Sécurité des Systèmes d'Information
Mandatory TP 3 - ZKIPs and Key Generation

27 Novembre 2019

Submit on **Moodle**, with both your Python 3 files **.py**, and your report in **PDF**, before **December 18, 2019 at 4pm (16h00)**.

Your code needs to be **commented**.

Reminder : The Feige-Fiat-Shamir protocol

The Feige-Fiat-Shamir protocol is a Zero Knowledge Proof (ZKIP) :

- **Generating secret and keys** : We ask a trusted third party T to generate two big prime numbers randomly, p and q, and then computes $n = pq$. Only the n is shared publicly. User A then generates a set of secrets $s_1, s_2, ..., s_k \in \mathbb{Z}_n^*$, and the corresponding public set $v_1, v_2, ..., v_k \in \mathbb{Z}_n^*$, with $v_i = s_i^2 \mod n$.
- **Proof of knowledge** :

1	$A \to B$:	$x = r^2 \mod n$;	A chooses a random r, computes x, and sends x to B.
2	$B \to A$:	$e_1, e_2, ..., e_k \in \{0, 1\}$;	B sends the challenges e_i.
3	$A \to B$:	$y = r \prod_{i=1}^{k} s_i^{e_i} \mod n$;	A answers with y.

- B accepts the proof if $y \neq 0$ et $y^2 = x \prod_{i=1}^{k} v_i^{e_i} \mod n$.

Reminder : Graph Isomorphism ZKIP

Two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ are isomorph if a permutation $\pi : V_1 \to V_2$ exists, such that $(u, v) \in E_1 \Leftrightarrow (\pi(u), \pi(v)) \in E_2$.

The ZKIP goes as follows :

- **Secret generation** : A generates a big (approx 1000 nodes) random graph G_1, then a permutation π, and creates G_2 by applying π to G_1. Then both graphs G_1 and G_2 are send publicly, and π is A's secret.

- **Proof of Knowledge** :

  1. A chooses a random permutation $\sigma : G_2 \rightarrow H$ (the permutation, applied to G_2, creates H). A then sends the graph H to B.
  2. B chooses a challenge $i \in \{1, 2\}$ and sends it to A.
  3. A computes λ such that $H = \lambda(G_i)$. Which means if $i = 2$, $\lambda = \sigma$, and if $i = 1$, $\lambda = \sigma \circ \pi$.
  4. B checks if $H = \lambda(G_i)$. If it is, the step is validated.
  5. Repeat steps 1 to 4 enough times to assure good soundness.

A combo ZKIP-KAP : Zero Knowledge Interactive Proof - Key Agreement Protocol

Now, we will modify the previous ZKIP with graphs, to allow A and B to mutually authenticate, and simultaneously create a shared symmetric key for encrypted communication between A and B, with a Key Agreement Protocol (KAP).

The idea of this KAP is :

- Create a bigger (around 2000 nodes) graph, which will be used as G_1. This G_1 will be used by both A and B to identify themselves (i.e. $G_1^A = G_1^B = G_1$).
- Let A create a permutation α which will modify only the order of the first half (the first 1000 nodes for a 2000 nodes graph) of the nodes, and the corresponding graph G_2^A. It is obvious that this will allow the ZKIP to function the same way and be as effective as defined before, except a little slower because the graph is bigger (but almost half of the graph is not modified).
- Now let B create a permutation β affecting only the second half of the nodes (the last 1000 nodes), and the corresponding graph G_2^B.
- Now let A and B apply the ZKIP twice, once for A to identify to B, and the second time for B to identify to A.
- Now, the shared key is $G_{Key} = \beta(G_2^A) = \alpha(G_2^B)$.

Now, we add another step to make the key more user friendly. Let us define $V_A = \{v_1, v_2, ...v_n\}$ and $V_B = \{v_{n+1}, ...v_w\}$ (w is the total number of nodes, so $w = 2n$ if the total number is even, $w = 2n - 1$ if that number is odd). With each pair (x, y) with $x \in V_A, y \in V_B$, let's define the one bit $b_{(x,y)} = 1$ if $(x, y) \in E_{G_{Key}}$ (i.e. if this edge exists in G_{Key}), or $b_{(x,y)} = 0$ if$(x, y) \notin E_{G_{Key}}$.

We can define the following vector of bits, which is just the one bit defined earlier for each such edge (The big point denotes the concatenation) :

$$k = \bullet_{i=1}^{n} \bullet_{j=n+1}^{2n} b_{(i,j)}$$

And finally, to create the fixed-length key (let's say that length is equal to 1024), just xor the first 1024 bits of k with the next 1024 bits, and xor that result with the next 1024, and etc... until you reach the end of k (add a padding of zeros at the end of k if needed to make it a multiple of 1024) :

$$KEY = (...((k_{1-1024} \oplus k_{1025-2048}) \oplus k_{2049-3072})... \oplus k_{(end-1023)-end})$$

That's a better looking key !

Exercice 1 : Feige-Fiat-Shamir

- *Theory :*
  1. Define the following properties : *Completeness*, *Soundness*, *Proof of Knowledge*, and *Zero-Knowledge Proof.*
  2. Show that the Feige-Fiat-Shamir protocol is a *Proof of Knowledge.*
  3. Show that it is also a *Zero-Knowledge Proof.*
  4. How many times should we repeat this process to ensure a good probability of *Soundness* ?
- *Implementation :* Implement the Feige-Fiat-Shamir protocol with Python 3.

Exercice 2 : Graph Isomorphism ZKIP

Implementation : You need to implement the described ZKIP based on graph isomorphism. You have to :

1. Generate big random graphs, around 1000 nodes (for the edges, just define a probability p, and generate each possible edge with probability p),
2. Generate a random permutation on the graph,
3. Create a function applying such permutations on a graph,
4. Clearly separate A and B's roles (use comments, and meaningful names for your functions and variabls),
5. Repeat the process enough times to have a good soundness.

Exercice 3 : The Graph Isomorphism ZKIP-KAP combo

- *Implementation* : Use what you did in exercice 2 to implement this new version of the protocol, with the bigger graphs, key generation, and shortening of the graph key into a 1024 bit key.
- *Theory* : You're receiving a call from Alice and Bob, and they want to discuss your protocol. They have doubts about what you modified in this protocol, and they're not sure it is secure.
  1. Explain what specificity of permutations is forcing us to have a bigger graph and modify the protocol, and why A and B both generate the same key.
  2. Explain why an attacker, knowing both G_2^A and G_2^B, still can't reconstruct G_{Key}. Which part of the key graph is only known by A and B ?
  3. Is the proposed key modification method secure ? Explain why.
  4. What tool should we use to create a better key from G_{Key} ?