Liam Brew
CS 306

Professor Triandopoulos
24 November 2020

Homework 02

Problem 1: Doman-extension MAC implementation

1.1) mac() and verify()

--see attached Java code—

1.2) Default Message

The hexadecimal MAC tag of the default message is **CEDE6D796ADB99036ED8C1D96B59908E**.

1.3) Algorithm Choice

To complete this code, I chose to implement a CBC-MAC algorithm. This has the benefit of not requiring an initial vector. By building off the previous message block in a chained manner, the CBC-MAC is more efficient than the domain-extension scheme and can authenticate larger messages than the PRF-based scheme.

1.4) Domain Extension

By using padding, any length message can be accounted for so long as they are a multiple of **n** and the size of them are fixed in advanced. Additionally, F_k must also be a secure PRF, but I assumed this to be the case as it was provided to me. If these criteria are not met, then this algorithm is ultimately insecure.

Problem 2: Data outsourcing and public-key infrastructure

2.1) Eve-Mallory Collaboration

As indicated by the term “collaboration” in the question prompt, I assume that Mallory is aware of and shares some complicity in Eve’s attack. Therefore, with Mallory’s assistance in this matter Eve can potentially perform a replay attack. To perform this attack, Mallory would manipulate the data storage of the program by maintaining Bob’s previous public key. The location of this key and instructions on how to access it will be relayed to Eve, allowing them to attain Bob’s former key. With this being done, then Mallory can begin rerouting all messages sent to Bob by providing Bob’s former key (which Eve possesses) instead of the new one he generated once he realized sk_b was compromised. Eve will therefore be able to read this message as she possesses the key it was addressed to; Bob is using the new one, which means he is unable to read or receive it. Once Eve has gleaned the required information from the message, she re-encrypts it (using Bob’s new public key this time, available in the public directory for all users) and sends it to him. This deliberate resending of the data classifies this attack as a replay one.

2.2) Timestamping

By using a timestamp, the CA can include the time of usage/generation onto D, C, and all subsequent iterations (D’, C’’, etc.). This will then direct queries to the most recent relevant iteration of the Merkle tree. If an older one is being accessed while a newer option exists, then the potential of tampering is realized. This would help prevent messages using Bob’s old key after he generated the new one.

I pledge my honor that I have abided by the Stevens Honor System

Problem 3: Intrusion resilience

3.1) Split Architecture

This split architecture improves security by making it drastically more difficult to get access to correct passwords. By splitting the password verification process in half, the compromise of one server won't give the attacker any advantage. To successfully gain a password, the attacker will need to compromise both servers simultaneously. The odds of this are greatly reduced when compared to the likelihood of accessing only one server. As the final accept/rejection decisions depend on the output of both servers to gain the entire password, the confidentiality of user password data is improved as it is not located all in one place.

3.2) Honeywords and Cracking

Honeywords make password cracking detectable by alerting the system in the event of a likely security breach. Good honeywords are very similar grammatically to actual passwords, and from the outside offer no real discernible difference. However, honeywords are designed to make the odds of the real user entering them (such as through a typo) nearly negligible. An example of this is the real password **pa$$word5** and the honeyword **p4$$w0rds**. While it would likely never occur to the user to enter the honeyword, from an outsider's point of view the two are very similar. This is meant to catch hackers who have access to password lists. As an outsider, they do not know which entry is the real password as both are likely to be valid. This is amplified by the fact that in practice there are multiple honeywords for each correct password, all appearing to be valid choices. This drastically reduces the probability of the attacker choosing the correct one. If a honeyword is chosen, the system can very reasonably infer that its password list has been compromised and can take the appropriate action.

3.3) Honeyword Generation

A reasonable honeyword for the real password **pa$$word5** can be **p4$$w0rds**. Both words have the same leetspeak translation of "passwords" but are spelled differently. Likely, the attacker would know which password is correct unless they have other information to go off.

3.4) Likely Password

Out of these potential passwords, I would immediately discount the one that appears to be a random string. It does not seem reasonable for a user to have come up with this password on their own and use it daily, and if they do then they should consider switching to the Office of Information Technology instead. This leaves me with **Blink-123** and **Blink-182**, one of which is most likely the correct password. I am not as sure in my elimination of the honeyword here as I was previously, as both could be reasonably assumed to be passwords. If I had to choose I would **Blink-182** as the password. This is because it is the name of a known music group and not a seemingly random word and number combination as **Blink-123** is. As mentioned though, I am not as sure about this, indicating that the honeyword system has worked.

Problem 4: On the RSA cryptosystem

4.1) Efficiency

The modular exponentiations at the core of the RSA cryptosystem are realized efficiently in practice by the repeated squaring algorithm. This increases the speed of which $a^p \bmod n$ is calculated as the algorithm has a logarithmic time complexity. The decryption/signing aspect of RSA is made more efficient using the Chinese remainder theorem, as the related modular exponentiations used for decryption contain smaller exponents and moduli, thereby completing faster.

I pledge my honor that I have abided by the Stevens Honor System

4.2) Code

--see attached Python code--

I pledge my honor that I have abided by the Stevens Honor System