Part I (5 Marks / Question):

Q1	**Q2**	**Q3**	**Q4**
ABCD	C	BCD	AD
Q5	**Q6**	**Q7**	
CD	ABD	AD	

Part II:

Q8 (6 marks):

When a server receives an HOTP from a user, it calculates one HOTP locally by $HOTP(k, C) = HMAC(k, C)\ mod\ 10^d$, where k is a pre-shared key, C is a counter shared with the user, and d is a public parameter. If the locally-calculated HOTP is equal to the user's submission, it is accepted.

That is because other persons who do not know k cannot generate the same HMAC value, even C and d are public. If the HOTP can pass the verification, it must be calculated from k. Since only the legitimate user owns k, it must be the legitimate user.

Q9 (4 marks):

That is because both client and server can make sure that all the shared information (e.g., parameters, algorithms) is the same. Otherwise, they cannot perform the correct encryption & decryption.

Q10 (6 marks):

Since $m \equiv c^d\ (mod\ n)$, we have $m\ mod\ n = c^d\ mod\ n$. If we let $c^d\ mod\ n = r$, we have $m\ mod\ n = r$, i.e., $m = kn + r$ where k is any integer. If we did not have such limitation, we would have infinite m that fulfills the requirements.

Q11 (8 marks):

If a verification fails, the revealed hash is different from the hash calculated from the received document. It is due to two reasons.

Reason 1: The public key and secret key are not in pairs. In the signing procedure, a signer uses sk to sign the hash $H(m)$. If pk and sk are not in pairs, a verifier cannot reveal the same hash from the signature as when it was signed, which will be different from the hash calculated from the received document m.

Reason 2: The document received is altered. If it is $m' \neq m$, $H(m') \neq H(m)$. In this case, a verifier will reveal the correct hash $H(m)$ from the signature σ, but its different from the hash calculated from the document received. Thus the verification fails.

Q12 (6 marks):

If a hash function $H(x)$ does not meet the weak collision resistance, given a file m, we may have a different file m' s.t. $H(m) = H(m')$. In a verifier's view, it will treat m and m' as the same file, even their contents are different. This breaks the integrity.

Q13 (12 marks):

Since the block size is 2 bits, the message 011010 can be divided into 3 blocks, 01, 10, and 10.

The initial vector of sponge construction is an all-zero vector 0000.

Absorbing:

Step 1: XOR the first block, the vector is 0100. Use it as the input of f, get the output 1001 from the table as the vector.

Step 2: XOR the second block, the vector is 0001. Use it as the input of f, get the output 0100 from the table as the vector.

Step 3: XOR the third block, the vector is 1100. Use it as the input of f, get the output 1111 from the table as the vector.

Squeezing:

Since the hash length is 4 bits, we need to squeeze out 2 blocks.

Step 1: Get the first block 11 from 1111. Use it as the input of f, get the output 0110.

Step 2: Get the second block 01 from 0110.

The hash is 1101.

Q14 (8 marks):

Attacker's steps:

Step 1: When the browser sends the signed challenge to the server, this attacker fetches this signed challenge.

Step 2: This attacker initiates a login attempt and sends this fetched signed challenge to the server.

Reasons:

For example, a server always sends 123456 as the challenge number. When this user (Alice) uses its key to sign this challenge, an attacker can fetch this signed 123456 from its communication to the server. This signed 123456 can be verified by the server from this user's public key.

When this attacker initiates a login attempt, it also receives a request to ask it to sign 123456. This attacker can send the fetched signed 123456. This can also be verified by the server from this user's public key. The server cannot distinguish whether this signed 123456 is an old one or a fresh one.

Q15 (15 marks):

Since the block size is 4 bits, the message 011011001101 can be divided into 3 blocks, 0110, 1100, and 1101.

The nonce is 10 and the counter starts from 00.

(Possible solution 1)

Block 1: The input of the block cipher should be 1000. The output of the block cipher should be 0011 from the table. XOR the first block to get its ciphertext 0101.

Block 2: The input of the block cipher should be 1001. The output of the block cipher should be 0000 from the table. XOR the second block to get its ciphertext 1100.

Block 3: The input of the block cipher should be 1010. The output of the block cipher should be 0100 from the table. XOR the third block to get its ciphertext 1001.

Thus, the ciphertext should be 010111001001.

(Possible solution 2)

Block 1: The input of the block cipher should be 0010. The output of the block cipher should be 1000 from the table. XOR the first block to get its ciphertext 1110.

Block 2: The input of the block cipher should be 0110. The output of the block cipher should be 1011 from the table. XOR the second block to get its ciphertext 0111.

Block 3: The input of the block cipher should be 1010. The output of the block cipher should be 0100 from the table. XOR the third block to get its ciphertext 1001.

Thus, the ciphertext should be 111001111001.