Lab 3 Theory

This assignment is due **at 10:00pm ET** on **Saturday, October 30, 2021**.

Please make note of the following instructions:

- Remember that your solutions must be submitted on Gradescope. Please sign-up for 6.S060 Fall 2021 on Gradescope, with the entry code `KYRBPK`, using your MIT email.
- We require that the solution to the problems is submitted as a PDF file, **typeset on LaTeX**, using the template available on the course website (`https://6s060.csail.mit.edu/2021/`). Each submitted solution should start with your name, the course number, the problem number, the date, and the names of any students with whom you collaborated.
- We will have bug bounties for Lab 3 and beyond. If you are the first to report a bug in either the Coding or the Theory parts of the lab you will receive a $10 Toscanini's Gift Certificate! Serialization is via Piazza – report the bug as a private question on Piazza and we will respond as soon as possible as to whether it is a real bug or not.

Problem 3-1. Private Album Encryption [40 points]

This problem will help you prepare for the coding part of Lab 3. Note that we refer to elements from the coding parts of previous labs. The problem focuses on combining several primitives that we saw in class, and to use these primitives to create new protocols with more complex security properties. We will start by describing and defining the primitives you can use to build your new protocols.

Global Sharing: You have access to a centralized server that can store data and metadata for you and send that data to any user that requests it. More precisely, you have access to a function `Store(key, obj)` that will store the object obj under the corresponding key onto the server, and a function `Load(key)` that will fetch the object associated with the key from the server.

Public Profiles: You have access to the PublicProfile feature you've developed in Lab 0 that makes it possible for any user to publicly advertise any data and metadata to other users. In particular, you have access to a function

`update_public_profile_infos(infos, metadata)`

that will update the info and metadata of your public profile and push it to the server. You also have access to a

`get_friend_public_profile(friend_username)`

function to get a friend's public profile form the server.

Public Key Signature Scheme: You have access to the public-key signature scheme from lab 2:

$$(\mathsf{Gen}() \rightarrow (\mathsf{sk_{sign}}, \mathsf{pk_{sign}}), \mathsf{Sig}(\mathsf{sk_{sign}}, \mathsf{m}) \rightarrow \sigma, \mathsf{Ver}(\mathsf{pk_{sign}}, \mathsf{m}, \sigma) \rightarrow \texttt{True/False})$$

with m in $\{0,1\}^*$ and $\mathsf{sk_{sign}}$, $\mathsf{pk_{sign}}$ and σ in $\{0,1\}^\lambda$, where λ is our security parameter.

Public Key Infrastructure: You have access to a function

`get_public_signing_key(friend_username)`

that gives you a trusted *signing* public key for the given username. (Note that this is equivalent to assuming that you have completed Lab2 successfully. Note that these keys **cannot be used for encryption**).

Symmetric Key Encryption: You have access to an authenticated symmetric-key encryption scheme

$$(\mathsf{Gen}() \rightarrow (\mathsf{sk_{enc}}), \mathsf{Enc}(\mathsf{sk_{enc}}, \mathsf{m}) \rightarrow \mathsf{C}, \mathsf{Dec}(\mathsf{sk_{enc}}, \mathsf{C}) \rightarrow \mathsf{m}/\bot)$$

with m and C in $\{0,1\}^*$ (where $|C| = O(|m|)$) and $\mathsf{sk_{enc}}$ in $\{0,1\}^\lambda$, where λ is our security parameter.

Public Key Authenticated Encryption: You have access to a new *authenticated public-key encryption scheme*, which simultaneously signs and encrypts:

$$\begin{aligned}
&(\mathsf{Gen}() \to (\mathsf{sk}_{\mathsf{enc\&auth}}, \mathsf{pk}_{\mathsf{enc\&auth}}),\\
&\mathsf{Enc\&Auth}(\mathsf{pk}_{\mathsf{enc\&auth,0}}, \mathsf{sk}_{\mathsf{enc\&auth,1}}, \mathsf{m}) \to \mathsf{C},\\
&\mathsf{Dec\&Ver}(\mathsf{sk}_{\mathsf{enc\&auth,0}}, \mathsf{pk}_{\mathsf{enc\&auth,1}}, \mathsf{C}) \to \mathsf{m}/\bot)
\end{aligned}$$

with m and C in $\{0,1\}^*$ (where $|C| = O(|m|)$) and $\mathsf{sk}_{\mathsf{enc\&auth,b}}$ ($b \in \{0,1\}$) , $\mathsf{pk}_{\mathsf{enc\&auth,b}}$ in $\{0,1\}^\lambda$, where λ is our security parameter.

This scheme is used as follows: Alice runs Gen to get

$$(\mathsf{sk}_{\mathsf{enc\&auth,A}}, \mathsf{pk}_{\mathsf{enc\&auth,A}}),$$

and Bob runs

$$(\mathsf{sk}_{\mathsf{enc\&auth,B}}, \mathsf{pk}_{\mathsf{enc\&auth,B}})$$

to generate key pairs. (Pay attention to the subscripts!) Once the pk are exchanged, Alice encrypts and authenticates for Bob with

$$\mathsf{Enc\&Auth}(\mathsf{pk}_{\mathsf{enc\&auth,B}}, \mathsf{sk}_{\mathsf{enc\&auth,A}}, \mathsf{m}) \to \mathsf{C};$$

Bob decrypts and authenticates C with

$$\mathsf{Dec\&Ver}(\mathsf{sk}_{\mathsf{enc\&auth,B}}, \mathsf{pk}_{\mathsf{enc\&auth,A}}, \mathsf{C}) \to \mathsf{m}/\bot.$$

Under the hood, this scheme uses a primitive you've already seen: key exchange.

Note that these public keys are not provided by the public key infrastructure! You will need a way to securely transfer them.

(a) Describe a protocol using these primitives that makes it possible for users to exchange their Enc&Auth public key with one another in an authenticated way assuming a malicious server that can tamper with its own storage.

[5 points]

(b) Describe a protocol using these primitives that makes it possible for a given user (the owner) to confidentially share a secret K with a group of other users (their friends) despite the presence of malicious users.

Assumption: Operations between the clients and the server happen sequentially and atomically. For instance, a client will always have the latest version of the server state when the client tries to modifying the state. You will also assume that (a) was correctly answered and that any user have access to other users' Enc&Auth public keys.

Threat Model: in this question we assume a honnest server (that will not tamper with the integrity of the messages) but we assume a server that has no security measures and will serve any request from any user. Finally we assume the presence of malicious users that will try to access photos that they are not supposed to see.

Security Properties:

- Correctness: At the end of the protocol, and in the absence of malicious users, each "friend" user should have access to the secret.
- Confidentiality: No user besides the owner of the secret and the friends listed in the album can have access to the secret and this, even in the presence of malicious users.
- Authenticity: An attacker should not be able to share a fake secret on behalf of an other user.

Note that if it is ok for a malicious user to prevent other users from accessing the secret, it is not ok for a malicious user to gain access to the secret itself.

In particular, you will describe the protocols (using pseudo code, text and/or drawings) for the following operations:
`create_shared_secret(secret_name,` K`, list_friends)`,
`get_shared_secret(secret_name)` $\rightarrow K$
[5 points]

(c) Let us now consider the actual setup of Lab 3. Alice (the owner) wants to share a private album of photos with a group of friends (Bob and Cedric). In order to be able to control who has access to these pictures we introduce the notion of *private albums*. You will design such a scheme by assuming the following assumptions, adversaries and by trying to archieve the security properties described here.

Assumption: Operations between the clients and the server happen sequentially and atomically. For instance, a client will always have the latest version of the server state when the client tries to modifying the state.

Threat Model:

- The server is functioning correctly and will not tamper with the integrity of messages. However, the server will not authenticate users.
- Malicious users will try to access photos that they are not supposed to see.

Security Properties:

- Correctness: Any friends listed in the album can read and add photos to the album that will then be visible to friends listed in the album, even if the owner is offline.
- Confidentiality: No user besides the owner of the secret and the friends listed in the album should be able to gain access to the album's photos.
- Revocability: The owner (and only the owner) can add *or remove* other users from the list of friends the album is shared with.

Note that we cannot guarantee that a user that has been removed from the list of friends of the album will never have access to photos uploaded to the album before they were removed (they might have downloaded the full album before being removed). Nevertheless, we should guarantee that they will not have access to **newly** uploaded photos.

In particular, you will describe the protocols (using pseudo code, text and/or drawings) for the following operations:

```
create_shared_album(album_name, photos, list_friends)
add_friend(album_name, friend_username),
remove_friend(album_name, friend_username),
add_photo(album_name, photo)
```

Describe two schemes that satisfy the following bandwidth limitations for the `remove_friend` operation assuming m friends and n photos of size $l >> \lambda$ with λ the security parameter as defined in the different primitives:

1. Bandwidth limited to $O(n * l + m * \lambda)$.
2. Bandwidth limited to $O((n + m) * \lambda)$

Note that you are encouraged to use the protocol you designed for part (b) and that doing this question concurrently with the coding part might help you. [20 points]

(d) Assuming that we replace the Public Key Authenticated Encryption primitive with a non-authenticated Public Key Encryption primitive , come up with an attack on one of your previous protocol that could lead a malicious user Eve to access photos not meant for her. [10 points]

Problem 3-2. Message authentication codes [30 points] Recall that in class we mentioned the hash-then-MAC paradigm, which uses a MAC for messages of fixed length n (such as AES, in which case $n = 128$), and a hash function with range $\{0,1\}^n$, to MAC messages of arbitrary length, by first hashing the message to an element in $\{0,1\}^n$ and then applying the underlying MAC to the hash value.

In class we mentioned that for the resulting MAC to be secure the hash function must be collision resistant, and thus this paradigm cannot be used with AES since its domain consists of messages of length 128, and there does not exist a collision resistant hash function with 128 bit security that maps to message of length 128 due to the birthday paradox.

In this problem we consider using the hash-then-MAC paradigm with AES but with a hash function with a *secret* seed. In particular, consider the following hash function H: It takes as input a (secret) seed $(a_1, \ldots, a_t)$ and a message $(M_1, \ldots, M_t)$, where each a_i and M_i are blocks of 128 bits, and it outputs

$$H((a_1, \ldots, a_t), (M_1, \ldots, M_t)) = \sum_{i=1}^{t} a_i M_i.$$

The arithmetic is done over a finite field of size 2^{128} (known as the Galois Field and denoted by $\mathsf{GF}[2^{128}]$), though how the arithmetic is done is not important, the only important thing is that the output is in $\{0,1\}^{128}$, and that in every (finite) field (and in particular in $\mathsf{GF}[2^{128}]$) every non-zero element has an inverse.

(a) Define the MAC scheme obtained by applying the hash-then-MAC paradigm to the hash function $H : (\{0,1\}^{128})^t \times (\{0,1\}^{128})^t \to \{0,1\}^{128}$ and AES as the underlying MAC.

[10 points]

(b)
1. Recall the attack for the hash-then-MAC using AES and a hash function without a secret seed (assuming the message space is $(\{0,1\}^{128})^t$ for $t \geq 2$). *Hint:* This attack takes time roughly 2^{64}.
2. Explain why this attack fails if we use the hash function above (with a secret seed), assuming the attacker sees significantly less than 2^{64} tags.
3. Is this hash-then-MAC scheme (with H as defined above and AES) secure if the attacker gets to see more than 2^{64} tags for messages of his choice? *Hint:* Use the fact that AES is injective (i.e., if $\mathsf{AES}(K, M) = \mathsf{AES}(K, M')$ then $M = M'$), and the fact that H is a linear function, together with the fact that one can efficiently solve t linear equations with t variables (using Gaussian elimination).

[20 points]

Problem 3-3. Weaknesses of CPA-secure cryptosystems [30 points]

Let $F\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ be a secure pseudorandom function. In class, we saw the CPA-secure encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ with keyspace $\mathcal{K}$, where

$$\mathsf{Enc}(k,m) := \begin{cases} r \xleftarrow{\mathrm{R}} \{0,1\}^n \\ c \leftarrow F(k,r) \oplus m \\ \text{output } (r,c) \end{cases} \qquad \mathsf{Dec}(k,(r,c)) := \begin{cases} m \leftarrow F(k,r) \oplus c \\ \text{output } m \end{cases} .$$

We will use a secure MAC scheme $\mathsf{MAC}\colon \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n$ over the same keyspace $\mathcal{K}$.

MIT uses encryption to protect communications between department offices and its central payroll system. In particular, the EECS department office and MIT's payroll office share a secret encryption key $k_{\mathsf{Enc}} \xleftarrow{\mathrm{R}} \mathcal{K}$ and a secret MAC key $k_{\mathsf{MAC}} \xleftarrow{\mathrm{R}} \mathcal{K}$.

On each pay date, a machine in the EECS department sends a sequence of messages to the payroll office. Each plaintext message is a 54-byte ASCII-encoded string with format:

$m =$ "`date:NNNN-NN-NN, mit_id:NNNNNNNNN, amount:NNNNNNNNN.NN`",

where each `N` represents an ASCII-encoded decimal digit (i.e., ASCII values `0x30`–`0x39`).

Unfortunately, the designers of the system make the wrong decision and use MAC-then-encrypt rather than encrypt-then-MAC. In particular, the EECS department encrypts each message m as:

$$\mathsf{ct} = \mathsf{Enc}(k_{\mathsf{Enc}}, m \,\|\, \mathsf{MAC}(k_{\mathsf{MAC}}, m)).$$

The machine at the payroll office operates on each ciphertext ct as follows:

- Compute $(m' \| t') \leftarrow \mathsf{Dec}(k_{\mathsf{Enc}}, \mathsf{ct})$.
- Check that the bytes of the string corresponding to the MIT ID number and the payment amount are valid ASCII decimal digits. If not, send a `FORMAT ERROR` message to the EECS machine.
- Check that the date corresponds to today's date. If not, send a `DATE ERROR` message to the EECS machine.
- Check that $t' = \mathsf{MAC}(k_{\mathsf{MAC}}, m')$. If not, send a `MAC ERROR` message to the EECS machine.
- Append the `(ID, amount)` pair to a log file for later processing and send an `OK` message to the EECS machine.

Explain how a network attacker that can intercept ciphertexts and interact with the payroll server can recover **all but one of the bits in each secret digit** in the plaintexts that the EECS department sends. Recovering these bits for a single plaintext using your attack should take no more than than 10,000 interactions with the payroll server.