Homework 1: Crypto Party

Due: Thursday, February 23 @ 11:59 pm EST

Overview and instructions

This homework has 6 problems:

- Problems 1–4 are required for all students
- Problems 5–6 are required for **CS1620/CS2660 students only**

Note on collaboration

You are welcome (and encouraged!) to collaborate with your peers, but the solutions you write down must be **your own work** (ie, written by you). You are responsible for independently understanding all work that you submit—after discussing a problem as a group, you should ensure that you are able to produce your own answers independently to ensure that you understand the problem. For more information, please see the course Collaboration Policy.

In your submission, we ask that you include a brief *collaboration statement* describing how you collaborated with others on each problem—see the next section for details.

How to submit

You will submit your work in PDF form on Gradesope. Your PDF should conform to the following requirements:

- **Do not** include any identifying information (name, CS username, Banner ID, etc.) in your PDF, since all homeworks are graded anonymously
- Each problem (where "problem" is one of the Problems 1–4 or 1–6) should start on a separate page. When you submit on Gradescope, you will be asked to mark which pages correspond to which problem
- At the start of each problem, write a brief *collaboration statement* that lists the names and CS usernames of anyone you collaborated with and what ideas you discussed together
- If you consulted any outside resources while answering any question, you should cite them with your answer

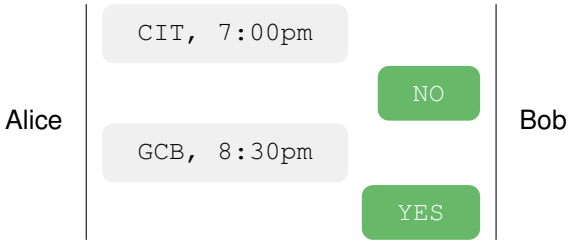
There are two separate Gradescope submissions for this assignment:

- All students should submit Problems 1–4 to the assignment labeled **"Homework 1: Problems 1–4"**
- CS1620/CS2660 students must also submit Problems 5–6 to the assignment labeled **"Homework 1: Problems 5–6"**. For this part, you can either make a separate PDF with problems 5–6, or just have one PDF and then mark the pages for these problems. Submissions for Problems 5–6 from CS1660-only students will not be graded (ie, there is no extra credit).

Problem 1: Dating with Public Keys

Alice and Bob, both Brown CS students, are secretly dating. In order to set up a meeting, they exchange encrypted messages using a *deterministic* public key encryption scheme (deterministic meaning that multiple encryptions of a given plaintext always produce the same ciphertext).

Alice and Bob both have their own public-private key pair, (PK_A, SK_A) and (PK_B, SK_B), respectively. When Alice wants to send a message, m, to Bob, she encrypts it using Bob's public key and sends the resulting ciphertext, $c = \mathsf{Enc}(m, PK_B)$, to him. Similarly, Bob's messages to Alice are computed from $\mathsf{Enc}(m, PK_A)$. In plaintext, the messages Alice and Bob exchange are always of the following form:



Assume that they always plan to meet at a building on Brown's campus, and that,when referring to a specific Brown building, the names they use are consistent (i.e. they won't alternate between "SciLi" and "Sciences Library").

Answer the following questions based on this scenario. Your responses to each question should be no more than one paragraph (around 150 words) each.

Question a) Eve wants to find out about the secret dates between Alice and Bob and knows both of their public keys, the form of their messages (including the name they use to refer to all of Brown's buildings), and can eavesdrop on the ciphertexts being exchanged. Describe how Eve can find out when and where the next meeting is going to be, even though Eve is unable to learn the secret keys. (*Hint:* The encryption scheme is *deterministic*).

Question b) TRUE or FALSE: Eve is an IND-CPA adversary. Explain.

Question c) Alice and Bob found out that Eve can learn about their meetings. Propose a *simple* modification to the protocol that prevents the attack from part (a) and explain why the protocol will prevent the attack, and explain why it works. By "simple", your new protocol cannot rely on sending additional messages or require changing any of the cryptographic functions involved or adding new cryptographic functions.

Sample TA solution

(a) Eve can take advantage of the fact that if we encrypt the same plaintext twice with a deterministic cryptosystem then we get the same ciphertext. Since the number of buildings around Brown is relatively small, as is the number of possible times that Alice or Bob could encode using their format, she can simply generate every possible plaintext by combining every possible time with every possible building name. Then she can encrypt each of these with each of the public keys, and build a table mapping ciphertexts to their corresponding plaintexts. Now whenever she intercepts a ciphertext, so long as the message that was used to generate it follows the agreed-upon format, the ciphertext will appear in her table, and she will learn the plaintext that it corresponds to.

(b) TRUE. Eve's use of public key encryption in part (a) to build her table of ciphertext mappings is equivalent to the "setup phase" from the definition of IND-CPA, where the adversary can perform a polynomially bounded number of queries to the encryption oracle. Eve has direct access to the "encryption oracle" in this case because she can perform public key encryptions herself (even though she doesn't have access to the private key).

(*Aside:* The notion of "IND-CPA adversary" really just means an adversary who has direct access to an encryption oracle.)

(c) We present two possible solutions.

Initialization Vector. One way for them to avoid the attack is to add a long, random string at the end of their message (an initialization vector). As long as two different plaintexts differ by even a bit, their ciphertexts will be completely unrelated. Thus, if the random strings are long enough, the probability of ever generating the same one over the lifetime of their communication will be vanishingly small.

In order for Eve to again brute-force the messages, she would have to, for every possible message, also try every possible random value. For long enough random strings, Eve's attack will be effectively useless.

(*Aside:* It's critical to specify that the strings should be "long" and "random" enough such that they're unguessable and effectively unique, otherwise, Eve can potentially brute-force an attack.)

Timestamp (with nanosecond precision). Alice and Bob can add a timestamp with enough precision (say, nanoseconds) to the end of each message. We assume that Alice and Bob won't send multiple messages in a given nanosecond, and that a nanosecond is enough precision such that Eve cannot brute force all possible timestamps with each building location and time, etc. This will cause each plaintext to differ, which will cause all ciphertexts to differ, which will cause them to not match.

For the same reasons as the "Initialization Vector" scenario, it is unfeasible for Eve to brute force all possible timestamps with each building location and time. In theory, Eve could build a "database" for a particular nanosecond in the future, but assuming nanoseconds have high time precision that Eve cannot feasibly do this for every nanosecond, the likelihood of any given message sent by Alice or Bob falling into the nanosecond Eve prepares for is low.

Problem 2: Exceptional Access

Law enforcement agencies have been lobbying for exceptional access to encrypted communications. However, many cybersecurity experts have argued that enabling exceptional access would have untenable security consequences. Two cryptographers from GCHQ (the UK's equivalent of the NSA) have proposed a method of exceptional access that they believe does not compromise the integrity of encryption in the following article:
`https://www.lawfareblog.com/principles-more-informed-exceptional-access-debate`.
Please read over this article and then consider the following questions.

For another example, this (optional) article describes a Cisco Webex vulnerability which unintentionally allows "ghost access" to meetings:
`https://www.securityweek.com/cisco-webex-vulnerability-allows-ghost-access-meetings`

These are open questions and will be graded for completeness or your responses and justification of your claims. Please reference and/or quote specific arguments from the readings in your answers.

Question a) The first article discusses several principles that can be used to evaluate exceptional access methods. What standards would need to be followed for you to find an exceptional access reasonable (3 standards max)? If you believe that exceptional access is never acceptable, please justify why. How does your answer differ from the GCHQ principles, if at all?

Question b) Do you believe that the GCHQ authors' fifth principle—that "any exceptional access solution should not fundamentally change the trust relationship between a service provider and its users"—is met by their proposed exceptional access mechanism? Why or why not? (4–6 sentences)

Question c) One argument when it comes to exceptional access is that if companies don't provide a mechanism for the government to access communications between users, then the only option left for government agencies is hacking, ie. finding or exploiting known vulnerabilities in a system. However, relying on such vulnerabilities can incentivize governments to avoid disclosing them to the public so they remain useful to law enforcement. Do you find hacking a compelling mechanism for providing exceptional access? Why or why not? (4–6 sentences)

Sample TA Solution

(a) There are two approaches that can be taken by students: arguing that exceptional access is never justified, and listing principles that they believe would make an exceptional access method acceptable.

Typically, arguing exceptional access is never justified would be done by claiming that any method that allows unintended recipients to access encrypted data necessarily constitutes a cybersecurity vulnerability, and that vulnerabilities have worse consequences (i.e., potential for significant cyberattacks, espionage, loss of trust in software services, potential compromise of journalists' ability to communicate with sources) than an inability to access encrypted data (failure to prevent and prosecute crimes).

See `https://dspace.mit.edu/bitstream/handle/1721.1/97690/MIT-CSAIL-TR-2015-026.pdf` for a justification of why exceptional access mechanisms necessarily create vulnerabilities, `https://www.washingtonpost.com/world/national-security/former-national-security-officials-urg 2015/12/15/3164eae6-a27d-11e5-9c4e-be37f66848bb_story.html` for background on national security vs. law enforcement disagreements on exceptional access, and `https://www.thirdway.org/report/weakened-encryption-the-threat-to-americas-national-security` for a pure national security argument against exceptional access.

For the second approach, below are three examples of standards that might be suggested, although there are many more that students could reasonably suggest. For additional examples, see pages 13-14 of `https://carnegieendowment.org/files/EWG__Encryption_Policy.pdf` (intended for on-device key escrow but nonetheless broadly applicable).

1. Software providers cannot be required to design their systems in preparation for exceptional access.

While one-off exceptional access assistance can create temporary vulnerabilities, designing systems in a way that compromises confidentiality or integrity upfront creates persistent vulnerabilities. Additionally, this is antithetical to security by design. Although this may lead to firms redesigning their systems to make exceptional access methods impossible (i.e., by allowing the client to verify whether any keys have been changed or added, which the ghost proposal requires), these decisions would necessarily improve the overall security of these systems. For context, system redesigns to enable persistent methods of exceptional access can be required by Australia's exceptional access legislation (*the Telecommunications and Other Legislation Amendment (Assistance and Access) Act*) through Technical Capability Notices. This standard is in conflict with the view expressed by former FBI Director James Comey, who argued that "it makes more sense to address any security risks by developing intercept solutions during the design phase, rather than resorting to a patchwork solution when law enforcement comes knocking after the fact" `https://www.fbi.gov/news/speeches/going-dark-are-technology-privacy-and-public-safety-on-a-collision-course`.

2. A judge, in determining whether to allow a particular form of exceptional access, must be presented with a report on the security consequences of the proposed method by an independent cybersecurity expert. The conclusions of the report must be made public.

   Although it is possible for firms to protest exceptional access orders, it is likely that they will consistently claim that methods are too risky while governments are likely to consistently claim that the risk is minimal. This may to lead to situations in which judges are forced to weigh well-defined national security consequences against contested cybersecurity risks. When the judge cannot evaluate the cybersecurity risks, they could inadvertently allow exceptional access orders where there is significant risk, or vice versa. An independent expert would provide the judge with unbiased advice on the consequences of the order, allowing them to evaluate its proportionality more effectively. Additionally, publicising the expert's finders would help to either assuage public concerns about the security of the products they use (if the orders are reasonable) or put pressure on government agencies to proactively suggest methods that maintain the cybersecurity of these systems (if the orders are reckless).

3. Requests for exceptional access must allow sufficient time for secure methods of retrieving the information to be developed.

   This would prevent access methods from being rushed, limiting the likelihood of serious vulnerabilities being introduced. Although the rhetorical examples typically used when justifying exceptional access are time-sensitive (i.e., a bomb will go off and the location is on an encrypted hard drive), situations like this are unlikely (even the case examples provided by Comey in the article cited earlier are all after the fact evidence gathering) and the risk caused by rushing exceptional access is tremendous. Given that a large part of the concern about exceptional access is that vulnerabilities may be created by developer error, shortening the timespan for implementation greatly increases the risk that serious vulnerabilities are introduced. Additionally, rushed implementation may compromise the validity of review processes both internally and by independent evaluators (such as in standard 1).

The key area of contention between these standards and the GCHQ principles is that the GCHQ principles leave the door open to changing the design of systems to make exceptional access easier. In fact, their ghost proposal is only possible if companies can be forced to actively change the design of existing messaging apps (including Signal and WhatsApp).

(b) *Argument against:*

Even if, as the GCHQ authors claim, "you don't even have to touch the encryption" in their proposed system, ghost access nonetheless constitutes a significant change in the trust relationship between a service provider and its users. That is because, as discussed in lecture 3, encryption is only half of the equation: users need to trust that their messages are not just encrypted, but also that their messages are only sent to the intended recipients. Additionally, this proposal requires the ability to suppress notifications that users have elected to receive, thus preventing the user from trusting the integrity of

the notification system.

Arguments for are difficult when using "trust" in a theoretical cybersecurity sense, but possible if viewing it through the same lens as the GCHQ cryptographers: if the risk of exploitation is minimal, and you believe that the government is not interested in your communications, then the trust relationship between user and service provider is not fundamentally changed. Arguments along these lines are valid, and are interesting in that they highlight a key point of contention in the exceptional access debate.

(c) I do not think that this is a valid argument. As seen with the Cisco-Webex vulnerability discussed in the second reading, there are independent security researchers working to find vulnerabilities with motivation of finding these first to receive a bounty. Although the government could find hacks into software and devices allowing them access to important user-data, it would only be a matter of time until another security researcher finds the same vulnerability used by the government and discloses this to the company who will then fix the bug. Therefore, it is not viable for the government to assume that hacking will always work. (4 sentences)

Problem 3: CIT-Branded Block Ciphers

Someone has ignored all of the warnings about hand-rolling your own cryptography (see the Lecture 3 readings) and has created a new block cipher mode called CIT mode. Figure a provides an overview of how CIT mode works—you can assume that the block cipher used is a secure, deterministic block cipher with a block size and key size of 256 bits.

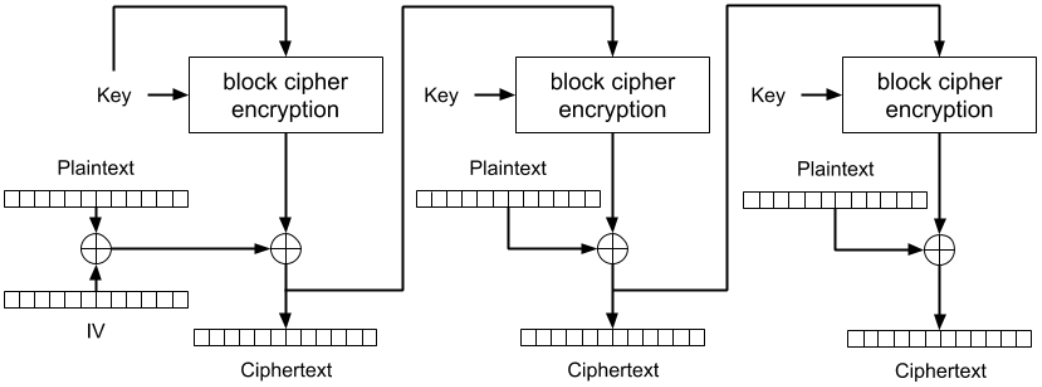


Figure 1: The encryption scheme for CIT mode.

(And yes, you're seeing that correctly—the CIT block cipher mode involves encrypting the encryption key with itself...)

For **parts (a) through (e)**, limit each of your answers to **100 words maximum**. Part (f) has no word limit.

Question a) Either draw a diagram or write out equations of the decryption scheme for CIT mode using the notation for describing cipher modes from the lectures[1]. Make sure to label all parts of your diagram or variables used.

Question b) Suppose CIT mode is used to encrypt two equal-length plaintext messages, m_1 and m_2, using the same key *and the same initialization vector* to produce two ciphertexts, c_1 and c_2. m_1 and m_2 differ in at least one bit at some index i.

Consider an adversary, Eve, who can eavesdrop on the ciphertexts; that is, Eve can see c_1 and c_2. What, if anything, can Eve learn about the underlying plaintexts m_1 and m_2?

Question c) Suppose that exactly one bit at index j of a given ciphertext, c, which was encrypted in CIT mode, is corrupted in transmission (that is, after the encryption is complete). When c is decrypted using CIT decryption, which bits of the plaintext will be corrupted? Explain.

Question d) TRUE or FALSE: Encryption in CIT mode is parallelizable.[2] Explain.

Question e) TRUE or FALSE: Decryption in CIT mode is parallelizable. Explain.

Question f) TRUE or FALSE: CIT mode is IND-CPA-secure. Explain.

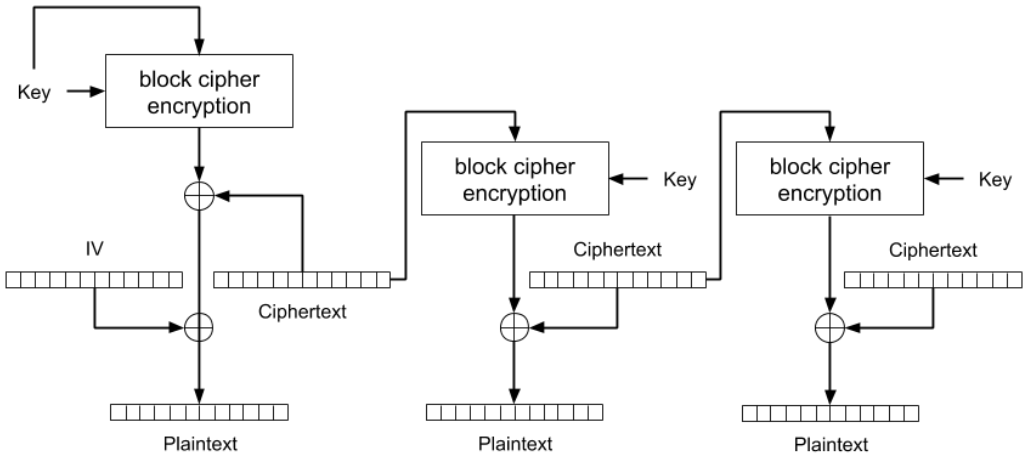
(*Hint:* In a IND-CPA game, the key remains fixed throughout the duration of the game, but the IV used will be different for every encryption.)

[1] You can see examples for writing cipher mode equations in the Cryptography II lecture, slides 12 and 16.

[2] "Parallelizable" in the context of block ciphers comes from our discussion of ECB mode in Lecture 3: that is, for a given input plaintext (split into fixed-length blocks), we should be able to compute ciphertext blocks without having to wait for previous ciphertext blocks (and thus we can parallelize the computation of the ciphertext blocks), and vice versa for decryption of ciphertext blocks to plaintext blocks. Thus, we say encryption and decryption in ECB mode is parallelizable.

Sample TA Solution

a) The decryption mode diagram looks like this:



(*Aside:* For full credit, remember to demonstrate that the key is one of the inputs to each application of the block cipher; also remember that "triple XOR" is not really a valid operation in this scenario, since $\oplus(1,1,1) = 0$ while $\oplus(1, \oplus(1,1)) = 1$.)

The decryption mode equations are as follows:

From the diagram, we can define an encryption scheme that takes $P_0, P_1, \ldots, P_n$ as plaintext inputs and outputs $IV, C_0, C_1, \ldots, C_n$. For the first block, we find that $C_0 = P_0 \oplus IV \oplus E_k(k)$. Then for any $n > 0$ we can define $C_n = P_n \oplus E_k(C_{n-1})$. Rearranging these equations by XORing terms, we can define a decryption scheme. We find $P_0 = C_0 \oplus IV \oplus E_k(k)$. Then, for any $n > 0$ we find $P_n = C_n \oplus E_k(C_{n-1})$.

b) Let i be the *first* index at which m_1 and m_2 differ for a given bit. You then learn the following:

- $E_k(k) \oplus IV$ will be constant, so the first ciphertext block will be constant if the underlying plaintext block is constant. From here, you can see that all following blocks are the same up until the block that contains i, since the ciphertexts will be the same.
- In the block that contains i, you learn all of the indices of matching bits in that block.
- In the blocks following the block containing i, you get no information.

c) Two things are corrupted, and nothing else:

- Bit j of the plaintext is corrupted.
- All of the bits in the plaintext block that comes directly after the block containing bit j are corrupted; all the other blocks following this block are fine.

d) FALSE. You need to know the ciphertext in the previous block to compute the next ciphertext.

e) TRUE. Each ciphertext can be independently converted back into the plaintext just by knowing the ciphertext block and the key (see the diagram from part (a)).

f) FALSE. There are a few possible adversary strategies; here are some examples.

Double IV XOR approach. Suppose you encrypt M twice with the same key but different IV. Let p_1 be the first block of the plaintext of M. The first blocks of the ciphertexts will be $E(k,k) \oplus \mathsf{iv}_1 \oplus p_1$ and $E(k,k) \oplus \mathsf{iv}_2 \oplus p_1$. You can thus xor the first blocks of the ciphertexts together to get $\mathsf{iv}_1 \oplus \mathsf{iv}_2$. Since the IVs are public (because they're part of the ciphertext), if xor-ing the first blocks of the ciphertexts produces $\mathsf{iv}_1 \oplus \mathsf{iv}_2$, then we know that p_1 was the first plaintext block in both messages.

Thus, the adversary strategy involves:

- Query phase: Send some m_1 and m_2 in the query phase where the *first blocks* of m_1 and m_2 differ in at least one bit to generate ciphertexts c_1 and c_2.
- Challenge phase: Send the same m_1 and m_2 from the query phase and receive the challenge ciphertext c'. You can then use the above XOR strategy to xor the first block of c' with the first block of c_1 (and same for the first block of c_2) to determine which of m_1 or m_2 was encrypted to produce c'.

Recover $E(k, k)$ approach. In the query phase:

- Send a single-block message m_i, which returns $\mathsf{ct} = \langle \mathsf{iv}; c_i \rangle$. Because $c_i = E(k, k) \oplus \mathsf{iv} \oplus m_i$, we can recover $E(k, k)$ via $E(k, k) = c_1 \oplus \mathsf{iv} \oplus m_i$.

In the challenge phase:

- Send two single-block messages m_1 and m_2 where $m_1 \neq m_2$. We are given $\mathsf{ct} = \langle \mathsf{iv}; c_i \rangle$, and we know $c_i = E(k, k) \oplus \mathsf{iv} \oplus m_i$. We can then rewrite our equation to show that $m_i = c_i \oplus E(k, k) \oplus \mathsf{iv}$. Since we recovered $E(k, k)$ in the query phase, it's easy for us to then compute m_i (since we are given c_i and iv) and thus we can determine which m_i was sent by the challenger.

Problem 4: TryHackMe Lab: Burp Suite

If you completed HW0, you should have already registered for a TryHackMe account. If so, you have been added to the CS1660 course and have been granted a premium account. If you have not completed HW0, please register for TryHackMe as soon as possible—you may have already received an invite to do so when we approved your account for premium access.

For this problem, you will gain some practice using Burp Suite, a set of tools for testing web applications, which you may find useful for the next project. Depending on when you start this lab, some of the topics about web applications may not have been discussed yet—we will review them in the next few web security lectures.

To begin the lab, log into TryHackMe and go to `https://tryhackme.com/jr/cs1660burpra`, then follow the instructions.

Grading note There is nothing to submit for this part. As you complete each task in the lab, your progress is automatically recorded such that we can view it. TryHackMe rooms are graded based on *completion*, not correctness. As long as you have answered all of the questions, you will get full credit. This lab should not take more than 1 hour to complete—if you are stuck or are dealing with technical issues, make sure to post a question on Edstem.

Problem 5: Hash Functions, RSA Edition? (1620/2660 only)

Note: Only CS1620/CS2660 students are required to complete this problem.

In the *Cryptography II* lecture, we discussed how public-key digital signature cryptosystems generally form signatures *over a cryptographic hash* of the intended message. Signing a hash of the message (rather than the message itself) has performance benefits for long messages, since asymmetric cryptography is quite expensive. However, if we are in a situation where the messages are always small (smaller than the output length of hash), does removing the hash function result in a more performant cryptosystem that's just as secure? This problem gives a bit more background on how RSA works, and explores a few scenarios to help us answer this question.

Background: RSA signatures, with hashing Consider the standard RSA public-key cryptosystem described below, which we'll denote as "RSA standard" ($\mathsf{RSA}_{\text{std}}$):

- A standard RSA key-pair is formed via public key $PK = \langle n, e \rangle$, where n is the "RSA modulus" (a large prime number that defines the key length) and e is some fixed, small, prime number, and private key $SK = d$, where d has the property $(x^e)^d = x \mod n$ for all x. (For the purpose of this problem, exactly how these numbers are generated doesn't matter.) In this problem, let's assume $e = 3$.
- In $\mathsf{RSA}_{\text{std}}$, the signing operation $\mathsf{Sign}_{SK}(M)$ works by computing $S = \mathsf{hash}(M)^d \mod n$, where S is the signature on M. To verify the signature S, a third-party can execute $\mathsf{Verify}(M, S, PK)$, which results in a correct verification if $S^e = \mathsf{hash}(M) \mod n$ holds true. (Note that the verification step does not require knowledge of d, only $PK = \langle n, e \rangle$.)

You can assume that the hash function satisfies all of the properties of cryptographic hash functions.

Now, consider the following simplified scheme $\mathsf{RSA}_{\text{simple}}$. In $\mathsf{RSA}_{\text{simple}}$, $\mathsf{Sign}_{SK}(M)$ is exactly the same as the corresponding operation from $\mathsf{RSA}_{\text{std}}$, except without the application of the hash function. That is, in $\mathsf{RSA}_{\text{simple}}$, $\mathsf{Sign}_{SK}(M)$ produces the signature $S = M^d \mod n$ and $\mathsf{Verify}(M, S, PK)$ checks if $S^3 = M \mod n$ holds true.

Question a) Alice and Bob are using $\mathsf{RSA}_{\text{simple}}$ to send messages with integrity guarantees to each other. Eve is a network attacker who can inject, modify, and drop messages sent between Alice and Bob. Eve wants to trick Bob into believing that Alice sent a message that Alice did not.

Explain how Eve, who knows Alice's public key PK, can find a (M, S) pair (and give a specific example of such a pair) such that S is a valid signature on M, even without knowledge of Alice's secret key SK. (For part (a), the message M does not have to be chosen in advance and can be random.)

Question b) Bernardo is holding an auction for his personal collection of balloon animals. In this auction, Alice and Bob submit bids to Bernardo, where their bids are signed using $\mathsf{RSA}_{\text{simple}}$. Each bid M is an integer (in dollars), and they will send their $\mathsf{RSA}_{\text{simple}}$ signature on M; that is, they send $S = M^d \mod n$. Bernardo will accept whichever bid is highest (and will expect that person to pay up however much they bid!).

Suppose Alice sends a bid M (and its corresponding signature S) to Bernardo. Is it possible for Eve to tamper with S in such a way that he can find a new signature, S', that corresponds to a bid that is some x times as much as Alice's original bid? Explain. (You can choose any value of $x > 0$, and you can assume that $xM < n$ for your chosen x.)

Question c) Explain why the use of a cryptographic hash function in $\mathsf{RSA}_{\text{std}}$ prevents the forgery attacks described in parts (a) and (b). (*Hint:* How do the properties—and which properties—of a cryptographic hash function prevent these kind of attacks?)

Sample TA Solution

(a) The trick is that it's difficult to go from some chosen M to a valid S, but it's easy to go from S to a valid M. In order to create a valid signature Eve can choose an arbitrary value for S and then calculate $S^3 \mod n$. For instance, the pair $(M = 1, S = 1)$ works.

(b) The idea is to multiply M by x and S by the eth root of x.

For instance, let $x = 64$. Then, multiply S by 4 (since 4 is the 3rd root of 64). This works because: $4S = 4M^d \mod n$ and thus $(4S)^3 = 64M^d \mod n$.

(c) The *one-way* property makes it difficult for us to find M's that will hash to the value that we need (using our strategy from part (a)). *Collision resistance* is generally helpful here to also avoid an attack where the adversary finds an M' that hashes to the same value of M (i.e. if they've seen a previous signature-plaintext pair).

Problem 6: Counting for Cryptographers is Hard (1620/1660 only)

Note: Only CS1620/CS2660 students are required to complete this problem.

In the *Cryptography II* lecture, we described how one can convert a block cipher into a stream cipher using a "counter" technique. This technique is actually more formally known as CTR block cipher mode encryption. Specifically, $\mathsf{CTR}_{\text{lecture}}$ (to denote the scheme specifically shown on Slide 21 in Lecture 3) works in the following way:

- The encrypting user knows the secret key $\langle k, t \rangle$, where k is a random bitstring and t is a "counter" with a number of bits equal to the block size.
- Given a plaintext $M = m_0||m_1||\cdots||m_{n-1}$, the ciphertext generated by $\mathsf{CTR}_{\text{lecture}}$ mode is

  $$C = c_0||c_1||\cdots||c_{n-1}$$

  where each $c_i = E_k(t+i) \oplus m_i$.
- After encrypting an entire message M (not after each individual m_i), the user updates their secret key $\langle k, t \rangle$ to $\langle k, t+1 \rangle$.

For this question, assume that the block size of the block cipher used is 32 bits.

Question a) TRUE or FALSE: $\mathsf{CTR}_{\text{lecture}}$ mode is IND-CPA-secure against an attacker with polynomially bounded resources. Explain. (*Hint:* The answer is FALSE.)

Question b) Is there a way to easily fix the security issues described in part (a)? Under your proposed mitigation, are there any limitations on the number of times the encrypting user can use a given secret key k under your new version of CTR mode? Explain. (*Hint:* How many times can you use the secret key k?)

Sample TA Solution

a) FALSE. Without loss of generality, set $t = 0$. In the query phase:

- Send $M = m_0||m_1$, where M is a two block message. This produces $C = c_0||c_1 = E_k(0) \oplus m_0||E_k(0+1) \oplus m_1$. Then, the internal t in the key is updated to $t = 1$.

In the challenge phase:

- Send m_1 and m_2, where both are single block messages that differ in some bit and m_1 is the same as the m_1 in M from the query phase. This returns $c' = E_k(1+0) \oplus m_i$, where m_i was the message chosen by the challenger.
- If $m_i = m_1$, then notice that $c' = c_1$ from the query phase, since $E_k(0+1) \oplus m_1 = E_k(1) \oplus m_1 = E_k(1+0) \oplus m_i$. Thus, if $c' = c_1$, the adversary outputs the guess $i = 1$, since they can distinguish that m_1 underlies the ciphertext c'.
- Similarly, if $c' \neq c_1$, it must be the other message that was encrypted (since $m_1 \neq m_2$), so the adversary outputs the guess $i = 2$.

b) A couple of solutions:

- Do $E_k(t||i)$ instead of $E_k(t+i)$ (and expand t and i out to 32 bits (i.e. pad them with leading 0's)).
- Instead of adding 1 to t between each message encryption, add n to t, where n is the number of blocks that were encrypted.

The idea here is to simply ensure that you can never have "collisions" between the representation of the $\langle t, i \rangle$ pair passed into the E_k function.The number of times the encrypting user can use a given secret key k is limited by the block size of block cipher, which in this case is 32 bits. Thus, we can encrypt 2^{32} *blocks* before we hit a stream reuse vulnerability.