# CS 161 Computer Security



## Q1 *The Red Hood* (15 points)

Jason Todd decides to launch a communications channel in order to securely communicate with the Red Hood Gang over an insecure channel. Jason wants to test different schemes in his attempt to attain confidentiality and integrity.

Notation:

- $M$ is the message Jason sends to the recipient.
- $K_1$, $K_2$, and $K_3$ are secret keys known to only Jason and the recipient.
- ECB, CBC, and CTR represent block cipher encryption modes for a secure block cipher.
- Assume that CBC and CTR mode are called with randomly generated IVs.
- $H$ is SHA2, a collision-resistant, one-way hash function.
- HMAC is the HMAC construction from lecture.

Decide whether each scheme below provides confidentiality, integrity, both, or neither. For all question parts, the ciphertext is the value of $C$; $t$ **is a temporary value that is not sent as part of the ciphertext.**

Q1.1 (3 points)

$$t = \mathsf{CBC}(K_1, M) \qquad C_1 = \mathsf{ECB}(K_2, t) \qquad C_2 = \mathsf{HMAC}(K_3, t) \qquad C = (C_1, C_2)$$

- ○ Confidentiality only
- ○ Integrity only
- ● Both confidentiality and integrity
- ○ Neither confidentiality nor integrity

**Solution:** This is a typical encrypt-then-MAC scheme with a twist: Instead of including the ciphertext $t$ directly, the ciphertext (but not the MAC) is additionally encrypted with ECB mode. Even though both the HMAC and ECB leak information about $t$, $t$ doesn't leak information about the plaintext, so the scheme is confidential. The HMAC over $t$ ensures that the input passed to CBC decryption can't be tampered with, so the scheme maintains integrity.

Q1.2 (3 points)

$$t = \mathsf{ECB}(K_1, M) \qquad C_1 = \mathsf{CBC}(K_2, t) \qquad C_2 = \mathsf{HMAC}(K_3, t) \qquad C = (C_1, C_2)$$

- ○ Confidentiality only
- ● Integrity only
- ○ Both confidentiality and integrity
- ○ Neither confidentiality nor integrity

**Solution:** Notice that $t$ leaks information about the message because it uses insecure ECB mode. $C_2$ then leaks information about $t$, which leaks information about the plaintext, so confidentiality is lost (in this case, $C_2$ is deterministic). However, because the HMAC is computed over $t$, which is decryptable to the message, integrity is maintained.

Q1.3 (3 points)

$$C_1 = \mathsf{ECB}(K_1, M) \qquad C_2 = H(K_2 \| C_1) \qquad C = (C_1, C_2)$$

- ○ Confidentiality only
- ○ Integrity only
- ○ Both confidentiality and integrity
- ● Neither confidentiality nor integrity

**Solution:** $C_1$ leaks information about $M$ it uses insecure ECB mode, so confidentiality is lost. $C_2$ does not maintain integrity as it vulnerable to length extension attacks—an attacker could forge $C_2' = H(K_2 \| C_1 \| x)$ and $C_1' = C_1 \| x$, which would be accepted by anyone verifying the hash.

Q1.4 (3 points) For this subpart only, assume that $i$ a monotonically, increasing counter incremented per message.

$$C_1 = \mathsf{CTR}(K_1, M) \qquad C_2 = \mathsf{HMAC}(i, H(C_1)) \qquad C = (C_1, C_2)$$

*Clarification issued during exam:* Assume that the counter, $i$, starts at 0.

- ● Confidentiality only
- ○ Integrity only
- ○ Both confidentiality and integrity
- ○ Neither confidentiality nor integrity

**Solution:** Because $i$ is a known value, the key to the HMAC can be predicted, and the scheme does not maintain integrity. However, since the ciphertext is encrypted with secure CTR mode, and the insecure HMAC is computed only over the ciphertext, the scheme maintains confidentiality.

Q1.5 (3 points) For this subpart only, assume that the block size of block cipher is $n$, the lengths of $K_1$ and $K_2$ are $n$, the length of $M$ must be $2n$, and the length of the hash produced by $H$ is $2n$.

$$C_1 = \mathsf{CBC}(K_1, K_2) \qquad C_2 = M \oplus C_1 \oplus H(C_1) \qquad C = (C_1, C_2)$$

- ○ Confidentiality only
- ○ Integrity only
- ○ Both confidentiality and integrity
- ● Neither confidentiality nor integrity

**Solution:** Notice that the attacker already knows the value of $C_1$ since it is sent with the ciphertext. Because of this, the adversary can just compute $H(C_1)$ then $C_2 \oplus C_1 \oplus H(C_1)$ in order to recover $M$, so the scheme is not confidential. Additionally, there is no MAC, so the scheme does not have integrity.

## Q2 *PRNGs and Diffie-Hellman Key Exchange* **(15 points)**

Eve is an eavesdropper listening to an insecure channel between Alice and Bob.

1. Alice and Bob each seed a PRNG with different random inputs.
2. Alice and Bob each use their PRNG to generate some pseudorandom output.
3. Eve learns both Alice's and Bob's pseudorandom outputs from step 2.
4. Alice, without reseeding, uses her PRNG from the previous steps to generate $a$, and Bob, without reseeding, uses his PRNG from the previous steps to generate $b$.
5. Alice and Bob perform a Diffie-Hellman key exchange using their generated secrets ($a$ and $b$). Recall that, in Diffie-Hellman, neither $a$ nor $b$ are directly sent over the channel.

For each choice of PRNG constructions, select the minimum number of PRNGs Eve needs to compromise (learn the internal state of) in order to learn the Diffie-Hellman shared secret $g^{ab} \bmod p$. Assume that Eve always learns the internal state of a PRNG between steps 3 and 4.

Q2.1 (3 points) Alice and Bob both use a PRNG that outputs the same number each time.

- ● (A) Neither PRNG
- ○ (B) One PRNG
- ○ (C) Both PRNGs
- ○ (D) Eve can't learn the secret
- ○ (E) ——
- ○ (F) ——

**Solution:** Eve observes the PRNG outputs. Since both PRNGs output the same number each time, Eve also learns the values of $a$ and $b$. She can use this to compute the shared secret $g^{ab} \bmod p$ without compromising any PRNGs.

Q2.2 (3 points) Alice uses a secure, rollback-resistant PRNG. Bob uses a PRNG that outputs the same number each time.

- ● (G) Neither PRNG
- ○ (H) One PRNG
- ○ (I) Both PRNGs
- ○ (J) Eve can't learn the secret
- ○ (K) ——
- ○ (L) ——

**Solution:** Eve observes Bob's PRNG output and learns the value of $b$. Alice will send $g^a \bmod p$ in his half of the exchange. Eve can compute $(g^a)^b \bmod p$ to learn the shared secret without compromising any PRNGs.

Q2.3 (3 points) Alice and Bob both use a secure, rollback-resistant PRNG.

○ (A) Neither PRNG

● (B) One PRNG

○ (C) Both PRNGs

○ (D) Eve can't learn the secret

○ (E) —

○ (F) —

> **Solution:** Eve only needs to compromise one PRNG to learn one of the secrets. For example, if Eve compromises Alice's PRNG, she learns $a$ and can compute $(g^b)^a \bmod p$ to learn the shared secret (because Bob sends $g^b \bmod p$ in his half of the exchange). Alternatively, if Eve compromises Bob's PRNG, she learns $b$ and can compute $(g^a)^b \bmod p$ to learn the shared secret (because Alice sends $g^a \bmod p$ in her half of the exchange).

For the rest of the question, consider a different sequence of steps:

1. Alice and Bob each seed a PRNG with different random inputs.
2. Alice uses her PRNG from the previous step to generate $a$, and Bob uses his PRNG from the previous step to generate $b$.
3. Alice and Bob perform a Diffie-Hellman key exchange using their generated secrets ($a$ and $b$).
4. Alice and Bob, without reseeding, each use their PRNG to generate some pseudorandom output.
5. Eve learns both Alice's and Bob's pseudorandom outputs from step 4.

As before, assume that Eve always learns the internal state of a PRNG between steps 3 and 4.

Q2.4 (3 points) Alice and Bob both use a secure, but not rollback-resistant PRNG.

○ (G) Neither PRNG

● (H) One PRNG

○ (I) Both PRNGs

○ (J) Eve can't learn the secret

○ (K) —

○ (L) —

> **Solution:** Because there is no rollback resistance, if Eve compromises one PRNG, Eve can deduce previous PRNG output and learn a secret (either $a$ or $b$), which is enough to compute the shared secret (as in the previous part).

Q2.5 (3 points) Alice and Bob both use a secure, rollback-resistant PRNG.

○ (A) Neither PRNG

○ (B) One PRNG

○ (C) Both PRNGs

● (D) Eve can't learn the secret

○ (E) —

○ (F) —

> **Solution:** Even if Eve compromises both PRNGs, because they are rollback-resistant, Eve cannot deduce the secrets $a$ and $b$ (i.e. previous PRNG output).

## Q3 *Bonsai* (10 points)

EvanBot wants to store a file in an *untrusted* database that the adversary can read and modify.

Before storing the file, EvanBot computes a hash over the contents of the file and stores the hash separately. When retrieving the file, EvanBot re-computes a hash over the file contents, and, if the computed hash doesn't match the stored hash, then EvanBot concludes that the file has been tampered with.

*Clarification during exam:* Assume that EvanBot does not know if hashes or files have been modified in the untrusted datastore.

Q3.1 (4 points) What assumptions are needed for this scheme to guarantee integrity on the file? Select all that apply.

■ (A) An attacker cannot tamper with EvanBot's stored hash

☐ (B) EvanBot has a secret key that nobody else knows

☐ (C) The file is at most 128 bits long

■ (D) EvanBot uses a secure cryptographic hash

☐ (E) None of the above

☐ (F) ——

**Solution:** In order to guarantee integrity on this file, we need two assumptions to hold.

First, the attacker shouldn't be able to tamper with the stored hash. If they could, then the attacker could simply replace the file with an arbitrary file of the attacker's choice, and replace the original stored hash with a hash over this new file. EvanBot's check on the file would succeed.

If EvanBot had a secret key, then EvanBot could change the scheme to use a MAC using the secret key instead of a hash. However, since this scheme uses a hash, a secret key doesn't help us here.

The file being 128 bits long has no relevance to this question.

Finally, the hash must be a secure cryptographic hash. A quick counterexample: if EvanBot used a hash function that mapped every input to the hash value "1", then the attacker could choose an input of their choice, and the check on the hash would always succeed.

For the rest of this question, we refer to two databases: a *trusted database* that an adversary cannot read or modify, and an *untrusted database* that an adversary can read and modify.

Assume that $\mathsf{H}$ is a secure cryptographic hash function and $\|$ denotes concatenation.

EvanBot creates and stores four files, $F_1$, $F_2$, $F_3$, and $F_4$, in the untrusted database. EvanBot also computes and stores a hash on each file's contents in the untrusted database:

$$h_1 = \mathsf{H}(F_1) \qquad h_2 = \mathsf{H}(F_2) \qquad h_3 = \mathsf{H}(F_3) \qquad h_4 = \mathsf{H}(F_4)$$

Then, EvanBot stores $h_{\text{root}} = \mathsf{H}(h_1 \| h_2 \| h_3 \| h_4)$ in the *trusted* database.

Q3.2 (3 points) If an attacker modifies $F_2$ stored on the server, will EvanBot be able to detect the tampering?

● (G) Yes, because EvanBot can compute $h_{\text{root}}$ and see it doesn't match the stored $h_{\text{root}}$

○ (H) Yes, because EvanBot can compute $h_2$ and see it doesn't match the stored $h_2$

○ (I) No, because the hash doesn't use a secret key

○ (J) No, because the attacker can re-compute $h_2$ to be the hash of the modified file

○ (K) ——

○ (L) ——

**Solution:**

In this scheme, we have a trusted database that an adversary cannot read or modify. Because we have this trusted database, it's possible to ensure integrity through the use of hashes, despite them not being signed (like MAC's).

Let's walk through what happens if an attacker modifies $F_2$. If the attacker modifies this file and nothing else, then it's easy for Bot to detect tampering: Bot just has to recompute a hash over $F_2$ and realize that it doesn't match $h_2$.

However, an attacker can also modify $h_2$ to be the hash of the malicious file, since it's in the untrusted database. Because of this, in order to detect tampering, Bot has to use the only thing that the attacker doesn't have access to: $h_{root}$, which is stored in the trusted database.

Based on this information: the simplest way to verify the integrity of $F_2$ is to:

1. Recompute a hash over $F_1$, $F_2$, $F_3$, and $F_4$.
2. Recompute $h_{root}$ using these hashes.
3. Compare this $h_{root}$ to the stored version of $h_{root}$.

If the attacker modifies $F_2$, then Bot will **always** be able to detect the tampering, since the check on the root hashes will fail.

Q3.3 (3 points) What is the minimum number of hashes EvanBot needs to compute to verify the integrity of all four files?

○ (A) 1 ○ (C) 3 ● (E) 5

○ (B) 2 ○ (D) 4 ○ (F) More than 5

**Solution:**

Because the attacker has the ability to modify all files and hashes in the insecure database, Bot needs to make sure that the attacker hasn't modified any single file/hash pair. To do this, Bot need to follow the procedure discussed in Q3.2's solution - recompute a hash over each file (4 hashes in total), and recompute the root hash (1 hash in total).