Q1. Explain Data Encryption Standard (DES) and Rivest-Shamir-Adleman (RSA) Algorithms

Data Encryption Standard (DES):

Data Encryption Standard (DES) is a block cipher with a 56-bit key length that has played a significant role in data security. Data encryption standard (DES) has been found vulnerable to very powerful attacks therefore, the popularity of DES has been found slightly on the decline. DES is a block cipher and encrypts data in blocks of size of 64 bits each, which means 64 bits of plain text go as the input to DES, which produces 64 bits of ciphertext. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits.

DES is based on the two fundamental attributes of cryptography: substitution (also called confusion) and transposition (also called diffusion). DES consists of 16 steps, each of which is called a round. Each round performs the steps of substitution and transposition. Let us now discuss the broad-level steps in DES.

In the first step, the 64-bit plain text block is handed over to an initial Permutation (IP) function.

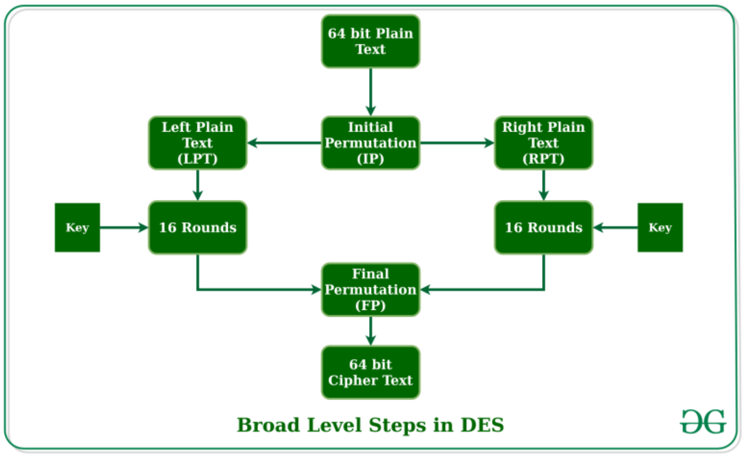
The initial permutation is performed on plain text.

Next, the initial permutation (IP) produces two halves of the permuted block; saying Left Plain Text (LPT) and Right Plain Text (RPT).

Now each LPT and RPT go through 16 rounds of the encryption process.

In the end, LPT and RPT are rejoined and a Final Permutation (FP) is performed on the combined block

The result of this process produces 64-bit ciphertext.



Initial Permutation (IP)

As we have noted, the initial permutation (IP) happens only once and it happens before the first round. It suggests how the transposition in IP should proceed, as shown in the figure. For example, it says that the IP replaces the first bit of the original plain text block with the 58th bit of the original plain text, the second bit with the 50th bit of the original plain text block, and so on.

This is nothing but jugglery of bit positions of the original plain text block. the same rule applies to all the other bit positions shown in the figure.

As we have noted after IP is done, the resulting 64-bit permuted text block is divided into two half blocks. Each half-block consists of 32 bits, and each of the 16 rounds, in turn, consists of the broad-level steps outlined in the figure.

Step 1: Key transformation

We have noted initial 64-bit key is transformed into a 56-bit key by discarding every 8th bit of the initial key. Thus, for each a 56-bit key is available. From this 56-bit key, a different 48-bit Sub Key is generated during each round using a process called key transformation. For this, the 56-bit key is divided into two halves, each of 28 bits. These halves are circularly shifted left by one or two positions, depending on the round.

For example: if the round numbers 1, 2, 9, or 16 the shift is done by only one position for other rounds, the circular shift is done by two positions. The number of key bits shifted per round is shown in the figure.

After an appropriate shift, 48 of the 56 bits are selected. From the 48 we might obtain 64 or 56 bits based on requirement which helps us to recognize that this model is very versatile and can handle any range of requirements needed or provided. for selecting 48 of the 56 bits the table is shown in the figure given below. For instance, after the shift, bit number 14 moves to the first position, bit number 17 moves to the second position, and so on. If we observe the table , we will realize that it contains only 48-bit positions. Bit number 18 is discarded (we will not find it in the table), like 7 others, to reduce a 56-bit key to a 48-bit key. Since the key transformation process involves permutation as well as a selection of a 48-bit subset of the original 56-bit key it is called Compression Permutation.

Because of this compression permutation technique, a different subset of key bits is used in each round. That makes DES not easy to crack.

Step 2: Expansion Permutation

Recall that after the initial permutation, we had two 32-bit plain text areas called Left Plain Text(LPT) and Right Plain Text(RPT). During the expansion permutation, the RPT is expanded from 32 bits to 48 bits. Bits are permuted as well hence called expansion permutation. This happens as the 32-bit RPT is divided into 8 blocks, with each block consisting of 4 bits. Then, each 4-bit block of the previous step is then expanded to a corresponding 6-bit block, i.e., per 4-bit block, 2 more bits are added.

This process results in expansion as well as a permutation of the input bit while creating output. The key transformation process compresses the 56-bit key to 48 bits. Then the expansion permutation process expands the **32-bit RPT** to **48-bits**.

RSA algorithm is an asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. Public Key and Private Key. As the name describes that the Public Key is given to everyone and the Private key is kept private.

An example of asymmetric cryptography:

A client (for example browser) sends its public key to the server and requests some data.

The server encrypts the data using the client's public key and sends the encrypted data.

The client receives this data and decrypts it.

Since this is asymmetric, nobody else except the browser can decrypt the data even if a third party has the public key of the browser.

The idea! The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is a multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024-bit keys could be broken in the near future. But till now it seems to be an infeasible task. we will implement a simple version of RSA using primitive roots.

Step 1: Generating Keys

To start, we need to generate two large prime numbers, p and q. These primes should be of roughly equal length and their product should be much larger than the message we want to encrypt.

We can generate the primes using any primality testing algorithm, such as the Miller-Rabin test. Once we have the two primes, we can compute their product n = p*q, which will be the modulus for our RSA system.

Next, we need to choose an integer e such that 1 < e < phi(n) and gcd(e, phi(n)) = 1, where phi(n) = (p-1)*(q-1) is Euler's totient function. This value of e will be the public key exponent.

To compute the private key exponent d, we need to find an integer d such that d*e = 1 (mod phi(n)). This can be done using the extended Euclidean algorithm.

Our public key is (n, e) and our private key is (n, d).

Step 2: Encryption

To encrypt a message m, we need to convert it to an integer between 0 and n-1. This can be done using a reversible encoding scheme, such as ASCII or UTF-8.

Once we have the integer representation of the message, we compute the ciphertext c as c = m^e (mod n). This can be done efficiently using modular exponentiation algorithms, such as binary exponentiation.

Step 3: Decryption

To decrypt the ciphertext c, we compute the plaintext m as m = c^d (mod n). Again, we can use modular exponentiation algorithms to do this efficiently.

Q2. Explain Diffie-Hellman Key Exchange Algorithm With an Example

Diffie-Hellman Key Exchange Algorithm:
Diffie-Hellman is a mathematical problem that is the foundation for many cryptographic protocols. Diffie-Hellman is one of the greatest inventions in Cybersecurity. This revolutionary algorithm was proposed by Whitfield Diffie and Martin Hellman in 1976, enabling two entities to agree upon a secret key without prior arrangements, even in the presence of potential eavesdroppers. Diffie-Hellman offers a powerful solution to

secure key exchange which has always been challenging and prone to alteration, thus ensuring confidentiality and integrity in information.

A key exchange algorithm is needed in communication and cryptography for several reasons:

- It enables two or more parties to agree upon a secret key without exposing it to potential eavesdroppers, the key is then used for encryption and decryption, which is vital for maintaining confidentiality in communication.
- Preserving the data integrity was also a major challenge in digital communication where data is always vulnerable to tempering while transmission. A key exchange algorithm helps in preserving the integrity of the transmitted data, it prevents unauthorized alteration or tampering of data during transmission.
- A key exchange algorithm facilitates authentication of the communicating parties, verifies who they claim to be, thus escalating the risk of man-in-the-middle (impersonation) attack.

Thus along with encrypting the data for maintaining the confidentiality of the communication, a key exchange algorithm was also needed to maintain the integrity and authorized access of the information.

Method of Operation in Diffie-Hellman Key Exchange

Diffie-Hellman key exchange algorithm is based on the principles of modular exponentiation and discrete logarithms to allow two parties to securely establish a shared secret key over an insecure communication channel. Here is an operational overview of the process in context to Alice and Bob :

1. Parameters Setup

Alice and Bob must agree upon two number:

- A large prime number **p**,
- A generator **g** of p, which is the primitive root of p

These two number are shared and are not kept secret.

2. Key Generation

- Alice and Bob randomly chose a private key, say xa and xb, where xa is the private key of Alice and xb is the private key of Bob.
- These private keys are kept secret and not being shared.

3. Public Key Exchange

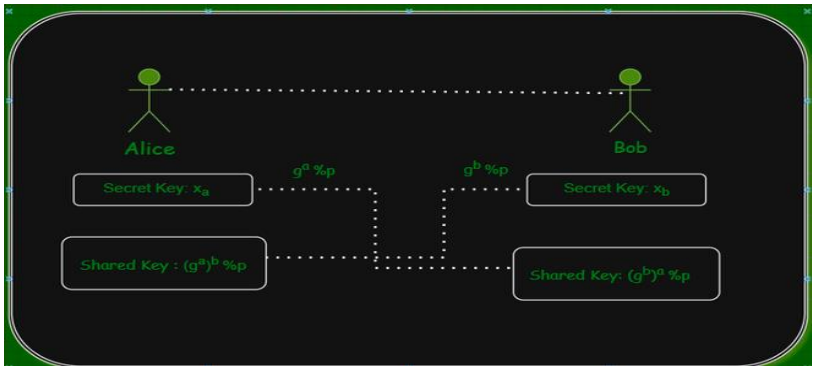
- Both Alice and Bob perform a calculation to generate their corresponding public keys.
  $ya = ga \ (mod\ p)$
  $yb = gb \ (mod\ p),$
  where ya is the public key of Alice and yb is the public key of Bob
- The public key are then shared with each other, ya is shared with Bob and yb is shared with Alice.

4. Shared Secret Key Calculation

- Alice then calculates the shared secret using the yb received from Bob and her private key as:
  $k = (yb)xa \ (mod\ p)$
- Bob also calculates the shared secret using the ya received from Alice and his private key xb as:
  $k = (ya)xb \ (mod\ p)$

5. Resulting Secret

Alice and Bob will end upon the same shared secret key, which can be used for encryption and decryption of information using symmetric key algorithms.



Numerical Example on Diffie-Hellman Key Exchange

Question: Suppose Alice and Bob agreed on p as 7 and g as 5. Find the value of secret keys?

Solution:

Let us suppose xa be 3, then ya can be given as:

ya = 53 % 7 = 6

Also let us assume xb as 4, then yb can be calculated as:

yb = 54 % 7 = 2

The value of k which is the secret key, can be calculated and verified as:

k = 23 % 7 = 64 % 7 = 1

k = 1, for both the calculations, which is the shared secret.

Strength of Diffie-Hellman Key Exchange Algorithm

The Diffie-Hellman key exchange is secure because of the difficulty of calculation discrete logarithms. An eavesdropper listening to the communication channel for the exchanged value of ya and yb would find it extremely difficult to determine shared secret without knowing the value of xa or xb which are private keys and a limited to the one party.

Thus it allows two parties (say Alice and Bob) to securely establish a shared secret key over an insecure channel without the need to transmit the key itself, establishing a secure means for encrypted communication eliminating the vulnerabilities associated in direct transmission of keys.

Perfect Forward Secrecy (PFS)

Perfect Forward Secrecy is the property in the cryptography that prevents the exposure of long-term secret keys from compromising the past or future communication. In context to Diffie-Hellman, prefect forward secrecy means that even if an attacker were somehow gain / compute the private keys used during a session, he would not be able to decrypt past communications or use those keys to decrypt any of the future communication. It's an important property of a systems where the long-term security of data is crucial, it helps to prevent the accumulation of data over time, making it more complex for attackers to decrypt large amounts of data even if they obtain private keys or have the ability to eavesdrop on communications.

Key Aspects of Perfect Forward Secrecy

- **Use of Session Keys:** Systems that implements Perfect Forward Secrecy generates a unique session key for every session, so even if an attackers manages to know the current session key, it cannot use it to decrypt past or future communications, as each session keys becomes invalid after session is over.
- **Temporary Keys:** The use of temporary keys generated by Perfect Forward Secrecy system for each session, which is not use for other sessions, ensuring that if one key is compromised, it doesn't compromises other communications.
- **Zero Dependence on Long-term Keys:** The long-term keys used for authentication or key exchange, in case they are compromised other communication remains secured. As they are used for the purpose of establishing the session keys.

- **Enhanced Security:** Perfect Forward Secrecy add a layer to the security, in scenarios where long-term keys might be at risk due to various factors such as complex cyber attacks, compromised servers, or future cryptographic development.

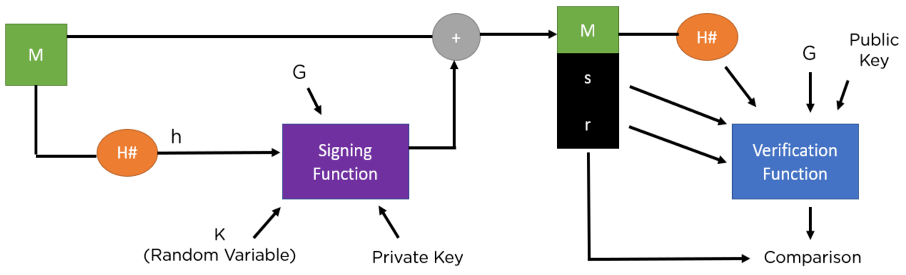
Q3. Explain Digital Signature Algorithm (DSA) With an Example

Digital Signature Algorithm (DSA):

Digital Signatures Algorithm is a FIPS (Federal Information Processing Standard) for digital signatures. It was proposed in 1991 and globally standardized in 1994 by the National Institute of Standards and Technology (NIST). It functions on the framework of modular exponentiation and discrete logarithmic problems, which are difficult to compute as a force-brute system.

DSA Algorithm provides three benefits, which are as follows:

- Message Authentication: You can verify the origin of the sender using the right key combination.
- Integrity Verification: You cannot tamper with the message since it will prevent the bundle from being decrypted altogether.
- Non-repudiation: The sender cannot claim they never sent the message if verifies the signature.



The image above shows the entire procedure of the DSA algorithm. You will use two different functions here, a signing function and a verification function. The difference between the image of a typical digital signature verification process and the one above is the encryption and decryption part. They have distinct parameters, which you will look into in the next section of this lesson on the DSA Algorithm.

Steps in DSA Algorithm

Keeping the image above in mind, go ahead and see how the entire process works, starting from creating the key pair to verifying the signature at the end.

1. Key Generation

There are two steps in the key generation process: parameter generation and per-user keys.

Parameter Generation

- Initially a user needs to choose a cryptographic hash function (H) along with output length in bits |H|. Modulus length N is used in when output length |H| is greater.
- Then choose a key length L where it should be multiple of 64 and lie in between 512 and 1024 as per Original DSS length. However, lengths 2048 or 3072 are recommended by NIST for lifetime key security.
- The values of L and N need to be chosen in between (1024, 60), (2048, 224), (2048, 256), or (3072, 256) according to FIPS 186-4. Also, a user should chose modulus length N in such a way that modulus length N should be less than key length (N<L) and less than and equal to output length (N<=|H|).
- Later a user can choose a prime number q of N bit and another prime number as p of L bit in such a way that p-1 is multiple of q. And then choose h as an integer from the list (2……..p-2).
- Once you get p and q values, find out

g = h^(p-1)/q*mod(p). If you get g = 1, please try another value for h and compute again for g except 1.

p, q and g are the algorithm parameters that are shared amongst different users of the systems.

Per-user Keys

To compute the key parameters for a single user, first choose an integer x (private key) from the list (1…….q-1), then compute the public key, y=g^(x)*mod(p).

2. Signature Generation

- It passes the original message (M) through the hash function (H#) to get our hash digest(h).
- It passes the digest as input to a signing function, whose purpose is to give two variables as output, s, and r.
- Apart from the digest, you also use a random integer k such that 0 < k < q.
- To calculate the value of r, you use the formula r = (gk mod p) mod q.
- To calculate the value of s, you use the formula s = [K-1(h+x . R)mod q].
- It then packages the signature as {r,s}.
- The entire bundle of the message and signature {M,r,s} are sent to the receiver.

3. Key Distribution

While distributing keys, a signer should keep the private key (x) secret and publish the public key (y) and send the public key (y) to the receiver without any secret mechanism.

Signing

Signing of message m should be done as follows:

- first choose an integer k from (1……q-1)
- compute

r = g^(k)*mod(p)*mod(q). If you get r = 0, please try another random value of k and compute again for r except 0.

- Calculate

s=(k^(-1)*(H(m)+xr))*mod(q). If you get s = 0, please try another random value of k and compute again for s except 0.

- The signature is defined by two key elements (r,s). Also, key elements k and r are used to create a new message. Nevertheless, computing r with modular exponential process is a very expensive process and computed before the message is known. Computation is done with the help of the Euclidean algorithm and Fermat's little theorem.

4. Signature Verification

- You use the same hash function (H#) to generate the digest h.
- You then pass this digest off to the verification function, which needs other variables as parameters too.
- Compute the value of w such that: s*w mod q = 1
- Calculate the value of u1 from the formula, u1 = h*w mod q
- Calculate the value of u2 from the formula, u2 = r*w mod q
- The final verification component v is calculated as v = [((gu1 . yu2) mod p) mod q].
- It compares the value of v to the value of r received in the bundle.
- If it matches, the signature verification is complete.

Having understood the functionality of the DSA Algorithm, you must know the advantages this algorithm offers over alternative standards like the RSA algorithm.

Example of a DSA Signing by Alice

Alice creates a message, M, that she wants to send to Bob. She also creates a digital signature, S, for the message using her private key. Then she sends the message and digital signature to Bob. Then act follows,

- Alice Generated an M message she would like to send to Bob.
- Then Alice Generates a random number, k, and computer = (g^k mod p) mod q.
- After that, compute, s = (k^-1 * (H(M) + x*r)) mod q, where H is a cryptographic hash function, and x is Alice's private key.
- Alice's digital signature, S, is the pair (r, s).
- Alice sends the message M and her digital signature S to BoB

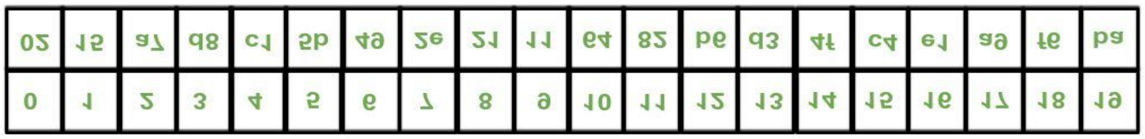
Q4. Explain the Following Types of One-time Password (OTP) Algorithms with Examples: a. Time-based OTP (TOTP) b. HMAC-based OTP (HOTP)

Authentication, the process of identifying and validating an individual is the rudimentary step before granting access to any protected service (such as a personal account). Authentication has been built into the cyber security standards and offers to prevent unauthorized access to safeguarded resources. Authentication mechanisms today create a double layer gateway prior to unlocking any protected information. This double layer of security, termed as two factor authentication, creates a pathway that requires validation of credentials (username/email and password) followed by creation and validation of the **One Time Password (OTP)**. The OTP is a numeric code that is randomly and uniquely generated during each authentication event. This adds an additional layer of security, as the password generated is fresh set of digits each time an authentication is attempted and it offers the quality of being unpredictable for the next created session. The two main methods for delivery of the OTP is:

1. **SMS Based:** This is quite straightforward. It is the standard procedure for delivering the OTP via a text message after regular authentication is successful. Here, the OTP is generated on the server side and delivered to the authenticator via text message. It is the most common method of OTP delivery that is encountered across services.
2. **Application Based:** This method of OTP generation is done on the user side using a specific smartphone application that scans a QR code on the screen. The application is responsible for the unique OTP digits. This reduces wait time for the OTP as well as reduces security risk as compared to the SMS based delivery.

The most common way for the generation of OTP defined by The Initiative For Open Authentication (OATH) is the **Time Based One Time Passwords (TOTP)**, which is a Time Synchronized OTP. In these OTP systems, time is the cardinal factor to generate the unique password. The password generated is created using the current time and it also factors in a secret key. An example of this OTP generation is the Time Based OTP Algorithm (TOTP) described as follows:

1. Backend server generates the secret key
2. The server shares secret key with the service generating the OTP
3. A hash based message authentication code (HMAC) is generated using the obtained secret key and time. This is done using the cryptographic SHA-1 algorithm. Since both the server and the device requesting the OTP, have access to time, which is obviously dynamic, it is taken as a parameter in the algorithm. Here, the Unix timestamp is considered which is independent of time zone i.e. time is calculated in seconds starting from January First 1970. Let us consider "0215a7d8c15b492e21116482b6d34fc4e1a9f6ba" as the generated string from the HMAC-SHA1 algorithm.
4. The code generated is 20 bytes long and is thus truncated to the desired length suitable for the user to enter. Here dynamic truncation is used. For the 20-byte code "0215a7d8c15b492e21116482b6d34fc4e1a9f6ba", each character occupies 4 bits. The entire string is taken as 20 individual one byte string.



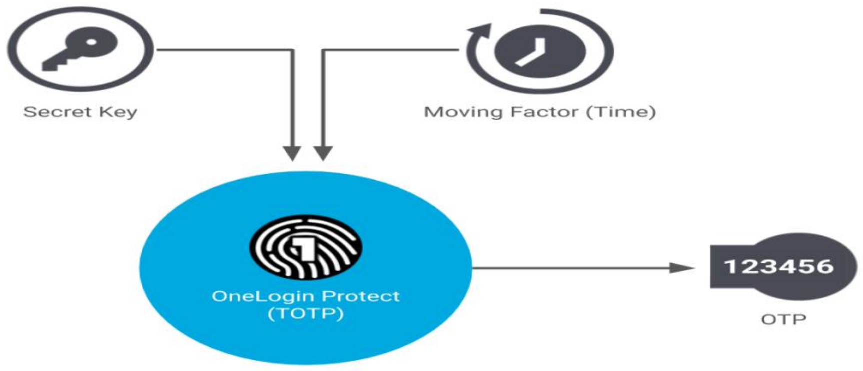
02	15	a7	d8	c1	5b	49	2e	21	11	64	82	b6	d3	4f	c4	e1	a9	f6	ba
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

We look at the last character, here a. The decimal value of which is taken to determine the offset from which to begin truncation. Starting from the offset value, 10 the next 31 bits are read to obtain the string "6482b6d3". The last thing left to do, is to take our hexadecimal numerical value, and convert it to decimal, which gives 1686288083. All we need now are the last desired length of OTP digits of the obtained decimal string, zero-padded if necessary. This is easily accomplished by taking the decimal string, modulo 10 ^ number of digits required in OTP. We end up with "288083" as our TOTP code.
5. A counter is used to keep track of the time elapsed and generate a new code after a set interval of time
6. OTP generated is delivered to user by the methods described above.

Apart from the time-based method described above, there also exist certain mathematical algorithms for OTP generation for example a one-way function that creates a subsequent OTP from the previously created OTP.

The two factor authentication system is an effective strategy that exploits the authentication principles of "something that you know" and "something that you have".The dynamic nature of the latter principle implemented by the One Time Password Algorithm is crucial to security and offers an effective layer of protection against malicious attackers.
Time-based One-time Password (TOTP) is a time-based OTP. The seed for TOTP is static, just like in HOTP, but the moving factor in a TOTP is time-based rather than counter-based.

The amount of time in which each password is valid is called a **timestep**. As a rule, timesteps tend to be 30 seconds or 60 seconds in length. If you haven't used your password within that window, it will no longer be valid, and you'll need to request a new one to gain access to your application.



Limitations and Advantages

While both are far more secure than not using MFA at all, there are limitations and advantages to both HOTP and TOTP. TOTP (the newer of the two technologies) is easy to use and implement, but the time-based element does have a potential for time-drift (the lag between the password creation and use). If the user doesn't enter the TOTP right away, there's a chance it will expire before they do. So the server has to account for that and make it easy for the user to try again without automatically locking them out.

Since HOTP doesn't have the time-based limitation, it's a little more user-friendly, but may be more susceptible to brute force attack. That's because of a potentially longer window in which the HOTP is valid. Some forms of HOTP have accounted for this vulnerability by adding a time-based component to their code, somewhat blurring the lines between these two types of OTP.