Final Exam – Practice

MATH 481/581 — Cryptography
Final Exam: Wednesday, May 6, 2026

Please print your name:

Bonus challenge. Let me know about any typos you spot in the posted solutions (or lecture sketches). Any typo, that is not yet fixed by the time you send it to me, is worth a bonus point.

Problem 1. The final exam will be comprehensive, that is, it will cover the material of the whole semester.

- Make sure that you have completed all homework.
- Review the practice problems for both midterms (for the material up to Midterm #2).
- The problems below cover the material since Midterm #2.

Problem 2. We use the (silly) hash function $H(x) = x \pmod{25}$.

Alice's public RSA key is $(N, e) = (55, 13)$, her private key is $d = 17$.

(a) How does Alice sign the message $m = 3141592$?

(b) How does Bob verify her message?

(c) Verify whether the message $(m, s) = (1234, 9)$ was signed by Alice.

(d) Give an example of a collision of our hash function.

Solution.

(a) $H(m) = 17$. The signature therefore is $s = H(m)^d = 17^{17} \pmod{55}$.

Doing the usual binary exponentiation ($17^2 \equiv 14$, $17^4 \equiv 31$, $17^8 \equiv 26$, $17^{16} \equiv 16$), $17^{17} = 17^{16} \cdot 17 \equiv 52 \pmod{55}$. Hence, the signature is $s = 52$.

(b) Bob receives the signed message $(m, s) = (3141592, 52)$.

He computes $H(m) = 17$ and then checks using the public key whether $H(m) \equiv s^{13} \pmod{55}$. Indeed, doing the usual binary exponentiation, $52^{13} \equiv 17 \pmod{55}$, so the signature checks out.

(c) For $m = 1234$, we compute $H(m) = 9$ and then check, using the public key, whether $H(m) \equiv s^{13} \pmod{55}$.

Doing the usual binary exponentiation, $9^{13} \equiv 14 \pmod{55}$. Since this does not equal $H(m) = 9$, the signature is not Alice's.

(d) For instance, the messages $m = 1$ and $m = 26$ have the same hash value $H(m) = 1$.

Problem 3. Consider the compression function $\tilde{H}\colon \{4\,\text{bits}\} \to \{2\,\text{bits}\}$ defined by

x	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
$\tilde{H}(x)$	00	10	11	01	10	00	01	11	11	01	00	10	01	11	10	00

(a) Let $H(x)$ be the hash function obtained from $\tilde{H}$ using the Merkle–Damgård construction (using initial value $h_1 = 00$). Compute $H(111101)$.

Armin Straub
straub@southalabama.edu

(b) Find a collision with $H(111101)$.

Solution.

(a) Since $\tilde{H}$ compresses by 2 bits, we need to chop $x = 111101$ into blocks x_i of 2 bits: $x_1 = 11$, $x_2 = 11$, $x_3 = 01$.

$h_1 = 00$

$h_2 = \tilde{H}(h_1, x_1) = \tilde{H}(0011) = 01$

$h_3 = \tilde{H}(h_2, x_2) = \tilde{H}(0111) = 11$

$h_4 = \tilde{H}(h_3, x_3) = \tilde{H}(1101) = 11$

Hence, $H(111101) = h_4 = 11$.

(b) Our computation above shows that, for instance, $H(1111) = 11$ as well. Other collisions include $H(10) = 11$.

Problem 4.

(a) Does Alice have to choose a new y if she sends several messags to Bob using ElGamal? Explain.

(b) The movie "Swordfish" features a DoD system using 128 bit RSA, which is broken by one of the actors. What is your reaction to that?

(c) Can encryption and/or decryption of RSA be sped up by the Chinese Remainder Theorem?

(d) Give two examples of side-channels that can be exploited in a side-channel attack.

(e) What is a NOBUS backdoor?

(f) Let p be a large prime. State the discrete logarithm problem, the computational Diffie-Hellman problem as well as the decisional Diffie-Hellman problem, all modulo p. Rank these three problems by their difficulty.

(g) Which primes p are called safe? What is the implication of using a safe prime for ElGamal?

(h) Due to using a poor PRG, the same prime p is used for two RSA public keys (N_1, e_1) and (N_2, e_2). Explain how to break both keys.

Solution.

(a) Yes, she absolutely has to randomly choose a new y every time! Here's why:

If she was using the same y to encrypt messages $m^{(1)}$ and $m^{(2)}$, Alice would be sending the ciphertexts $\left(c_1^{(1)}, c_2^{(1)}\right) = (g^y, g^{xy} m^{(1)})$ and $\left(c_1^{(2)}, c_2^{(2)}\right) = (g^y, g^{xy} m^{(2)})$.

That means, Eve can immediately figure out $c_2^{(1)} / c_2^{(2)} = m^{(1)} / m^{(2)}$ (the divison is a modular inverse and everything is modulo p). That's a combination of the plaintexts, and Eve should never be able to get her hands on such a thing.

(Also, Eve would know right away that Alice is doing the mistake of reusing y because $c_1^{(1)} = c_1^{(2)}$.)

Comment. The situation is just like for the one-time pad (in that case, reusing the key reveals $m^{(1)} \oplus m^{(2)}$).

(b) Looks like someone confused AES and RSA. While AES exists in a 128 bit version, key sizes for RSA are at least 1024 bit. 128 bit RSA would not provide any security.

https://blog.cryptographyengineering.com/2012/01/30/bad-movie-cryptography-of-week/

(c) Decryption can be sped up using the CRT, but encryption cannot. That's because using the CRT requires knowledge of the factorization of $N = pq$. (See Example 162(a).)

(d) Typical examples of side-channels include timing or power consumption.

(e) A NOBUS backdoor ("nobody but us") is a backdoor into a cryptosystem which can only be used by the person who knows a secret (which is infeasible to obtain by anyone else even if they know about the backdoor).

The term is also used to describe vulnerabilities in a cryptosystem which only a powerful agency (like the NSA) has the capabilities to exploit (in which case the agency might be happy to keep these vulnerabilities alive).

https://en.wikipedia.org/wiki/NOBUS

(f) The DL problem is the following: given $g, g^x \pmod p$, find x.

The CDH problem is the following: given $g, g^x, g^y \pmod p$, find $g^{xy} \pmod p$.

The DDH problem is the following: given $g, g^x, g^y, r \pmod p$, decide whether $r \equiv g^{xy} \pmod p$.

DL is harder than CDH, which is harder than DDH.

(g) A prime p is called safe if $(p-1)/2$ is a prime as well.

In general, checking whether a residue g is a primitive root (or, at least, has large order), so that we can use it for ElGamal, is difficult (because it typically involves factoring $p-1$).

On the other hand, if p is a safe prime, then all residues $g \not\equiv 0, \pm 1 \pmod p$ have (large!) order $(p-1)/2$ or $p-1$. In other words, all these residues are primitive roots (order $p-1$) or squares of primitives roots (order $(p-1)/2$).

(h) Since $\gcd(N_1, N_2) = p$ (except in the exceedingly unlikely case that, by accident, $N_1 = N_2$), we can factor both N_1 and N_2, and determine the secret key d_1 (and, likewise, d_2) by computing $d_1 \equiv e_1^{-1} \pmod{\phi(N_1)}$ where $\phi(N_1) = (p-1)\left(\frac{N_1}{p} - 1\right)$.

Problem 5.

(a) A hash function $h(x)$ is called one-way if [ ].

(b) A hash function $h(x)$ is called (strongly) collision-resistant if [ ].

(c) Does using a hash function provide authenticity?

(d) What's the difference between a compression function and a hash function? Which construction allows us to create the latter from the former?

(e) Is SHA-2 considered a secure password hashing algorithm?

(f) What does it mean to salt a password?

(g) In which sense are MD5 and SHA-1 broken? For which purposes must they not be used anymore? For which purposes is it still acceptable to use these hash functions?

(h) Explain why using a hash with 128 output bits is not appropriate for digital signatures.

(i) List the main ideas for storing human passwords for authentication.

(j) You need to hash (salted) passwords for storage. Unfortunately, you only have SHA-2 available. What can you do?

(k) We have learned about the birthday paradox. What is its implication for hash functions?

(l) Let H be a cryptographic hash function. What is a simple way to construct a MAC from H?

(m) Both digital signatures and MACs provide authenticity. What aspect of authenticity do digital signatures provide that MACs don't?

Solution.

(a) $h(x)$ is called one-way if, given y, it is computationally infeasible to compute m such that $h(m) = y$. Such a function is also called preimage-resistant.

(b) $h(x)$ is called (strongly) collision-resistant if it is computationally infeasible to find two messages m_1, m_2 such that $h(m_1) = h(m_2)$.

(c) No, everybody can use the same hash function. To provide authenticity, a digital signature or a MAC can be used.

(d) A hash function H is a function, which takes an input x of arbitrary length, and produces an output $H(x)$ of fixed length, say, b bit. On the other hand, a compression function $\tilde{H}$ takes input x of length $b + c$ bits, and produces output $\tilde{H}(x)$ of length b bits.

One popular construction to create hash functions from compression functions is the Merkle–Damgård construction.

(e) No, it is too fast.

(f) It means that we don't compute the hash $H(m)$ of a password m but instead $H(s, m)$ where s is some random data, called the salt. We must then pass on both s and $H(s, m)$.

(g) MD5 and SHA-1 have been demonstrated to not be collision-resistant.

As a consequence, they must not be used for applications like digital signatures, which rely on collision-resistance.

However, MD5 and SHA-1 are still considered one-way. Hence, it is acceptable (and still widespread practice) to use these hash functions for file integrity checking or, when iterated sufficiently to slow them down, for password storage.

(h) Digital signatures require a collision-resistant hash function. Using a birthday attack, for a hash with 128 output bits, a collision can be found by computing about 2^{64} hashes. That's not easy but doable with some effort (keep in mind that DES was brute-forced as early as 1997; that required computing about 2^{56} cases; 2^{64} is only $2^8 = 256$ times as large).

[On the other hand, brute-force is currently infeasible for 2^{128} cases, which is why AES-128 is considered secure despite its 128 bit key size.]

(i) Instead of passwords m, the hashes $H(s, m)$ should be stored together with s, a unique (typically random) value called "salt". Moreover, it is important to use a (slow!) hash function H designed for password storage. The usual hash functions like SHA-2 are too fast (thus making brute-force attacks practical).

(j) Iterate many times! (In order to slow down the computation of the hash.) The naive way would be to simply set $h_0 = H(m)$ and $h_{n+1} = H(h_n)$. Then use as hash the value h_N for large N.

In current applications, it is typical to choose N on the order of $100,000$ or higher (depending on how long is reasonable to have your user wait each time she logs in and needs her password hashed for verification).

(k) For collision-resistance, the output size of a hash function needs to have twice the number of bits that would be necessary to prevent a brute-force attack.

(l) We can simply produce a MAC $M_k(x)$ (usually referred to as a HMAC) as follows:

$$M_k(x) = H(k, x)$$

Comment. This seems to work fine for instance for SHA-3. On the other hand, this does not appear sufficiently secure for certain other common hashes. Instead, it is common to use $M_k(x) = H(k, H(k, x))$ (as well as certain padding).

https://en.wikipedia.org/wiki/Hash-based_message_authentication_code

(m) A MAC does not offer non-repudiation because several parties know the private key. Hence, it cannot be proven to a third party who among those computed the MAC (and, in any case, such a discussion would make it necessary to reveal the private key, which is usually unacceptable).