

Cryptography and Security — Final Exam

Serge Vaudenay

29.1.2018

- duration: 3h
- no documents allowed, except one 2-sided sheet of handwritten notes
- a pocket calculator is allowed
- communication devices are not allowed
- the exam invigilators will **not** answer any technical question during the exam
- readability and style of writing will be part of the grade

1 Collision on Sponge-Based Hash Functions

The sponge is a construction for cryptographic hashing. It uses a cryptographic permutation $f : \{0, 1\}^b \rightarrow \{0, 1\}^b$. To instantiate the sponge with f , we have to pick the parameters r (called the “rate”) and c (called the “capacity”), such that $r + c = b$. The sponge also allows to set an arbitrary length h of the output. We fix f , c and h and set $H = \text{Sponge}[f, c, h](M)$ defined as follows. We first define $\text{pad}(a) = 10^{r-(a+2 \bmod r)}1$ for an integer a (i.e. the bitstring consisting of one bit 1 followed with $r - (a + 2 \bmod r)$ bits 0, then one bit 1: this is not a power of 10). Here, $|m|$ denotes the length of the bitstring m . We use the notation $\|$ for the concatenation of bitstrings. To compute the hash $H(m)$ of a message m (specified as a bitstring), we use the following pseudocode:

Input: m

```
1:  $M \leftarrow m \| \text{pad}(|m|)$  { $\text{pad}(|m|) = 10^{r-(|m|+2 \bmod r)}1$ }
2:  $\ell \leftarrow |M|/r$ 
3: split  $M = M_1 \| M_2 \| \dots \| M_\ell$  {split  $M$  into  $\ell$  blocks of  $r$  bits  $M_1, \dots, M_\ell$ }
4:  $S \leftarrow 0^b$ 
5: for  $i = 1$  to  $\ell$  do
6:    $S \leftarrow S \oplus (M_i \| 0^c)$ 
7:    $S \leftarrow f(S)$ 
8: end for
9:  $Z \leftarrow$  empty string
10: while  $|Z| < h$  do
11:    $Z \leftarrow Z \| \text{left}_r(S)$ 
12:    $S \leftarrow f(S)$ 
13: end while
14: return  $\text{left}_h(Z)$ 
```

where $\text{left}_r(Z)$ returns the r leftmost bits of a binary string Z . (We similarly define $\text{right}_c(Z)$ so that $Z = \text{left}_r(Z) \| \text{right}_c(Z)$ whenever Z is of b bits.) We see that the rate impacts performance, the bigger it is, the less times we need to call f to hash a message. Now we investigate how the capacity influences the security of a sponge-based hash function.

Q.1 Say why it is called a sponge and in which well-known algorithm is this construction used.


Q.2 Briefly describe a generic collision-search attack on the hash function H . What is its time and memory complexity? (It is convenient to measure the time complexity in computations which are equivalent to one call of f .)



Q.3 Prove that given two randomly chosen strings $x \neq x'$ such that $|x| = |x'| = b$, the probability that $\text{right}_c(f(x)) = \text{right}_c(f(x'))$ is about 2^{-c} . (To make this estimation, you can assume that f is a “random permutation”.)

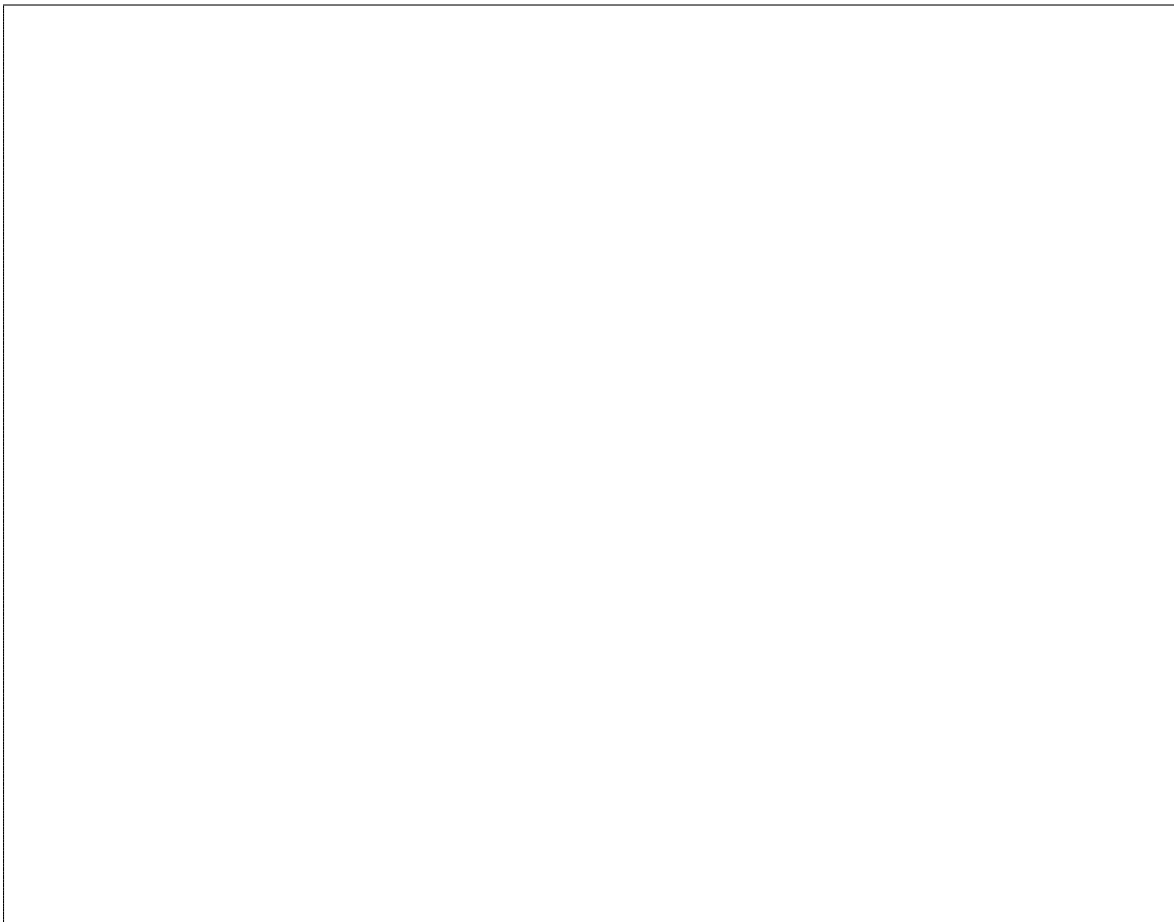


Q.4 Describe an algorithm that will find two $x \neq x'$ such that $|x| = |x'| = b$ and $\text{right}_c(f(x)) = \text{right}_c(f(x'))$. What is its time and memory complexity?



Q.5 Assume that $h > c$. Describe a collision-search attack on the hash function $H = \text{Sponge}[f, c, h](M)$ that is more efficient than the generic collision search from Q.2. What is its time and memory complexity?

HINT: You can assume that after processing $\lceil c/2r \rceil$ randomly chosen message blocks, the state S behaves as $f(x)$ with a random input x .



2 LPN with Extreme Noise Parameters

We consider a set of parameters consisting of an integer k and a probability τ . We assume that a secret vector $s \in \{0, 1\}^k$ is initially selected with a uniform distribution. We define a source \mathcal{S} which generates some random pairs (v, b) as follows: first, the source picks a random vector $v \in \{0, 1\}^k$ with a uniform distribution and (independently) a random bit e such that $\Pr[e = 1] = \tau$. Then, the source produces $b = \langle v, s \rangle \oplus e$ where we use the notation $\langle \cdot, \cdot \rangle$ for the dot product in \mathbf{Z}_2^k defined by

$$\langle v, s \rangle = (v_1 s_1 + \cdots + v_k s_k) \bmod 2$$

and we use the notation \oplus for the addition in \mathbf{Z}_2 . The components of v and s are the v_i and s_i for $i = 1, \dots, k$, respectively. The output (v, b) of the source \mathcal{S} is called a *sample*. Upon a query to \mathcal{S} , the source produces a sample. By making a query, we get only the freshly generated sample (v, b) but we see neither the secret s nor the noise bit e .

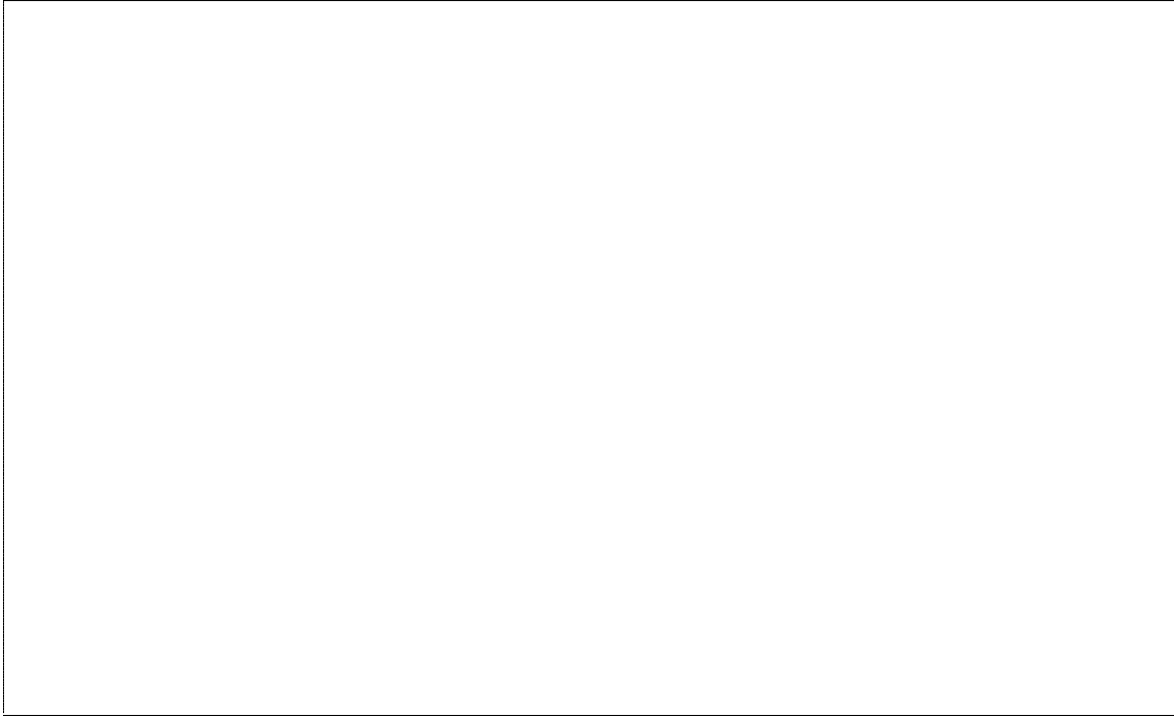
The *Learning Parity with Noise problem* (LPN) consists of designing an algorithm $\mathcal{A}^{\mathcal{S}}(k, \tau)$ which recovers the secret vector s by only querying the source \mathcal{S} and knowing the parameters k and τ , with a “large enough” probability of success (e.g. at least 50%):

$$\Pr[\mathcal{A}^{\mathcal{S}}(k, \tau) \rightarrow s] \geq \frac{1}{2}$$

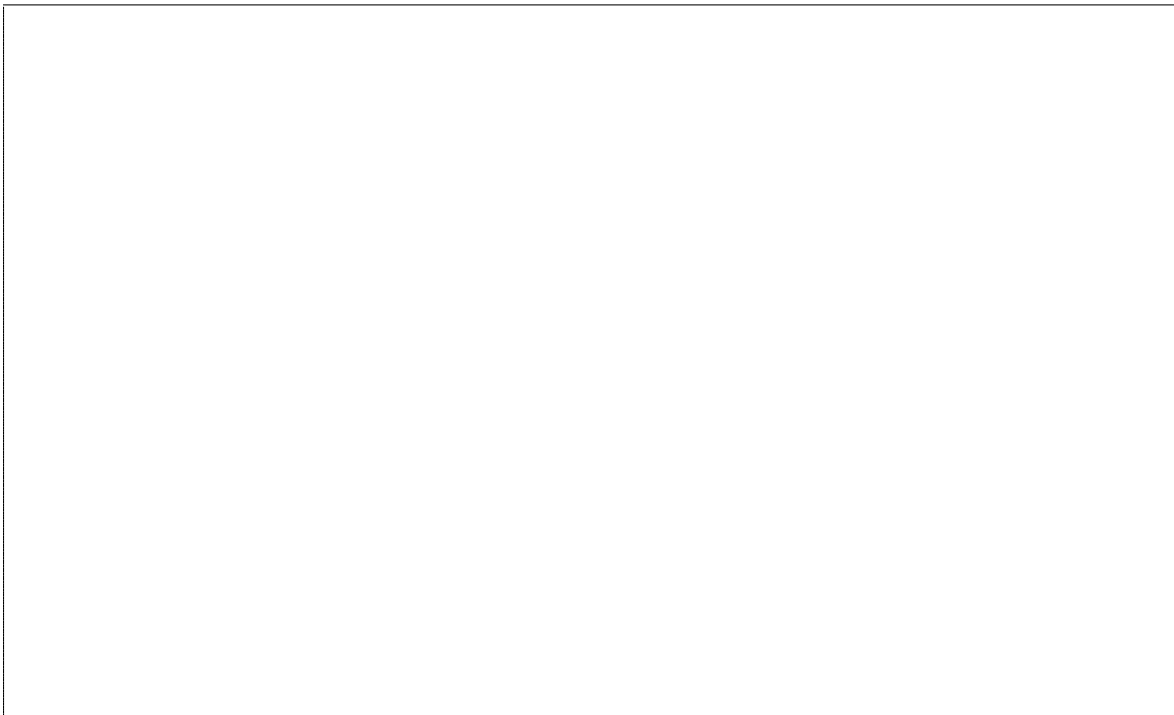
We assume that τ is a function of k and we measure the *complexity* of \mathcal{A} with a *time* complexity and a *query* complexity. Both are asymptotic in terms of k . The time complexity is the asymptotic expected complexity in number of binary operations. The query complexity is the number of queries to the source. The LPN problem is often used in cryptography with τ between $\frac{\text{cte}}{\sqrt{k}}$ and $\frac{1}{2} - \frac{\text{cte}}{k}$, for a constant cte , as it is believed to be hard (even with the help of quantum computers!). In this exercise, we investigate values of τ which are not in this range.

Q.1 For $\tau = 0$, prove that the LPN problem is easy to solve: there exists a polynomially bounded (in terms of k) algorithm to solve it. Briefly describe this algorithm and its complexity (time and query).

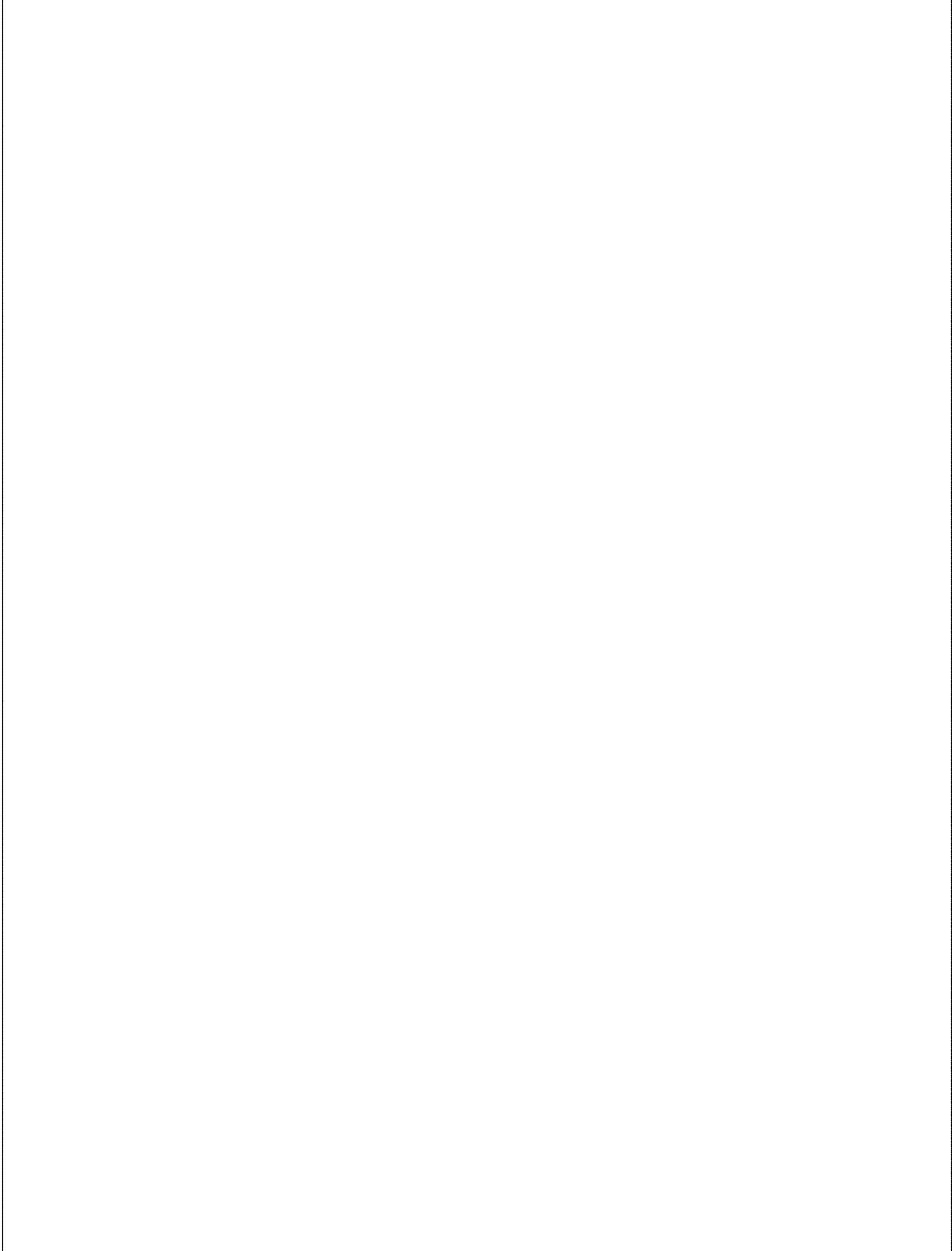
Q.2 For $\tau = \frac{1}{2}$, prove that the LPN problem is impossible to solve.
HINT: first show that the source is uniform whatever s .



Q.3 For $\tau = \mathcal{O}(\frac{1}{k})$, we show in the next questions that the LPN problem is still easy to solve.
Q.3a Prove that we can easily get k samples (v_i, b_i) , $i = 1, \dots, k$ such that all v_i are linearly independent. Briefly describe this algorithm with a pseudocode and its complexity (time and query).



Q.3b We denote $b_i = \langle v_i, s \rangle \oplus e_i$. Prove that the number X of $i \in \{1, \dots, k\}$ such that $e_i = 1$ is a random variable which has an expected value equal to $k\tau$ and a variance of $k\tau(1 - \tau)$.



Q.3c By using the previous questions, prove that for $\tau = \mathcal{O}(\frac{1}{k})$ we can recover s from the obtained samples. Briefly describe an algorithm with a pseudocode and its complexity (time and query).

HINT: there may be an exhaustive search on a tiny space.

HINT²: $\Pr[X > k\tau] \leq 37\%$.

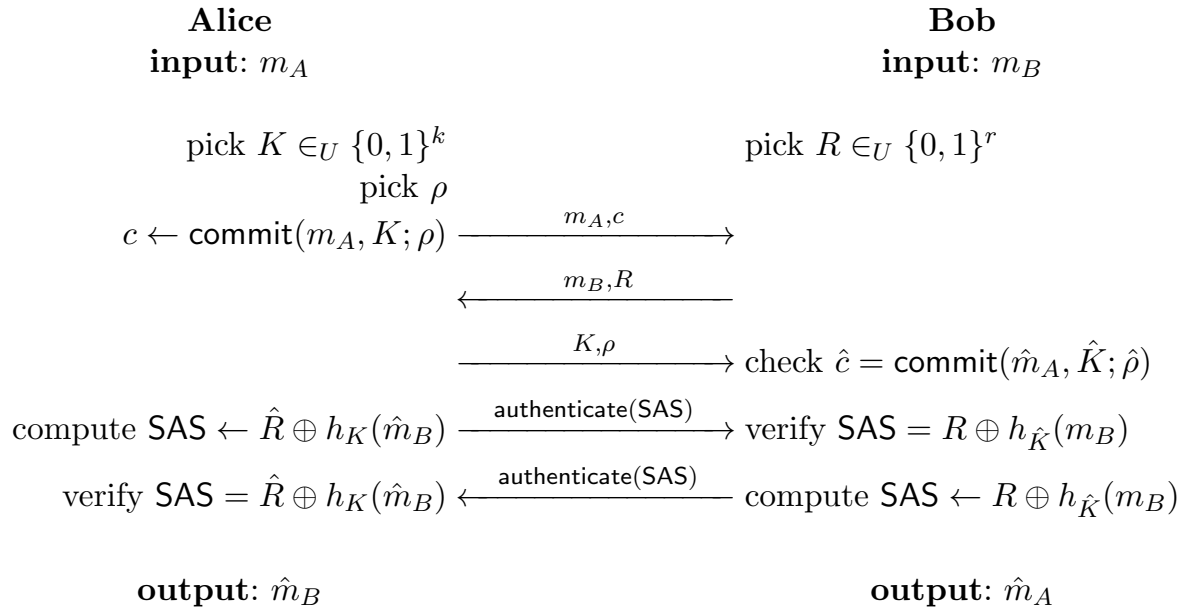
3 Cross-Authentication based on Short Authenticated Strings

Let r and k be two integers.

We consider a commitment scheme `commit` with which we commit to (m_A, K) (where m_A is a message and K is a hash key) by picking some coins ρ and producing $c = \text{commit}(m_A, K; \rho)$ and we open the commitment by releasing (m_A, K, ρ) .

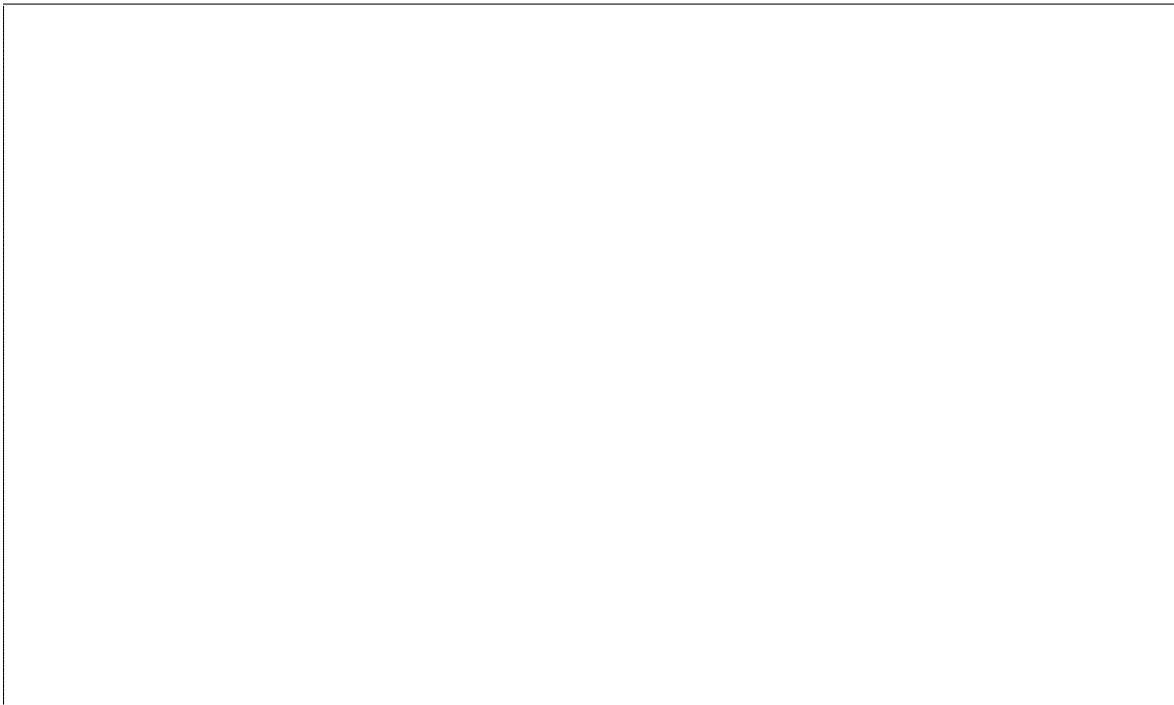
We also consider a strongly universal hash function family h_K from the set of all finite bitstrings to $\{0, 1\}^r$, defined by a key $K \in \{0, 1\}^k$. This is such that for any fixed m, m' such that $m \neq m'$, the pair $(h_K(m), h_K(m'))$ is uniformly distributed in $(\{0, 1\}^r)^2$ when $K \in \{0, 1\}^k$ is uniformly distributed.

We define the following SAS-based message cross-authentication protocol: if Alice wants to send the message m_A and Bob wants to send the message m_B , they run the following protocol. Communications with the `authenticate` tags are authenticated (i.e. sent through a special authenticated channel) in the sense that an adversary cannot create or modify the message. Other communications are done through an insecure channel which may be corrupted by an adversary. For this reason, we put a hat on notations to designate received values. For instance, if Alice sends K to Bob through an insecure channel, Bob receives \hat{K} which is equal to K only if the adversary lets the message unchanged.



In the considered attacks in this exercise, we have a (wo)man-in-the-middle (let's call her Eve) who controls the (unauthenticated) channels and substitutes all variables by some variants with a hat that she computes. She wants that for any $m_A, m_B, \hat{m}_A, \hat{m}_B$, there exists an *efficient* way to make Alice and Bob succeed (i.e., both verifications succeed), with input/output m_A/\hat{m}_B and m_B/\hat{m}_A , respectively. If not specified otherwise, the attack is supposed to succeed with probability 100%. We assume that h and `commit` are *easily* computable. We assume that r is small so that a loop with 2^r iterations is considered as *efficient*. Eve can use $m_A, m_B, \hat{m}_A, \hat{m}_B$ as input before the attack starts.

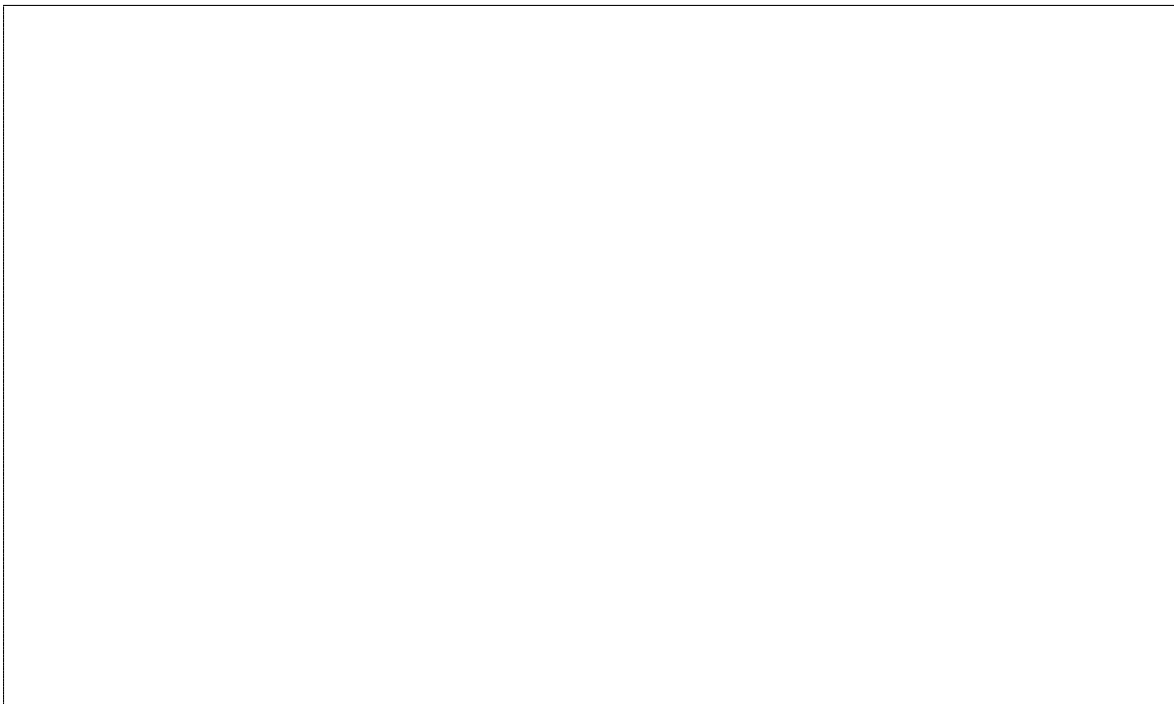
Q.1 Explain what this protocol could be used for and a typical use case.



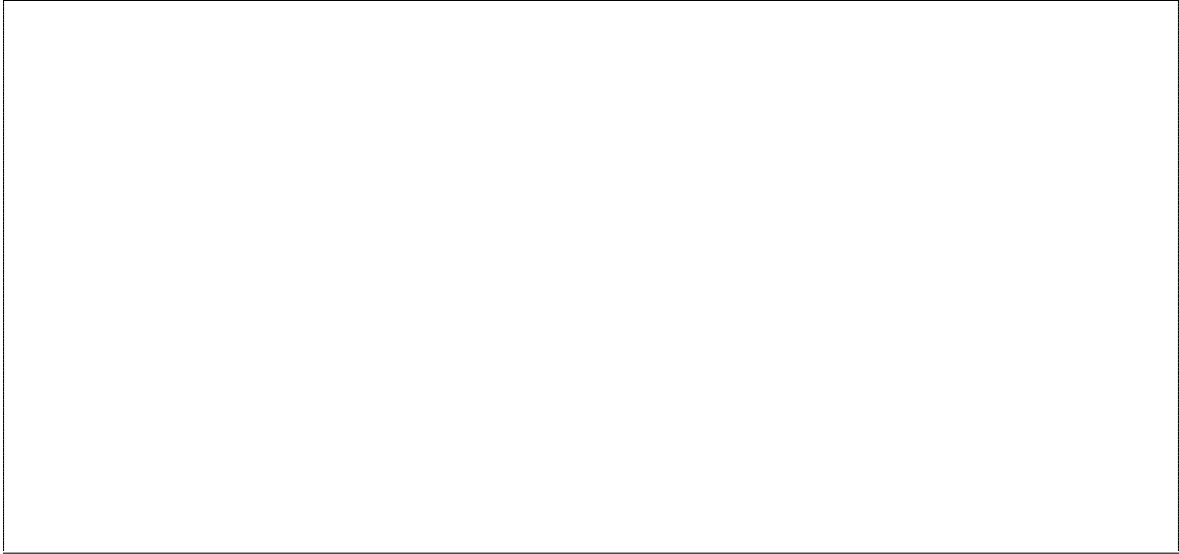
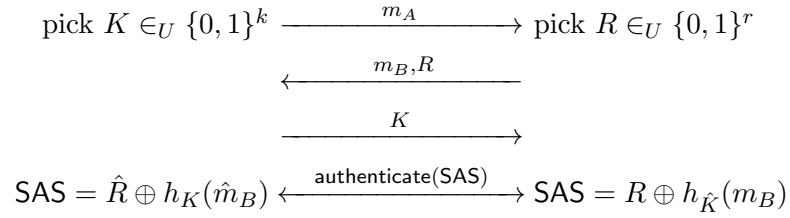
Q.2 (Case of a non-hiding commitment.) If there exists an efficiently implementable function reveal such that for all m, K, ρ , we have

$$\text{reveal}(m, \text{commit}(m, K; \rho)) = K$$

prove that there is an attack.



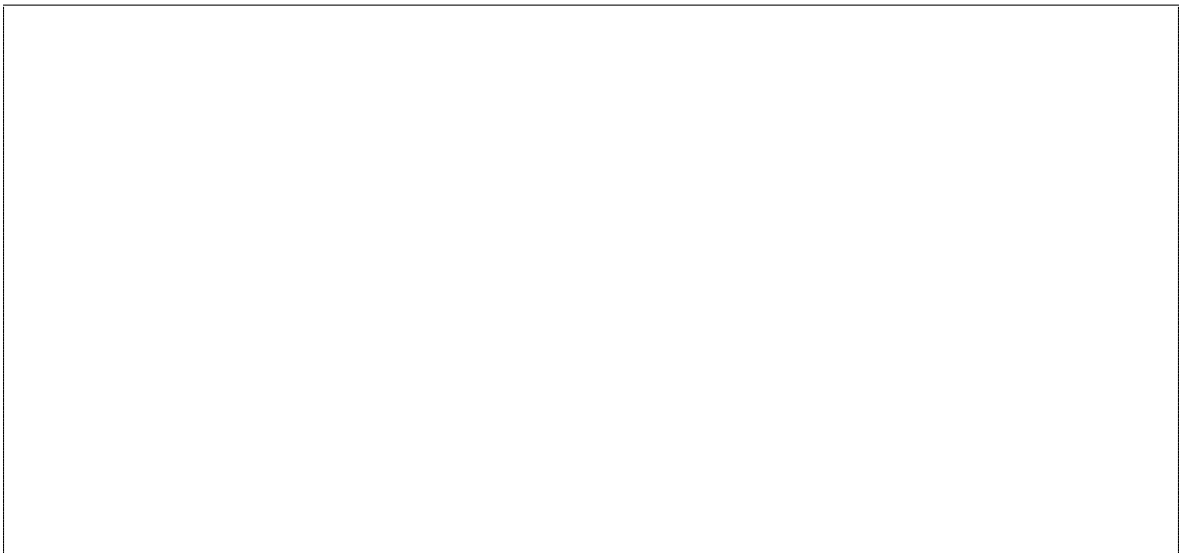
Q.3 (Case of a modified protocol.) If we modify the protocol by removing the commitment (as shown below), prove that there is an attack using a loop of 2^r iterations.



Q.4 (Case of a non-binding commitment.) If there exists an efficiently implementable function `equivocate` returning random coins ρ such that for all c, \hat{m}, \hat{K} , we have

$$\text{commit}(\hat{m}, \hat{K}; \text{equivocate}(c, \hat{m}, \hat{K})) = c$$

prove that there is an attack using a loop of 2^r iterations.



Q.5 (General case: attack with success probability 2^{-r} .) Prove that there is an attack which succeeds with probability 2^{-r} .