CS 161 Computer Security
Spring 2010 Paxson/Wagner
HW 4 Soln

1. (40 pts.) Blind Signatures and Digital Cash

(a) It would sometimes be useful to have electronic forms of payment that provide anonymity the way that paper cash does. Blind signatures provide a way to do that. In a blind signature, the contents of the document being signed are never revealed to the signer.

Here is one blind signature scheme that uses RSA signatures. Bob has a public key n (his RSA modulus) and a private key d. Alice wants Bob to sign the message m blindly.

1. Alice chooses a random value r uniformly at random from the set $\{r \in \mathbb{Z} : 1 \leq r < n \text{ and } \gcd(r,n) = 1\}$. Alice blinds the message by computing $t = mr^3 \bmod n$.
2. Bob signs t by computing $u = t^d \bmod n$ and returning u to Alice.

Show how Alice can unblind u to obtain a valid signature $s = m^d \bmod n$ on m. In other words, show how Alice can compute s, based upon the information she knows and the information she has received from Bob. (Of course, Alice does not know Bob's private key d, so your solution must not assume knowledge of d.)

Answer: Remember, $u = t^d = (mr^3)^d = m^d r^{3d} = m^d r \bmod n$. So Alice can unblind Bob's response by computing $s = \frac{u}{r} = \frac{m^d r}{r} = m^d \bmod n$.

(b) In the protocol above, suppose Bob does not know the message m that Alice wants signed, but Bob somehow knows it must be either m_0 or m_1. Can Bob distinguish which is the case, more effectively than random guessing (e.g., is there some strategy he can use to guess correctly with probability $3/4$ or greater)? Justify your answer carefully.

HINT: The function $f(r) = r^3 \bmod n$ is one-to-one: if $r \neq s \bmod n$, then $r^3 \neq s^3 \bmod n$.

Answer 1: No. Given $t = mr^3 \bmod n$, but with no knowledge of r, Bob cannot determine whether $m = m_0$ or $m = m_1$. Since r is uniformly distributed on the set $S = \{r \in \mathbb{Z} : 1 \leq r < n \text{ and } \gcd(r,n) = 1\}$, and since $f : S \to S$ is one-to-one, it follows that $f(r)$ is uniformly distributed on the set S, too. In other words, $r^3 \bmod n$ is uniformly distributed on S.

Also, the function $g : S \to S$ given by $g(x) = m_0 x \bmod n$ is one-to-one, so $g(f(r))$ is uniformly distributed on S. In other words, $m_0 r^3 \bmod n$ is uniformly distributed on S, as is $m_1 r^3 \bmod n$. Since they both have the exact same distribution, Bob cannot distinguish between them.

Answer 2: No. Bob receives $t = mr^3 \bmod n$.

Suppose Bob hypothesizes that the message was m_0. Then there is exactly one value of r that is consistent with this hypothesis, namely, the value $r_0 = f^{-1}(t/m_0)$. (Here we are using the fact that f is one-to-one, to ensure that such an inverse exists and is unique.)

Similarly, if Bob hypothesizes that the message was m_1, there will be exactly one value of r that is consistent with this hypothesis, namely, $r_1 = f^{-1}(t/m_1)$.

Either way, there is exactly one value of r consistent with Bob's hypothesis, so we cannot rule out either hypothesis. Moreover, since r is uniformly distributed, we have no reason to favor one hypothesis over the other. (This can be formalized by a calculation using conditional probability, which we omit.)

Answer 3: No. Define $\alpha = r^3 \bmod n$. The hint tells us that, since r is uniformly distributed on the set $S = \{r \in \mathbb{Z} : 1 \leq r < n \text{ and } \gcd(r,n) = 1\}$, so too is α, since $\alpha = f(r)$. Bob receives t, where $t = m\alpha \bmod n$, but Bob has no knowledge of α. This is effectively a one-time pad encryption of m under the pad α, except that instead of using xor to combine the pad with the message, we use multiplication modulo n. The proof of security for the one-time pad can be adapted to prove that t gives us no information—not even partial or probabilistic information—about whether the message was m_0 or m_1.

Comment: What this tells you is that Bob learns nothing about Alice's message m. Bob does not even learn partial or probabilistic information about m. Even if Bob had some way to narrow down m to one of a few possibilities through some other means, the blind signature protocol does not give him any additional information about m. That's why it is called a *blind* signature protocol: Bob blindly signs a message, without any idea what message he just signed. Normally, blindly signing an unknown message would be a terrible idea, but as we see in the rest of this question, the blind signature protocol can be a useful building block for designing digital cash schemes.

(c) Going back to our desire for digital cash, our requirements for a digital cash protocol are:

1. The bank cannot tell where the customer is spending his/her digital cash. In particular, the bank does not learn which merchant the customer spends his/her money at.
2. The merchant cannot identify the customer from the digital cash he/she provides to the merchant.
3. The merchant has some way of checking that the digital cash is not counterfeit.
4. The digital cash cannot be copied and spent twice.

Below we lay out one digital cash protocol. In our protocol there are three players: Carol, the customer; BankOBits, the bank; and Maxine, the merchant. Let coin $= (c,s)$ where $s = \text{Sign}_{\text{BOB}}(c)$ and $c =$ the value of the coin in dollars.

Imagine the following protocol:

a. Carol wants some digital cash in the amount of c dollars. She blinds c, obtaining $t = cr^3 \bmod n$.
b. Carol opens a secure channel to BankOBits and sends BankOBits the following: c,t.
c. BankOBits authenticates Carol (you can assume this is done correctly), decrements Carol's account balance by c dollars, and returns the signed, blinded coin: $u = t^d \bmod n$.
d. Carol can now unblind u to recover the signed coin, coin $= (c, c^d \bmod n)$.
e. When Carol presents the coin to Maxine some time later to buy valuable goods from Maxine, Maxine can verify the bank's signature and feel confident the coin is valid.
f. Finally, Maxine can present the coin $(c, c^d \bmod n)$ to BankOBits to get the money deposited into her account. BankOBits will first verify the signature before honoring the coin and increasing Maxine's account balance by c dollars.

With this protocol, one way that Carol can cheat is to double-spend. That is, she can make a digital copy of c^d and present it to two different merchants. Name one other way that Carol can cheat the bank to get free money.

Answer 1: Carol can send to the bank c,t where c = \$5 and $t = 10r^3 \bmod n$. The bank will sign a coin worth \$10, but only deduct \$5 from Carol's account.

Answer 2: Carol can withdraw a valid coin worth \$10, by using the protocol above:

$$\text{coin}_1 = (c_1, s_1) = (10, 10^d \bmod n).$$

Then, she can withdraw a valid coin worth \$6, again using the protocol above:

$$\text{coin}_2 = (c_2, s_2) = (6, 6^d \bmod n).$$

She can then use these two valid coins to create a third coin:

$$\text{coin}_3 = (c_1 c_2, s_1 s_2 \bmod n) = (60, 6^d \times 10^d \bmod n) = (60, 60^d \bmod n).$$

Both the merchant and bank will accept this coin as valid, since the signature is valid. Note that once Carol obtains the first two coins from the bank, she has all the information needed to form the third coin entirely on her own (without interacting with the bank). In the process of withdrawing the first two coins, Carol's account will be reduced by \$16. However, Carol now has three valid coins worth a total of \$76. Profit!

(d) Dr. Alizarin suggests that we fix the protocol by making the denomination c the same for all coins: the bank will announce in advance that every coin signed under the modulus n will be worth exactly one dollar. Now, the coin will be derived from a random 128-bit string z. In particular, the coin will be given by $\text{coin} = (z, s)$ where $s = H(z)^d \bmod n$.

Whenever a merchant receives a putative coin from a customer, the merchant immediately deposits it with the bank. When a merchant presents a signed coin to the bank for deposit, BankOBits checks its database to make sure a coin with the same string z hasn't been previously deposited. If the coin has not been previously deposited and is validly signed, BankOBits increments the merchant's balance, adds z to the database, and returns a success message to the merchant; otherwise, BankOBits returns an error message to the merchant. The merchant ships the goods to the customer only after receiving a success message from the bank.

Will Dr. Alizarin's scheme prevent counterfeit/copied coins? In other words, does it satisfy requirements 3 and 4 above?

Answer: Yes.

Comment: Because the bank stores a list of all previously-deposited coins, each coin can be spent only once. Because the coin is signed by a secure signature scheme, coins cannot be counterfeited. The cryptographic hash function H prevents the attack listed in "Answer 2" to part (c).

(e) Does Dr. Alizarin's scheme satisfy requirement 2 above?

Answer: Yes.

(f) Suppose Carol is one of 10,000 customers withdrawing coins on Monday, and 500 merchants deposit coins on Tuesday. Assuming Alice spent some of her coins at some merchant on Tuesday, can the bank identify exactly which merchant Alice spent her digital cash at?

Based on this, do you think that Dr. Alizarin's scheme satisfies requirement 1 above? You can assume BankOBits has been wildly successful and has many customers, and that customers tend to withdraw coins from the bank well in advance and spend them only much later.

Answer: The bank cannot identify at which merchant Alice spent her cash; the scheme does satisfy requirement 1. When the bank receives a coin $(z, H(z)^d)$ from by the merchant, the bank cannot tell which customer withdrew this coin, due to the blind signature.

In more detail, suppose there were 10,000 instances of the withdrawal protocol, so the bank sees $t_1, \ldots, t_{10000}$ (t_i is the blinded version of $H(z_i)$, for the ith coin withdrawn; i.e., $t_i = H(z_i) r_i^3 \bmod n$). Now suppose that some coin $(z_i, H(z_i)^d)$ is presented to the bank for deposit. The bank cannot identify which of the 10,000 withdrawal requests this corresponds to, for reasons similar to those articulated in response to question 1(b). In short, the bank has no way to link withdrawals to deposits. As far as the bank can tell, each deposited coin might have come from any of the customers who previously withdrew a coin of the same denomination.

Comment: In fact, Dr. Alizarin's scheme is a well-known scheme for anonymous digital cash. It was invented by the cryptographer David Chaum, while he was a Ph.D. student at Berkeley, and was later deployed for a brief time by a company called Digicash. However, Digicash eventually folded, along with many other startups developing payment schemes; Paypal eventually beat out most of the competition.

(g) Prof. Jasper is unhappy that Dr. Alizarin's scheme requires the merchant to contact the bank immediately. Prof. Jasper proposes an alternate scheme that allows the merchant to accept digital cash, even if the merchant is not Internet-connected at the time of purchase. In particular, Prof. Jasper proposes that a coin should have the form coin = $(c||z, s)$ where $s = \text{Sign}_{\text{BOB}}(c||z)$, i.e., $s = m^d \bmod n$ where $m = H(c||z)$. (Here $||$ denotes concatenation of bit strings.) Whenever a merchant presents a signed coin to the bank for reimbursement, BankOBits checks its database to make sure a coin with the same string z hasn't been previously deposited.

In addition, if the bank does detect double-spending it would like to know if it was the merchant or a customer who tried to cheat. To aid in this, BankOBits requires that after the merchant has verified the signature on $(c||z, s)$, the merchant asks the customer to provide a random 128-bit string *id*, and the merchant provides $(c||z, s, id)$ to BankOBits. If, when the merchant presents $(c||z, s, id)$ to the bank, the bank detects double-spending (i.e., some other coin with the same z was previously deposited), the bank will then compare the value of id on the coin provided by Maxine to the value of *id* stored in the database with the doubly-spent coin. If the values of *id* are the same then the bank knows the merchant cheated. If they are different then the bank knows that the customer cheated.

Will Prof. Jasper's scheme work? That is, will the bank be able to tell whether it was the merchant or Carol who cheated in the case of doubly-spent money?

Answer 1: Prof. Jasper's scheme will not work. Carol can frame the merchant by using the same id twice. If the bank is not tracking which merchant presented which coin then Carol can use the same id at two different merchants. If the merchants are not tracking which id's they've already seen then Carol can use the same id twice at the same merchant.

Answer 2: Prof. Jasper's scheme will not work. The merchant can frame Carol. The merchant can present the same coin to the bank twice, but change the id to a new value each time.

2. (30 pts.) Onion Routing

(a) Give two reasons why usability might be considered a security *requirement* for using onion routing for anonymity (such as by systems like Tor).

Answer: Four possible answers are:

(a) Onion routing protocols hides each user within a large population of users. If the protocol is hard to use, it will attract fewer users, providing less anonymity for those who do use the protocol. Anonymity in Tor relies upon being just one in a crowd; if only N people use Tor, then an attacker can deduce that any particular Tor connection must have come from one of those N people, so the larger N is, the better anonymity everyone receives.

(b) If the protocol is hard to use, there will be fewer users, making it easier for an adversary to control a large proportion of onion routers. The larger the proportion of onion routers controlled by the attacker, the greater the likelihood that for any individual Tor connection, all onion routers on the path will be controlled by the attacker; when that happens, all anonymity is lost.

(c) If the protocol is hard to use, users may be more likely to set up their nodes improperly, making the protocol insecure for all users whose traffic passes through those nodes.

(d) If the protocol is hard to use, users will turn it off completely, losing all anonymity.

(b) Bob is concerned about his privacy online so he always uses an onion router (the same as a "mix" discussed in lecture). He makes sure that all his outgoing traffic (e.g., HTTP requests, DNS requests) goes via the onion router. However, Bob does not take any steps to scrub the contents of his requests of identifying information. For each of the following, briefly describe whether it could be used by a web server Bob visits to learn Bob's identity, and explain why or why not.

i. Cookies

Answer: If Bob accepts cookies from a web server, all future requests to that server will be accompanied by the cookies. The server may not learn Bob's IP address, but it could easily track all of Bob's activities on the server (i.e., link these activities and deduce that all of these activities were performed by the same person). Certainly, if Bob reveals any identifying information in one request (e.g. if he provides his name on some form), that information can be tied to all of his requests.

ii. Web bugs

Answer: Web bugs allow third parties to track visitors to your site. This would let a server track all of Bob's activities, not just at one site, but across all sites that include a web bug from that third party. The third party may not learn Bob's IP address or name, but will be able to put together a trace of sites visited by Bob. Depending on what kind of sites Bob visits, this could be used to put together a likely profile of Bob.

iii. Referer field

Answer: The Referer field will reveal information about the previous site that Bob visited. This wouldn't reveal Bob's IP or name, but it could leak some identifying information (e.g., if the previous site was `DegrassiHigh2005Renunion.com`).

iv. Google Analytics

Answer: Google Analytics collects and reports a great deal of information, including the location and host name of the client. This is personally identifiable information.

(c) Suppose Alice sends a message to Bob using the onion routing protocol described in class. Neither Bob nor any of the onion router mixes (other than the first) along the circuit know the identity of Alice. How would you augment the protocol to allow Bob to send a reply to Alice without learning

the identity of Alice? What could Alice include in her initial message that Bob could then use to get a message back without learning her identity? You may add new functionality to all onion routers, but you must explain how your new functionality would work.

Answer: There are a bunch of ways to do it, but one simple way involves setting up a reply channel back through the mixes. When Alice sets up a route to Bob through mixes M_i, M_h, M_k (in that order), she also sends instructions to the individual mixes to establish a return channel that Bob can use to send responses back to her. This sets up everything Bob needs to get a response back to Alice, without revealing Alice's identity.

Let's see how this could work, starting with a simple case where Alice's path contains just a single Tor router (a single hop). Suppose she decides to route through mix M_i, whose public key is K_i. She establishes a pseudonym P, and sends M_i the following instructions encrypted under K_i:

$$\{\text{“Whenever you receive a message addressed to pseudonym } P\text{, please forward it on to Alice.”}\}_{K_i}$$

When she sends a message to Bob, she sends him the pseudonym P, and instructs him to encrypt his reply under key K_{Alice}, address his reply to pseudonym P at mix M_i, and encrypt the whole thing under key K_i. In other words, if Bob wants to reply with the message X, he sends

$$\{P, \{X\}_{K_{\text{Alice}}}\}_{K_i}$$

to M_i. Now the mix M_i can decrypt, see that this is addressed to pseudonym P, look up pseudonym P in its table of pseudonyms, and find out that the corresponding message should be forwarded to Alice; so M_i sends $\{X\}_{K_{\text{Alice}}}$ to Alice. That's a single hop.

If Alice wants better anonymity, she can use two hops. Maybe she decides to route through mixes M_i and M_j, whose public keys are K_i and K_j. Then she would set up a pseudonym P_1 and send M_i the following instructions, encrypted under K_i:

$$\{\text{“Whenever you receive a message addressed to pseudonym } P_1\text{, please forward it on to Alice.”}\}_{K_i}$$

Alice would also set up a pseudonym P_2 and send M_j the instructions:

$$\{\text{“Whenever you receive a message addressed to pseudonym } P_2\text{, please forward it on to } M_i\text{, addressed to } P_1 \text{ and encrypted under key } K_i\text{.”}\}_{K_j}$$

The instructions are encrypted under key M_j and send it to M_j over an anonymized path that goes through M_i, so M_j does not learn Alice's identity. Finally, when Alice contacts Bob over the anonymous channel, she instructs him to send his reply to M_j and address it to pseudonym P_2. So if Bob wants to reply with message X, he sends

$$\{P_2, \{X\}_{K_{\text{Alice}}}\}_{K_j}$$

to M_j. Mix M_j can decrypt, look up the pseudonym P_2, and send

$$\{P_1, \{X\}_{K_{\text{Alice}}}\}_{K_i}$$

to M_i. Mix M_i can decrypt, look up P_1, and forward to Alice:

$$\{X\}_{K_{\text{Alice}}}$$

Note that neither mix is able to deduce that Alice is corresponding with Bob. M_j knows that someone is corresponding with Bob through M_i, but doesn't know the client's identity; M_i knows that Alice is corresponding with someone through M_j, but doesn't know who is sending replies back to her. Thus anonymity is preserved.

You can generalize this to multiple hops.

3. (30 pts.) Reasoning about memory-safety

We'd like to know the conditions under which `sanitize()` is memory-safe, and then prove it.

Find the blank space labelled `(a) requires:` and fill it in with the precondition that's required for `sanitize()` to be memory-safe. (If several preconditions are valid, you should list the most general precondition under which it is memory-safe.)

Then, find the four blanks (b)–(e) inside `sanitize()` and fill each of these with invariants so that (1) each invariant is guaranteed to be true whenever that point in the code is reached, assuming that all of `sanitize()`'s callers respect the precondition that you identified, and (2) your invariants suffice to prove that `sanitize()` is memory-safe, assuming that it is called with an argument that satisfies the precondition that you identified.

Answer:

```
/* (a) requires: s != NULL && 0 <= n <= size(s) */
int sanitize(char s[], size_t n) {
  size_t i=0, j=0;
  while (j<n) {
    /* (b) invariant: s != NULL && 0 <= i <= j < n <= size(s) */
    if (issafe(s[j])) {
      s[i] = s[j];
      i++; j++;
      /* (c) invariant: s != NULL && 0 < i <= j <= n <= size(s) */
    } else {
      j++;
      /* (d) invariant: s != NULL && 0 <= i < j <= n <= size(s) */
    }
    /* (e) invariant: s != NULL && 0 <= i <= j <= n <= size(s) */
  }
  return i;
}
int issafe(char c) {
  return ('a' <= c && c <= 'z') || ('0' <= c && c <= '9') || (c == '_');
}
```