Applied Cryptography

Public Key Cryptography, Assignment 6, Tuesday, November 26, 2024

Remarks:

- Hand in your answers through Brightspace.
- Hand in format: PDF. Either hand-written and scanned in PDF, or typeset and converted to PDF. Please, **do not** submit photos, Word files, LaTeX source files, or similar.
- Assure that the name of **each** group member is **in** the document (not just in the file name).

Deadline: Sunday, December 8, 23.59

Goals: After completing these exercises you should have an understanding of zero knowledge proofs, hash-based signatures, Merkle trees, and the ECDSA

1. (**15 points**) For this exercise, we will write $G = (V, E)$ if G is a graph with set of vertices V, and set of edges E. An isomorphism $\phi : G_0 \to G_1$ between graphs $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$ is a bijection $\phi : V_0 \to V_1$ such that $\{v, w\} \mapsto \{\phi(v), \phi(w)\}$ describes a bijection $E_0 \to E_1$. Any isomorphism $\phi : E_0 \to E_1$ has an inverse isomorphism $\phi^{-1} : G_1 \to G_0$.

   The graph isomorphism problem is the problem of determining if two graphs are isomorphic. There is currently no known way to solve this problem in polynomial time. (Although there is a quasi-polynomial time algorithm by László Babai[1].)

   (a) Given a large, unstructured graph G_0, show that we can easily find a pair (G_1, ϕ), where G_1 is a graph such that $G_1 \neq G_0$, and $\phi : G_0 \to G_1$ is an isomorphism.

   Within the lecture, the following zero knowledge proof was shown, where the prover $\mathcal{P}$ seeks to prove to the verifier $\mathcal{V}$ that they know a graph isomorphism $\phi : G_0 \to G_1$.

   - The prover $\mathcal{P}$ generates an isomorphism $\psi : G_1 \to G_2$ for a graph $G_2 \notin \{G_0, G_1\}$, and commits to G_2 by sending it to $\mathcal{V}$.
   - The verifier $\mathcal{V}$ randomly chooses a challenge $b \leftarrow \{0, 1\}$, and sends it to $\mathcal{P}$.
   - The verifier $\mathcal{V}$ accepts $\mathcal{P}$ if it returns an isomorphism $G_b \to G_2$.

   (b) Explain how a prover $\mathcal{P}$ which knows $\phi : G_0 \to G_1$ can succeed in the final step.

   (c) Proof rigorously why this protocol has a soundness error of 1/2.

   (d) To achieve a protocol without soundness error, the prover could simply show both the isomorphism $G_0 \to G_2$ and the isomorphism $G_1 \to G_2$. What security property would we lose by doing this?

   For $n \geq 2$, consider the following variant of the zero-knowledge protocol:

   - The prover $\mathcal{P}$ generates a sequence of isomorphisms $G_1 \to G_2 \to \cdots \to G_{n-1} \to G_n$, and commits to $G_2, \ldots, G_n$ by sending them to $\mathcal{V}$.
   - The verifier $\mathcal{V}$ randomly chooses a challenge $b \leftarrow \{1, \ldots, n\}$, and sends it to $\mathcal{P}$.
   - The verifier $\mathcal{V}$ accepts $\mathcal{P}$ if it can return an isomorphism $G_i \to G_{i+1 \bmod n+1}$ for every $i \in \{1, \ldots, n\} \backslash \{b\}$.

   (e) What is the soundness error of this protocol? Formally prove your claim.

   (f) How does the answer to (e) compare to the soundness error of $n - 1$ repeats of the first protocol?

[1] Babai's paper on this can be found here: https://dl.acm.org/doi/10.1145/2897518.2897542

A Hamiltonian cycle through a graph is a cycle C (a path which starts and ends in the same vertex) which visits each vertex of the graph exactly once. Finding such a cycle is an NP-hard problem.

(g) Argue that, if we have a Hamiltonian cycle through a graph G_1, and an isomorphism $\phi : G_1 \to G_2$, we can find a Hamiltonian cycle through G_2.

(h) Use (g) to design a zero-knowledge proof protocol where the prover $\mathcal{P}$ shows they know a secret Hamiltonian cycle C through a public graph G, with a soundness error of 1/2. Proof the security properties of your protocol. (Hint: try to think of two challenges related to (g) based on two different hard problems.)

2. **(15 points)** Consider the following generalization of one-time signatures:

- **Key generation:**
  - Generate a pair of secret and public key $(\mathsf{sk}_1, \mathsf{pk}_1)$ using Lamport's OTS for 256-bit messages using SHA-256 as the internal hash function H_0;
  - Set a state to $S = ()$ and $\sigma_0 = ()$.
- **Signing:** To sign messages M_i, $i = 1, 2, \ldots$ the signer operates as follows:
  - Generate a new key pair $(\mathsf{sk}_{i+1}, \mathsf{pk}_{i+1})$;
  - Compute $s_{i,\mathrm{OTS}} = \mathsf{Sign}_{\mathsf{sk}_i}(H_1(M_i, \mathsf{pk}_{i+1}))$ where $\mathsf{Sign}_{\mathsf{sk}_i}$ is the signing algorithm of Lamport OTS using the secret key sk_i, and H_1 some hash function;
  - Construct the signature $\sigma_i = (M_i, \mathsf{pk}_{i+1}, s_{i,\mathrm{OTS}}, \sigma_{i-1})$;
  - Add $(M_i, \mathsf{pk}_{i+1}, \mathsf{sk}_{i+1}, s_{i,\mathrm{OTS}})$ to the state S.
- **Verification:** To verify the signature $\sigma_i = (M_i, \mathsf{pk}_{i+1}, s_{i,\mathrm{OTS}}, \sigma_{i-1})$:
  - Check $\mathsf{Vf}_{\mathsf{pk}_j}(M_j, \mathsf{pk}_{j+1}, s_{j,\mathrm{OTS}}) = 1$ for all $j \in \{1, 2, \ldots, i\}$.

(a) What should the length of the output of the hash function H_1 be?

(b) Use recursion to compute the length of the signature of the 12-th message. Denote the length of the i-th message by L_{M_i}.

(c) What is the advantage of this scheme compared to MSS introduced in the lectures? What is the disadvantage?

(d) Show that a forgery is possible if instead of $\mathsf{Sign}_{\mathsf{sk}_i}(H_1(M_i, \mathsf{pk}_{i+1}))$ the signer includes $\mathsf{Sign}_{\mathsf{sk}_i}(H_1(M_i))$.

(e) Show that a forgery is possible, if in the signature generation of σ_i, we omit σ_{i-1}, and in the verification process we set $j \in \{i\}$, and provide the OTS public key pk_i together with the signature.

(f) Show that a forgery is possible if the adversary is able to find second preimages for H_1.

3. **(20 points)** Recall from Introduction to Cryptography the basics of elliptic curve cryptography: An elliptic curve (in Weierstrass form) is a set of the form

$$E(\mathbb{F}_p) = \{(x, y) \in \mathbb{F}_p^2 \mid y^2 = x^3 + Ax + B\} \sqcup \{\mathcal{O}\}$$

over some finite field $\mathbb{F}_p$, where $A, B \in \mathbb{F}_p$ are *curve parameters*. The extra point $\mathcal{O}$ is called the *point at infinity*. This $E(\mathbb{F}_p)$ forms a group, so we can add two points $P_1, P_2 \in E(\mathbb{F}_p)$ together to get a third point $P_3 = P_1 + P_2 \in E(\mathbb{F}_p)$. The neutral element of this group is $\mathcal{O}$, so $P + \mathcal{O} = P$ for any $P \in E(\mathbb{F}_p)$. For a point $P \in E(\mathbb{F}_p)$ and an $n \in \mathbb{N}$, we write $[n]P$ for the sum of n copies of P.

$$[n]P = \underbrace{P + P + \ldots + P}_{n \text{ times}}$$

We say P has order n if n is the lowest positive integer such that $[n]P = \mathcal{O}$.

For this exercise, we will use the curve `Curve22103` from the Relic toolkit. Use the following code in `Sage` to obtain this curve, and a point G on the curve:

```
p = 2^221 - 3
F = FiniteField(p)
A = 2246662222262553316222251257251635889469762035211714373411374400767
B = 2121847654359078131987681742959878340054775255477730874138797701479
E = EllipticCurve(F, [A,B])

Gx = 1337986418373986027481331603085719890755810485220461707389578529200
Gy = 357140777035609340439344712909994219660040722634117794161168213942
G = E(Gx,Gy)
```

(a) Use `Sage`'s built-in commands to find the size of `Curve22103`, and the order of G.

(b) The group structure of `Curve22103` can be given as

$$E(\mathbb{F}_p) \cong \mathbb{Z}_q \times \mathbb{Z}_c,$$

where q is a large prime number, while c is a smaller cofactor. Based on your answer to (a), what are the values of c and q?

Now, we will consider the Elliptic Curve Digital Signature Algorithm, or ECDSA, which was discussed in the final slides of the last lecture. It can be given using the following diagram:

KeyGen:

(a) Let G be a base point of $E(\mathbb{F}_p)$ of order q, where p is an odd prime or a power of 2

(b) Choose a random $d \in \mathbb{Z}_q^*$ and compute $Q = dG$

(c) Output public key $\mathsf{pk} = Q$ and private key $\mathsf{sk} = d$

Sign: Given message M,

(a) $k \xleftarrow{\$} \mathbb{Z}_q, (x_1, y_1) \leftarrow kG$, and $r \leftarrow x_1 \pmod{q}$

(b) Set $h = H(M)$ and calculate $s \leftarrow (h + dr)k^{-1}$

(c) Set $\sigma = (r, s)$ and output message-signature pair (M, σ)

Verify: To verify the message-signature pair (M, σ)

(a) Parse $\sigma = (r, s)$, verify $0 < r, s < q$ and calculate $h = H(M)$

(b) Compute $u_1 = hs^{-1} \pmod{q}$ and $u_2 = rs^{-1} \pmod{q}$

(c) Compute $X = u_1 G + u_2 Q$

(d) If X is point of infinity, output Fail

(e) Take $X = (x_1, y_1)$, check $r \overset{?}{=} x_1 \pmod{q}$ and output Accept if check succeeds, otherwise Fail

We will use the curve `Curve22103` and the point G specified in the `Sage` code earlier for this algorithm.

(c) Show the correctness of ECDSA, i.e., that indeed $r = x_1 \mod q$ for valid signatures.

(d) Suppose we choose the secret key
`154594568703641690998489183597663211003678858273761150657308524442`
What is the corresponding public key?

(e) For the keys from from (d), give a signature (r, s) for a message with hash $h(M) = 8$. The following code in sage might help:

```
Fq = FiniteField(q)
k = Fq.random_element()
K = int(k)*G //do you see why int() is necessary?
r = mod(K[0], q)
```

(f) Give an element $T \in E(\mathbb{F}_p)\backslash\{\mathcal{O}\}$ such that $[c]T = \mathcal{O}$, where c is from exercise (b). Explain how you found it. (Hint: you can find random elements of the curve with `E.random_element()`.)

(g) For a public key Q, let $\sigma = (r, s)$ be a valid signature for a message M, and suppose that $u_2 = rs^{-1} (\text{mod } q) \in \{0, \ldots, q-1\}$ is divisible by c, interpreting u_2 and c as integers. Let T be as described in (f). Show how T can be used to create other public keys which accept (M, σ).

`Curve25519` is a standardised curve, which is commonly used in practice. Due to its similar structure to `Curve22103`, the trick from (g) also works for `Curve25519`. The cryptocurrency *Monero* uses ECDSA on `Curve25519` to create signatures and to sign transactions. In short, users spend their coins by creating a valid signature, and it prevents double spending of a coin by checking if the associated public key has already been used to spend that coin.

(h) Explain how an attacker can spend his coins multiple times by using (g). (This attack was only prevented in May 2017, and so it was possible perform this attack for more than three years.)