if the tag t is obtained by applying a pseudorandom function to the message m, then forging a tag on a previously unauthenticated message requires the adversary to correctly guess the value of the pseudorandom function at a "new" input point. The probability of guessing the value of a *random* function on a new point is 2^{-n} (if the output length of the function is n). The probability of guessing such a value for a *pseudorandom* function can be only negligibly greater.

The above idea, shown in Construction 4.5, gives a secure *fixed-length* MAC for *short* messages. In Section 4.3.2, we show how to extend this to handle messages of arbitrary length. We explore more efficient constructions of MACs for arbitrary-length messages in Sections 4.4, 4.5, and 6.3.2.

Variable Length MAC:

CONSTRUCTION 4.7

Let $\Pi' = (\mathsf{Mac}', \mathsf{Vrfy}')$ be a fixed-length MAC for messages of length n. Define a MAC as follows:

- Mac: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^*$ of (nonzero) length $\ell < 2^{n/4}$, parse m as d blocks $m_1, \ldots, m_d$, each of length $n/4$. (The final block is padded with 0s if necessary.) Choose a uniform message identifier $r \in \{0,1\}^{n/4}$.
  For $i = 1, \ldots, d$, compute $t_i \leftarrow \mathsf{Mac}'_k(r \| \ell \| i \| m_i)$, where i, ℓ are encoded as strings of length $n/4$.[†] Output the tag $t := \langle r, t_1, \ldots, t_d \rangle$.
- Vrfy: on input a key $k \in \{0,1\}^n$, a message $m \in \{0,1\}^*$ of nonzero length $\ell < 2^{n/4}$, and a tag $t = \langle r, t_1, \ldots, t_{d'} \rangle$, parse m as d blocks $m_1, \ldots, m_d$, each of length $n/4$. (The final block is padded with 0s if necessary.) Output 1 if and only if $d' = d$ and $\mathsf{Vrfy}'_k(r \| \ell \| i \| m_i, t_i) = 1$ for $1 \le i \le d$.

[†] Note that i and ℓ can be encoded using $n/4$ bits because $i, \ell < 2^{n/4}$.

To define this construction just make story around different components.Proof is long af!

20 b)

CBC-MAC was one of the first message authentication codes to be standardized. A basic version of CBC-MAC, secure when authenticating messages of any *fixed* length, is given as Construction 4.9. (See also Figure 4.1.) We caution that this basic scheme is *not* secure in the general case when messages of different lengths may be authenticated; see further discussion below.

CONSTRUCTION 4.9

Let F be a pseudorandom function, and fix a length function $\ell(n) > 0$. The basic CBC-MAC construction is as follows:

- Mac: on input a key $k \in \{0,1\}^n$ and a message m of length $\ell(n) \cdot n$, do the following (set $\ell = \ell(n)$ in what follows):

  1. Parse m as $m = m_1, \ldots, m_\ell$ where each m_i is of length n.
  2. Set $t_0 := 0^n$. Then, for $i = 1$ to ℓ, set $t_i := F_k(t_{i-1} \oplus m_i)$.

  Output t_ℓ as the tag.

- Vrfy: on input a key $k \in \{0,1\}^n$, a message m, and a tag t, do: If m is not of length $\ell(n) \cdot n$ then output 0. Otherwise, output 1 if and only if $t \stackrel{?}{=} \mathsf{Mac}_k(m)$.

4 Failure of basic CBC-MAC for arbitrary length messages

Following simple MAC forgery attack demonstrates why basic CBC-MAC is not secure against arbitrary length messages:-

- Choose an arbitrary one block message, $m \in X$.
- Request tag $t = F_k(m)$ for message m.
- Output t as the forged tag for new 2 block message, $m' = (m||t \oplus m)$. Clearly, $CBC_k(m, t \oplus m) = F_k(F_k(m) \oplus (t \oplus m)) = F_k(t \oplus t \oplus m) = F_k(m) = t$

As a valid forged message tag pair, (m', t) could be generated, we conclude that basic CBC-MAC is not secure against arbitrary length messages.

https://www.csa.iisc.ac.in/~arpita/Cryptography15/CT3.pdf

20 c) To be done

21 a)

Hash function –

A hash function H is a transformation that takes a ***variable sized input m*** and returns a ***fixed size string*** called a ***hash value (h = H(m))***. Hash functions chosen in cryptography must satisfy the following requirements:

- The input is of variable length,
- The output has a fixed length,
- H(x) is relatively easy to compute for any given x,
- H(x) is one-way,
- H(x) is collision-free.

A hash function H is said to be one-way if it is hard to invert, where "hard to invert" means that given a hash value h, it is computationally infeasible to find some input x such that ***H(x) = h***.

If, given a message x, it is computationally infeasible to find a message y not equal to x such that ***H(x) = H(y)*** then H is said to be a weakly collision-free hash function.

A strongly collision-free hash function H is one for which it is computationally infeasible to find any two messages x and y such that ***H(x) = H(y)***.

For Birthday Attack : https://medium.com/math-simplified/the-birthday-attack-93b7b3522aa1

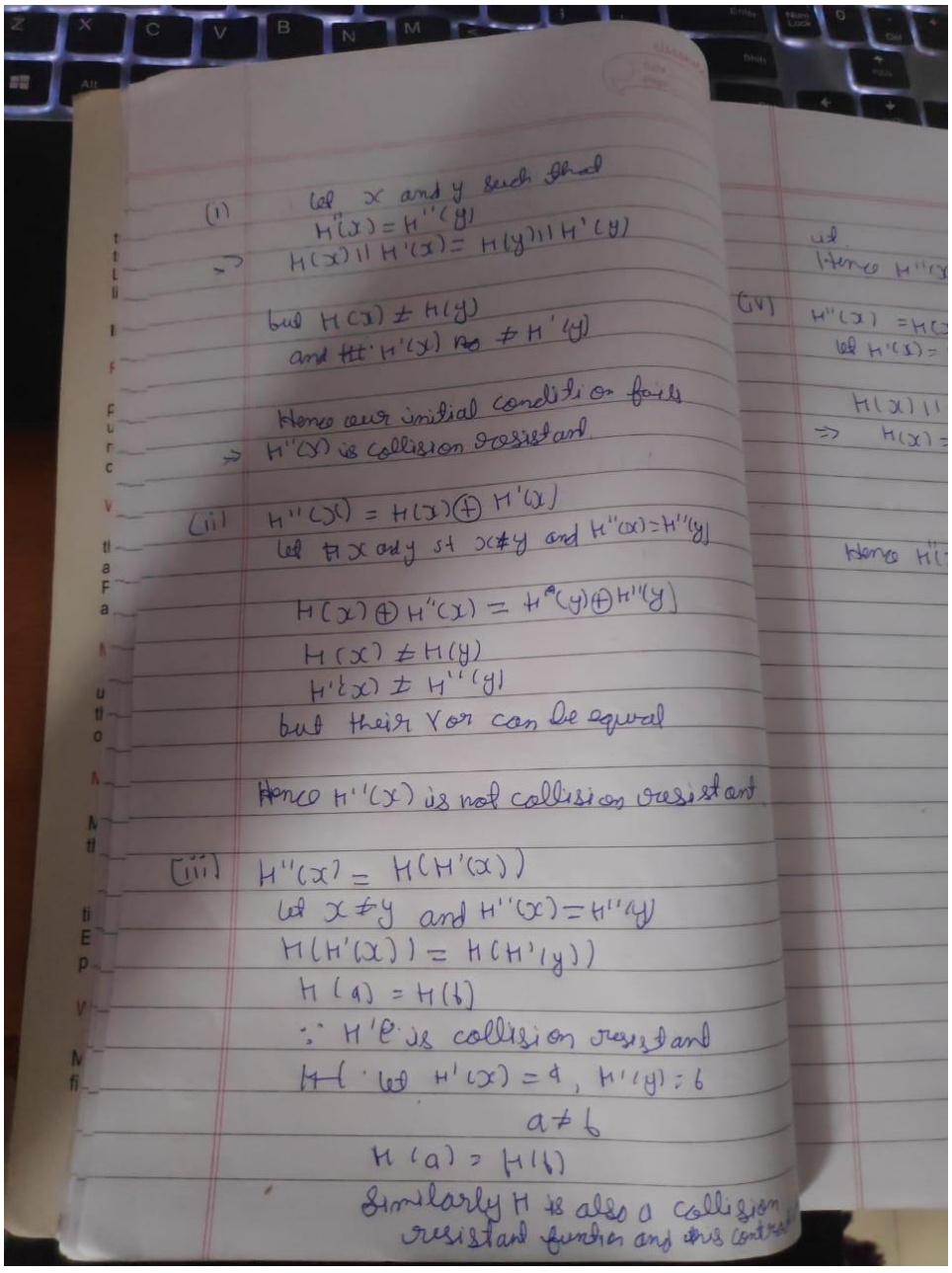
We show here that a birthday attack allows an attacker to find a collision in any hash function having an

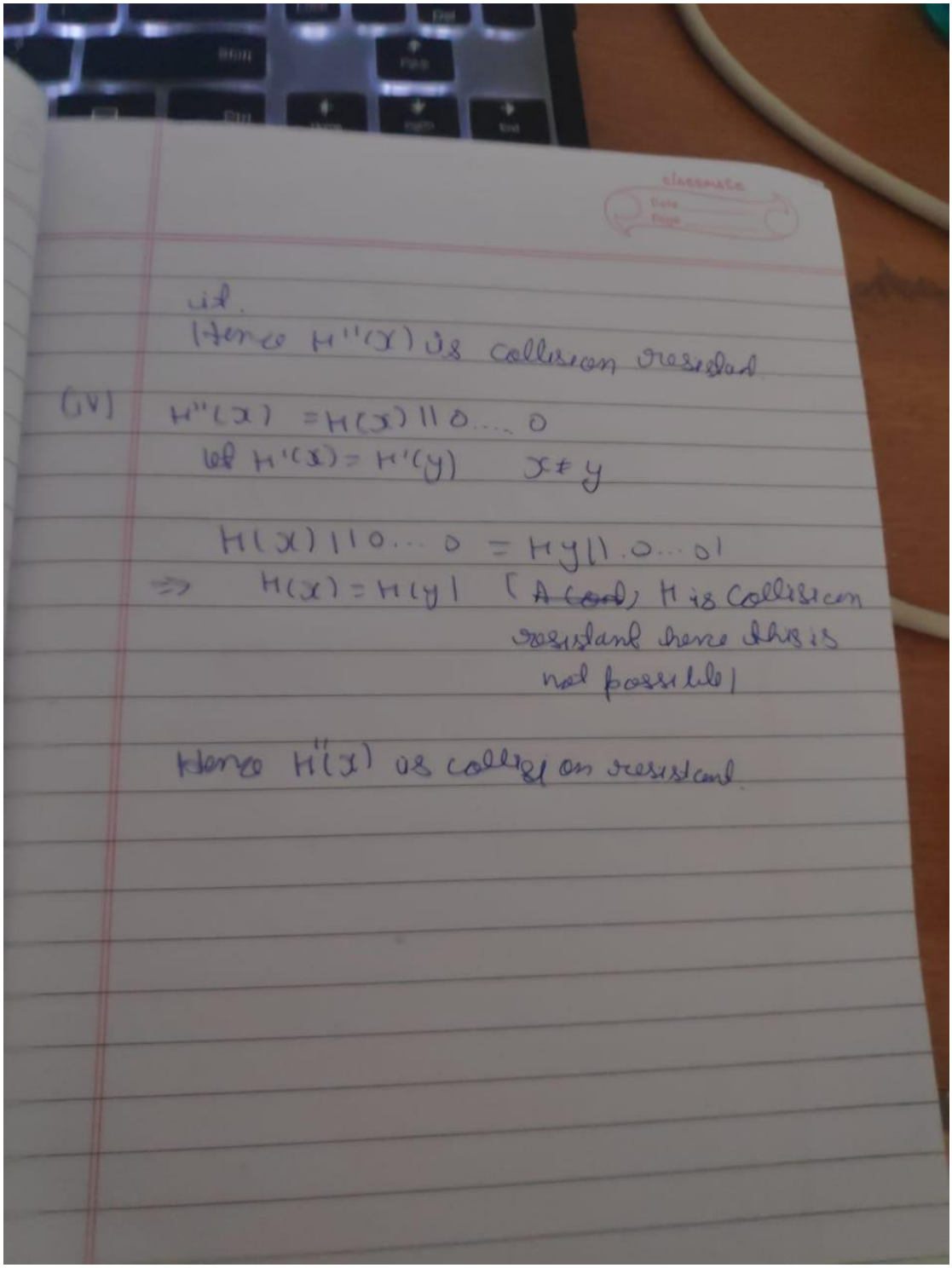
Thus, if we want to ensure collision resistance against attackers running in time 2n we need to use hash functions whose output is at least 2n bits long| twice the length of secret keys providing comparable security guarantees.

Encrypt and authenticate: To be found (all three)

b) To be found

c)





Please cross check these answers

1 a)

2.1 Strong One Way Function

A Strong One-Way function is a function which is easy to compute and can be inverted only with a negligible probability on a random input or it is hard to invert on all but a negligible fraction of inputs.

Definition 1. *A function* $f : \{0,1\}^* \rightarrow \{0,1\}^*$ *is called strongly one way if two condition hold*

1. *easy to compute: There exists a polynomial-time algrithm, A, so that on input x algorithm A outputs f(x) (i.e f(x)=A(x)).*
2. *hard to invert: For every probabilistic polynomial-time algorithm* $A^{'}$*,every polynomial p() , and all sufficently large n's*

$$Pr(A^{'}(f(x)) \in f^{-1}f(x)) < \frac{1}{p(n)}$$

2.2 Weak One Way Function

A Weak One-Way function is a function which is easy to compute and slightly hard to invert for random inputs or easy to invert on some non-negligible fraction of the inputs.

Definition 2. *A function* $f : \{0,1\}^n \rightarrow \{0,1\}^n$ *is called weak one-way, if*

f is a polynomial-time computable function
there exists a polynomial $p(.)$, for every probabilistic polynomial-time algorithm A, and all sufficiently large $n^{'}s$
$Pr(A^{'}(f(x)) \in f^{-1}f(x)) < 1 - \frac{1}{p(n)}$
where x is chosen uniformly in $0, 1^n$ and the probability is also over the internal coin flaps of A flops

Example Integer Factoring

Consider $f(x, y) = x.y$

Easy to compute
Is it **one-way**?
No: if $f(x, y)$ is even can set inverse as $(f(x, y)/2, 2)$
If factoring a number into prime factors is hard
Specially given $N = P.Q$, the product of two random large (n-bit) primes, it is hard to factor
Then somewhat hard - **there are a non-negligible fraction of such numbers** $1//n^2$ from the density of primes.
Hence a **weak** one-way function.

b) https://www3.cs.stonybrook.edu/~omkant/S04.pdf (See Theorem 2) (Not fully relevant, couldn't find anything else!)

c)

1 Collections of One-Way Functions

Definition 1 *A collection of one-way functions is a family of functions* $\mathcal{F} = \{f_i : \mathcal{D}_i \to \mathcal{R}_i\}_{i \in I}$ *such that:*

1. *It is easy to sample a function: There is some* $Gen \in \mathsf{PPT}$ *such that* $Gen(1^n)$ *outputs an index* $i \in I$.
2. *It is easy to sample the domain: There is a* PPT *machine which, on input* $i \in I$, *samples from* $\mathcal{D}_i$.
3. *It is easy to evaluate: There is a* PPT *machine which, on input* $i \in I, x \in \mathcal{D}_i$, *can compute* $f_i(x)$.
4. *It is hard to invert:*

$$(\forall A \in \mathsf{nuPPT})(\exists\ neg\ \epsilon)(Pr[i \leftarrow Gen(2^n); x \leftarrow \mathcal{D}_i : A(1^n, i, f_i(x)) \in f_i^{-1}(f_i(x))] \leq \epsilon(n))$$

Here are some candidates:

- Multipling large primes: $I = \mathbb{N}$, $\mathcal{D}_n = \{p, q : p \text{ and } q \text{ are } n\text{-bit primes}\}$, $Gen(1^n) \to n$, and $f_i(x, y) \to xy$.
- Exponentiation: $Gen(1^n) \to (p, g)$ where p is a random n-bit prime and g sis a generator for $\mathbb{Z}_p^*$. In this case, $f_{p,g}(x) \to g^x \mod p$. This function is one-to-one, ie. is a *permutation*. The Discrete Log Assumption states that this gives us a collection of one-way functions.
- RSA Collection: $Gen(1^n) \to (n, e)$ where $n = pq$ for random n-bit primes p and q, and e is a random element in $\mathbb{Z}_{\varphi(n)}$. In this case, $f_{n,e}(x) = x^e \pmod n$. This setup gives us a trapdoor permutation (ie. it's invertible with some extra information; more in this to come).

Proposition 1 *There exists a collection of one-way functions iff there exists a one-way function.*

Proof. If we have a one-way function g, define $Gen(1^n) \to n$, and $\mathcal{D}_n = \{0,1\}^n$. To sample from the domain for index set n, we generate a random string in $\{0,1\}^n$. Then we can define $f_i(x) = g(x)$.
Now suppose we have a collection of one-way functions with index generator $Gen : \{0,1\}^n \to I$ and sampling function $\sigma : i \to \mathcal{D}_i$. Then we can define a one-way function $g(r_1, r_2)$ with $|r_1| = |r_2|$ by setting $i \leftarrow Gen(r_1)$ and then using r_2 to sample from $\mathcal{D}_i$.

2)

Example 3.25

We can gain familiarity with the definition by considering an insecure example. Define the keyed, length preserving function F by $F(k,x) = k \oplus x$. For any input x, the value of $F_k(x)$ is uniformly distributed (when k is uniform). Nevertheless, F is not pseudorandom since its values on any *two* points are correlated. Consider the distinguisher D that queries its oracle $\mathcal{O}$ on distinct points x_1, x_2 to obtain values $y_1 = \mathcal{O}(x_1)$ and $y_2 = \mathcal{O}(x_2)$, and outputs 1 if and only if $y_1 \oplus y_2 = x_1 \oplus x_2$. If $\mathcal{O} = F_k$, for any k, then D outputs 1. On the other hand, if $\mathcal{O} = f$ for f chosen uniformly from Func_n, then

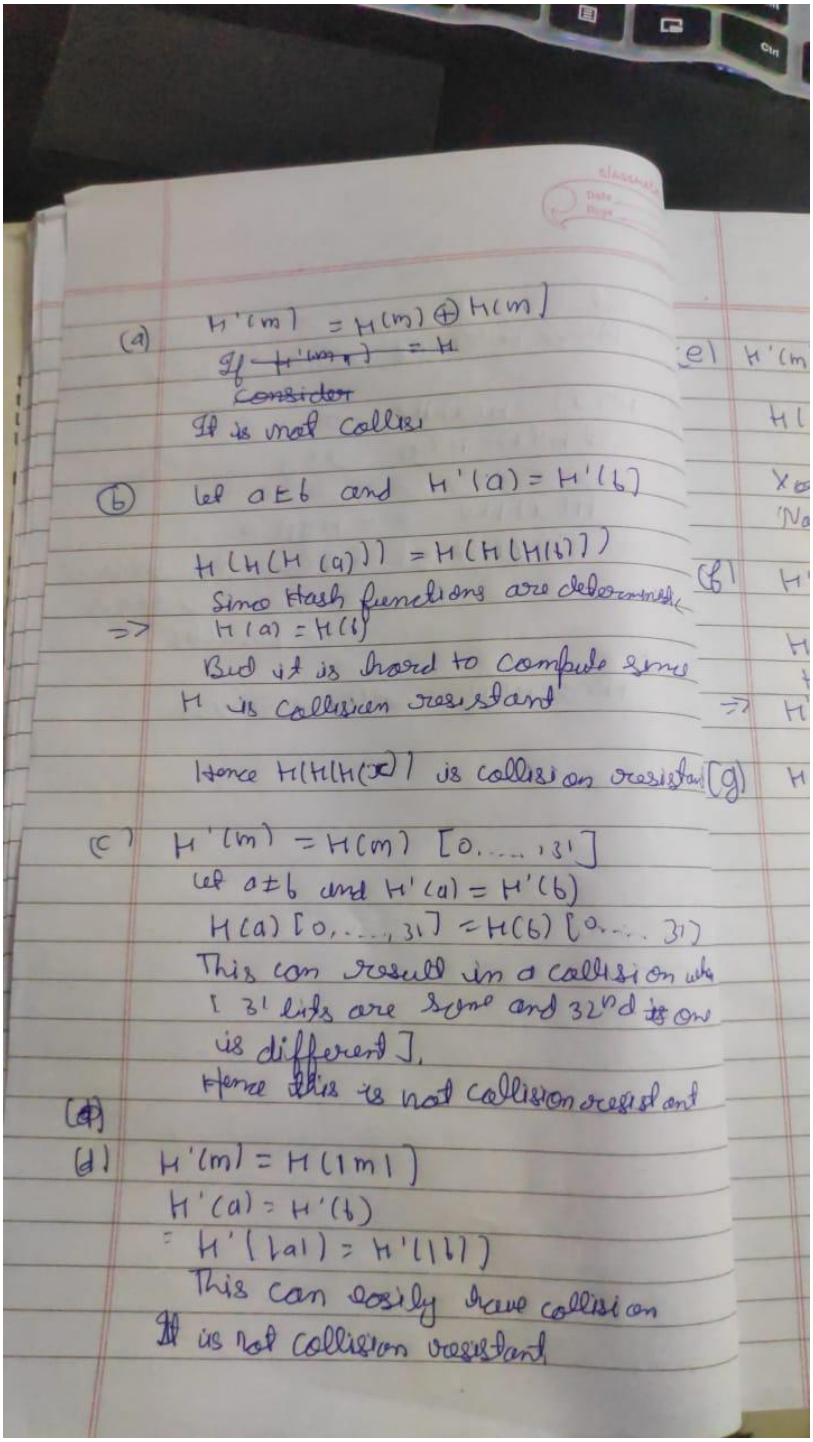
$$\Pr[f(x_1) \oplus f(x_2) = x_1 \oplus x_2] = \Pr[f(x_2) = x_1 \oplus x_2 \oplus f(x_1)] = 2^{-n},$$

since $f(x_2)$ is uniform and independent of x_1, x_2, and $f(x_1)$. We thus have $\Pr[D^{F_k(\cdot)}(1^n) = 1] = 1$ and $\Pr[D^{f(\cdot)}(1^n) = 1] = 2^{-n}$, and the difference between these two is not negligible. ♢

3) Idea taken from
https://crypto.stackexchange.com/questions/63548/construct-prf-with-longer-output-from-existing-prf
G(k,m) = F(k,m)||F(F(k,m),m) (Confirm this once)
An adversary can't break this scheme since key is derived using the same prf and it is being used to create another prf the resulting prf should be secure (Not sure though!)

4) a) Not collision resistant since it is giving same output 0 for every input

b)



(a) $H'(m) = H(m) \oplus H(m)$

~~If $H'(m_1) = H'(m_2)$~~

~~Consider~~

It is not collision

(b) let $a \neq b$ and $H'(a) = H'(b)$

$H(H(H(a))) = H(H(H(b)))$

Since Hash functions are deterministic

$\Rightarrow H(a) = H(b)$

But it is hard to compute since H is collision resistant

Hence $H(H(H(x)))$ is collision resistant

(c) $H'(m) = H(m)\ [0, \ldots, 31]$

let $a \neq b$ and $H'(a) = H'(b)$

$H(a)[0, \ldots, 31] = H(b)[0, \ldots, 31]$

This can result in a collision where [31 bits are same and 32nd bit one is different].

Hence this is not collision resistant

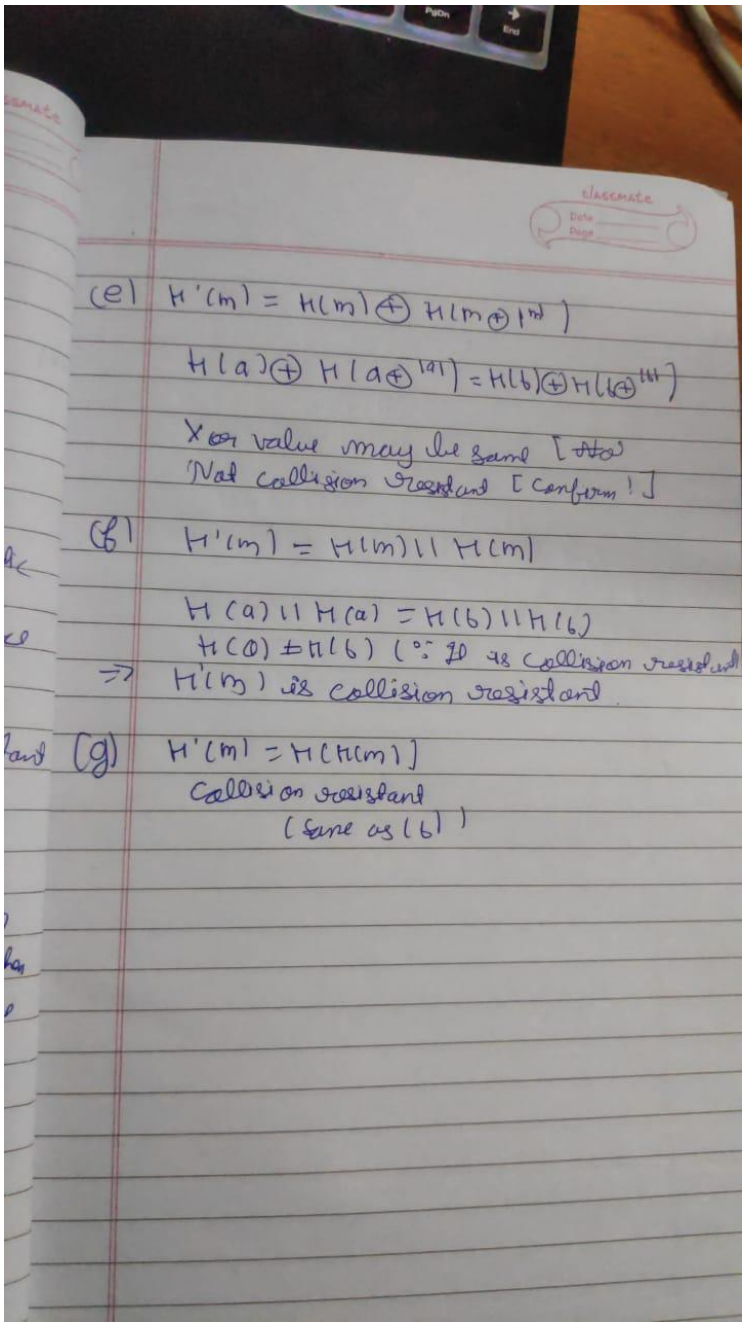
(d) $H'(m) = H(|m|)$

$H'(a) = H'(b)$

$= H'(|a|) = H'(|b|)$

This can easily have collision

It is not collision resistant



(e) $H'(m) = H(m) \oplus H(m \oplus 1^{|m|})$

$H(a) \oplus H(a \oplus 1^{|a|}) = H(b) \oplus H(b \oplus 1^{|b|})$

XOR value may be same [~~sta~~]
Not collision resistant [Confirm!]

(f) $H'(m) = H(m) \| H(m)$

$H(a) \| H(a) = H(b) \| H(b)$
$H(a) = H(b)$ ($\because$ H is collision resistant)
$\Rightarrow$ $H'(m)$ is collision resistant

(g) $H'(m) = H(H(m))$
Collision resistant
(same as (b))

5 a)

6.2 The Merkle–Damgård Transform

Many applications require "full-fledged" collision-resistant hash functions that can handle very long inputs, or even inputs of arbitrary length. But it is much easier to construct fixed-length hash functions (i.e., compression functions) that only accept "short" inputs—something we will return to in Section 7.3. Fortunately, the *Merkle–Damgård transform* allows us to convert the latter to the former. This approach for domain extension of hash functions has been used frequently in practice, including for the hash function MD5 and the SHA hash family (cf. Section 7.3). The Merkle–Damgård transform is also interesting from a theoretical point of view since it implies that compressing by a single bit is as easy (or as hard) as compressing by an arbitrary amount.

For concreteness, assume the compression function (Gen, h) takes inputs of length $n + n' \geq 2n$, and generates outputs of length n. (The construction can be generalized for other input/output lengths, as long as h compresses.) Applying the Merkle–Damgård transform, defined in Construction 6.3 and depicted in Figure 6.1, yields a hash function (Gen, H) that maps inputs of *arbitrary* length to outputs of length n.

CONSTRUCTION 6.3

Let (Gen, h) be a compression function for inputs of length $n + n' \geq 2n$ with output length n. Fix $\ell \leq n'$ and $IV \in \{0,1\}^n$. Construct hash function (Gen, H) as follows:

- Gen: remains unchanged.
- H: on input a key s and a string $x \in \{0,1\}^*$ of length $L < 2^\ell$, do:
  1. Append a 1 to x, followed by enough zeros so that the length of the resulting string is ℓ less than a multiple of n'. Then append L, encoded as an ℓ-bit string. Parse the resulting string as the sequence of n'-bit blocks $x_1, \ldots, x_B$.
  2. Set $z_0 := IV$.
  3. For $i = 1, \ldots, B$, compute $z_i := h^s(z_{i-1} \| x_i)$.
  4. Output z_B.

5 b)

PROOF We show that for any s, a collision in H^s yields a collision in h^s. Let x and x' be two different strings of length L and L', respectively, such that $H^s(x) = H^s(x')$. Let $x_1, \ldots, x_B$ be the B blocks of the padded x, and let $x'_1, \ldots, x'_{B'}$ be the B' blocks of the padded x'. Let $z_0, z_1, \ldots, z_B$ (resp., $z'_0, z'_1, \ldots, z'_{B'}$) be the intermediate results during computation of $H^s(x)$ (resp., $H^s(x')$). There are two cases to consider:

Case 1: $L \neq L'$. In this case, the last step of the computation of $H^s(x)$ is $z_B := h^s(z_{B-1} \| x_B)$, and the last step of the computation of $H^s(x')$ is $z'_{B'} := h^s(z'_{B'-1} \| x'_{B'})$. Since $H^s(x) = H^s(x')$ we have $h^s(z_{B-1} \| x_B) = h^s(z'_{B'-1} \| x'_{B'})$. However, $L \neq L'$ and so $x_B \neq x'_{B'}$. (Recall that the last ℓ bits of x_B encode L, and the last ℓ bits of $x'_{B'}$ encode L'.) Thus, $z_{B-1} \| x_B$ and $z'_{B'-1} \| x'_{B'}$ are a collision with respect to h^s.

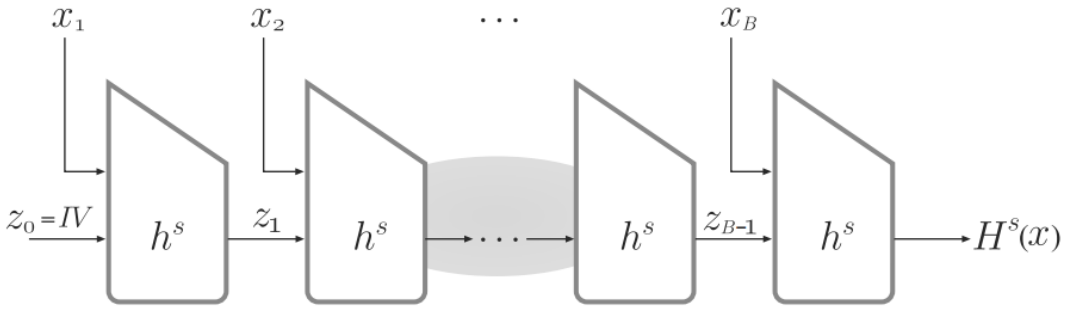


FIGURE 6.1: The Merkle–Damgård transform.

Case 2: $L = L'$. This means that $B = B'$. Let $I_i \stackrel{\text{def}}{=} z_{i-1} \| x_i$ denote the ith input to h^s during computation of $H^s(x)$, and define $I_{B+1} \stackrel{\text{def}}{=} z_B$. Define $I'_1, \ldots, I'_{B+1}$ analogously with respect to x'. Let N be the largest index for which $I_N \neq I'_N$. Since $|x| = |x'|$ but $x \neq x'$, there is an i with $x_i \neq x'_i$ and so such an N certainly exists. Because

$$I_{B+1} = z_B = H^s(x) = H^s(x') = z'_B = I'_{B+1},$$

we have $N \leq B$. By maximality of N, we have $I_{N+1} = I'_{N+1}$ and in particular $z_N = z'_N$. But this means that I_N, I'_N collide under h^s.

We leave it as an exercise to turn the above into a proof by reduction. ■

6 Reasoning:

For some applications, security requirements weaker than collision resistance suffice. Security notions that are sometimes considered include:

- *Second-preimage resistance:* Informally, a hash function is said to be second-preimage resistant if given s and a uniform x it is infeasible for a PPT adversary to find $x' \neq x$ such that $H^s(x') = H^s(x)$.
- *Preimage resistance:* Informally, a hash function is preimage resistant if given s and $y = H^s(x)$ for a uniform x, it is infeasible for a PPT adversary to find a value x' (whether equal to x or not) with $H^s(x') = y$. (Looking ahead to Chapter 8, this basically means that H^s is *one-way*.)

It is immediate that any hash function that is collision resistant is also second-preimage resistant. It is also true that if a hash function is second-preimage resistant then it is preimage resistant. We do not formally define the above

a) True
b) False
c) To be done
d) True

3. Let $\mathsf{H}_1, \mathsf{H}_2 : \{0,1\}^m \to \{0,1\}^n$ be two hash functions. Define a hash function $\mathsf{H} : \{0,1\}^m \to \{0,1\}^{2n}$ as $\mathsf{H}(x) = \mathsf{H}_1(x) \| \mathsf{H}_2(x)$. Prove that if at least one of $\mathsf{H}_1, \mathsf{H}_2$ is collision resistant, then H is collision resistant.

A: If x, x' is a collision for H, then $\mathsf{H}(x) = \mathsf{H}(x')$ i.e., $\mathsf{H}_1(x) \| \mathsf{H}_2(x) = \mathsf{H}_1(x') \| \mathsf{H}_2(x')$ i.e., $\mathsf{H}_1(x) = \mathsf{H}_1(x')$ and $\mathsf{H}_2(x) = \mathsf{H}_2(x')$. That means x, x' is a collision for both H_1 and H_2. So, if atleast of $\mathsf{H}_1, \mathsf{H}_2$ is collision resistant, then so is H.

7) To be done
8)

If we define a hash function (or compression function) h that will hash an n-bit binary string to an m-bit binary string, we can view h as a function from $\mathbb{Z}_{2^n}$ to $\mathbb{Z}_{2^m}$. It is tempting to define h using integer operations modulo 2^m. We show in this exercise that some simple constructions of this type are insecure and should therefore be avoided.

(a) Suppose that $n = m > 1$ and $h : \mathbb{Z}_{2^m} \to \mathbb{Z}_{2^m}$ is defined as

$$h(x) = x^2 + ax + b \pmod 2$$

Prove that it is easy to solve **Second Preimage** for any $x \in \mathbb{Z}_{2^m}$ without having to solve a quadratic equation.

(b) Suppose that $n > m$ and $h : \mathbb{Z}_{2^n} \to \mathbb{Z}_{2^m}$ is defined to be a polynomial of degree d:

$$h(x) = \sum_{i=0}^{d} a_i x^i \pmod{2^m}$$

where $a_i \in \mathbb{Z}$ for $0 \leq i \leq d$. Prove that it is easy to solve **Second Preimage** for any $x \in \mathbb{Z}_{2^n}$ without having to solve polynomial equation.

Solution:

(a) A hash function $h : \mathbb{Z}_{2^n} \to \mathbb{Z}_{2^m}$, where $n = m$, defined as:

$$h(x) = (x^2 + ax + b) \pmod{2^m} \tag{14}$$

is not Second preimage resistant, because every $\bar{x} \in \mathbb{Z}_2^n$ of the form:

$$\bar{x} = -x - a, \tag{15}$$

represents a valid solution to the Second Preimage Problem.

To see this, let's input $\bar{x}$ into the given hash function:

$$\begin{aligned} h(\bar{x}) &= (-x-a)^2 + a(-x-a) + b = \\ &= x^2 + 2ax + a^2 - ax - a^2 + b = x^2 + ax + b \end{aligned} \tag{16}$$

9) To be done

10) a)

Solution:

Let's prove by contradiction that the hash function h_2 is collision resistant: let's assume that there exist $x_1, x_2 \in \{0,1\}^{4m}$ such that $x_1 \neq x_2$, but $h_2(x_1) = h_2(x_2)$. Let's further define x_1 and x_2 as:

$$\begin{aligned} x_1 &= x_1' \| x_1'' \text{ where } x_1', x_1'' \in \{0,1\}^{2m} \\ x_2 &= x_2' \| x_2'' \text{ where } x_2', x_2'' \in \{0,1\}^{2m} \end{aligned}$$

Since $h_2(x_1) = h_2(x_2)$, we can write:

$$h_1[h_1(x_1') \| h_1(x_1'')] = h_1[h_1(x_2') \| h_1(x_2'')] \tag{4}$$

Using the fact that the hash function h_1 is collision resistant, i.e. there does not exist $x_i, x_j \in \{0,1\}^{2m}$ such that $x_i \neq x_j$, but $h_1(x_i) = h_1(x_j)$, equation (4) can be rewritten as:

$$h_1(x_1') \| h_1(x_1'') = h_1(x_2') \| h_1(x_2'') \tag{5}$$

Using the property of concatenation operation that two binary strings can be equal only if all concatenated parts are equal, we can rewrite equation (5) as:

$$\begin{aligned} h_1(x_1') &= h_1(x_2') \\ h_1(x_1'') &= h_1(x_2'') \end{aligned} \tag{6}$$

Using again the fact that function h_1 is collision resistant, it follows that:

$$\begin{aligned} x_1' &= x_2' \\ x_1'' &= x_2'' \end{aligned} \tag{7}$$

From equation (7), it finally follows that $x_1 = x_2$, which contradicts the initial assumption. Therefore, hash function h_2 is collision resistant.

10 b)

Solution:

(b) Let's prove by induction that hash function h_i is collision resistant:

First step: $i = 2$

As proved in part (a) of this problem, the hash function $h_2(x) = h_1[h_1(x')\|h_1(x'')]$, where $x \in \{0,1\}^{4m}$ and $x', x'' \in \{0,1\}^{2m}$, is collision resistant.

Intermediate step: $i = k - 1$

The hash function $h_{k-1}(x) = h_1[h_{k-2}(x')\|h_{k-2}(x'')]$, where $x \in \{0,1\}^{2^{k-1}m}$ and $x', x'' \in \{0,1\}^{2^{k-2}m}$, is collision resistant.

Final step: $i = k$

Let prove by contradiction that the hash function $h_k(x) = h_1[h_{k-1}(x')\|h_{k-1}(x'')]$, where $x \in \{0,1\}^{2^k m}$ and $x', x'' \in \{0,1\}^{2^{k-1}m}$ is collision resistant: let assume there exist two distinct elements $x_1 = x_1'\|x_1'', x_2 = x_2'\|x_2'' \in \{0,1\}^{2^k m}, x_1', x_1'', x_2', x_2'' \in \{0,1\}^{2^{k-1}m}$ such that $x_1 \neq x_2$, but $h_k(x_1) = h_k(x_2)$. We can now write:

$$h_1[h_{k-1}(x_1')\|h_{k-1}(x_1'')] = h_1[h_{k-1}(x_2')\|h_{k-1}(x_2'')] \tag{28}$$

Using the fact that the hash function h_1 is collision resistant, we can rewrite equation (28) as:

$$h_{k-1}(x_1')\|h_{k-1}(x_1'') = h_{k-1}(x_2')\|h_{k-1}(x_2'') \tag{29}$$

Combining the properties of concatenation operation and the assumption that the hash function h_{k-1} is collision resistant, from equation (29) it follows that $x_1 = x_2$, which contradicts the initial assumption about h_k. Therefore, hash function h_k is collision resistant.

11)

2 (Each ciphertext

blocks affects only the current plaintext block and the next.)

1) To be found