Advanced Cryptography — Final Exam
Solution

Serge Vaudenay

20.6.2011

I Σ-Protocol for $\mathcal{P}$

We consider an alphabet Z, a polynomial P, and a predicate R. We assume that R can be computed in polynomial time. Given $x \in Z^*$, we let

$$R_x = \{w \in Z^*; R(x,w) \text{ and } |w| \leq P(|x|)\}$$

where $|x|$ denotes the length of x. We define the language L from R by

$$L = \{x \in Z^*; R_x \neq \emptyset\}$$

Q. In this question, we assume that there is an algorithm $\mathcal{A}$ such that for any $x \in L$, we obtain $\mathcal{A}(x) \in R_x$ and that for any $x \in Z^*$, the running time of $\mathcal{A}(x)$ is bounded by $P(|x|)$.
Construct a Σ-protocol for L. Carefully specify all protocol elements and prove all properties which must be satisfied.

Let ε be a word of length 0.

- *We define $\mathcal{P}(x,w) = \varepsilon$ and $\mathcal{P}(x,w,e) = \varepsilon$.*
- *We take the set of challenges $E = \{\varepsilon\}$. We could actually take any set of challenges with polynomially bounded length.*
- *The verification algorithm $V(x,a,e,z)$ first computes $w = \mathcal{A}(x)$, then checks if $R(x,w)$ holds.*
- *Clearly, this protocol satisfies completeness ($x \in L$ is accepted by the verifier when the protocol is honestly run).*
- *Clearly, the algorithms run in polynomial time in terms of $|x|$.*
- *To define a polynomial time extractor based on some values x,a,e,e',z,z' such that $V(x,a,e,z)$ and $V(x,a,e',z')$ hold, and $e \neq e'$, we simply compute $w = \mathcal{A}(x)$. Clearly, we obtain a polynomial-time extractor.*
- *To define a simulator $S(x,e)$, we just take $(a,z) = (\varepsilon,\varepsilon)$. Clearly,*

  $$\Pr[S(x,e) = (a,z)] = \Pr[\mathcal{P}(x,w) = a, \mathcal{P}(x,w,e) = z]$$

  So, we obtain a polynomial-time simulator.

So, all properties of a Σ-protocol are satisfied.

II OR Proof

The exercise is inspired by Proof of Partial Knowledge and Simplified Design of Witness Hiding Protocols *by Cramer, Damgård and Schoenmakers. Published in the proceedings of Crypto'94 pp. 174–187, LNCS vol. 839, Springer 1994.*

Let $Z = \{0,1\}$ be an alphabet. We consider two Σ-protocols Σ_1 and Σ_2 for two languages L_1 and L_2 over the alphabet Z defined by two predicates R_1 and R_2. We assume that Σ_1 and Σ_2 use the same challenge set E which is given a group structure with a law $+$. For Σ_i, $i \in \{1,2\}$, we denote $\mathcal{P}_i$ the prover algorithm, V_i the verification predicate, $\mathcal{E}_i$ the extractor, and $\mathcal{S}_i$ the simulator.

Q.1 (**AND proof**) Construct a Σ protocol $\Sigma = \Sigma_1$ AND Σ_2 for the language defined by

$$R((x_1,x_2),(w_1,w_2)) \iff R_1(x_1,w_1) \text{ AND } R_2(x_2,w_2)$$

The prover and the verifier are simply defined by a parallel execution of Σ_1 and Σ_2 together with the same challenge. So are the extractor and the simulator.
More precisely, $\mathcal{P}((x_1,x_2),(w_1,w_2);r_1,r_2)$ runs $\mathcal{P}_i(x_i,w_i;r_i) = a_i$ for $i = 1,2$ and yield (a_1,a_2). Uppon challenge $e \in E$, $\mathcal{P}((x_1,x_2),(w_1,w_2),e;r_1,r_2)$ runs $\mathcal{P}_i(x_i,w_i,e;r_i) = z_i$ for $i = 1,2$ and yield (z_1,z_2). The verification holds $V((x_1,x_2),(a_1,a_2),e,(z_1,z_2))$ if and only if both $V_i(x_i,a_i,e,z_i)$ hold for $i = 1,2$. The extractor $\mathcal{E}((x_1,x_2),(a_1,a_2),e,e',(z_1,z_2),(z'_1,z'_2))$ runs $w_i = \mathcal{E}_i(x_i,a_i,e,e',z_i,z'_i)$ for $i = 1,2$ and yield (w_1,w_2). The simulator $\mathcal{S}((x_1,x_2),e)$ runs $(a_i,z_i) = \mathcal{S}_i(x_i,e)$ for $i = 1,2$ and yields $((a_1,a_2),(z_1,z_2))$.
Note: it is important to use the same challenge for both protocols in order to avoid troubles in the extraction.

(**OR proof**) In the remaining of the exercise, we now let

$$R((x_1,x_2),w) \iff R_1(x_1,w) \text{ OR } R_2(x_2,w)$$

This predicate defines a new language L. We construct a new Σ-protocol $\Sigma = \Sigma_i$ OR Σ_2 for L by

- $\mathcal{P}((x_1,x_2),w;r_1,r_2)$ finds out i such that $R_i(x_i,w)$ holds, sets $j = 3-i$, then picks a random $e_j \in E$ and runs $\mathcal{S}_j(x_j,e_j;r_1) = (a_j,e_j,z_j)$. Then, it runs $\mathcal{P}(x_i,w;r_2) = a_i$ and yield (a_1,a_2).
- Upon receiving e, $\mathcal{P}((x_1,x_2),w,e;r_1,r_2)$ sets $e_i = e - e_j$, runs $\mathcal{P}(x_i,w,e_i;r_2) = z_i$ and yields (e_1,e_2,z_1,z_2).

The verification predicate is

$$V((x_1,x_2),(a_1,a_2),e,(e_1,e_2,z_1,z_2)) \iff \begin{cases} e = e_1 + e_2 \text{ AND} \\ V_1(x_1,a_1,e_1,z_1) \text{ AND} \\ V_2(x_2,a_2,e_2,z_2) \end{cases}$$

Q.2 Show that Σ is complete and works in polynomial time.

The protocol $\mathcal{P}$ is a finite sequence of polynomial time operations or subroutines, so it is polynomial. Since V_1 and V_2 have a polynomially bounded complexity, so does V. We already know that E is polynomially samplable. So Σ works in polynomial time (except that we did not specify yet the extractor and the simulator).
If the protocols are honestly run, we have $\mathcal{S}_j(x_j,e_j) \to (a_j,e_j,z_j)$. So, by the property of the simulator for Σ_j, we have that $V_j(x_j,a_j,e_j,z_j)$ holds. Since w is a correct witness for x_i in Σ_i, since $\mathcal{P}(x_i,w;r_2) = a_i$ and $\mathcal{P}(x_i,w,e_i;r_2) = z_i$, due to the completeness of Σ_i we have that $V_i(x_i,a_i,e_i,z_i)$ holds. Since we further have $e_i = e - e_j$, the last condition for $V((x_1,x_2),(a_1,a_2),e,(e_1,e_2,z_1,z_2))$ to hold is satisfied. So, Σ satisfies the completeness property of Σ-protocols.

Q.3 Construct an extractor $\mathcal{E}$ for Σ and show that is works, in polynomial time.

If $V((x_1,x_2),(a_1,a_2),e,(e_1,e_2,z_1,z_2))$ and $V((x_1,x_2),(a_1,a_2),e',(e'_1,e'_2,z'_1,z'_2))$ hold with $e \neq e'$, we must have either $e_1 \neq e'_1$ or $e_2 \neq e'_2$. Let assume that $e_1 \neq e'_1$. Then, we know that $V_1(x_1,a_1,e_1,z_1)$ and $V_1(x_1,a_1,e'_1,z'_1)$ hold. So, we can run the $\mathcal{E}_1$ extractor on $(x_1,a_1,e_1,e'_1,z_1,z'_1)$ to extract a witness w for x_1 in L_1. Clearly, w is also a witness for (x_1,x_2) in L. The method is similar in the case $e_2 \neq e'_2$.
Clearly, we obtain a polynomially bounded extractor.

Q.4 Construct a simulator $\mathcal{S}$ for Σ and show that is works, in polynomial time.

Given (x_1,x_2) and e, we pick a random e_1 and let $e_2 = e - e_1$. Then, we run $\mathcal{S}_1(x_1,e_1) \to (a_1,e_1,z_1)$ and $\mathcal{S}_2(x_2,e_2) \to (a_2,e_2,z_2)$. The output is $((a_1,a_2),e,(e_1,e_2,z_1,z_2))$. This defines our simulator $\mathcal{S}$.
Clearly, this works in polynomial time.
We let $a = (a_1,a_2)$ and $z = (e_1,e_2,z_1,z_2)$. We have

$$\Pr[\mathcal{S} \to a,e,z|e] = \sum_{e_1+e_2=e} \Pr[e_1]\Pr[\mathcal{S}_1 \to a_1,e_1,z_1|e_1]\Pr[\mathcal{S}_2 \to a_2,e_2,z_2|e_2]$$

Since $\mathcal{S}_1$ and $\mathcal{S}_2$ are simulators for Σ_1 and Σ_2, we have

$$\Pr[\mathcal{S} \to a,e,z|e] = \sum_{e_1+e_2=e} \Pr[e_j]\Pr[\Sigma_j \to a_j,e_j,z_j|e_j]\Pr[\mathcal{S}_i \to a_i,e_i,z_i|e_i]$$

for whatever pair (i,j) such that $\{i,j\} = \{1,2\}$. We let i be random defined by $\mathcal{P}$. Clearly, the above sum equals $\Pr[\Sigma \to a,e,z|e]$. So, $\mathcal{S}$ satisfies the property of a simulator for Σ.

III Smashing SQUASH-0

The exercise is inspired by Smashing SQUASH-0 *by Ouafi and Vaudenay. Published in the proceedings of Eurocrypt'09 pp. 300–312, LNCS vol. 5479, Springer 2009.*

We consider an access control protocol called SQUASH-0 in which a client and a server hold a secret key K. In the protocol, the server sends a challenge C. The client must respond with

$$S = (\mathsf{stoi}(C \oplus K))^2 \bmod N$$

for a given modulus N, where stoi is a function transforming a bitstring into an integer by $\mathsf{stoi}(\varepsilon) = 0$ for the zero-length bitstring ε, and

$$\mathsf{stoi}(b \| s) = b + 2 \times \mathsf{stoi}(s)$$

for any bit $b \in \{0,1\}$ and any bitstring s. By convention, the least significant bit has position 0. We further assume that N is larger than K and C.

Q.1 Let c_i be -1 raised to the power of the bit position i in C. Let k_i be -1 raised to the power of the bit position i in K.
Show that

$$S = \left(\frac{1}{4} \sum_{i,j} 2^{i+j} c_i c_j k_i k_j - \frac{2^\ell - 1}{2} \sum_i 2^i c_i k_i + \frac{(2^\ell - 1)^2}{4} \right) \bmod N$$

where ℓ is the bitlength of N.

The XOR of two bits in the ± 1 representation is obtained by a regular multiplication. The ± 1 representation of bits can be converted to a 0-1 representation by $x \mapsto \frac{1-x}{2}$. So,

$$\mathsf{stoi}(C \oplus K) = \sum_i 2^i \frac{1 - c_i k_i}{2} = \frac{2^\ell - 1}{2} - \frac{1}{2} \sum_i 2^i c_i k_i$$

By squaring it we obtain the result for S.
The SQUASH-0 proposal suggests to use Mersenne numbers for N. incidentally, we obtain $2^\ell - 1 = N$. We deduce

$$S = \left(\frac{1}{4} \sum_{i,j} 2^{i+j} c_i c_j k_i k_j \right) \bmod N$$

In what follows, we assume that $N = 2^\ell - 1$. Deduce

$$S = \left(\frac{1}{4} \sum_{i,j} 2^{i+j} c_i c_j k_i k_j \right) \bmod N$$

Q.2 Deduce that by using about ℓ^2 challenges and their responses, an adversary could recover K by solving a linear system of $O(\ell^2)$ equations with $\frac{\ell(\ell-1)}{2}$ unknowns.
As an example, consider $\ell = 1\,024$. What is the complexity of the attack?
Hint: define $\kappa_{i,j} = k_i k_j$.

We let $\kappa_{i,j} = k_i k_j$ for $i < j$. For $i = j$, we have $k_i k_j = 1$. For $i > j$, we have $k_i k_j = \kappa_{j,i}$. So, all $k_i k_j$ can be expressed in terms of κ's. This way, the equation becomes linear. We have $\frac{(\ell(\ell-1)}{2}$ unknowns κ. So, by collecting enough equations (namely, about ℓ^2), we can solve the linear system. The complexity of such algorithm is essentially $O(\ell^6)$. For $\ell = 2^{10}$, we need 2^{20} known challenges and we reach a complexity of 2^{60}, which is not practical.

Q.3 Given a function φ mapping a bitstring of length d to a real number, we define

$$\hat{\varphi}(V) = \sum_x (-1)^{x \cdot V} \varphi(x)$$

where $\cdot$ denotes the dot product between two bitstrings and the sum goes on all bitstrings x of length d. For the function $\varphi(x) = (-1)^{x \cdot U}$, show that $\hat{\varphi}(V) = 2^d$ if $V = U$ and $\hat{\varphi}(V) = 0$ otherwise. We write it $\hat{\varphi}(V) = 2^d 1_{V=U}$.

We have

$$\hat{\varphi}(V) = \sum_x (-1)^{x \cdot (U \oplus V)}$$

When $U \oplus V \neq 0$, this is zero. When $U = V$, this is clearly 2^d.

Q.4 In a chosen challenge attack, an adversary creates d challenges $C^1, \ldots, C^d$ and all linear combinations of these challenges. Namely, $C(x_1 \ldots x_d) = x_1 C^1 \oplus \cdots \oplus x_d C^d$. Given a d-bit vector x, we thus define $C(x)$. We write x as an argument of S and c_i as well so that $S(x)$ is the response to challenge $C(x)$ and $c_i(x)$ is -1 raised to the power of the bit position i in $C(x)$. Let U_i be the d-bit vector consisting of the bit at position i of $C^1, \ldots, C^d$.
Deduce that

$$\hat{S}(V) = \frac{1}{4} \sum_{i,j} 2^{d+i+j} k_i k_j 1_{V=U_i \oplus U_j}$$

Hint: observe $c_i(x) = (-1)^{x \cdot U_i}$ and use Q.1 then Q.3.

The bit at position i of $C(x)$ is clearly $x \cdot U_i$. So,

$$c_i(x) = (-1)^{x \cdot U_i}$$

We now use Q.1. By the definition of $\hat{S}$, we have

$$\hat{S}(V) = \sum_x (-1)^{x \cdot V} \left(\frac{1}{4} \sum_{i,j} 2^{i+j} c_i(x) c_j(x) k_i k_j \right) \bmod N$$

We can now use our observation and permute the two sums and obtain

$$\hat{S}(V) = \frac{1}{4} \sum_{i,j} 2^{i+j} k_i k_j \sum_x (-1)^{x \cdot (V \oplus U_i \oplus U_j)}$$

We can then use Q.3.

Q.5 With the same notations, we assume that the function mapping a non-ordered pair $\{i, j\}$ with $i \neq j$ to $U_i \oplus U_j$ behaves like a random function. We further assume that d is pretty small. For each V, estimate the number of non-ordered pairs $\{i, j\}$ with $i \neq j$ such that $V = U_i \oplus U_j$.
Deduce that we get 2^d equations modulo N with $\ell(\ell-1)2^{-d-1}$ unknowns $\kappa_{i,j}$ on average taking values in $\{-1, +1\}$.

> *We have $\frac{\ell(\ell-1)}{2}$ non-ordered pairs $\{i, j\}$ with $i \neq j$. The vector $U_i \oplus U_j$ takes values in a set of 2^d elements. So, each V has (on average) $\ell(\ell-1)2^{-d-1}$ pairs. Therefore, each equation $\hat{S}(V)$ uses this amount of unknowns $\kappa_{i,j} = k_i k_j$.*

Q.6 We take $d = 2\log_2 \ell$ and solve each equation by exhaustive search. Deduce a chosen-challenge attack to break the algorithm.
How many chosen challenges does it use, asymptotically?
What is its complexity?

> *With $d = 2\log_2 \ell$, each equation has $\frac{1}{2}$ unknown on average. So, exhaustive search works in constant time. We just solve $O(\ell^2)$ equations using $O(\ell^2)$ chosen challenges.*
>
> 1: *pick $C^1, \ldots, C^d$*
> 2: *for each x, define $C(x)$ and get $S(x)$*
> 3: *do an FFT transform on S to get the table $\hat{S}$*
> 4: *for each V, make an exhaustive search on the expressed $\kappa_{i,j} = \pm 1$ in $\hat{S}(V)$ to recover the κ's*
> 5: *pick k_1 at random and infer k_i from $\kappa_{1,i}$*
>
> *The FFT complexity is $O(d2^d)$. So, the overall complexity is $O(\ell^2 \log \ell)$. This is much better than $O(\ell^6)$.*

IV PIF Implies PAF

We consider a function family F_k taking inputs of length λ, making outputs of length λ, and where the key k is also of length λ. We consider the two following games:

Game $\mathsf{PIF}(\mathcal{A}, 1^\lambda)$:

pick some random coins k of length λ
pick ρ
run $\mathcal{A}(\rho) \to x$
if $|x| \neq \lambda$, output 0 and stop
pick a random bit b
if $b = 0$ **then**
  compute $y = F_k(x)$
else
  pick a random y of λ bits
end if
run $\mathcal{A}(y; \rho) \to b'$
output $b \oplus b' \oplus 1$

Game $\mathsf{PAF}(\mathcal{A}, 1^\lambda)$:

pick some random coins k of length λ
pick ρ
pick a random x of length λ
compute $y = F_k(x)$
run $\mathcal{A}(y; \rho) \to x'$
output $1_{x=x'}$

We say that F_k is PIF-secure (resp. PAF-secure) if for all polynomially bounded $\mathcal{A}$, we have that $\Pr[\mathsf{PIF}(\mathcal{A}, 1^\lambda) = 1] - \frac{1}{2}$ (resp. $\Pr[\mathsf{PAF}(\mathcal{A}, 1^\lambda) = 1]$) is a negligible function in terms of λ.

Q. Show that if F_k is PIF-secure, then it is PAF-secure.

Hint: based on a PAF-adversary $\mathcal{A}$ and some coins $\rho' = r' \| \rho \| b''$, define $\mathcal{A}'(\rho') = x$ picked at random from r' then $\mathcal{A}'(y, \rho') = 1$ if $\mathcal{A}(y; \rho) = x$ and $\mathcal{A}'(y, \rho') = b''$ otherwise. By considering $\mathcal{A}'$ as a PIF-adversary, look at the link between $\Pr[\mathsf{PIF}(\mathcal{A}', 1^\lambda) = 1] - \frac{1}{2}$ and $\Pr[\mathsf{PAF}(\mathcal{A}, 1^\lambda) = 1]$.

Consider an adversary $\mathcal{A}$ who is polynomially bounded. We want to show that $p = \Pr[\mathsf{PAF}(\mathcal{A}, 1^\lambda) = 1]$ is negligible.
For this, we define the adversary $\mathcal{A}'$ as follows: we let $\rho' = r' \| \rho \| b''$ and $\mathcal{A}'(\rho')$ picks a random x using r'. Then, $\mathcal{A}'(y; \rho')$ runs $\mathcal{A}(y; \rho) = x''$. If $x = x''$, it answers 1. Otherwise, it answers by b''.
When running the game $\mathsf{PIF}(\mathcal{A}', 1^\lambda)$, in the $b = 0$ case, we have $x = x''$ with probability p and $\mathcal{A}'$ never answers 0. We have $x \neq x''$ with probability $1 - p$ and $\mathcal{A}'$ answers 0 with probability $\frac{1}{2}$. So, $\mathcal{A}'$ answers 0 with probability $\frac{1-p}{2}$. So,

$$\Pr[\mathsf{PIF}(\mathcal{A}', 1^\lambda) = 1 | b = 0] = \frac{1-p}{2}$$

When $b = 1$, $\mathcal{A}(y; \rho)$ has no information about x, so x is independent from x'' and we have $\Pr[x = x''] = 2^{-\lambda}$. Thus,

$$\Pr[\mathsf{PIF}(\mathcal{A}', 1^\lambda) = 1 | b = 1] = 2^{-\lambda} + \frac{1 - 2^{-\lambda}}{2}$$

Finally, we have

$$\begin{aligned} \Pr[\mathsf{PIF}(\mathcal{A}', 1^\lambda) = 1] - \frac{1}{2} &= \frac{1}{2}\left(\frac{1-p}{2} + 2^{-\lambda} + \frac{1-2^{-\lambda}}{2}\right) - \frac{1}{2} \\ &= -\frac{p}{4} + \frac{2^{-\lambda}}{4} \end{aligned}$$

Since F_k is PIF-secure, we know that $\Pr[\mathsf{PIF}(\mathcal{A}', 1^\lambda) = 1] - \frac{1}{2}$ must be negligible. Thus, $-\frac{p}{4} + \frac{2^{-\lambda}}{4}$ is negligible. Since $\frac{2^{-\lambda}}{4}$ is negligible, we obtain that $\frac{p}{4}$ is negligible. So, p is negligible.