Applied Cryptography

Symmetric Cryptography, Assignment 3, Monday, March 4, 2024

Exercises with answers and grading.

1. **(10 points)** On 23 February 2017, researchers from CWI Amsterdam and Google Researchers broke SHA-1 in practice. In this exercise, you will generate your own SHA-1 collision. There is no need to dive into the technical details of SHA-1, the only important thing to know is that SHA-1 is a Merkle-Damgård design (see slide 10). In more detail, SHA-1 operates on a state of 160 bits, and compresses message blocks of 512 bits at a time.

   The attackers of SHA-1 derived two different messages M and M' of size 1024 bits that, if preceded by a common prefix S of size 1536 bits, resulted in SHA-1($S\|M$) = SHA-1($S\|M'$). The messages S, M, M' are given at `https://www.cs.ru.nl/~bmennink/sha1attack/`, alongside a SHA-1 digest computation tool.

   (a) Compute SHA-1($S\|M$) and SHA-1($S\|M'$).

   (b) Consider the following message M'':

   ```
   8B AD F0 0D 8B AD F0 0D 8B AD F0 0D 8B AD F0 0D
   8B AD F0 0D 8B AD F0 0D 8B AD F0 0D 8B AD F0 0D
   8B AD F0 0D 8B AD F0 0D 8B AD F0 0D 8B AD F0 0D
   8B AD F0 0D 8B AD F0 0D 8B AD F0 0D 8B AD F0 0D
   ```

   Compute SHA-1($S\|M\|M''$) and SHA-1($S\|M'\|M''$).

   (c) Consider a 1024-bit message block M_a defined as:

   ```
   C0 FF EE C0 FF EE C0 FF EE C0 FF EE C0 FF EE C0
   FF EE C0 FF EE C0 FF EE C0 FF EE C0 FF EE C0 FF
   EE C0 FF EE C0 FF EE C0 FF EE C0 FF EE C0 FF EE
   C0 FF EE C0 FF EE C0 FF EE C0 FF EE C0 FF EE C0
   FF EE C0 FF EE C0 FF EE C0 FF EE C0 FF EE C0 FF
   EE C0 FF EE C0 FF EE C0 FF EE C0 FF EE C0 FF EE
   C0 FF EE C0 FF EE C0 FF EE C0 FF EE C0 FF EE C0
   FF EE C0 FF EE C0 FF EE C0 FF EE C0 FF EE C0 FF
   ```

   Find a prefix P for M_a and a second message M_b such that SHA-1($P\|M_a$) = SHA-1(M_b), where $P\|M_a$ and M_b are distinct and of equal size. Hint: use S, M, M'.

   (d) Compute the corresponding SHA-1 hash digest for your message of question 1c.

Begin Secret Info:. . . . .

(a) F9 2D 74 E3 87 45 87 AA F4 43 D1 DB 96 1D 4E 26 DD E1 3E 9C.

(b) 4A 01 05 56 63 7b 3F E7 00 1C B1 61 98 04 0D 31 EB 7A A1 D1.

(c) Take $P = S\|M$ and $M_b = S\|M'\|M_a$.

(d) If you put $P = S\|M$ or $P = S\|M'$: 60 11 17 14 1F C3 CB B6 DC DE A2 D4 4D B1 09 5F 1F C0 E1 19.

End Secret Info . . . .

2. **(10 points)** In this exercise you build your own sponge-based hash function, assuming you already have a permutation p on $b = 400$.

   Using the KECCAK-f permutation that you can find in `permutation.py`, build the full KECCAK hash function. Make sure you can control the capacity c and the rate r (with $c + r = b$), as you will use that feature in exercise 2. You can verify your implementation against the server using

   ```
   nc appliedcrypto.cs.ru.nl 4143
   ```

   (Note that this is only possible from within the Radboud network, or via a VPN connection.)

   Begin Secret Info:..
   See corresponding python file.
   End Secret Info..

3. **(20 points)** In this exercise, we look at some of the indifferentiability-based bounds for sponge constructions that were stated in the lecture slides, and show that these bounds are tight: is it possible to "break" these constructions in this number of queries? Use your implementation from exercise 2 for exercises 3b and 3d with parameters $b = 400$, $c = 40$, $r = 360$, and $n = 720$.

   (a) Sketch the generic approach to find a collision for sponge-based hash functions, and argue what the computational complexity is.

   (b) Use your implementation from exercise 2 to find an actual collision, i.e., provide two messages M, M' with $M \neq M'$ with the same hash $h(M) = h(M')$.

   (c) For the general case, e.g., if c, b, r and n are variables, what is actually the proven minimal complexity to find a collision against a sponge construction? Relate this bound to your attack.

   (d) Use your implementation from exercise 2 to find a second preimage for your name, i.e., if the encoding of your name (run 'python encoding.py your_name') is the message M, find another $M' \neq M$ such that $h(M) = h(M')$.

   (e) For the general case, e.g., if c, b, r and n are variables, what is actually the proven minimal complexity to find a second preimage against a sponge construction? Relate this bound to your attack.

   Begin Secret Info:..

   (a) Just $\mathcal{O}(2^{n/2})$ by general birthday-based collision finding.

   (b) The idea here is that if you try random messages M and M and keep track of their inner states, then at some point you will find a collision between these in $\mathcal{O}(2^{c/2})$ and from such an inner-collision, you can pick the right message to agree on the whole state. From that collision on, you apply the same permutation and so you get the same hash.

   (c) Together with (a) and (b) this leads to $\min(2^{c/2}, 2^{n/2})$ which agrees with what we saw in the lecture.

   (d) Now you cannot use two different messages to start with. Instead, let us try to find a second preimage for $h(M)$ for fixed M. We check the state right after the last squeeze, call this X. Then we can start with random messages M starting from the beginning of the sponge, and we can start with random messages M' working backwards from state X until we again find an inner collision. This should take $2^{c/2}$ again.

(e) *Note that we now do not have* $2^{n/2}$ *from (a)! We get* $\min(2^{c/2}, 2^n)$*.*

End Secret Info ....

4. **(10 points)** Consider SpongeWrap from lecture 5 slide 10. This construction is only a secure authenticated encryption scheme if the distinguisher cannot repeat nonces for encryption queries (it may repeat nonces for decryption queries, though).

   (a) Suppose above condition is violated and the distinguisher can repeat nonces for encryption queries. Describe an attacker that breaks the confidentiality of SpongeWrap in a constant number of queries.

   (b) Suppose we *omit* the domain separating bits 1/0 from SpongeWrap, and associated data and message are both padded with simple 10^*-padding (i.e., the last, incomplete, block of both A and M is padded with a 1 and a sufficient number of 0s to get an r-bit block). Describe an attacker that breaks the authenticity of SpongeWrap in a constant number of queries. (Note: the distinguisher is *not* allowed to repeat nonces for encryption queries.)

Begin Secret Info: ....

(a) *The attacker can make two SpongeWrap encryption queries with identical* N *and* A*, and differing first message blocks* $M_1 \neq M_1'$*:*

- $\mathit{SpongeWrap}(N, A, M_1 \| M_{rest}, r) = (C_1 \| C_{rest}, T)$,
- $\mathit{SpongeWrap}(N, A, M_1' \| M_{rest}', r) = (C_1' \| C_{rest}', T)$.

The resulting ciphertexts satisfy $C_1 \oplus M_1 = C_1' \oplus M_1'$*. This is unlikely to hold for a random oracle* \$*.*

Note: the attacker can also simply ask for two different tag lengths for the same message. In literature, this would not be considered a valid attack, but here, I will.

(b) *The attacker can make a single SpongeWrap encryption query, move the first block of* M *to the last block of* A*, and obtain a valid forgery. Below, we already consider data to be padded, and associated data and message are explicitly described in terms of their blocks:*

- $\mathit{SpongeWrap}(N, A, M_1 \| M_2, r) = (C_1 \| C_2, T)$,
- $\mathit{SpongeWrap}^{-1}(N, A \| M_1, C_2, T) = M_2$.

End Secret Info ....