Advanced Cryptography — Final Exam
Solution

Serge Vaudenay

28.6.2010

- all documents are allowed
- a pocket calculator is allowed
- communication devices are not allowed
- answers to the exercises must be provided on a separate sheet
- readability and style of writing will be part of the grade
- do not forget to put your name on your copy!

1 Chameleon Hash Function from Σ-Protocol

This exercise is inspired from Bellare-Ristov, Hash Functions from Sigma Protocols and Improvements to VSH*, published in the proceedings of ASIACRYPT 2008, LNCS vol. 5350, Springer.*

In this exercise we define a **partial Σ-protocol** a tuple consisting of

- a relation $R(x, w)$ holding on two words x and w called an *instance* and a *witness* respectively, defining a language $L_R = \{x; \exists w \quad R(x, w)\}$;
- a function $a = P(x, w; r_P)$ computing a commitment a on an instance x, a witness w, and random coins r_P;
- a samplable domain E_x called *set of challenges*;
- a function for $z = P(x, w, e; r_P)$ computing a *response* z on an instance x, a witness w, a challenge e, and random coins r_P;
- a verification relation $V(x, a, e, z)$ on an instance x, a commitment a, a challenge e, and a response z.

We further define by A_x and Z_x the set of all possible values for a and z, respectively. These objects shall satisfy the following properties:

- $R(x, w)$, $P(x, w; r_P)$, $P(x, w, e; r_P)$, and $V(x, a, e, z)$ can be evaluated in a time which is polynomial with respect to the length of the instance x;
- we can generate a random element of E_x with uniform distribution in a time which is polynomial with respect to the length of x;
- for every x, w, r_p, e such that $e \in E_x$ and $R(x, w)$ holds, if we define $a = P(x, w; r_P)$ and $z = P(x, w, e; r_P)$, then $V(x, a, e, z)$ holds;

The partial Σ-protocol is executed as follows:

- the prover uses as input (x, w) such that $R(x, w)$ holds and receives some random coins r_p;
- the verifier uses as input x and receives some random coins to generate $e \in E_x$ uniformly distributed;

– the prover sends $a = P(x, w; r_P)$ to the verifier;
– the verifier sends e to the prover;
– the prover sends $z = P(x, w, e; r_P)$ to the verifier;
– the verifier checks that $V(x, a, e, z)$ holds and accept.

In the case of any deviation (e.g. the final check or any computation failed), the protocol fails.

Q.1 Which objects are missing to define a Σ-protocol?

> *An extractor $E(x, a, e, z, e', z')$ to compute a witness from two accepted transcripts (a, e, z) and (a, e', z') with same commitment a and different challenges $e \neq e'$, and a simulator $S(x, e; r_S)$ to generate a transcript (a, e, z) from x and e with correct distribution.*

We call a **strong Σ-protocol** a partial Σ-protocol together with a function $H_x(e, z) = a$ and a function $E(x, a, e, z, e', z')$ such that

– $H_x(e, z)$ can be evaluated in a time which is polynomial for any $e \in E_x$ and $z \in Z_x$;
– $E(x, a, e, z, e', z')$ can be evaluated in a time which is polynomial for any $a \in A_x$, $e, e' \in E_x$, and $z, z' \in Z_x$;
– we can generate a random elements of Z_x with uniform distribution in a time which is polynomial with respect to the length of x;
– for all x, $e \in E_x$, $z \in Z_x$, $V(x, H_x(e, z), e, z)$ holds;
– honestly executing the partial Σ-protocol makes z uniformly distributed in Z_x and independent from e;
– for all x, $a \in A_x$, $e \in E_x$, and $z \in Z_x$, $V(x, a, e, z)$ holds if and only if $a = H_x(e, z)$;
– for every x, a, e, e', z, z' such that $e, e' \in E_x$, $z, z' \in Z_x$, $(e, z) \neq (e', z')$, $V(x, a, e, z)$ holds, and $V(x, a, e', z')$ holds, if we define $w = E(x, a, e, z, e', z')$, then $R(x, w)$ holds.

Q.2 What is the difference between the hypothesis on E and the special soundness property of Σ-protocols?

> *Now it works whenever $(e, z) \neq (e', z')$ instead of $e \neq e'$. Somehow, the new property for E is stronger than the property of special soundness.*

Show that a strong Σ-protocol is a Σ-protocol.

> *Computability and completeness are already satisfied by the definition of a partial Σ-protocol. Special soundness is implied by the new definition of E. We construct a simulator $S(x, e; r) = (H_x(e, z), e, z)$ where $z \in Z_x$ is generated with uniform distribution in Z_x given r. The honest execution of the protocol with instance x generates a transcript (a, e, z) with a given distribution such that $V(x, a, e, z)$ holds and e is uniformly distributed in E_x. Due to the definition of strong Σ-protocols, z is uniformly distributed and independent from e and $a = H_x(e, z)$. So, the transcript has the same distribution as the one from the $S(x, e; r)$ when $e \in E_x$ is random.*

Q.3 Show that given x and w such that $R(x, w)$ holds, we can create a collision on the function H_x.

With some random r_P and two different $e, e' \in E_x$ we can compute $a = P(x, w; r_P)$, $z = P(x, w, e; r_P)$, and $z' = P(x, w, e'; r_P)$. Since $V(x, a, e, z)$ and $V(x, a, e', z')$ hold, we must have $a = H_x(e, z)$ and $a = H_x(e', z')$, so $H_x(e, z) = H_x(e', z')$. Since $e \neq e'$, this is a collision.

Q.4 Show that given $x \in L_R$, finding a collision on H_x implies finding a witness for $x \in L_R$.

Assume that $a = H_x(e, z) = H_x(e', z')$ with $(e, z) \neq (e', z')$. We know that $V(a, e, z)$ and $V(a, e', z')$ hold due to the property of a strong Σ-protocol. Since $(e, z) \neq (e', z')$, $w = E(x, a, e, z, e', z')$ is a witness for x.

Deduce that if R is such that given $x \in L_R$ it is hard to find w such that $R(x, w)$ holds, we can define a trapdoor collision resistant hash function by using x as a common reference string.

We generate x and w such that $R(x, w)$ holds and declare x as being the common reference string. Then, w is a trapdoor. We have shown that making a collision implies recovering the trapdoor so H_x is collision-resistant.

Q.5 Recall the Goldwasser-Micali-Wigderson Σ-protocol based on graph isomorphism.

The relation is $R((G_0, G_1), \varphi)$ where the witness φ is invertible and such that $\varphi(G_0) = G_1$

Prover		***Verifier***
φ *st* $\varphi(G_0) = G_1$	(G_0, G_1)	
pick π *invertible*		*pick* $e \in \{0, 1\}$
$H = \pi(G_0)$	$\xrightarrow{H}$	
	$\xleftarrow{e}$	
$\sigma = \pi \circ \varphi^{-e}$	$\xrightarrow{\sigma}$	$\sigma(G_e) \stackrel{?}{=} H$

Show that the Golwasser-Micali-Wigderson Σ-protocol is not a strong Σ-protocol.

If we have a non-trivial automorphism τ of the graph G_e, then if (H, e, σ) is an accepted transcript, then $(H, e, \sigma \circ \tau)$ as well. However, we cannot extract a witness from the two transcripts.

Q.6 Recall the Fiat-Shamir Σ-protocol.

The relation $R((n, v), s)$ holds if and only if $s^2 v \bmod n = 1$.

Prover		***Verifier***
s *st* $s^2 v \bmod n = 1$	(n, v)	
pick $r \in \mathbf{Z}_n^*$		*pick* $e \in \{0, 1\}$
$x = r^2 \bmod n$	$\xrightarrow{x}$	
	$\xleftarrow{e}$	
$y = rs^e \bmod n$	$\xrightarrow{y}$	$y^2 v^e \bmod n \stackrel{?}{=} x$

Show that the Fiat Shamir Σ-protocol is not a strong Σ-protocol.

We can have two accepted transcripts (x, e, y) and $(x, e, -y \bmod n)$ with same x which are not enough to extract a witness.

Q.7 Recall the Schnorr Σ-protocol.

The relation $R((G, q, g, y), x)$ holds if and only if $g^x = y$ in group G, where q is a prime greater than 2^t, and g has order q in G.

Prover		***Verifier***
x *st* $g^x = y$	(G, q, g, y)	
pick $k \in \mathbf{Z}_q$		*pick* $e \in \{1, \ldots, 2^t\}$
$r = g^k$	$\xrightarrow{r}$	q *prime* $> 2^t$
	$\xleftarrow{e}$	g, y *of order* q
$s = ex + k \bmod q$	$\xrightarrow{s}$	$ry^e \stackrel{?}{=} g^s$

Show that the Schnorr Σ-protocol is a strong Σ-protocol.

If (r, e, s) and (r, e', s') are accepted transcripts, we have $s, s' \in \mathbf{Z}_q$, $ry^e = g^s$ and $ry^{e'} = g^{s'}$. If $e \neq e'$ we know that we can extract a witness. If $e = e'$, we obtain that $g^s = g^{s'}$. Since g has order q, we must have $s = s'$ in $\mathbf{Z}_q$. This is not possible if $(e, s) \neq (e', s')$.
Furthermore, (r, e, s) is accepted if and only if $r = g^s y^{-e}$ so we can define $H_y(e, s) = g^s y^{-e}$.
Finally, s is uniformly distributed in $\mathbf{Z}_q$. So, we have a strong Σ-protocol.

Deduce a trapdoor hash function based on this protocol. Does it remind you something?

Let x be a trapdoor and $y = g^x$ be a CRS. We define $H_y(e, s) = g^s y^{-e}$ which looks like the Pedersen commitment.

2 Instances of the ElGamal

Let p be a large prime number and g be an element of $\mathbf{Z}_p^*$. We denote by q the order of g. We let $\mathcal{G}$ be a subgroup of $\mathbf{Z}_p^*$ which include g. We let $\mathcal{M} = \{0,1\}^\ell$ be the message space. We assume an injective function $e : \mathcal{M} \to \mathcal{G}$ which is called an *embedding function*. We further assume that given a random $m \in \mathcal{M}$, $e(m)$ "looks like" uniformly distributed in $\mathcal{G}$. In this exercise, we consider the ElGamal cryptosystem using domain parameters (p, g, q, e) with different choices on how to select them. Namely, a secret key is a value $x \in \mathbf{Z}_q$, its public key is $y = g^x \bmod p$. For any message $m \in \mathcal{M}$, the encryption of m with public key y is a pair (u, v) such that $u = g^r$ with $r \in \mathbf{Z}_q$ random and $v = e(m)y^r$. The decryption of (u, v) with secret key x is $m = e^{-1}(vu^{-x})$.

Q.1 We assume here that g is a generator of $\mathbf{Z}_p^*$. What is the value of q?

$q = p - 1$.

Is the cryptosystem IND-CPA secure? Why?

The IND-CPA security is equivalent to the hardness of the decisional Diffie-Hellman problem with generator g. However, the order of g is even so the least significant bit of the discrete logarithm of any $z \in \mathbf{Z}_p$ is easy to compute from the Legendre symbol $\left(\frac{z}{p}\right)$. Hence, we can easily distinguish (g, g^x, g^r, g^{xr}) from (g, g^x, g^r, g^s) by checking that the least significant bit of xr is the product of the least significant bits of x and r. *Indeed, and adversary can select two messages m_0 and m_1 such that the least significant bit of $\log e(m_b)$ is b. (Given a random m, $\log e(m)$ is a random bit with distribution close to uniform, so we can easily find m_0 and m_1.) Then, given the encryption (u, v) of m_b, he can compute $b = \log(vu^{-x}) = \log v - (\log y)\log u$. So, the ElGamal cryptosystem is not IND-CPA secure.*

Q.2 We assume here that q is prime and that $\mathcal{G} = \mathbf{Z}_p^*$.
Is the cryptosystem IND-CPA secure? Why?

The same IND-CPA adversary works here. So, the ElGamal cryptosystem is not IND-CPA secure.

Q.3 We assume here that q is a large prime but much smaller than p, and that $\mathcal{G}$ is generated by g.
Is the cryptosystem IND-CPA secure? Why?

In this case, the decisional Diffie-Hellman problem is assumed to be hard. We know that the IND-CPA security in this case is equivalent to the decisional Diffie-Hellman problem. So, the ElGamal cryptosystem is IND-CPA secure.

In practice, how to propose an efficient embedding function e?

It is pretty hard because e must be invertible in practice.

Q.4 We assume here that $p = 1 + 2q$ with q prime and that $\mathcal{G}$ is generated by g.
Is the cryptosystem IND-CPA secure? Why?

Yes. This is a particular case of the previous question.

Show that $\mathcal{G}$ is the subgroup of all quadratic residues in $\mathbf{Z}_p^*$.

We know that the group of quadratic residues include exactly $\frac{p-1}{2} = q$ elements. We know that $\mathcal{G}$ has q elements. Furthermore, $g^{\frac{p-1}{2}} = g^q = 1$ so g is a quadratic residue. So, all elements of $\mathcal{G}$ are quadratic residues. Therefore, all quadratic residues are in $\mathcal{G}$.

Compute $\left(\frac{-1}{p}\right)$.

We have $(-1)^{\frac{p-1}{2}} = (-1)^q = -1$ so the Legendre symbol is -1.

Deduce that for any $x \in \mathbf{Z}_p^*$ then either x or $-x$ is in $\mathcal{G}$.

Either x or $-x$ is a quadratic residue but not both since -1 is not a quadratic residue. So, either x or $-x$ is in $\mathcal{G}$.

Finally, if $\ell = \lfloor \log_2 q \rfloor$, propose a practical embedding function e.

Let $e_0(m) - 1$ be the integer with binary expansion m. We have $0 < e_0(m) \leq q$. Let now $e(m) = e_0(m)$ if $\left(\frac{e_0(m)}{p}\right) = +1$ and $e(m) = -e_0(m)$ otherwise. We have $e(m) \in \mathcal{G}$. Since we cannot have $e(m) = e(m')$ whenever $m \neq m'$, this is a practical embedding function. Its inverse is also easy to compute.

3 Trusted Agent Setup

This exercise is inspired from Mateus-Vaudenay, On Tamper-Resistance from a Theoretical Viewpoint: The Power of Seals*, published in the proceedings of CHES 2009, LNCS vol. 5747, Springer.*

In this exercise we consider protocols between two participants A and B involving a trusted third participant C. This participant C is assumed to always be honest, which means that he always follows the protocol he is expected to follow. We further assume that communication channels between the participants are authenticated and protect message integrity.

As an example we can specify this way a commitment protocol in which A can commit on a value x to B:

- A's protocol: send the input x to C
- C's protocol: receive some message x' form A then send a message "commit" to B
- B's protocol: receive a message form C and check that it is "commit"

For the opening phase:

- A's protocol: send a message "open" to C
- C's protocol: receive a message from A, check that it is "open", and send x' to B
- B's protocol: receive a message x'' from C and take it as the output

The protocol is *perfectly hiding* since what B receives in the commit phase is independent from x. The protocol is *perfectly binding* since whatever x'' that B receives from C must be equal to x' (due to message integrity and authentication assumption) sent by C which must be the x' received by C in the first phase (due to the honesty assumption on C), which must be equal to x (due to message integrity and authentication assumption).

Q.1 Propose a zero-knowledge proof of knowledge for a relation $R(x, w)$ in which A can prove to B that he knows w such that the predicate $R(x, w)$ holds, being given a public x. The protocol shall be perfectly complete, perfectly sound, and perfectly zero-knowledge. Specify the protocol/algorithm for A, B, and C.

- *A's protocol: send x and w to C*
- *C's protocol: receive x' and w' from A, verify that $R(x', w')$ holds, and send x' to B*
- *B's protocol: receive x'' from C and verify that $x = x''$*

Clearly, this protocol is perfectly complete: if everyone is honest, B always accept. If A is malicious and B accept, it must have received x form C. Since C is honest and has sent x (due to the assumption on the channel), he must have received x and w' such that $R(x, w')$ holds. So, A must have sent w' such that $R(x, w')$ holds (due to the assumption on the channel). Thus, by running the algorithm on the malicious A we can extract w' such that $R(x, w')$ holds. This protocol is thus perfectly sound. Clearly, what B sees (namely: x) can be simulated. So, the protocol is perfectly zero-knowledge.

We now change the setup assumption a bit. We assume that there are several participants $C_1, \ldots, C_n$ such that when queried for the first time with a program p they boot on this program and follow it honestly. Additionally, participants can query them with a special query "Code" on which they return by p so that we can see what program they were booted with. These extra participants called *trusted agents* can have a pseudorandom generator embedded.

Q.2 Redo the previous question in this model.

- *A's protocol: send code p to a new C_i, then send x and w to C_i and i to B*
- *code p: receive x' and w' from a participant P, verify that $R(x', w')$ holds, and send $[P, x']$ to B*
- *B's protocol: receive i' from A, send "Code" to $C_{i'}$, receive p', then receive $[P', x'']$ from C_j, verify that $i' = j$, $p' = p$, $P' = A$, and $x = x''$*

The updated proof is straightforward.

Q.3 For any proof system in the standard model (that is, who does not involve any trusted agent), show that a malicious adversary using a trusted agent can break the deniability property. Namely, the malicious verifier can later prove that w such that $R(x, w)$ holds is known by someone.

We use a malicious verifier who program the following code p, boot a trusted agent with p, then forward all message between A and the trusted agent. Code p works as follows:

- *simulate the honest verifier algorithm*
- *if succeeds, answer any query by the message "x proven"*

After the protocol succeeds, a new participant D can query B's trusted agent and receive the message "x proven". Clearly, no simulator can end up to such state.