# CS 161 Fall 2024 — Introduction to Computer Security — Exam Prep 6

## Q1 *Bob's Birthday* (11 points)

It's Bob's birthday! Alice wants to send an encrypted birthday message to Bob using ElGamal.

Recall the definition of ElGamal encryption:

- $b$ is the private key, and $B = g^b \bmod p$ is the public key.
- $\mathsf{Enc}(B, M) = (C_1, C_2)$, where $C_1 = g^r \bmod p$ and $C_2 = M \times B^r \bmod p$
- $\mathsf{Dec}(b, C_1, C_2) = C_1^{-b} \times C_2 \bmod p$

Q1.1 (2 points) Mallory wants to tamper with Alice's message to Bob. In response, Alice decides to sign her message with an RSA digital signature. Bob receives the signed message and verifies the signature successfully. Can he be sure the message is from Alice?

- ● Yes, because RSA digital signatures are unforgeable.
- ○ Yes, because RSA encryption is IND-CPA secure.
- ○ No, because Mallory could have blocked Alice's message and replaced it with a different one.
- ○ No, because Mallory could find a different message with the same hash as Alice's original message.

**Solution:** RSA digital signatures, when paired with a secure hash function, are believed to be unforgeable. See the textbook for a game-based definition of what exactly we mean by unforgeable.

As we discussed in class, ElGamal is malleable, meaning that a man-in-the-middle can change a message in a *predictable* manner, such as producing the ciphertext of the message $2 \times M$ given the ciphertext of $M$.

Q1.2 (3 points) Consider the following modification to ElGamal: Encrypt as normal, but further encrypt portions of the ciphertext with a block cipher $E$, which has a block size equal to the number of bits in $p$. In this scheme, Alice and Bob share a symmetric key $K_{\mathsf{sym}}$ known to no one else.

Under this modified scheme, $C_1$ is computed as $E_{K_{\mathsf{sym}}}(g^r \bmod p)$ and $C_2$ is computed as $E_{K_{\mathsf{sym}}}(M \times B^r \bmod p)$. Is this scheme still malleable?

- ○ Yes, because block ciphers are not IND-CPA secure encryption schemes
- ○ Yes, because the adversary can still forge $k \times C_2$ to produce $k \times M$
- ● No, because block ciphers are a pseudorandom permutation
- ○ No, because the adversary isn't able to learn anything about the message $M$

**Solution:** While block ciphers aren't IND-CPA secure, they are secure when encrypting "random-looking" values because of their properties as pseudorandom permutations. As long as the values you encrypt are unique, the output of the block cipher will always be secure. ElGamal's $C_1$ and $C_2$ both appaer random.

Additionally, because block ciphers are a PRP, the scheme is no longer malleable, because modifying the ciphertext in any way causes an unpredictable change to the result of decrypting the block cipher with $D_{K_{\mathsf{sym}}}$.

The remaining parts are independent of the previous part.

For Bob's birthday, Mallory hacks into Bob's computer, which stores Bob's private key $b$. She isn't able to read $b$ or overwrite $b$ with an arbitrary value, but she can multiply the stored value of $b$ by a random value $z$ known to Mallory.

Mallory wants to send a message to Bob that appears to decrypt as normal, but **using the modified key** $b \cdot z$. Give a new encryption formula for $C_1$ and $C_2$ that Mallory should use. Make sure you only use values known to Mallory!

*Clarification during exam:* For subparts 3 and 4, assume that the value of B is unchanged.

Q1.3 (3 points) Give a formula to produce $C_1$, encrypting $M$.

**Solution:** Mallory should send $g^r$ with some randomly chosen $r$, as usual.

Q1.4 (3 points) Give a formula to produce $C_2$, encrypting $M$.

**Solution:** Mallory should send $C_2 = m \times B^{rz} \mod p$.

## Q2 *Cryptography: EvanBot Signature Scheme* **(12 points)**

EvanBot decides to make a signature scheme!

To initialize the system, a Diffie-Hellman generator $g$ and prime $p$ are generated and shared to all parties. The private key is some $x \bmod p$ chosen randomly, and the public key is $y = g^x \bmod p$.

To sign a message $m$ such that $2 \leq m \leq p-2$:

1. Choose a random integer $k$ between 2 and $p-2$.
2. Set $r = g^k \bmod p$.
3. Set $s = (\mathsf{H}(m) - xr)k^{-1} \bmod (p-1)$. If $s = 0$, restart from Step 1.
4. Output $(r, s)$ as the signature.

*Clarification after exam: $k$ is chosen to be coprime to $p-1$.*

To verify, check that $g^{H(m)} \equiv$ _____ $\bmod p$. We will fill in this blank in the next few subparts.

Q2.1 (3 points) Select the correct expression for $H(m)$ in terms of $x, r, k, s$ and $p-1$.
*HINT: Use Step 3 of the signature algorithm.*

- ○ $k(xr)^{-1} + s \bmod (p-1)$
- ○ $k^{-1} + xr \bmod (p-1)$
- ○ $ks - xr \bmod (p-1)$
- ● $ks + xr \bmod (p-1)$

**Solution:** From step 3:

$$\begin{aligned} s &\equiv (\mathsf{H}(m) - xr)k^{-1} \bmod (p-1) \\ sk &\equiv \mathsf{H}(m) - xr \bmod (p-1) \\ sk + xr &\equiv H(m) \bmod (p-1) \\ H(m) &\equiv ks + xr \bmod (p-1) \end{aligned}$$

Q2.2 (4 points) Using the previous result, select the correct value for the blank in the verification step. *HINT: Replace the $H(m)$ in $g^{H(m)}$ with your results from the previous subpart.*

- ○ $y^s r^2 \bmod p$
- ○ $r^y r^s \bmod p$
- ● $y^r r^s \bmod p$
- ○ $r g^{yr} \bmod p$

**Solution:**

$$\begin{aligned} & g^{ks+xr} \bmod p \\ & \equiv g^{ks} \cdot g^{xr} \bmod p \\ & \equiv (g^k)^s \cdot (g^x)^r \bmod p \\ & \equiv r^s y^r \equiv y^r r^s \bmod p \end{aligned}$$

Q2.3 (5 points) Show how to recover the private key $x$ if a signature is generated such that $s = 0$ (i.e. the check on Step 3 is ignored).

**Solution:** If $s = 0$, then $0 \equiv (H(m) - xr)k^{-1} \bmod (p-1)$ per Step 3, which means $xr = H(m) \bmod (p-1)$ and we can solve for $x$. Note that $k$ is implicitly coprime to $p-1$ by construction in the protocol.

## Q3 *Hash Functions: YAAS (Yet Another Authentication Scheme)* **(8 points)**

EvanBot decides to design a new authentication scheme.

Define `pwd` to be a secure password that only EvanBot knows.

Also, define $H^k$ to be the result of repeatedly applying $H$, a cryptographically secure hash function, $k$ times. Note: $H^0(x) = x$.

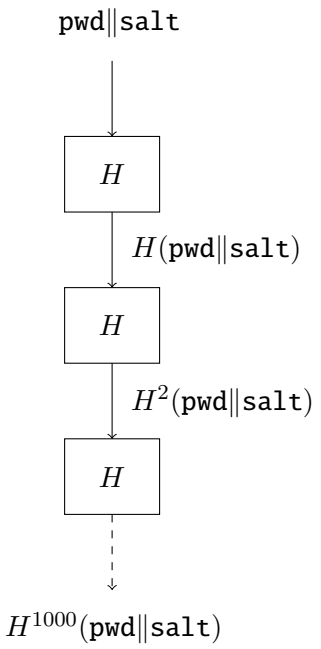
$$H^k(x) = \underbrace{H(H(...H}_{k \text{ times}}(x)))$$

To sign up:

1. EvanBot securely generates a 128-bit salt `salt`.
2. EvanBot sends $H^{1000}(\texttt{pwd}\|\texttt{salt})$ to the server.
3. The server maps EvanBot's username to a variable called `stored`. The server sets `stored` to be the value received in Step 2.

To log in for the $n$-th time ($n$ starts at 1):

1. EvanBot sends $H^{1000-n}(\texttt{pwd}\|\texttt{salt})$ to the server.
2. The server checks whether [ANSWER TO Q3.1].
3. If Step 2 succeeds, the server updates `stored` to [ANSWER TO Q3.2].



Q3.1 (1 point) Let `Step1` be the value received in Step 1 of the login process. Select the correct option for the blank in Step 2.

- ● $H(\texttt{Step1}) = \texttt{stored}$
- ○ $H(\texttt{stored}\|\texttt{salt}) = \texttt{Step1}$
- ○ $H(\texttt{stored}) = \texttt{Step1}$
- ○ $H(\texttt{Step1}\|\texttt{salt}) = \texttt{stored}$

**Solution:**

When EvanBot first signs up, `stored` is set to:

$\texttt{stored} = H^{1000}(\texttt{pwd}\|\texttt{salt})$.

To log in for the first time, set $n = 1$, so $1000 - n = 999$. EvanBot sends this value to the server:

$\texttt{Step1} = H^{999}(\texttt{pwd}\|\texttt{salt})$

From these two values, we can write:

$H(H^{999}(\texttt{pwd}\|\texttt{salt})) = H^{1000}(\texttt{pwd}\|\texttt{salt})$

$H(\texttt{Step1}) = \texttt{stored}$

At this point, `stored` gets set to $H^{999}(\texttt{pwd}\|\texttt{salt})$.

On the next login, $n = 2$, so EvanBot sends this value to the server:

$\texttt{Step1} = H^{998}(\texttt{pwd}\|\texttt{salt})$

From these two values, we can again write:

$H(H^{998}(\texttt{pwd}\|\texttt{salt})) = H^{999}(\texttt{pwd}\|\texttt{salt})$

This pattern continues for subsequent logins.

Q3.2 (1 point) Select the correct option for the blank in Step 3.

- ○ `stored` (no update needed)
- ● `Step1`
- ○ $H(\texttt{stored}\|\texttt{salt})$
- ○ $H^2(\texttt{Step1})$

**Solution:** Since the server always wants to store the next value in the hash chain, we need to update `stored` to current client value.

Q3.3 (1 point) Does the server need to know `salt` in order to complete the login process?

- ○ Yes
- ● No

Q3.4 (1 point) Eventually, EvanBot logs in by sending $H^{700}(\texttt{pwd}\|\texttt{salt})$. Given only `pwd`, `salt`, and $H^{700}(\texttt{pwd}\|\texttt{salt})$, how many calls to $H$ does EvanBot need to make on the next login request?

○ 0 ○ 1 ● 699 ○ 700

**Solution:** For the next login request, EvanBot needs to compute $H^{699}(\texttt{pwd}\|\texttt{salt})$.

(This is because on the next login, $n$ increases by 1, so $1000 - n$ decreases by 1. Specifically, $1000 - n$ changes from 700 to 699.)

From the previous subpart, we can write:

$H(H^{699}(\texttt{pwd}\|\texttt{salt})) = H^{700}(\texttt{pwd}\|\texttt{salt})$

$H$ is a secure cryptographic hash function, so given the output of the 700th hash (right-hand side), we have no way to find an input to the 700th hash ($H^{699}$, input to $H$ on the left-hand side).

The only way to compute $H^{699}(\texttt{pwd}\|\texttt{salt})$ is to take `pwd` and `salt` and compute the 699 hashes from the start again.

Note: In practice, when computing $H^{1000}(\texttt{pwd}\|\texttt{salt})$ during sign-up, EvanBot could cache $H^k(\texttt{pwd}\|\texttt{salt})$ values for $1 \leq k \leq 1000$ to avoid the recomputation later. But in this question, we specifically say that EvanBot is only given those three values, and cannot rely on previously-cached values.

Q3.5 (2 points) Eve is an on-path attacker.

Which of these sets of values, if seen by Eve, would allow Eve to learn the password? Each answer choice is independent.

☐ The first login attempt only
■ The 1000th login attempt only
☐ Any two login attempts in a row
☐ All of the first 999 login attempts
☐ The 999th login attempt only
■ All of the first 1000 login attempts
☐ None of the above

**Solution:** The key realization is that $n = 1000$ means EvanBot will send $H^{1000-1000}(\texttt{pwd}\|\texttt{salt}) = H^0(\texttt{pwd}\|\texttt{salt}) = \texttt{pwd}\|\texttt{salt}$, from which Even can read the password in plaintext. Other options are incorrect because the given hash function is one-way, so an attacker cannot reverse the chain to get to the original value. The salt provides brute-force resistance as well.

Q3.6 (2 points) Assume an attacker has compromised the server and can modify `stored`. Can the attacker login as EvanBot?

- ● Yes, without knowing $n$, `pwd`, or `salt`.
- ○ Yes, but only if they know `salt`.
- ○ Yes, but only if they know $n$ and `salt`.
- ○ Yes, but only if they know $n$.
- ○ No, even if they know $n$ and `salt`.

**Solution:** Since the server only checks that $H(\texttt{Step1}) = \texttt{stored}$, the attacker can set `stored` to $H(x)$ for any value $x$ that the attacker picks. Then the attacker would simply provide $x$ to login as EvanBot.