## (60 points) Tcpcrypt is a TCP extension designed to make end-to-end encryption of TCP traffic the default, not the exception. To facilitate adoption tcpcrypt provides backwards compatibility with legacy TCP stacks and middle boxes. The paper, “The case for ubiquitous transport-level encryption”, is available for you to go through all the details. You can also go to http://tcpcrypt.org/ and find more information about Tcpcrypt. Read the paper and answer the following questions:

### (20 points) Tcpcrypt is a transport layer security protocol. As we discussed in the class, TLS is also a transport layer security protocol. Compare the differences between Tcpcrypt and TLS protocol (list at list three differences)

* Differences: design, efficiency/ performance/ scalability, and integration requirements
* TLS - application layer
  + requires a detailed handshake process for session establishment, which includes key exchange, server authentication using X.509 certificates, and optionally client authentication [1].
  + [1] <https://www.ibm.com/docs/en/zos/2.4.0?topic=protocols-tls-ssl>
  + This process contains over-head
    - must be deliberately integrated into applications (OpenSSL), and is
    - used selectively, typically only in situations that are deemed particularly “sensitive”
* Tcpcrypt challenges what is considered sensitive by advocating for a shift towards automatic and universal encryption to make the internet more secure by default.
  + Tcpcrypt - transport layer
  + minimizes performance impact of encryption
  + minimizes need for any specific application-level support or configuration [2].
  + provide integrity protection
    - Unlike SSL, provides integrity protection for the TCP session itself,
    - defends against attacks that might reset the connection, insert data into it, or otherwise interfere with its progress.
    - To do this, tcpcrypt adds a MAC option to every TCP packet after the INIT1/INIT2 exchange.
  + this design prioritizes efficiency, reduces complexity of key negotiations, leverages efficient key management strategies,
  + can not only be deployed without disrupting existing network infrastructures, but it is able to support a higher rate of connections than TLS
    - This efficiency makes Tcpcrypt strong in environments with high volumes of network traffic
* [1] <https://www.ibm.com/docs/en/zos/2.4.0?topic=protocols-tls-ssl>
* [2] paper

### (20 points) What is the design consideration of Tcpcrypt protocol? Why is it necessary?

* provide not just encryption and integrity protection, but also a firm foundation upon which higher-level authentication mechanisms can build.
* all TCP sessions are protected against passive eavesdroppers
* all applications that require authentication should, as a side effect, enjoy protection against active man-in-the-middle attacks, all without duplication of effort.
* tcpcrypt is fundamentally different as it requires no user setup.
* encryption ubiquitous/ streamlined/ efficient
* embedded directly into TCP
* simplify encryption across the internet through default end-to-end encryption
* backward compatible with existing TCP implementations
* necessary: current state of internet security
  + traffic is generally encrypted only when considered absolutely necessary due to performance overheads and complexities involved in deploying current encryption techniques.
* Tcpcrypt aims to reduce much of the server load
  + embedding security features at the transport layer to simplify the key exchange processes and offload much of the cryptographic work to clients.
    - This design minimizes the number of messages exchanged during the key negotiation process.
    - For example, Tcpcrypt integrates the initial key exchange directly into the TCP handshake - reducing the overhead associated with separate encryption protocols like TLS that require multiple round-trips to establish a secure connection.
  + ephemeral keys during session initiation,
    - forward secrecy /optimizing the key negotiation time.
  + session caching strategies
    - once a key exchange has occurred between a particular client and server, the resultant session information is cached.
    - Subsequent connections between the same client and server can leverage this cached data to bypass the full key negotiation, significantly speeding up connection times and reducing server load.
  + These features collectively make Tcpcrypt an efficient solution capable of handling high volumes of connections while maintaining low latency and high throughput, thereby facilitating widespread encryption across the internet without significant performance degradation.
  + Therefore, As digital threats continue to evolve and networks continue to expand, Tcpcrypt pushes towards a future where secure communications are the norm, not the exception

### (20 points) Tcpcrypt was originally proposed in 2010. What are the major challenges to adopt Tcpcrypt

* Key exchange is the biggest challenge to tcpcrypt’s performance.
  + Tcpcrypt aims to ensure forward secrecy, which means the compromise of long-term keys does not compromise past session keys. This requires the exchange of a new, ephemeral key each time a connection is established. Unlike a regular TCP connection that primarily relies on simpler handshakes and ACK messages, establishing a secure connection with forward secrecy involves more complex cryptographic operations.
  + The section on Tcpcrypt emphasizes that key exchange is the primary challenge to its performance, particularly due to the computational demands associated with ensuring forward secrecy. Forward secrecy necessitates that each new session between hosts involves a fresh, ephemeral key exchange to prevent the compromise of long-term keys from affecting past communications. This key exchange, particularly when involving public-key cryptography such as RSA, is significantly more computationally intensive than a standard TCP connection setup. For instance, the document mentions that while a server can perform thousands of RSA encryptions per second, it can only manage a small fraction of that number in RSA decryptions, which are more resource-demanding.
  + To alleviate the burden on servers, which typically handle more connections than individual clients, Tcpcrypt strategically shifts the more demanding decryption tasks to the clients. This helps in maintaining server performance by distributing the computational load more evenly across the network. The protocol initiates with the client sending a 'HELLO' message, followed by the server's 'PKCONF' which lists compatible ciphers and may include a cookie for client verification. The client responds with 'INIT1', which includes its ephemeral public key and a nonce, and the server completes the exchange with 'INIT2', encrypting the session key using the client's public key.
  + To further enhance efficiency, Tcpcrypt employs session caching. Once a secure session is established, the session keys are cached for future interactions between the same hosts, eliminating the need for repetitive public-key operations in subsequent connections. This significantly reduces latency and the computational overhead involved in setting up new sessions. Additionally, Tcpcrypt ensures the integrity of each TCP packet by incorporating Message Authentication Codes (MACs), safeguarding against data tampering during transmission. Thus, while the initial key exchange presents performance challenges due to its complexity and demand on computational resources, Tcpcrypt's design mitigates these through intelligent distribution of tasks and session caching, balancing robust security with performance efficiency.
  + Bottlenecks
    - Applications such as the Web often establish more than one TCP connection between the same pair of hosts in rapid succession. When they do this, the amount of data transferred per connection can be quite small—often a few KBytes. If we have to pay the full cost of running the public key operations to establish these short-lived sessions, tcpcrypt can become a bottleneck. Fortunately we can use the same solution as SSL—cache the cryptographic state from one TCP connection and use it to bootstrap subsequent connections.
* Compatible
  + Ideally the key exchange for tcpcrypt would be performed in TCP’s three-way connection setup handshake, as this would add no additional network latency to establishing encrypted sessions
    - We can’t quite achieve this for the first connection between two hosts—rather, we require adding information to the first four packets of the session, as shown in Figure 1. To be backwards compatible with regular TCP, any data we can add to the SYN and SYN/ACK packets must fit within the TCP options field, which is limited to 40 bytes, some of which are required to negotiate other TCP functionality.
    - Careful with the way data is stored in the TCP packet as some middleboxes modify packets.
    - In Tcpcrypt, managing TCP RST (Reset) packets poses a security challenge. While Tcpcrypt ideally requires these packets to contain a valid Message Authentication Code (MAC) for security, legitimate RST packets may not have a MAC if one side has lost the connection state, such as after a crash. To balance security with functionality, Tcpcrypt allows these RST packets without a MAC if they pass basic sequence number checks. For applications needing higher security, like BGP or SSH, Tcpcrypt offers a setting to mandate MACs on all RST packets, though this could delay connection resets if a side loses state.

Despite the advantages, some of the major challenges that limit widespread adoption of Tcpcrypt primarily revolve around issues of, performance, compatibility , and resistance from end users. The authors discuss the challenge of ensuring that the protocol's interaction with existing network infrastructures is stable, specifically its backwards compatibility with legacy TCP stacks and network middleboxes such as NAT devices and firewalls. This compatibility is crucial because Tcpcrypt must operate seamlessly alongside traditional TCP operations without requiring changes to existing network hardware or software. The report emphasizes the design of Tcpcrypt to avoid breaking existing setups, especially given that it modifies the transport layer directly, which is fundamental for ensuring widespread acceptance and deployment.

Another significant challenge relates to the performance impact of Tcpcrypt, particularly on server operations. Although Tcpcrypt is designed to minimize the performance overhead associated with encryption, any form of security implementation will require additional computational resources. The report indicates that while Tcpcrypt performs well under benchmarks and shows the ability to handle large numbers of connections efficiently, it still introduces additional latency and processing overhead compared to unencrypted TCP. For high-throughput servers, the incremental cost of processing encrypted connections can be substantial and impact overall service performance.

Lastly, the report discusses the broader issue of adoption motivation among application developers and end-users. Since the immediate benefits of implementing Tcpcrypt—such as enhanced privacy and security—are not always apparent or valued by users, there might be little incentive for developers to integrate this protocol into their applications. The report argues that without a clear and direct benefit that can be easily perceived by end-users, the motivation to adopt a new, complex technology like Tcpcrypt may remain low. This challenge is compounded by the potential complexity involved in configuring and maintaining encrypted sessions, which could deter adoption further.

Overall, while Tcpcrypt offers a promising approach to enhancing the security of TCP traffic across the Internet, these challenges highlight the technical and practical hurdles that need to be addressed to encourage its widespread use. The successful deployment of Tcpcrypt will likely depend on overcoming these barriers through continued technical refinement and efforts to demonstrate and communicate the value of encrypted network traffic to a broader audience.

(15 points) Figure 1 shows a diagram of how to retrieve public keys using a public key authority. In steps 3, 6, and 7, two nonce N1, N2 are used. Explain why N1, N2 are used in the protocol and what security service can be ensured by using N1, and N2 in the protocol. (Hints: think about non-repudiation service in OSI model)

A diagram of a public-key authority

Description automatically generated

Figure 1 Retrieve Public Keys using a Public-key Authority

### [Notes]Steps 3, 6, 7:

* + 1. 3 - : communicated directly with by encrypting a message with the public key of . This message includes the Initiator's identity and a nonce
       1. - identifier for Initiator A so B can recognize the sender
       2. (generated by A) - needs to be acknowledged by B in a subsequent message to prove B is present and actively involved in the communication
    2. 6 – : Responder B sends an encrypted message back to Initiator A to establish mutual authentication and two-way communication. This message includes nonce 1 and 2:
       1. (originally sent by Initiator A) – by including this, it proves to A that B received and decrypted A’s message, thus authenticating B to A.
       2. (generated by B) - needs to be acknowledged by A in a subsequent message to prove A is present and actively involved in the communication
    3. 7 - : Initiator A sends a message back to Responder B, which includes the nonce provided by B in the previous communication. By sending nonce ​back to Responder B, Initiator A proves it has decrypted B's previous message and is actively participating in the session.
       1. - public key of Responder B used by Initiator A to encrypt the message.
       2. - The nonce generated by Responder B and included in its previous message to Initiator A. This acts as a receipt acknowledgment of the message from B, confirming the integrity of the exchange and providing assurance that both parties can decrypt messages from one another.

### Explain why N1, N2 are used in the protocol:

* + 1. In step 3, Nonce () is included along with the identity of Initiator A in the message encrypted with Responder B’s public key. The inclusion of is important because it acts as a marker of the session’s initiation and ensures that any response from B that includes confirms B's receipt and processing of A's message. By acknowledging in a subsequent message, B is able to prove that it is present and actively engaged in the communication.
    2. In step 6, Responder B responds to Initiator A by including both , which was originally sent by Initiator A, and a new nonce, . The retransmission of authenticates B to A and proves that B received and decrypted A's initial message. Then, by introducing , B contributes a way for A to respond with to confirm and validate communication.
    3. For step 7, Initiator A sends a message back to Responder B which includes encrypted with B's public key. This step ensures that A has successfully received and decrypted B's prior message. Sending back not only verifies A's presence but also confirms a mutual authentication process, as it proves A’s active involvement and completion of the nonce exchange cycle.
    4. Overall The use of nonces and in the protocol serves multiple important functions. Since each nonce is unique to its session, by using it, both parties are able to ensure that their messages are current and no old messages are being replayed by an attacker, which is essential in protecting against replay attacks. This uniqueness also binds each message to a specific session and participant, which means that even if a key were compromised, it would not be possible to use any intercepted messages in a different context. Secondly, nonces help authenticating the parties to each other. When a nonce generated by one party is signed or encrypted and sent back by another party, it ensures that the response is not pre-recorded and that the other party is indeed present and active at the time of the exchange. Then, by successfully processing and responding to the nonce, each party demonstrates their live participation in the communication and implicitly confirms receipt of the message.

### what security service can be ensured by using N1, and N2 in the protocol:

* + 1. The three primary security services that can be ensured by using N1, and N2 in the protocol are Authentication, Data Integrity, and Non-repudiation. The most direct service provided by the nonces is authentication. By including and correctly responding to nonces, both Initiator A and Responder B authenticate each other's identities. This process occurs as each party proves they have received, decrypted, and processed a message that could only have originated from the stated sender, due to the inclusion of a nonce known only within the context of their direct exchange. This can be seen in steps 6 and 7, where B and A authenticate each other by correctly handling the nonces they respectively generated.
    2. Nonces can also ensure data integrity as the presence of a nonce in each communication step allows the recipient to verify that the message has not been tampered with and is indeed the message sent by the peer. If a message were tampered with, the nonce would not match what the recipient expects. Similarly, it provides a way to detect replay attacks since an old message replayed by an attacker would contain an outdated nonce.
    3. Lastly, nonces also support aspects of non-repudiation as a party can not deny receipt of a message after it has engaged in communication with the other party. For example, Responder B cannot deny receiving A’s message in step 3 because step 6 included nonce , which was uniquely generated by Initiator A and included in the initial communication.

## (15 points) Users A and B uses the Diffie-Hellman key exchange technique with a common prime q = 71 and a primitive root α = 7. (Hints: refer to text book Figure 10.1 in Page 303)

### (5 points) If user A has private key XA = 5, what is A’s public key YA ?

Given , , , what is ?

### (5 points) If user B has private key XB = 12, what is B’s public key YB ?

Given , , , what is ?

### (5 points) What is the shared secret key?

A computes

B computes

Their shared key is 30.

## (10 pints) Consider a one-way authentication technique based on asymmetric encryption:

1. A->B:
2. B->A:
3. A->B:

* Note: pg457

1. Explain the protocol;
   * Initiator A sends their identity to Responder B so that B can recognize who is
   * trying to authenticate.
   * B needs this identity to retrieve A's public key from a trusted public key repository or from its own database.
   * Responder B then generates a challenge *,* or randomly generated piece of data, and encrypts it using the public key of Initiator A . Since A is the only one who has the corresponding private key, they should be the only one able to decrypt this message. This step verifies whether A is the owner of and has access to the private key
   * Initiator A responds to B by sending back the decrypted challenge to prove that they successfully decrypted the message using their private key. A successful decryption proves that A possesses the corresponding private key associated with the public key that B used to encrypt . Essentially, If A correctly sends back , it confirms A's identity and completes the authentication process. B now knows that A is indeed the owner of and is therefore the legitimate holder of the identity
2. What type of attack is this protocol susceptible to?
   1. the protocol does not protect against replays (pg458)
   2. A replay attack involves an attacker capturing a valid data transmission and retransmitting it to create an unauthorized effect or gain unauthorized access.
   3. In this protocol, during Step 2, B sends a challenge encrypted with A’s public key . A decrypts this challenge and sends it back in plaintext in Step 3.
   4. However, an attacker could capture the decrypted challenge sent to B by A. Since is sent in plaintext, the attacker can extract it and resend it later to B, pretending to be A.
   5. Since there are no Nonces or Timestamps in this protocol to verify If the correspondence is outdated, B would accept the old as valid and mistakenly authenticating the attacker as A.
      1. It is also important to note that the delays present in in real-world asynchronous communication could make timestamps less effective for mitigating these types of attacks.
   6. In this protocol, during Step 2, B sends a challenge encrypted with A’s public key . A decrypts this challenge and sends it back in plaintext in Step 3. However, an attacker could capture the decrypted challenge sent to B by A. Since is sent in plaintext, the attacker can extract it and resend it later to B, pretending to be A. Since there are no nonces or timestamps in this protocol to verify if the correspondence is outdated, B would accept the old as valid and mistakenly authenticate the attacker as A. However, It is also important to note that the delays present in in real-world asynchronous communication could make timestamps less effective for mitigating these types of attacks.