Network Security

Professor Gaurav S. Kasbekar
Department of Electrical Engineering
Indian Institute of Technology, Bombay

Week - 03

Lecture - 17

Message Integrity, Cryptographic Hash Functions and Digital Signatures: Part 2

Hello, in this lecture, we continue our discussion of message integrity, cryptography, hash functions, and digital signatures. In the previous lecture, we discussed that cryptographic hash functions can be used to achieve message integrity. Let's now discuss some properties of cryptographic hash functions. So, for analyzing a cryptographic hash function, it's useful to have a simple model for it. So, a simple model is as follows.

A cryptographic hash function should appear to be a completely random mapping from the input to the output. So, what do you mean by a completely random mapping? So, for a cryptographic hash function that produces an l-bit output, for example, $l = 128$, a completely random mapping from the input to the output means the following. So, this mapping is similar to a symmetric key encryption algorithm, but there are some differences. For a cryptographic hash function that produces an l-bit output, a completely random mapping means the following.

For a given input, the output is selected from among the 2^l possible outputs uniformly at random. For a different input, the output is independently selected uniformly at random from the 2^l possible outputs, and so on and so forth. So, it should appear that for an arbitrary input, the output is a value chosen uniformly at random from the 2^l possible output values. And for every input, the output value should appear to be independent of the output value for a different input value. So, what is the reason for requiring the mapping to appear completely random?

If the input were to map in a predictable way to the output, then it may be possible to efficiently find an input that results in a pre-specified hash value or to find two different inputs that result in the same hash value. So, these are the disadvantages of having mapping in a predictable fashion from the input to the output. So, for this reason, we require the mapping to appear completely random. Hence, a simple model for a cryptographic hash function with a l-bit output is to assume that for any given input, the output is selected

uniformly at random from the 2^l possible outputs. So, we will use this simple model to gain further understanding into a cryptographic hash function.

The question is, what should the number of output bits l be? For example, should $l = 64$, or should it be 128, 256, 512, and so on? Recall that for achieving message integrity, it should be computationally infeasible to find the message m, that is, the pre-specified hash value h. It should be computationally infeasible to find the message m such that H(m) equals the target h. Suppose an intruder is trying to break this cryptographic hash function and is trying to find a message m that has this hash value h. Suppose the intruder finds the hash values of different random inputs until he or she finds an input that has a hash value h, which is the target hash value. How many inputs would the intruder have to try to find an input with a hash value h with high probability?

- it should be computationally infeasible to find a message m that has a pre-specified hash value h, i.e., m such that $H(m) = h$

Suppose the intruder tries n different inputs; then the expected number of inputs whose hash value equals h is this. The claim is that the expected number of inputs whose hash value equals h is $\frac{n}{2^l}$.

So, let's justify this claim. Define the indicator random variable X_i to be 1 if the i^{th} input has hash value h and 0 otherwise. So, this random variable has only two possible values, 1 and 0. X_i is 1 if the i^{th} input has hash value h, and it is 0 otherwise. Such a random variable is called an indicator random variable.

$$X_i = \begin{cases} 1, & if\ i^{th}\ input\ has\ hash\ value\ h \\ 0, & else \end{cases}$$

- If intruder tries n different inputs, expected number of inputs whose hash value equals h is:
  - $\frac{n}{2^l}$
  - **Proof**:
    - Let $X_i = \begin{cases} 1, & \text{if } i\text{'th input has hash value } h, \\ 0, & \text{else.} \end{cases}$
    - Let $Y = X_1 + \cdots + X_n$
    - Then $E(Y) = nE(X_1) = \frac{n}{2^l}$
- So number of inputs that must be tried, on average, until success:
  - $n = 2^l$
- Hence, larger the value of l, more difficult it is for intruder
- E.g.:
  - if $l = 64$, then intruder must try $2^{64} \approx 1.8 \times 10^{19}$ inputs, which is computationally infeasible
  - if $l = 32$, then intruder must try $2^{32} \approx 4.3 \times 10^9$ inputs, which is feasible

It indicates whether the i^{th} input has a hash value h or not. Define the random variable $Y = X_1 + X_2 + \cdots + X_n$. By the definition of X_i, we see that this random variable Y equals the number of inputs that have the hash value h. So, by the definition of X_i, it follows that Y is the number of inputs that have the hash value h. Now, let's take expectation on both sides of this equation. Then, we get $E(Y) = E(X_1) + E(X_2) + \cdots + E(X_n)$, but $E(X_1)$, $E(X_2)$, …, $E(X_n)$ are equal. They are all equal to the $E(X_1)$ because these random variables all have the same distribution.

Each random variable is 1 with this probability $1/2^l$, and the random variable is 0 with probability $1 - (1/2^l)$. So, all these random variables X_1 to X_n are identically distributed. So, hence, their expectations are equal. So, hence, we get $E(Y) = nE(X_1)$. But notice that $E(X_1)$ is just one times the probability that the i^{th} input has the hash value h, which is $1/2^l$ in our simple model. So, hence, we get that $E(Y) = n/2^l$. Hence, we have proved this property that the expected number of inputs whose hash value equals the target h is $n/2^l$.

So, how many inputs must we try on average until we get an input that has the hash value h? So, we must get one input whose hash value is the target h. So, let us set this $n/2^l$ to 1. Then, we get $n = 2^l$. This is the number of inputs that must be tried on average until we get an input that has the hash value h. So, this is, we see that n is 2^l. It is 2 to the power the number of output bits of the hash function. This shows that the larger the value of l, the more difficult it is for the intruder to find an input with a specified output value h. This is also consistent with intuition. If there are more possible outputs, then on average the intruder should try more inputs to find an input that has a pre-specified hash value h.

Here is a numerical example. If $l = 64$, then the intruder must try 2^{64} inputs, which turns out to be 1.8×10^{19} inputs. This is computationally infeasible. Even if the intruder uses a multi-gigahertz processor, it would take a long time to try these many inputs. It would take several years to try these many inputs, and hence it's computationally infeasible to find an input with a particular hash value.

If $l = 32$, then the intruder must try 2^{32} inputs on average. That comes out to be 4.3×10^9 inputs. So, this is quite feasible. If the intruder uses, say, a multi-gigahertz processor, then just in some time the intruder can find an input which has the pre-specified hash value h. This analysis suggests that a value of $l = 64$ is adequate for achieving message integrity.

So, for some applications, we require the following stronger property. This property is that it is computationally infeasible to find two messages m and m' such that H(m') = H(m).

We call that this is one of the properties of a cryptographic hash function, which we discussed earlier in the definition. So, later, we'll discuss an example of such an application. Now let's discuss how large should l be so that it is computationally infeasible to find two different messages, m and m', which have the same hash value.

- Let $p = P$(two different inputs m and m' have the same hash value, i.e., $H(m') = H(m)$)
- How large must n be such that $p \geq \frac{1}{2}$?
- **Exercise**: For n larger than $\approx 2^{l/2}$, $p \geq \frac{1}{2}$
- That is, *if we find the hash values of* $n \approx 2^{l/2}$ *different inputs, then with a high probability* $\left(p \geq \frac{1}{2}\right)$, *we will find two inputs with the same hash value*

Suppose we find the hash values of n inputs. The hash value of each input is selected uniformly at random from the 2^l possible outputs. Let p be the probability that two different inputs m and m' have the same hash value, that is, H(m') = H(m). So, how large must the number of inputs n be such that $p \geq \frac{1}{2}$?

So, this $1/2$ is a number which is a sufficiently high probability. So, if $p \geq \frac{1}{2}$, then with a

reasonably high probability we have found two different inputs which have the same hash value H(m') = H(m). A simple exercise is to show that for n larger than $\approx 2^{l/2}$, $p \geq \frac{1}{2}$.

So, by using some simple probabilistic analysis, you can show that for n larger than $\approx 2^{l/2}$, $p \geq \frac{1}{2}$.

So, the conclusion is that if we find the hash values of n, which is $\approx 2^{l/2}$ different inputs, then with a higher probability, in particular $p \geq 1/2$, we will find two different inputs with the same hash value. So, consider this property. It is computationally infeasible to find two messages m and m', such that H(m') = H(m). If an intruder finds out the hash values of n, which is roughly $2^{l/2}$ different inputs, then with a high probability of $p \geq 1/2$, the intruder will find two different inputs with the same hash value, and hence, this property will be broken. Here is an example.

If $l = 64$, then the intruder must try $2^{64/} = 2^{32}$, which comes out to be 4.3×10^9 inputs. So, as we discussed earlier, this is quite feasible. If $l = 128$, then the intruder must try $2^{128/2} = 2^{64}$, which comes out to be 1.8×10^{19} inputs, which is computationally infeasible. So, this analysis suggests that in applications where we require this property, that it is computationally infeasible to find two messages m and m' such that H(m') = H(m),

a value of $l = 64$ is not adequate. We require a larger value of l. So, for this reason, a value of $l = 128$ or larger is used in practice.

So, we discussed earlier the first property where the intruder tries different inputs until they get an input which has a pre-specified hash h. So, for that property, we require $l = 64$, but for this stronger property that it is computationally infeasible to find two messages, m and m', which have the same hash value. So, for achieving this property, we require $l = 128$ or larger. $l = 64$ is not adequate to achieve this property. So, recall that p is the probability that out of n inputs, two different inputs m and m' have the same hash value; that is, H(m') = H(m).

p can be found as in the solution to the birthday problem, which is a well-known problem from basic probability. The birthday problem is this. Out of a set of n randomly chosen people, what is the probability that two of them have the same birthday? So, in this case, assuming that there are 365 days in a year, ignoring leap years, for each person, the birthday is on one of the days, uniformly at random, chosen from the 365 possible days. So, in the context of the hash function, the situation is similar.

For each input, the output is one of the values from the 2^l possible output values. So, for the birthday problem, the situation is similar. Instead of 2^l possible output values for the hash function case, the birthday can be on one of the 365 possible days. So, this is a similar problem to this one with just 2^l replaced with 365. For this reason, an attack on a cryptographic hash function in which the intruder tries n inputs to find two different inputs with the same hash value is called the birthday attack.

This nomenclature is due to this well-known problem from probability. Here is an example of the birthday attack. Tom applies for a tenured faculty position. He requests his department chairperson, Marilyn, who thinks highly of his work, for a letter of recommendation. So, Marilyn outlines the contents of the letter to her secretary, Ellen, and asks her to compose the letter.

So, the scheme that is used is as follows. So, Ellen will compose the letter and send it to Marilyn for her approval. Once Marilyn approves the letter, Ellen will send the letter to the dean, and Marilyn will just compute and email the 64-bit hash value of the letter to the dean for verification. So, this verification will check whether or not the letter has been modified during transit. So, if the hash value received by the dean is the same as the hash value of the letter received by the dean, then the dean can be confident that it is the same letter which was approved by Marilyn.

This is the scheme used for integrity for the dean to check that the dean has received the correct message. However, Ellen has a grudge against Tom and wants to damage his application. So, Ellen wants to send a negative letter to the dean, but Ellen still wants to get the letter approved by Marilyn, but wants to send a negative letter to the dean. So, can Ellen do this? She launches the following attack.

She creates two sets of 2^{32} letters each. One set provides a positive recommendation, and the other set provides a negative recommendation. If she finds a letter providing a positive recommendation that has the same hash value as a letter providing a negative recommendation, then she sends the letter providing a positive recommendation to Marilyn for her approval but sends the letter with the negative recommendation to the dean. Since the two letters have the same hash value, the dean thinks that the message has not been modified. So, now the challenge is how do we create so many letters, all providing a positive recommendation, and how do we create 2^{32} letters, all providing a negative recommendation? So, a trick to create a set of 2^{32} positive letters is shown here.

So, we have here 32 brackets, each with two choices. So, for example, we might say, 'This letter is to give my honest opinion,' or we can say, 'This message is to give my honest opinion,' and so on and so forth. So, we can either say 'this letter' or 'this message'; they have the same meaning. Similarly, 'honest' and 'frank' have the same meaning, and so on and so forth. So, here, if we use 'a candidate' or we say 'up,' they have the same meaning, and so on.

So, there are 32 brackets in this way. There are 32 brackets which are shown here, and we can choose one of the words in each bracket. So, from the first bracket, we can choose a word in two ways; then, from the second bracket, we can choose one of the choices in two ways, and so on and so forth. So, there are 2^{32} possible choices. And regardless of which choice we make in each bracket, the meaning of the letter remains roughly the same.

So, these are all positive letters. They all say positive things about Tom. So, he is an outstanding or excellent researcher of great talent or ability, known worldwide for his brilliant insights into many challenging problems, and so on and so forth. So, these are all positive letters. Similarly, we create a set of 2^{32} negative letters.

This set of letters also has 32 brackets, each with two choices, and they all say negative things about the candidate. His research hardly ever shows insight or understanding of the key problems of the day, and so on and so forth. So, these 2^{32} letters all say negative things. Now, Ellen creates two sets of 2^{32} letters each, one set providing a positive

recommendation and the other set providing a negative recommendation. With a high probability, she finds one positive letter, which we'll call m, and one negative letter, m', with the same hash value, that is, H(m) = H(m').

Then, Ellen emails the positive letter, m, to Marilyn for her approval and the negative letter, m', to the dean. So, Marilyn computes the hash of m and sends it to the dean. So, H(m) is sent to the dean, and the dean receives the letter m', which is a negative letter. But since H(m) = H(m'), the verification at the dean succeeds. So, hence, Ellen succeeds in her attack.

She manages to send a negative letter to the dean but is able to get it approved by Marilyn. So, in this case, the integrity of this scheme is broken, and that's because the length of the output of the hash function is small; it's only 64 bits. So, if you were to choose a hash function with a larger output value, say for example 128 bits output, in that case, Allen would have to create two sets of 2^{64} letters each, and it would be computationally infeasible to try out, to find out the hash values of so many letters. So, this scheme fails because we use a hash function with whose output length value is small. Just a note that the probability that Ellen finds a positive letter and a negative letter with the same hash value is roughly one fourth.

It is not roughly half. The reason is that earlier, when we derived that the probability is half when $n \approx 2^{l/2}$, the probability is half. When we stated that when n is $2^{l/2}$, the probability is roughly half. In that situation, the intruder tries out n different inputs until the intruder gets two different inputs with the same hash value. But in this case, the intruder tries out, creates two sets of 2^{32} letters each, and must find a positive letter and a negative letter with the same hash value.

So, it's not enough to find two positive letters with the same hash value. It's not enough to find two negative letters with the same hash value. But the intruder must find one positive letter and one negative letter with the same hash value. For this reason, the probability is lower. It's roughly one fourth instead of roughly half.

But still, this is a sufficiently high probability. So, with a high probability, Ellen is able to find a positive letter and a negative letter with the same hash value, thereby breaking this scheme. Thank you.