Université Libre de Bruxelles
Faculté des Sciences

2022-2023

INFO-F-405: Introduction to cryptography

3. Hashing

Definitions and requirements

Exercise 1

For a given hash function, does second preimage resistance imply collision resistance, or vice-versa, and why?

Answer of exercise 1

Collision resistance implies second preimage resistance (and not vice-versa).

Let us consider an adversary who is able to find a second preimage of $y = \text{hash}(x)$, i.e., he is able to find $x' \neq x$ such that $\text{hash}(x') = y = \text{hash}(x)$. By definition of the second preimage, he is given the value x. This means he is also able to produce a collision since $x' \neq x$ and $\text{hash}(x') = \text{hash}(x)$. So, an second preimage attack implies a collision attack.

By contraposition, assume that a hash function is collision-resistant, i.e., it is infeasible to find a collision. Then, if an adversary could find a second preimage, he would also be able find a collision, which was assumed to be infeasible. Hence, it is also infeasible to find a second preimage. Therefore, collision resistance implies second preimage resistance.

If a hash function is second preimage resistance, i.e., it is infeasible to find a second preimage, a collision could nevertheless be found by other means than via a second preimage attack. Intuitively, a collision attack does not require a specific image y, so a collision attack in general does not help finding a preimage for the given image y. Therefore, collision resistance and second preimage resistance are not equivalent.

Exercise 2

Choose a standard cryptographic hash function (or XOF) arbitrarily. Write a small program that computes the digest of a given string truncated to n bytes. For a given

value n, compute the digest of arbitrary (but distinct) input strings, and look for a collision at the output.

1. Start with $n = 1$, so $\ell = 8$ bits of output. After how many iterations t do you get a collision?

2. Increase n until this starts to take too much time. Do you recognize the law that gives you $t(n)$?

Answer of exercise 2

You should be able to recognize the birthday paradox with $t \approx 2^{\ell/2} = 2^{4n}$.

Exercise 3

Use the previously written piece of code and do the following test. Choose an arbitrary message m and compute $H = \text{hash}(m)$. Then, flip the first bit of m in order to obtain m'. Can you predict the value of $H' = \text{hash}(m')$ from that of H without explicitly computing it?

Repeat the experiment with different values of m and/or flipping bits at different positions. Can you see a predictable relationship between the corresponding H and H'? What is the average number of differences (i.e., the Hamming distance) between H and H'?

Answer of exercise 3

No, H and H' should have no apparent relationship, even if m and m' differ only in one bit position.

The average of the Hamming distance between H and H' should be about $\ell/2$.

Exercise 4

A friend of yours designed his/her own hash function CATSHA320, with 320 bits of output. It is an iterated hash function that cuts the input message into blocks of 192 bits, processes them sequentially and has a chaining value of 224 bits. Without knowing more about this hash function, what is its expected collision resistance (in number of attempts), and why?

Same question for CATSHA320++ with the same specifications but a block size of 384 bits and a chaining value of 480 bits.

Answer of exercise 4

The adversary can either find an output collision or an internal collision.

In the former case, considering CATSHA320 as a black box, we can expect to find a collision after about $\sqrt{2^{320}} = 2^{160}$ attempts, following the birthday paradox. In the latter case, the chaining value after processing some input blocks can itself be viewed as a black box. Since it has 224 bits, we can expect to find a collision after about $\sqrt{2^{224}} = 2^{112}$ attempts, following the birthday paradox. Hence, the weakest link is the internal collision, and CATSHA320 cannot have more than 112 bits of collision resistance, i.e., after 2^{112} attempts.

For CATSHA320++ with a chaining value of 480 bits, the weakest link is now the output collision, and CATSHA320++ cannot have more than 160 bits of collision resistance, i.e., after 2^{160} attempts.

Note that the block size does not play a role in this exercise. (The only role the block size could play is in the number of blocks required for each attempt to have enough degrees of freedom. In CATSHA320, the 192 bits of a single block are enough to generate 2^{112} different messages for the 2^{112} attempts necessary to get the internal collision. Hence the adversary can generate attempts using 1-block messages. If the block size was, for instance, 16 bits, then at least 7 blocks of messages are necessary to be able to generate 2^{112} different messages.)

Zooming from Merkle-Damgård into SHA-1

Exercise 5

[1] The core of some hash functions is the compression function. Let assume that we build a compression function on top of the block cipher

$$\text{output block} = \text{AES}_{\text{key}}(\text{input block})$$

in one of the following ways:

- $f_1(x, y) = \text{AES}_x(y) \oplus x$
- $f_2(x, y) = \text{AES}_x(x) \oplus y$

We ask you to find collisions for f_1 and for f_2. In other words you should find values a_1, b_1 and a_2, b_2 such that $f_1(a_1, b_1) = f_1(a_2, b_2)$ and also find values c_1, d_1 and c_2, d_2 such that $f_2(c_1, d_1) = f_2(c_2, d_2)$.

Answer of exercise 5

[1] This exercise was inspired by an exercise from the Security course given by Dan Boney.

- For f_1: find x, y, x', y' such that $\text{AES}_x(y) \oplus x = \text{AES}_{x'}(y') \oplus x'$. Pick x, y, x' at random and set $y' = \text{AES}_{x'}^{-1}(\text{AES}_x(y) \oplus x \oplus x')$. We can invert AES because we know x'.
- For f_2: find x, y, x', y' such that $\text{AES}_x(x) \oplus y = \text{AES}_{x'}(x') \oplus y'$. Pick x, x' at random and set $y = \text{AES}_{x'}(x')$ and $y' = \text{AES}_x(x)$

Exercise 6

Many well-known hash functions, such as MD5, SHA-1 and the SHA-2 family, use a block cipher in the Davies-Meyer construction. However, these are all specific, dedicated, block ciphers. What about the AES? So let us build SHAES by first building a compression function from the AES-256 in the Davies-Meyer construction, then putting the compression function in the Merkle-Damgård construction.

1. What is the block size of SHAES?
2. What is the chaining value size of SHAES?
3. Is SHAES a valid solution in practice?

Answer of exercise 6

1. As it uses Davies-Meyer, the block size of SHAES is equal to the key size of AES-256, that is, 256 bits.
2. As it uses Davies-Meyer, the chaining value size of SHAES is equal to the block size of AES-256, that is, 128 bits.
3. With only 128 bits of chaining value, SHAES is vulnerable to internal collisions with a complexity of only 2^{64} attempts. This is weak in today's standards.

Exercise 7

Some designs, like the hash function SHA-1 and the SHA-2 family, use a combination of modular addition (in $\mathbb{Z}_{2^n}$) and bitwise addition or XOR (in $\mathbb{Z}_2^n$), plus rotations (or circular shifts). In the litterature, this combination is often referred to "ARX" for addition-rotation-XOR. The idea is that the modular addition is non-linear from the point of view of $\mathbb{Z}_2^n$, and vice-versa, the XOR is non-linear in $\mathbb{Z}_{2^n}$.

Formally, an element $x = (x_0, x_1, \ldots, x_{n-1}) \in \mathbb{Z}_2^n$ is mapped to the integer $X = \sum_i x_i 2^i$ in $\mathbb{Z}_{2^n}$. Let us define $X = \mathsf{integer}(x)$ as the function that converts an element of $\mathbb{Z}_2^n$ to an element of $\mathbb{Z}_{2^n}$, and let $x = \mathsf{vector}(X)$ be its inverse.

1. Show that $f(x, y) = \mathsf{vector}(\mathsf{integer}(x) + \mathsf{integer}(y))$, i.e., the modular addition of x and y interpreted as integers, is non-linear in $\mathbb{Z}_2^n$. *Hint:* It is sufficient to give a counter-example. Also, you may set $n = 2$ for simplicity.

2. Show that $F(X, Y) = \mathsf{integer}(\mathsf{vector}(X) \oplus \mathsf{vector}(Y))$, i.e., the XOR of X and Y interpreted as n-bit vectors, is non-linear in $\mathbb{Z}_{2^n}$.

Answer of exercise 7

1. Let $n = 2$. Note that $f(0, 0) = 0$, so we have to show that $f(x \oplus x', y \oplus y') \neq f(x, y) \oplus f(x', y')$ for some counterexample. Take $(x, y) = (1, 0)$ and $(x', y') = (0, 1)$. Then,

$$f(1, 1) = 2 \quad \neq \quad f(1, 0) \oplus f(0, 1) = 1 \oplus 1 = 0 \ .$$

2. Similarly, we have to show that $f(X + X', Y + Y') \neq F(X, Y) + F(X', Y')$ for some counterexample. Take $(X, Y) = (1, 0)$ and $(X', Y') = (0, 1)$. Then,

$$F(1, 1) = 0 \quad \neq \quad F(1, 0) + F(0, 1) = 1 + 1 = 2 \ .$$

Modern generic security

Exercise 8

Let $\mathcal{RO}(x)$ be a random oracle that outputs an infinite sequence of uniformly and independently distributed bits for each input $x \in \{0, 1\}^*$. If the random oracle is queried twice with the same input, it always returns the same sequence. We denote with $\mathcal{RO}(x, \ell)$ the output of $\mathcal{RO}(x)$ truncated after the first ℓ output bits, so $\mathcal{RO}(x, \ell) \in \{0, 1\}^\ell$.

1. *Preimage attack.* Given some value $y \in \{0, 1\}^\ell$, an adversary would like to find a preimage, i.e., an input x such that $\mathcal{RO}(x, \ell) = y$. What is the probability of success after t attempts? (We can approximate assuming that $t \ll 2^\ell$.)

2. *Multi-target preimage attack.* Given a set of m distinct values $\{y_1, \ldots, y_m\}$, with $y_i \in \{0, 1\}^\ell$, an adversary would be happy to find a preimage of any one of these images. What is the probability of success after t attempts?

Let $h(x)$ be a concrete extendable output function (XOF) that outputs a potentially infinite sequence of bits, and we similarly denote $h(x, \ell)$ its truncation to ℓ output bits. As of today, the only known attack against $h(x)$ has a probability of success $t/2^{c/2}$, for some security parameter c, and where t is the time spent by the adversary expressed in the number of attempts. This means that, except for this probability $\epsilon(t)$, we can model $h(x)$ as a random oracle.

3. What is the range of output sizes ℓ and parameter values c such that $h(x)$ offers 128 bits of preimage resistance?

4. What is the range of output sizes ℓ and parameter values c such that $h(x)$ offers 128 bits of multi-target preimage resistance with $m = 2^{32}$?

You may use the approximation $\log(a + b) \approx \max(\log a, \log b)$.

Answer of exercise 8

1. The probability that an adversary finds x such that $\mathcal{RO}(x, \ell) = y$ after t attempts is approximately $t2^{-\ell}$.

   After one attempt, the probability that $\mathcal{RO}(x, \ell) = y$ is $2^{-\ell}$, as the range of $\mathcal{RO}(x, \ell)$ has a size of 2^{ℓ}. It is tempting to say that after t attempts, the probability will be $t2^{-\ell}$, but this would be rigorously correct in the case of drawing without replacement. However, this remains approximately true when $t \ll 2^{\ell}$. To see this, let us consider the probability that the correct output does *not* appear after t attempts: $(1 - 2^{-\ell})^t$. Using binomial expansion, we see that $(1 - 2^{-\ell})^t = 1 - t2^{-\ell} + o(t2^{-\ell})$.

2. Similarly, the probability is approximately $mt2^{-\ell}$ since each attempt has a m chances out of 2^{ℓ} to hit one of the y_i's.

3. We are not given any information about the known attack against h, so we have to consider the worst case that this attack helps find the preimage. After t attempts the adversary can find the preimage $h(x) = y$ either by chance (i.e., $t2^{-\ell}$) or because it results from the attack on h (i.e., $t2^{-c/2}$). The probability of the union of two events is upper bounded by the sum of the probability of the two events, i.e., $\Pr[A \text{ or } B] \leq \Pr[A] + \Pr[B]$, so we have to consider as success probability $\epsilon(t) = t2^{-\ell} + t2^{-c/2}$. Working this out:

$$\begin{aligned} & t2^{-\ell} + t2^{-c/2} \\ =& t(2^{-\ell} + 2^{-c/2}) \\ \approx& t2^{\max(-\ell, -c/2)} \\ =& t2^{-\min(\ell, c/2)}. \end{aligned}$$

In order to have 128 bits of security, we need to ensure that $t2^{-\min(\ell,c/2)}$ does not reach 1 before $t = 2^{128}$. Therefore, we must have $\ell \geq 128$ and $c/2 \geq 128$ or, equivalently, $c \geq 256$.

4. The reasoning is similar, but this time with $\epsilon(t) = mt2^{-\ell} + t2^{-c/2}$. Working this out:

$$\begin{aligned} & mt2^{-\ell} + t2^{-c/2} \\ =& t(2^{-\ell+\log_2 m} + 2^{-c/2}) \\ \approx& t2^{\max(-\ell+\log_2 m, -c/2)} \\ =& t2^{-\min(\ell-\log_2 m, c/2)}. \end{aligned}$$

In order to have 128 bits of security, we must have $\ell - \log_2 m \geq 128 \Leftrightarrow \ell \geq 128 + 32 = 160$ and $c/2 \geq 128 \Leftrightarrow c \geq 256$.

Inside SHA-3

Exercise 9

Let π be a cryptographically secure permutation that maps $b = 1600$-bit strings onto b-bit strings. We assume that π and its inverse π^{-1} are both equally efficiently implementable. We build a hash function by applying the sponge construction on top of π. The hash function has a fixed output length of $n = 256$ bits. However, for the sake of this question, we set the capacity to $c = 0$.

1. Please explain how to obtain a collision with this hash function.
2. Given two prefixes x and y of b bits each, show how to obtain a collision between one input that must start with x and the other one with y.
3. Given an output value z of n bits, describe how to find a preimage with this hash function.
4. Now let us assume that this hash function is used exclusively in an application that hashes input strings made of ASCII-encoded text, for which each byte has its most significant bit always set to 0. More precisely, we restrict the input strings m such that every 8th bit is 0, and for the sake of simplicity this is the only restriction we consider (i.e., we do not care whether m is actually valid ASCII-encoded text or not). Please explain how to obtain a collision with this hash function such that the colliding inputs have this restriction. What is the complexity of your attack?

5. Given an output value z of n bits, describe how to find a preimage with this hash function such that the preimage has the aforementioned restriction. What is the complexity of your attack?

Answer of exercise 9

1. When $c = 0$, the sponge construction claims no security, but how can we obtain a collision? The simplest way consists in obtaining a state collision, i.e., two different sequences of input blocks that lead to the same value of the state. Because there is no inner part ($c = 0$), we can control the entire state value by appropriately choosing the input blocks as they are XORed in, and we can easily set the state to an arbitrarily chosen value.

   With this in mind, there are numerous ways to make a collision. The simplest consists in resetting the state to its initial value 0^b. Let $\mathsf{tozero} = \pi^{-1}(0^b)$. Then, after absorbing tozero the state is back to 0^b. Hence, for any arbitrary string s, we have $h(\mathsf{tozero}|s) = h(s)$.

2. After absorbing as first block x (resp. y), the state is $\pi(x)$ (resp. $\pi(y)$). We need to find one block x' to put after x and one block y' to put after y so that the same state is reached after they are XORed in. In other words,

   $$\pi(x) \oplus x' = \pi(y) \oplus y' \ .$$

   Any choice such that $x' \oplus y' = \pi(x) \oplus \pi(y)$ is suitable, for instance $(x', y') = (\pi(x), \pi(y))$ or $(x', y') = (\pi(y), \pi(x))$. Then, we have $h(x|x') = h(y|y')$.

3. We extend z by appending $1600 - 256$ arbitrary bits z' and consider this as the last state of the sponge function, $s_1 = z|z'$. We then compute $s_0 = \pi^{-1}(s_1)$, the state after padding and absorbing the first block. We unpad s_0 by removing the last bits 0 until a 1 is found and removed. This works only if $s_0 \neq 0^b$; in the unlikely case where $s_0 = 0^b$, we just restart with another arbitrary value z'.

4. When the input is restricted and every 8th bit must be set to zero, the adversary cannot directly influence the value of the state in those bits, exactly as for the inner part. There are $b/8 = 200$ bits that are restricted, and hence the behavior is identical to a sponge function with an inner part of $c = 200$ bits, except for the location of this inner part. In the sequel, let us call the *virtual inner part* every 8th bit of the state that coincides with the input bits that are stuck to 0.

   To get a collision in the state, we start with finding two input blocks x and y (satisfying the restriction) such that, after being absorbed, the value of the

virtual inner part is identical. With these, the state is $\pi(x)$ and $\pi(y)$, respectively. Then we proceed as in the sub-question above, i.e., we find two next input blocks x' and y' that cancel the difference between $\pi(x)$ and $\pi(y)$. Because the virtual inner part is identical, x' and y' can satisfy the restriction. Then, we have $h(x|x') = h(y|y')$.

The time-consuming part is to find the collision in the virtual inner part. Because of the birthday paradox, this takes about $\sqrt{2^{200}} = 2^{100}$ attempts.

5. Here we can proceed as in sub-question 3, except that the state $s_0 = \pi^{-1}(z|z')$ must have the virtual inner part equal to zero for the input to satisfy the restriction. A simple attack consists in trying different values z' until this happens. For every attempt, the probability is about 2^{-200}, so the complexity is about 2^{200}.

   (For completeness #1, the unpadding must also succeed. This typically requires that the last byte of s_0 contains $(10000000)_2$, which adds 7 more restrictions, for a total complexity of 2^{207}.)

   (For completeness #2, there exist more advanced preimage attacks on sponge functions that exploit the birthday paradox, and they can be transposed to this case. Using these, the complexity therefore drops to around 2^{100}.)

Exercise 10

Consider FIPS 202 and extendable output functions.

1. What is (approximately) the performance ratio between SHAKE128 and SHAKE256? And between SHAKE128 and SHA3-512?

2. For SHAKE128, what is the output size you need to choose to have 128-bit security for collision, preimage and second preimage resistance?

3. Same question, but only for preimage and second preimage resistance (no collision resistance required)?

4. Let us fix an ouput size ℓ. Given a set of m distinct values $\{y_1, \ldots, y_m\}$, with $y_i \in \{0,1\}^\ell$, an adversary would be happy to find a preimage of any one of these images. What is (approximately) the probability of success after t random attempts against a random oracle?

5. Say that $m = 2^{40}$. Can we use SHAKE128 to resist against multi-target attacks with a security level of 128 bits? If so, what would be the required output size ℓ?

Answer of exercise 10

1. We need to compare the corresponding rates. SHAKE128 is based on Keccak with $c = 256$, SHAKE256 on Keccak with $c = 512$, and SHA3-512 has $c = 1024$[2]; and the rate r is given by the formula $r = b - c$ with b=1600, the usual value for Keccak.
   As a result, for SHAKE128, $r = 1600 - 256 = 1344$, for SHAKE256, $r = 1600 - 512 = 1088$ and for SHA3-512, $r = 1600 - 576$.
   We can now compute the rates. SHAKE128 is $\frac{1344}{1088}$ times faster than SHAKE256 and $\frac{1344}{576}$ times faster than SHA3-512.

2. We know that preimage resistance is $2^{\min(n,c/2)}$, second-preimage resistance is $2^{\min(n,c/2)}$ and collision resistance is $2^{\min(n/2,c/2)}$ for the sponge construction[3].
   We can observe that the collision resistance is the critical one among the three $2^{\min(n/2,c/2)}$ bits of security. Because of the birthday paradox, we need 256 bits of output to reach 128 bits of collision resistance.

3. Here 128 bits of output are sufficient.

4. One attempt has a probability of $m2^{-\ell}$ of hitting one of the m given images. After $t \ll \frac{2^\ell}{m}$ attempts, the probability is approximately t times greater, hence $mt2^{-\ell}$. So an adversary would need about $t \approx \frac{2^\ell}{m} = 2^{\ell - \log_2 m}$ attempts to find one of the preimages. Hence, for the same output length, finding a multi-target preimage is about $\log_2 m$ bits easier than a regular (single-target) preimage attack.

5. Yes. SHAKE128 should resist against any attack up to complexity 2^{128} unless easier on a random oracle. So if we can find a solution that achieves 128 bits of security for the random oracle, the same applies to SHAKE128. (If the question was on more than 128 bits of security, the answer would have been no.) So, if we output at least $\ell = 128 + \log_2 m = 168$ bits, SHAKE128 can still achieve 128 bits of security against this attack.

Others

[2] See slide set 3-Hashing, page 43.
[3] From slide 31 of the slide set "3-Hashing"

Exercise 11

How (not?) to build message authentication codes from hash functions.

Let K be a k-bit secret key (e.g., $k = 128$ bits).

1. Let us assume that we build a MAC function as follows:

   $$\mathrm{MAC}_K(\text{message}) = \mathrm{hash}(\text{message}) \oplus K,$$

   where $\mathrm{hash}(\cdot)$ is a secure k-bit hash function and $\oplus$ denotes the bitwise mod-2 addition (or XOR). Is this MAC function secure? If so, please justify briefly. If not, please describe an attack in the light of the EU-CMA game.

2. Let us consider another MAC function. We now assume that the MAC function is constructed as follows:

   $$\mathrm{MAC}_K(\text{message}) = \mathrm{hash}(K_1 \| \text{message}) \oplus K,$$

   where $\mathrm{hash}(\cdot)$ is a secure k-bit hash function, $\|$ denotes the concatenation and $K = K_1 \| K_2$ with $|K_1| = |K_2| = k/2$. What is the security strength of this MAC function, and why?

Answer of exercise 11

1. No, it is not secure. From a single known (message, tag) pair, the attacker can recover the key K by computing $K = \mathrm{hash}(\text{message}) \oplus \text{tag}$.

   In the language of the EU-CMA game, the adversary can produce a forgery with probability 1 by doing the following:

   - The adversary queries $\text{tag} = \mathrm{MAC}_K(\text{message})$ with an arbitrary message. From this, K is recovered as explained above.
   - Knowing the key, the adversary computes $\text{tag}' = \mathrm{MAC}_K(\text{message}')$ with a different message.
   - The adversary wins as $(\text{message}', \text{tag}')$ is a valid pair that was not queried before.

2. The security strength is $k/2$ bits. The attack works similarly, except that we need to exhaustively search for the correct value of K_1, which means a time complexity of $2^{k/2}$. In more details, the attack works as follows.

   - Get a known (message, tag) pair by querying $\text{tag} = \mathrm{MAC}_K(\text{message})$ with an arbitrary message.

- For each candidate value K_1^* for the first $k/2$ bits of K:
  - Compute $K^* = \text{hash}(K_1^* \| \text{message}) \oplus \text{tag}$.
  - If the first $k/2$ bits of K^* do not match K_1^*, this cannot be a correct guess for the candidate K_1^*, so we continue. Otherwise, if they match, K^* might be the correct key; we double-check with another (message, tag) pair; if correct, we found the secret key K.