COMP5631: Cryptography and Security
2025 Spring – Written Assignment Number 4
Handed out: On April 30
Due on May 18 by 23:50 (please submit your solution paper to Canvas)
Please write your name and student ID on your solution paper

Q1. Consider the Digital Signature Standard covered in Lecture 14 and answer the following questions.

- After receiving $m||s||r$ from Alice, Bob had successfully verified the signature (i.e., the signature verification was successful). Today Bob has found a weak collision m' of m, i.e., $h(m) = h(m')$. Is $s||r$ the signature of m' by Alice? Justify your answer. 10 marks **Answer:** Yes, $s||r$ is the signature of m' by Alice. The proof is as follows:

  Step 1: Understanding the DSS Verification Algorithm
  The DSS verification algorithm includes the following steps (modulo q):
  1. $w = s^{-1} \bmod q$
  2. $u_1 = (h(m) \cdot w) \bmod q$, $u_2 = (r \cdot w) \bmod q$
  3. $v = ((g^{u_1} \cdot y^{u_2}) \bmod p) \bmod q$
  4. The signature is valid if and only if $v = r$

  Step 2: Analyzing the Signature Equation
  The signature s satisfies the equation:

$$s \equiv k^{-1} \cdot (h(m) + xr) \bmod q \tag{1}$$

  This can be transformed to obtain the value of k:

$$k \equiv s^{-1} \cdot (h(m) + xr) \bmod q \tag{2}$$

  Step 3: Verifying the Signature for the Original Message
  During verification, we calculate:

$$\begin{aligned} g^{u_1} \cdot y^{u_2} &\equiv g^{h(m)\cdot w} \cdot y^{r\cdot w} \bmod p && (3)\\ &\equiv g^{h(m)\cdot w} \cdot g^{x\cdot r\cdot w} \bmod p && (4)\\ &\equiv g^{(h(m)+xr)\cdot w} \bmod p && (5)\\ &\equiv g^{(h(m)+xr)\cdot s^{-1}} \bmod p && (6)\\ &\equiv g^{k} \bmod p && (7) \end{aligned}$$

  Since $r \equiv g^k \bmod p \bmod q$, we have $v = r$, and the verification succeeds.

  Step 4: Verifying the Signature for the Collision Message
  For the weak collision m' where $h(m) = h(m')$, substituting into the verification algorithm:

$$\begin{aligned} g^{u_1} \cdot y^{u_2} &\equiv g^{h(m')\cdot w} \cdot y^{r\cdot w} \bmod p && (8)\\ &\equiv g^{h(m')\cdot w} \cdot g^{x\cdot r\cdot w} \bmod p && (9)\\ &\equiv g^{(h(m')+xr)\cdot w} \bmod p && (10) \end{aligned}$$

Since $h(m) = h(m')$, this is equivalent to the original verification process:

$$g^{(h(m')+xr)\cdot w} \equiv g^{(h(m)+xr)\cdot w} \bmod p \tag{11}$$

$$\equiv g^k \bmod p \tag{12}$$

Therefore, we still have $r \equiv g^k \bmod p \bmod q$, so $v = r$, and the verification succeeds.

Conclusion: Since $h(m) = h(m')$, the verification process depends on the hash value rather than the message itself. Therefore, the same signature $s||r$ is also valid for the collision message m'. This demonstrates that DSS signatures are vulnerable to hash collisions.

- Should the underlying hash function h have the weak collision resistance property? Justify your answer. 10 marks

  Answer: Yes, the underlying hash function h must have the weak collision resistance property.

  The weak collision resistance (or second preimage resistance) means that given a message m, it should be computationally infeasible to find another message $m' \neq m$ such that $h(m) = h(m')$.

  As demonstrated in the previous part, if an adversary finds a weak collision m' for a message m that has been signed by Alice (with signature $s||r$), then the same signature will also verify correctly for m'. This is because the verification process depends only on the hash value $h(m)$, not on m itself.

  Without weak collision resistance, the following attack becomes possible:

  1. Alice signs a benign document m with signature $s||r$
  2. An adversary finds m' such that $h(m') = h(m)$, where m' contains malicious content
  3. The adversary can claim that Alice signed m', since signature verification will succeed
  4. Alice cannot prove she never signed m', which violates the non-repudiation property

  This attack directly undermines the fundamental security goal of digital signatures, which is to provide unforgeable evidence of the signer's consent to the specific document content. Therefore, weak collision resistance is essential for the security of DSS.

Q2. Consider the format of transmitted messages in PGP from A to B on Slide No. 24 of Lecture 16, where all the security services (signer nonrepudiation, data origin authentication, data integrity and data confidentiality) are required. Suppose that the digital envelop $E_{k_e^{(B)}}(k_s)$ was replaced by an adversary during transmission. Can this replacement be detected by the PGP package in B's computer? Justify your answer briefly. 10 marks

Answer: Yes, the replacement of the digital envelope $E_{k_e^{(B)}}(k_s)$ can be detected by the PGP package in B's computer.

Justification: When the digital envelope $E_{k_e^{(B)}}(k_s)$ is replaced with $E_{k_e^{(B)}}(k_s')$ by an adversary, the following detection mechanism occurs:

1. B uses private key $k_d^{(B)}$ to decrypt the tampered digital envelope, obtaining an incorrect session key k_s' instead of the original k_s.
2. B attempts to decrypt the message body $E_{k_s}(M)$ using the incorrect session key k_s', resulting in garbage data rather than the original message M.
3. During signature verification, B decrypts $E_{k_d^{(A)}}(H(M))$ using A's public key $k_e^{(A)}$ to obtain the hash $H(M)$ of the original message.
4. B computes the hash of the decrypted garbage data and compares it with $H(M)$.
5. Since these hash values will not match, the signature verification fails, alerting B to the tampering.

This demonstrates that the integrity and authentication services provided by the digital signature effectively detect the envelope replacement attack, even though the confidentiality service has been compromised.

Q3. Consider the Kerberos Authentication Protocol described in Lecture 17 and answer the following questions:

1. How does the Client C authenticate the sender after the Client receives the message in Step 2 from the AS? 8 marks
2. How does the Client C check the integrity of the received message in Step 2 from the AS? 7 marks
3. How does the TGS check if the Ticket_{tgs} was indeed issued by the AS or not? 7 marks
4. In addition to the TGS and the AS, who else can verify the validity of the ticket Ticket_{tgs}? Justify your answer briefly. 8 marks

Answer:

1. The client C authenticates the sender (AS) after receiving the message in Step 2 through the following mechanism:
   - In Step 2, the client receives $E_{K_c}(K_{c,tgs}, TGS, time, lifetime)$ from the AS
   - K_c is a key derived from the client's password, known only to the client and the AS
   - The client attempts to decrypt this message using K_c
   - If decryption succeeds and yields meaningful data (session key $K_{c,tgs}$ and expected parameters), then the client confirms that the message was encrypted by an entity knowing K_c (i.e., the AS)
   - Only the legitimate AS could have encrypted the message with the correct K_c, thus authenticating the AS as the sender
2. The client C checks the integrity of the received message in Step 2 through:
   - Successful decryption of $E_{K_c}(K_{c,tgs}, TGS, time, lifetime)$ using K_c, which verifies that the message has not been tampered with during transmission
   - Verification that the decrypted contents contain expected and consistent information:
     - TGS identifier matches the requested TGS

  - The timestamp is recent (within acceptable skew)
  - The lifetime parameters are reasonable
- Any modification to the encrypted message would result in decryption failure or inconsistent content, indicating integrity violation

3. The TGS checks if the Ticket_{tgs} was indeed issued by the AS through:
   - The $\text{Ticket}_{tgs} = E_{K_{tgs}}(C, TGS, time, lifetime, K_{c,tgs})$ is encrypted using K_{tgs}
   - K_{tgs} is a secret key shared only between the AS and TGS
   - The TGS attempts to decrypt the ticket using K_{tgs}
   - Successful decryption proves the ticket was encrypted by an entity knowing K_{tgs} (i.e., the AS)
   - The TGS also verifies the decrypted contents (TGS identifier, timestamp validity) to fully authenticate that the AS issued the ticket
4. No entity other than the AS and TGS can verify the validity of the ticket Ticket_{tgs}.
   Justification:
   - The Ticket_{tgs} is encrypted with K_{tgs}, a secret key shared exclusively between the AS and TGS
   - Verification requires decryption of the ticket, which is only possible with knowledge of K_{tgs}
   - Even the client C who possesses the ticket cannot decrypt or verify its contents
   - In the standard Kerberos protocol, no other system components have access to K_{tgs}
   - This design ensures that the ticket's validation remains strictly controlled, enhancing the security of the authentication process

Q4. Explain why SSL needs an alert protocol, while IPSec does not need such a protocol? 20 marks

Answer: SSL/TLS includes a dedicated Alert Protocol while IPSec does not require such a mechanism due to fundamental differences in their design philosophies, operational layers, and usage contexts:

Reasons why SSL needs an Alert Protocol:

- **Application Layer Operation:** SSL/TLS functions at the application layer (between TCP and application protocols) and needs to provide detailed error information to applications for appropriate handling and user feedback.
- **Connection-Oriented Nature:** SSL/TLS establishes and maintains stateful connections that require proper termination signals. The Alert Protocol allows for graceful connection closure through messages like `close_notify`.
- **Session Management:** SSL/TLS creates sessions with specific parameters that must be maintained. The Alert Protocol provides notifications when session parameters are violated or compromised.
- **Interactive Applications:** SSL/TLS commonly secures interactive applications (web browsers, email clients) where real-time error reporting is essential for user experience.

- **Granular Error Reporting:** Different types of errors require different responses. The Alert Protocol provides specific error codes (certificate expired, bad record MAC, etc.) enabling targeted remediation.
- **Handshake Complexity:** SSL/TLS has a complex multi-step handshake process where failures at different stages require specific handling.

Reasons why IPSec does not need an Alert Protocol:

- **Network Layer Operation:** IPSec functions at the network layer, operating transparently to applications. Error handling can be delegated to existing network protocols like ICMP.
- **Connectionless Design:** IPSec can operate in a connectionless manner, protecting individual packets without maintaining continuous session state.
- **Administrative Configuration:** IPSec is typically configured by system administrators rather than end-users, making extensive real-time alerts less necessary.
- **Reliance on IKE:** Internet Key Exchange (IKE) protocol, which establishes security associations for IPSec, includes its own notification mechanisms for key exchange errors.
- **Implicit Error Handling:** Failed IPSec packets are simply dropped or rejected, with errors handled implicitly by existing IP protocols through timeouts and retransmissions.
- **Transport Independence:** IPSec is designed to protect IP traffic regardless of the transport protocol, making it impractical to implement a universal alert system.

In summary, SSL's Alert Protocol exists because its application-layer, connection-oriented nature requires explicit error communication, while IPSec's network-layer, packet-oriented approach can rely on existing network protocols for error handling.

Q5. I use everyday the SSH in my laptop to access my Unix account. The client authentication is password-based. Assume that I used the SSH in my laptop to connect my Unix account two times yesterday and sent the Unix command "ls -a" from my laptop to the Unix server two times in the two separated SSH connections. Was the command "ls -a" encrypted using the same set of security parameters by the SSH in my laptop? Justify your answer briefly. 20 marks

Answer: No, the command "ls -a" was not encrypted using the same set of security parameters in the two separate SSH connections. This is because:

Key Security Parameters in SSH:

- Each SSH connection establishes a new and independent secure channel with its own unique cryptographic parameters
- During each connection's handshake phase, the following unique elements are generated:
  - Session keys: Unique symmetric encryption keys generated using Diffie-Hellman key exchange
  - Initialization vectors (IVs): Randomly generated for each connection
  - Message Authentication Code (MAC) keys: Unique for each session

  - Sequence numbers: Started fresh with each connection
- SSH implements *perfect forward secrecy* through ephemeral key exchange, ensuring each session uses unique cryptographic parameters
- The SSH protocol deliberately avoids reusing security parameters across different connections to prevent various cryptographic attacks:
  - Replay attacks: Prevented by unique session parameters and sequence counters
  - Traffic analysis: Complicated by different encryptions of identical commands
  - Known-plaintext attacks: Mitigated by different encryption parameters for identical data
- Even though the command ("ls -a") and authentication method (password-based) were identical in both connections, the underlying cryptographic parameters were completely different
- SSH client and server negotiate fresh parameters during each connection's setup phase, regardless of how recently a previous connection was established

Therefore, despite being the same command, "ls -a" would have been encrypted with entirely different security parameters in each of the two separate SSH sessions.