Homework 5: Review

This homework is due **Wednesday, December 10, 2025** at **11:59 p.m.** and counts for 5% of your course grade. Late submissions will be accepted up until the exam starts—with no penalty. Submissions after the exam will not be accepted. If you have a conflict due to travel, interviews, etc., please plan accordingly and turn in your homework early.

We encourage you to discuss the problems and your general approach with other students in the class. However, the answers you turn in must be your own original work, and you are bound by the Honor Code. Solutions must be submitted electronically via Gradescope in PDF. Answers may be as long or short as you like.

Answer the following questions:

1. **Key Terms.** Define **30** of the terms at the end of this document—understand them all:

2. **Applied Cryptography.** Alice and Bob, two CS 4264 alumni, have been stranded on a desert island for several weeks. Alas, these one-time partners are still fighting over whether Bob really pulled his weight on Project 4, and so they've decided to separate themselves until they work out their differences. Alice has built a hut on the beach, while Bob lives high in the forest branches. They plan to communicate silently by tossing coconuts over the treeline.

   Compounding Alice and Bob's misfortune, on this island there also lives an intelligent, literate, and man-eating panther named Mallory. The pair can cooperate to warn each other when they see the animal approaching each others' shelters, but they fear that Mallory will intercept or tamper with their messages in order to make them her next meal.

   (a) Fortunately, Alice and Bob each have an RSA key pair, and each knows the other's public key. Design two protocols, such that Alice and Bob can authenticate each other and agree on a shared secret for use in further communication, one protocol that provides forward secrecy and one that does not.

   (b) After arriving at a shared secret, Alice and Bob plan to use symmetric cryptography to protect their messages, but they disagree about how to apply it. Alice believes it is best to encrypt their plaintext then add a MAC to the ciphertext, while Bob wants to MAC first then encrypt. Explain whose approach is safer, and why.

3. **HTTPS.** A *self-signed certificate* makes the claim that a public key belongs to a particular server, without any trusted certificate authority (CA) to verify it. Browsers display a warning message when a site presents such a certificate, but users often override these warnings. Some websites use self-signed certs to avoid the trouble of obtaining a cert from a trusted CA.

   (a) Briefly explain how using HTTPS with a self-signed certificate provides protection against a passive eavesdropper.

   (b) How might a man-in-the-middle (MITM) attacker compromise connections to a site that uses a self-signed certificate, assuming that the site's users are accustomed to ignoring the browser certificate warnings?

   (c) Briefly compare the security of these designs:

      i. a self-signed certificate for all pages;

      ii. a certificate signed by a trusted CA for all pages.

      iii. a certificate signed by a trusted CA for login pages, and HTTP for non-login pages.

4. **Authentication.** Many organizations (including VT) have deployed two-factor authentication through the use of key fob–sized devices that display pseudorandom codes at a fixed time interval. These codes are generated based on a built-in clock and a device-specific secret s that is also stored on a central authentication server tied to the user's account. Here is one way such a device might work: Let n be the number of minutes that have elapsed since the UNIX epoch; output the first 20 bits of $\text{HMAC}_s(n)$. Successful authentication requires the user's username and password and the current pseudorandom code from the user's device.

   (a) Name three common attacks against authentication that are mitigated by these devices.

   (b) Name one common attack against authentication that is not mitigated.

   Some devices use a counter instead of a clock and generate a single-use code each time the user presses a button on the device. One way this might work is as above, letting n be a register that is initially zero; upon each button press, display the current code for one minute and increment n on the device; on each successful authentication increment n on the server.

   (c) Describe one security advantage of single-use codes compared to time-based codes.

   (d) Describe one usability advantage of single-use codes compared to time-based codes.

   As more and more organizations adopt these devices, end-users are burdened with carrying multiple devices, one for each entity to which they authenticate. Suppose instead that a central authority distributed and managed time-based devices (like the ones described above) for all users and companies, and allowed servers to verify a user's code through a public API.

   (e) Describe at least two serious vulnerabilities that this would introduce.

5. **Web Attacks.** Consider a fictitious social networking site called FacePalm (unofficial motto: “Move fast and facepalm”). The site has millions of users, not all of whom are particularly security-conscious. To protect them, all pages on the site use HTTPS.

   (a) FacePalm’s homepage has a “Delete account” link which leads to the following page:

   ```
   <p>Are you sure you want to delete your account?</p>
   <form action="/deleteuser" method="post">
     <input type="hidden" name="user" value="{{username}}"></input>
     <input type="submit" value="Yes, please delete my account"></input>
   </form>
   ```

   (The web server replaces {{username}} with the username of the logged-in user.)

   The implementation of /deleteuser is given by the following pseudocode:

   ```
   if account_exists(request.parameters['user']):
       delete_account(request.parameters['user'])
       return '<p>Thanks for trying FacePalm!</p>'
   else:
       return '<p>Sorry, ' + request.parameters['user'] + ', an error occurred.</p>'
   ```

   Assume that the attacker knows the username of an intended victim. What’s a simple way that the attacker can exploit this design to delete the victim’s account without any direct contact with the victim or the victim’s browser?

   (b) Suppose that /deleteuser is modified as follows:

   ```
   if validate_user_login_cookie(request.parameters['user'], request.cookies['login_cookie']):
       delete_account(request.parameters['user'])
       return '<p>Thanks for trying FacePalm!</p>'
   else:
       return '<p>Sorry, ' + request.parameters['user'] + ', an error occurred.</p>'
   ```

   where validate_login_cookie() checks that the cookie sent by the browser is authentic and was issued to the specified username. Assume that login_cookie is tied to the user’s account and difficult to guess.)

   Despite these changes, how can the attacker use CSRF to delete the victim’s account?

   (c) Suppose that the HTML form in (a) is modified to include the current user’s login_cookie as a hidden parameter, and /deleteuser is modified like this:

   ```
   if request.parameters['login_cookie'] == request.cookies['login_cookie'] and
           validate_login_cookie(request.parameters['user'], request.cookies['login_cookie']):
       delete_account(request.parameters['user'])
       return '<p>Thanks for trying FacePalm!</p>'
   else:
       return '<p>Sorry, ' + request.parameters['user'] + ', an error occurred.</p>'
   ```

The attacker can still use XSS to delete the victim's account. Briefly explain how.

6. **Secure Programming.** StackGuard is a compiler-based technique for defending against stack-based buffer overflows. It detects memory corruption using a *canary*, a known value stored in each function's stack frame immediately before the return address. Before a function returns, it verifies that its canary value hasn't changed; if it has, the program halts.

   (a) In some implementations, the canary value is a 64-bit integer that is randomly generated each time the program runs. Explain why this prevents the basic form of stack-based buffer overflow attack discussed in lecture.

   (b) What is a security drawback to choosing the canary value at compile time instead of at run time? If the value must be fixed, why is 0 a particularly good choice?

   (c) No matter how the canary is chosen, StackGuard cannot protect against all buffer overflow vulnerabilities. List one kind of bug that can corrupt program execution—even with StackGuard in place.

   (d) You are attempting to exploit a buffer overflow in an application which uses the C `gets()` function. The program appears to be exploitable, but your attack isn't working. Whatever you do, the process immediately crashes as soon as it jumps to the instructions you injected onto the stack. What's going on? How can you bypass this security measure?

   (e) You are developing a simple buffer overflow exploit reminiscent of `target0` from the Application Security. After lots of trial and error, you finally find an input that succeeds—but then then you try again with exactly the same bytes and it doesn't seem to work anymore! What's going on? How can you bypass this security measure?

7. **Communication Protocols**

   You are trying to have a secure conversation with Alice, however, Mallory may be listening.

   (a) You and Alice are setting up a secure channel of communication. Describe how you would establish a key for symmetric encryption using diffie hellman with Alice. Be sure to show (draw a diagram or explain mathematically) what information is sent by you and Alice respectively as well as what information is shared publicly over the network.

   (b) Are there any requirements for the parameters of the protocol above? If so, what are they and why?

   (c) You have conducted the protocol above twice, such that you now have one key for symmetric encryption and another for hashing. Describe how you would authenticate that you are speaking with Alice.

   (d) Explain how you would provide integrity over the channel.

8. **Networking Rehash**

   Bob is an insecure programmer and is using AnotherSketchyCorp's WiFi Network.

(a) Bob is using FTP to download some potentially useful files from a server on the network. However, the command keeps failing with the message *'connection refused'*. Explain which type of FTP is being used, how you can tell, and the source of the error.

(b) Bob believes there may be a malicious entity on the network, searching for open ports on devices. Explain how Bob can locate the malicious entity's IP address.

9. **Cipher Modes of Operation**

Bob is trying to securely store his secret picture in an encrypted fashion. Please answer the following:

(a) Before he determines a method of encryption, Bob needs criteria on which to judge the security of his cipher output. Please define and explain the criteria Bob should use.

(b) Consider the plain text and encrypted image below. Which cipher mode of operation is Bob using and why does this perform badly?

Figure 1: Bob's secret image

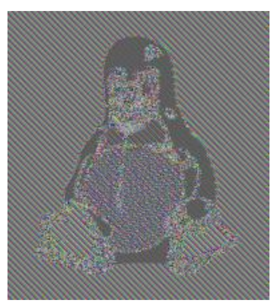
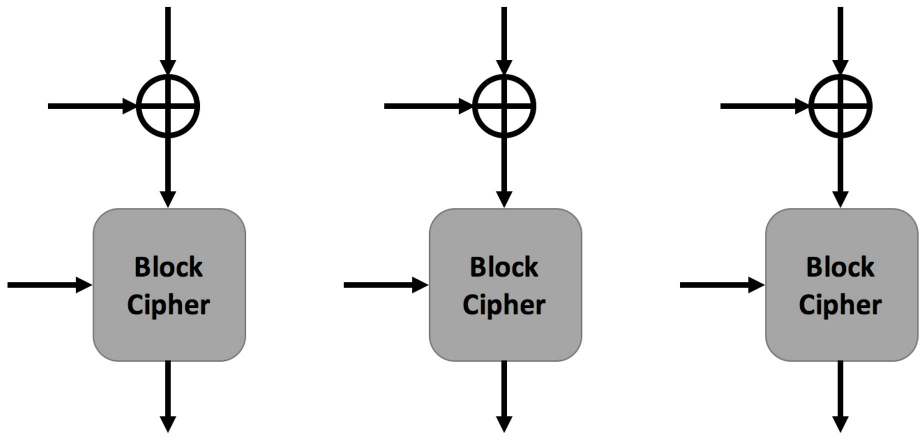


Figure 2: Bob's encrypted image

(c) After Alice tells Bob about the dangers of his current encryption scheme, Bob recommends method in which the output of one block cipher influences the encryption of the next. Please draw this method and explain the dangers of using it.

(d) After Bob realizes his mistake, he finally agrees to listen to Alice's ideas on block cipher modes of operation. She recommends using counter mode. Explain why this method is better than the two methods previously attempted by Bob.



10. **Ethics.** Consider the following scenario: A worm is infecting systems by exploiting a bug in a popular server program. It is spreading rapidly, and systems where it is deleted quickly become reinfected. A security researcher decides to launch a counterattack in the form of a defensive worm. Whenever a break-in attempt comes from a remote host, the defensive worm detects it, heads off the break-in, and exploits the same bug to spread to the attacking host. On that host, it deletes the original worm. It then waits until that system is attacked, and the cycle repeats.

    (a) Many people would claim that launching such a counterattack in this scenario is ethically unacceptable. Briefly argue in support of this view.

    (b) Are there circumstances or conditions under which an active security counterattack would be ethically justified? Briefly explain your reasoning.

acceptable use
access control list (ACL)
account expiration
Advanced Encryption Standard (AES)
advanced persistent threat (APT)
adware
anomaly-based monitoring
application black-listing
application firewall
application white-listing
application-level gateway (ALG)
ARP poisoning
ASLR
asymmetric key algorithm
attack vector
attribute-based access control (ABAC)
audit trails
authentication
authorization
availability
backdoors
BGP
block cipher
bluejacking
bluesnarfing
botnet
bring your own device (BYOD)
brute-force attack
buffer overflow
CAPTCHA
certificate authority (CA)
certificate revocation list (CRL)
certificates
chain of custody
Challenge Handshake Authentication Protocol (CHAP)
choose your own device (CYOD)
CIA triad
cipher
cipher block chaining (CBC)
closed-circuit television (CCTV)
CMAC
Common Vulnerabilities and Exposures (CVE)
computer security audits
confidentiality
content filters
cookies
cross-site request forgery (XSRF)
cross-site scripting (XSS)
cryptanalysis attack
cryptographic hash functions
cryptography
CSRF
cyclic redundancy check (CRC)
Data Encryption Standard (DES)
data loss prevention (DLP)
default account
defense in depth
demilitarized zone (DMZ)
denial-of-service (DoS)
DHCP
dictionary attack
Diffie-Hellman key exchange
digital signature
discretionary access control (DAC)
distributed denial-of-service (DDoS)
DNS
DNS poisoning
domain name kiting
due care
due diligence
due process
dumpster diving
Easter egg
eavesdropping
electromagnetic interference (EMI)
elliptic curve cryptography (ECC)
encryption
ephemeral key
ethical hacker
evil twin
explicit allow
explicit deny
Extensible Authentication Protocol (EAP)
fail-open mode
Faraday cage
federated identity management (FIM)
firewall
FISMA
forensics
fork bomb
FTP
fuzz testing
GDPR
Group Policy
group-based access control (GBAC)
hardening
hardware security module (HSM)
hash function
HIPAA
HMAC
hoax
honeynet
honeypot
host-based intrusion detection system (HIDS)
hotfix
HOTP
HSTS
HTML
HTTP
HTTP proxy
HTTPS
hypervisor
ICMP
IDS
IKE
implicit deny
incident management
incident response
information assurance
information security
input validation
integer overflow
integrity
Internet content filter
Internet Protocol Security (IPsec)
IoT
IP
IP proxy
IPv4
IPv6
IV attack
Kerberos
key
key escrow
key stretching
least privilege
Lightweight Directory Access Protocol (LDAP)
logic bomb
MAC filtering
MAC flooding
malware
man-in-the-browser
man-in-the-middle
mandatory access control (MAC)
mandatory access control (MAC)
mantrap
mean time between failures
message authentication code (MAC)
Message-Digest Algorithm 5 (MD5)
multifactor authentication (MFA)
mutual authentication
NDA
network access control (NAC)
network address translation (NAT)
network intrusion detection system (NIDS)
network intrusion prevention system (NIPS)
network perimeter
NFC
non-repudiation
nonce
OAuth
onboarding
one-time pad
Online Certificate Status Protocol (OCSP)
packet filtering
password cracker
patch
patch management
penetration testing
permissions
personally identifiable information (PII)
pharming
phishing
piggybacking
ping flood
Ping of Death
policy
pop-up blocker
POP3
port scanner
pretexting
Pretty Good Privacy (PGP)
private key
privilege escalation
promiscuous mode
Protected Extensible Authentication Protocol (PEAP)
protocol analyzer
proxy server
pseudorandom function
pseudorandom permutation
public key
public key cryptography
public key infrastructure (PKI)
qualitative risk assessment
quantitative risk assessment
radio frequency interference (RFI)
rainbow table
random function
random permutation
ransomware
RAT
Remote Authentication Dial-In User Service (RADIUS)
remote code execution
replay attack
residual risk
RFID
risk
risk acceptance
risk assessment
risk avoidance
risk management
risk mitigation
risk reduction
risk transference
role-based access control (RBAC)
rootkit
RSA
salted hash
sandbox
secure boot
secure code review
secure coding concepts
Secure Hash Algorithm (SHA)
Secure Shell (SSH)
Secure Sockets Layer (SSL)
security log files
security posture
security posture assessment
security template
security tokens
separation of duties
SFTP
shoulder surfing
signature-based monitoring
Simple Network Management Protocol (SNMP)
single point of failure
single sign-on
SMTP
Smurf attack
spam
spear phishing
spoofing
spyware
SQL
SSH
stateful packet inspection
steganography
stream cipher
symmetric key algorithm
SYN flood
Systems Development Life Cycle
tailgating
TCP
TCP reset attack
TCP/IP hijacking
TCSEC
teardrop attack
TEMPEST
Temporal Key Integrity Protocol
threat modeling
threat vector
time bomb
time of day restriction
TOTP
Transport Layer Security
Trojan horse
trusted platform module (TPM)
typosquatting
UDP
UDP flood attack
UEFI
uninterruptible power supply
URL
User Account Control
virtual machine
virtual private network (VPN)
virus
vishing
VLAN
VLAN hopping
vulnerability
vulnerability assessment
vulnerability management
vulnerability scanning
war-chalking
war-dialing
war-driving
watering hole attack
web of trust
web security gateway
whaling
white hat
white-box testing
Wi-Fi Protected Access (WPA)
Wi-Fi Protected Setup (WPS)
Wired Equivalent Privacy (WEP)
wiretapping
worm
X.509
zero day attack
zero trust
zombie