Wagner Spring 2014	CS 161 Computer Security	Homework 3

Due: Monday April 14, at 11:59pm

Instructions. This homework is due Monday April 14, at 11:59pm. It *must* be submitted electronically via Pandagrader (and not in person, in the drop box, by email, or any other method). Realistically, you should treat the deadline as 10:59pm; the 11:59pm deadline is strict, and Pandagrader may be overloaded immediately before it. Therefore, we *strongly* recommend you upload your solution by 10:59pm.

This assignment must be done on your own.

Please put your answer to each problem on its own page, in the order that the problems appear. For instance, if your answer to every problem fits on a single page, your solution will be organized as follows:

page 1: your solution to problem 1
page 2: your solution to problem 2
page 3: your solution to problem 3
page 4: your solution to problem 4
page 5: your solution to problem 5
page 6: your optional feedback, or a blank page ("problem 6")

If your solution to problem 3 takes up two pages, your solution would be organized as follows:

page 1: your solution to problem 1
page 2: your solution to problem 2
page 3: first page of your solution to problem 3
page 4: second page of your solution to problem 3
page 5: your solution to problem 4
page 6: your solution to problem 5
page 7: your optional feedback, or a blank page ("problem 6")

Scan your solution to a PDF—or, write it electronically and save it as a PDF. Then, upload it to Pandagrader. You must submit it as a PDF, not a series of images; you will receive no credit if you submit it as images.

Make sure to select pages for each question after uploading in Pandagrader. If you fail to complete the process (of page selection) so that your PDF does not render properly, your homework will be marked as late and therefore not accepted. If you have no response to the optional feedback, please attach a blank page at the end of the PDF and select that page for problem 6.

Problem 1 ***Short answer*** **(15 points)**

For this question, you only need to give a short answer (e.g., one sentence or less).

(a) Name one encryption scheme mentioned in class that protects against eavesdroppers but is vulnerable to man-in-the-middle attacks. You do not need to justify your answer.

(b) True or false: If the wind speeds over the golden gate bridge are truly random, then a good one-time pad would be the concatenation of wind speeds taken over time, since a one time pad needs to be random.

State true or false, then explain why in at most one sentence.

(c) True or false: The IV for CBC mode must be kept secret.

State true or false. You do not need to justify your answer.

(d) Alice and Bob share a symmetric key k. Alice sends Bob a message encrypted with k stating, "I owe you \$100", using some standard symmetric-key encryption scheme. True or false: Assuming the encryption scheme is secure, Bob can now go to a bank and prove to the bank teller that Alice does indeed owe him \$100.

State true or false, then explain why in at most one sentence.

(e) Alice and Bob share a symmetric key k. Alice sends Bob a message encrypted with k stating, "I owe you \$100", using some standard symmetric-key encryption scheme. True or false: Assuming the encryption scheme is secure, we can be confident that an active attacker cannot tamper with this message; its integrity is protected.

State true or false, then explain why in at most one sentence.

Problem 2 ***Why do RSA signatures need a hash?*** **(20 points)**

This question will help you work through why RSA signatures use a cryptographic hash function. In standard RSA signatures, the signature S on a message M is $S = H(M)^d \bmod n$, where d is the private key; to verify whether S is a valid signature on message M, we check whether $S^3 = H(M) \bmod n$ holds. But for this question, suppose we skipped using a hash function and just used M directly, so the signature S on a message M is $S = M^d \bmod n$. In other words, if Alice wants to send a signed message to Bob, she will send M, S to Bob, where $S = M^d \bmod n$ is computed using her private key d. Let's explore the implications.

(a) With this simplified RSA scheme, how can Bob verify whether S is a valid signature on message M? In other words, what equation should he check, to confirm whether M, S was validly signed by Alice? You don't need to justify your answer; just show the equation.

(b) Mallory learns of Alice and Bob's plans and decides to trick Bob. Mallory wants to send some M, S to Bob that Bob will think is from Alice, even though Mallory doesn't know the private key. Explain how Mallory can find M, S such that S will be a valid signature on M.

You should assume that Mallory knows the public key n, but not the private key d. She can choose both M and S freely. The message M does not have to be chosen in advance and can be gibberish.

Hint: If Mallory chooses M and then tries to find a corresponding S, she'll be at a dead-end, because finding S requires inverting a one-way function (cubing modulo n), and we know that is hard without knowledge of the trapdoor (the private key d). So instead, she should...

(c) Sameer is holding an auction. Alice and Bob will submit signed bids to the auctioneer Sameer, signed using this simplified RSA signature scheme. The message M is an integer that is their bid (in dollars), and they will send just their signature on M, signed using this simplified RSA scheme. Sameer will accept whichever bid is highest and expect that person to pay up however much they bid.

Mallory wants to mess with Bob (her rival). So, when Bob forms his bid M and sends Sameer the signed bid $S = M^d \bmod n$, Mallory intercepts the message from Bob containing S. Mallory would like to tamper with S to form a new signature S' that corresponds to a bid for $64\times$ as much as Bob's original bid, to force Bob to win the auction and pay through the nose for it. More precisely, she'd like to find a value S' such that S' is a valid signature on $64M$, so she can replace S with S' and forward the result onto Sameer. Help Mallory out: show how she can compute such a S'.

(Assume that M is small enough that $64M < n$, so $64M$ does not wrap around modulo n.)

(d) Are your attacks in parts (b) and (c) possible against the real RSA signature scheme (the one that includes the cryptographic hash function)? Why or why not?

Problem 3 ***Attacking encryption*** **(30 points)**

FlapChat is a happening web forum for folks who love to fly. FlapChat.com uses passwords and session cookies to authenticate their users. In particular, once a user logs in (by entering their username and password), the FlapChat.com server sets a cookie that contains information about the user. The cookie will be sent with all subsequent requests to FlapChat.com, and the FlapChat.com server will use that cookie to authenticate the user (so the user doesn't have to enter in their password again). Users are only given this cookie once they enter in their password correctly, so if the server sees a request accompanied by such a cookie, then it knows that it comes from an authenticated user.

You've learned that the cookie contains the encryption of a credential string, which contains the user's username, the user's display name, and the account-type (the letter O or A, which indicates whether this is an ordinary user or an administrator); the fields are separated by exclamation marks. For instance, the plaintext credential string for Alice might be

```
alice!Alice X. Jones!O
```

Once she has logged in, her cookie contains an encrypted version of this plaintext cre-

dential string.

You want to break into the web site and gain administrator-level access. This question has all the information needed to spot the vulnerabilities, so your attack should use only the information given here; no social engineering of site administrators, no eavesdropping on their traffic, no getting malware on admin's machines, no exploiting XSS bugs in the site, nothing like that.

(a) You haven't been able to find any information about how the encryption works, but looking at the cookie values, you suspect they must adding enough random bytes of padding to the end of the plaintext credential string to bring it up to a multiple of 16 bytes and then encrypting it somehow. You've been poking around the web site and you immediately notice that this is a public forum: anyone can create an account. In particular, you can create an ordinary account with any username and display name you want (assuming the username hasn't already been taken), but you can't create an administrator account. Describe an easy attack that will get you administrator-level access.

(Since you don't know how they are doing the encryption, your attack had better work no matter what encryption algorithm they use.)

(b) You show your buddy how to do the attack from part (a). Unfortunately, your buddy has a big mouth and starts bragging about his administrator-level access, and before you know it, the FlapChat developers spot the security hole and fix the problem. In particular, they change their system to require admins to manually approve each new account that is created, and they are careful not to approve any new account that would let you exploit the vulnerability mentioned in part (a). Curses! You've been foiled.

You go sulk for a week, but then by good luck you have a chance to chat up the developer of FlapChat at a party. You get him bragging about his elegant design and pretty soon you've learned how the plaintext credential string is encrypted: it is padded by appending random bytes until it is a multiple of 16 bytes long, then it is encrypted with AES-CTR mode, using a 128-bit AES key K stored on the FlapChat server. Thus, the cookie contains an AES-CTR ciphertext, i.e., $IV \;||\; C_1 \;||\; C_2 \;||\; \cdots \;||\; C_k$, where IV is a random initialization vector chosen randomly each time a user logs in, and where $C_i = Z_i \oplus M_i$ where $Z_i = \text{AES}_K(IV+i)$ and $M_1 \;||\; M_2 \;||\; \cdots \;||\; M_k$ is the padded plaintext credential string. When the server receives a cookie, it decrypts it using AES-CTR, pulls out the username, display name, and account-type by using the exclamation marks as separators (ignoring all padding bytes), checks that the username is in the database of registered users and that the account-type is either `O` or `A`, and accepts the user as authenticated if all of these checks pass. It then gives access based upon the account-type (for instance, if the HTTP request asks to perform some administrator-only action, the action is performed only if there is a valid cookie and its account-type is `A`).

Describe how to attack the crypto and get administrator-level access to the site.

(Your attack has to be something you can do on your own by attacking the crypto.)

(c) You get drunk one evening and use your administrator-level access to post a really funny prank message on the site. The site administrators are not amused, investigate a little further, and realize that maybe using CTR mode was not such a good idea. So, they beef up their security with a one-line fix: instead of using AES-CTR mode, they use AES-CBC mode. For good measure, they change the AES key so that people can't use old cookies; everyone will have to log in again and get a new cookie encrypted with the new scheme.

Describe how you can still break their crypto and get administrator-level access to the site.

(d) OK, so maybe AES-CBC mode wasn't such a great idea, either. Describe what they should have used instead of AES-CBC mode. If the plaintext credential is given by $M_1 \,||\, M_2 \,||\, \cdots \,||\, M_k$, what value should they store in the cookie? Be specific.

Problem 4 ***Using cryptographic building blocks*** **(20 points)**

This problem tests your understanding of how to use cryptographic algorithms. Alice and Bob conduct business extensively over the Internet, and they would like to be able to enter into binding contracts with other parties located around the world. To save the environment, they'd like to do it entirely electronically, over the Internet. Therefore, we'd like some cryptographic way that each party can agree to the terms of the contract. If Alice and Bob are contemplating a contract, Alice should be able to tell whether Bob has agreed to the contract, and similarly Bob should be able to tell whether Alice has agreed to the contract.

Once Alice and Bob have agreed to the terms of the contract, neither should be able to back out. If one party fails to live up to his/her obligations, or there is any dispute, we assume that the injured party will bring a lawsuit. In case of a lawsuit, it should be possible for Alice to convince the judge that Bob agreed to the terms of the contract (by showing the judge the messages she received from Bob). Similarly, it should be possible for Bob to convince the judge that Alice agreed to the contract. It should never be necessary for any party to give their private key to the judge, but it is OK to give the judge any session keys that are relevant.

The scheme should be secure against attackers with the ability to intercept and/or modify packets sent electronically, and with the ability to inject forged packets. There should be no way for an attacker to fool Alice into thinking that Bob has accepted any contract that he did not actually accept, and no way to fool Bob into thinking that Alice has accepted any contract that she did not actually accept. Also, in the event of a lawsuit, there should be no way for Alice to fool the judge into concluding that Bob accepted the contract when he actually didn't (even if Alice colludes with the attacker), and no way for Bob to fool the judge into concluding that Alice accepted the contract when she actually didn't (even if he colludes with the attacker). Let's look at several possible cryptographic protocols for this.

Assumptions: You can assume that Alice has a public key K_A and a matching private

key K_A^{-1}; Bob has a public key K_B and private key K_B^{-1}; and the court has a public key K_C and a private key K_C^{-1}. Assume that everyone knows everyone else's public key, with no possibility of spoofed public keys (e.g., Alice knows Bob's public key and the court's public key, and so on). The parties do not share their private key with anyone. k_1 is a random session key chosen by Alice (a new one is chosen for each contract Alice signs), and k_2 is a random session key chosen by Bob; both k_1 and k_2 are symmetric keys. Each party has his/her own personal computer, which is not shared, is adequately protected from compromise, and can be assumed free of malware. You may assume that all encryption, MAC, and digital signature algorithms are secure against chosen-plaintext attack and are implemented properly, but you shouldn't assume anything about which specific algorithm is used (we want the contract-signing scheme to be secure, as long as we plug in any secure encryption, MAC, and digital signature algorithm). Let T denote the text of the contract. T will include the identities of the parties (Alice and Bob) and all the terms of the contract. Both Alice and Bob already know T. There is no need to prevent the attacker from learning T. At least one of the schemes below is good.

Instructions: For each scheme listed below, write "Good" if the scheme meets the requirements above (no justification is needed in this case), or write "No good" if the scheme does not meet the requirements above and then explain briefly why the scheme is not acceptable (one or two sentences should be sufficient; if it is insecure, sketch the attack).

(a) Alice sends $E_{K_B}(T \mid\mid E_{K_C}(T))$ to Bob. Bob sends $E_{K_A}(T \mid\mid E_{K_C}(T))$ to Alice.

(b) Alice sends $E_{K_B}(k_1), \text{MAC}_{k_1}(0 \mid\mid T)$ to Bob. Bob sends $\text{MAC}_{k_1}(1 \mid\mid T)$ to Alice.

(c) Alice sends $\text{Sign}_{K_A^{-1}}(T)$ to Bob. Bob sends $\text{Sign}_{K_B^{-1}}(T)$ to Alice.

(d) Alice sends $\text{Sign}_{K_A^{-1}}(H(T))$ to Bob, where H is a cryptographic hash function. Bob sends $\text{Sign}_{K_B^{-1}}(H(T))$ to Alice.

(e) Alice sends $E_{K_B}(\text{Sign}_{K_A^{-1}}(T))$ to Bob. Bob sends $E_{K_A}(\text{Sign}_{K_B^{-1}}(T))$ to Alice.

(f) Alice sends $k_1, E_{k_1}(T), \text{Sign}_{K_A^{-1}}(k_1), \text{Sign}_{K_A^{-1}}(E_{k_1}(T))$ to Bob. Bob sends $k_2, E_{k_2}(T), \text{Sign}_{K_B^{-1}}(k_2), \text{Sign}_{K_B^{-1}}(E_{k_2}(T))$ to Alice.

(g) Alice sends $E_{K_B}(k_1 \mid\mid U_1), \text{Sign}_{K_A^{-1}}(U_1)$ to Bob where $U_1 = \ell \mid\mid E_{K_C}(k_1) \mid\mid E_{k_1}(T)$ and ℓ is the length of $E_{k_C}(k_1)$. Bob sends $E_{K_A}(k_2 \mid\mid U_2), \text{Sign}_{K_B^{-1}}(U_2)$ to Alice, where $U_2 = \ell \mid\mid E_{K_C}(k_2) \mid\mid E_{k_2}(T)$ (assume ℓ is the length of $E_{k_C}(k_2)$ too).

Problem 5 ***Voter registration matching*** **(15 points)**

A couple of weeks ago David was testifying at a hearing in Sacramento about technology for improving elections and the privacy and security risks, and he learned about the following problem.[1] In the US, voters must register to vote in the state where they

[1]Seriously. No kidding. This is a real problem, that actually came up at the hearing before the California Senate; it was mentioned as perhaps the primary barrier to doing cross-state voter registration matching! You get to try your hand at solving it, and seeing how the concepts we've learned in this class can help make democracy better.

reside. Unfortunately, sometimes voters move from one state to another and forget to cancel their registration in the old state, leaving them registered in both states—a no-no. Election officials in California and Nevada would like to compare their lists of registered voters, so they can find voters who are currently registered in both states and have them update their registration.

Unfortunately, the voter registration lists are pretty sensitive data: it's a list of all registered voters, with the birthdate and last 4 digits of the social security number (SSN) of each voter. One obvious approach would be for California to upload all of their voter registration lists to Nevada and let Nevada find all the matches, but California election officials have reasonable concerns about that: what if Nevada's server got breached and a hacker was able to access all that data? That'd be bad. Nevada has similar concerns about uploading their data to California. Nonetheless, it sure would be useful if we could find a way to do this without risking disclosure of private voter information.

Your job is to find a technical solution to this problem. In technical terms, we can think of California's voter database as a list $x_1, x_2, \ldots, x_n$, where x_i is a string that encodes the crucial information about the ith California voter (namely, x_i contains the voter's last name, their birthdate, and the last 4 digits of their SSN, encoded in a standard way and concatenated to get a single string). Similarly, we can think of Nevada's voter database as a list $y_1, y_2, \ldots, y_m$, where y_j is the same information about the jth Nevada voter. Our goal is to find all pairs (i, j) such that $x_i = y_j$. That is, we're treating two voter registration records as a match only if they have the same last name, the same birthdate, and the same SSN-last-4.

You can assume the existence of a third-party server, S, run by an independent non-profit organization. The non-profit organization tries their best to keep S secure, but hey, we all know that no one is perfect. California is going to transform their voter database to a list $x_1^*, x_2^*, \ldots, x_n^*$ (where x_i^* is somehow derived from x_i) and upload it to S. Similarly, Nevada is going to transform their database to a list $y_1^*, y_2^*, \ldots, y_m^*$ and upload it to S. Finally, S will find the set $P = \{(i, j) : x_i^* = y_j^*\}$ of all pairs (i, j) such that $x_i^* = y_j^*$ and send P to both California and Nevada. We want to ensure three requirements: (1) if $x_i = y_j$, then $(i, j) \in P$, (2) if $x_i \neq y_j$, then it is unlikely that $(i, j) \in P$, (3) if S is compromised by an attacker, then the information known to S does not let an attacker learn any voter's birthdate or last-4-of-SSN.

Fill in the details to get a solution. Your solution can use the cryptographic hash SHA256 and/or the AES block cipher, but you cannot use public-key cryptography or modular arithmetic. You can assume that California and Nevada share a key k that is not known to anyone else.

(a) In your solution, how should California compute x_i^* (as a function of x_i)? How should Nevada compute y_i^* (as a function of y_i)? Be specific.

(b) Explain why requirement (1) is met by your solution, i.e., explain why it is guaranteed that if $x_i = y_j$, then $x_i^* = y_j^*$ will hold.

(c) Explain why requirement (2) is met by your solution, i.e., if $x_i \neq y_j$, explain why it

is unlikely that $x_i^* = y_j^*$.

(d) Explain why requirement (3) is met by your solution, i.e., if S is compromised by an attacker, then the information known to S does not let an attacker learn any voter's birthdate or last-4-of-SSN.

Problem 6 ***Feedback*** **(0 points)**

Optionally, feel free to include feedback. What's the single thing we could do to make the class better? Or, what did you find most difficult or confusing from lectures or the rest of class, and what would you like to see explained better?