ZKIAP 2023: Math Building Blocks

HOMEWORK EXERCISES

Given integers x, m, write $QR(m,x) = 1$ if x is a quadratic residue mod m, i.e., there exists an integer s such that $s^2 \equiv x \pmod{m}$; write $QR(m,x) = 0$ if x is not a quadratic residue mod m.

Quadratic nonresidue

ASSUMPTION: The Prover can compute $QR(m,x)$ for all m,x

COMMON INPUT: positive integers m,x

GOAL: the Prover wants to convince the Verifer that $QR(m,x) = 0$

PROTOCOL:

1. The Verifier picks a random $s \in \mathbb{Z}_m$ uniformly among elements relatively prime to m, and also tosses a coin $b \leftarrow_R \{0,1\}$. Set

$$y \leftarrow \begin{cases} s^2 x & \text{if } b = 0, \\ s^2 & \text{if } b = 1. \end{cases}$$

   The Verifier sends y to the Prover and challenges the Prover to determine b.

2. The Prover computes $QR(m,y)$ and sends its value back to the Verifier

3. The Verifier accepts if the value it received from the Prover is indeed equal to b; otherwise it rejects.

You are asked to check:

(a) **Completeness**: if $QR(m,x) = 0$ and both parties behave according to the protocol, then the Verifier always accepts.

(b) **Soundness**: if $QR(m,x) = 1$, then no matter what the Prover does (the Prover does not have to follow the protocol), the Verifer rejects with probability $\geq 1/2$.

Quadratic residue

COMMON INPUT: positive integers m,x with $\gcd(m,x) = 1$

PRIVATE INPUT: the Prover knows a secret integer s such that $s^2 \equiv x \mod m$

GOAL: the Prover wants to convince the Verifier that $QR(m,x) = 1$ without revealing any information about s

(Note that the Prover can easily convince the Verifier that $QR(m,x) = 1$ by sending s to the Verifier, but this would reveal s completely.)

PROTOCOL:

1. The Prover picks a random $t \in \mathbb{Z}_m$ uniform among residues relatively prime to m. The Prover sends $y \leftarrow xt^2 \mod m$ to the Verifier.
2. The Verifier flips a coin $b \leftarrow_R \{0,1\}$ and sends b to the Prover.
3. The Prover receives b and sends to the Verifier the value of
$$u \leftarrow \begin{cases} t & \text{if } b = 0, \\ st & \text{if } b = 1. \end{cases}$$
4. The Verifier accepts if
$$y \equiv \begin{cases} u^2 x \mod m & \text{if } b = 0, \\ u^2 \mod m & \text{if } b = 1, \end{cases}$$
and rejects otherwise.

You are asked to check:

(a) **Completeness**: If both parties behaves according to the protocol, then the Verifier always accepts.

(b) **Soundness**: If $QR(m,x) = 0$ (in particular, the Prover does not know any valid s), then no matter what the Prover does, the Verifer rejects with probability $\geq 1/2$.

(b*) **Knowledge soundness**: For any Prover algorithm that leads the Verifier to accept, if the Verifier is allowed to simultaneously query both values of $b \in \{0,1\}$ (for the same t), then the Verifer can extract the secret s from the Prover. (The point here is that the Prover cannot convince the Verifier unless the Prover "knows" s.)

(c) **Zero knowledge**: No matter what the Verifier does (the Verifier could behave differently from the Protocol), the Verifier could have simulated the entire interaction on its own without interacting with the Prover and such that if the Verifier accepts, then the transcript is indistinguishable from the actual interaction.

Hint: imagine that the simulator generates a random u on its own instead of getting u from the Prover.

(This part is a little tricky, since the adversarial Verifier is allowed to behave differently from the protocol — remember what we are trying to show is that no matter what the Verifier does, it learns no information about s from the Prover that the Verifier could not have simply generated on its own. See the proof of Theorem 13.6 in Barak's notes if you want to see the solution.)

Self-pairing implies failure of DDH

Let $\mathbb{G}$ and $\mathbb{G}_T$ be abelian groups (written additively) of the same prime order q. Let $g \in \mathbb{G}$ be a generator. Suppose that we have an efficiently computable nondegenerate bilinear pairing

$$e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T.$$

Give an efficient algorithm for deciding, given $\alpha g, \beta g, y \in \mathbb{G}$ (with unknown $\alpha, \beta \in \mathbb{Z}_q$), whether $\alpha\beta g = y$.

Hint: compute the pairing against g.

This exercise shows that the Decisional Diffie–Hellman (DDH) assumption is false for groups with self-pairing.

BLS signature aggregation

SETUP:

- Pairing $e\colon \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$, where $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T$ are abelian groups (written additively) of prime order p, with generators g_0, g_1, g_T respectively
- Hash function H taking values in $\mathbb{G}_1$

The BLS signature scheme is defined as:

- **KeyGen**(): choose random (secret key) $sk = \alpha \leftarrow_R \mathbb{Z}_q$ and set (public key) $pk \leftarrow \alpha g_0 \in \mathbb{G}_0$.
- **Sign**(sk, m): output $\sigma \leftarrow \alpha H(m) \in \mathbb{G}_1$ as the signature on the message m
- **Verify**(pk, m, σ): accept if $e(g_0, \sigma) = e(pk, H(m))$.

Check that the verification algorithm always accepts a correctly signed signature. Also argue that it is computational infeasible to come up with a forged signature σ for any message m (the forger is free to chose m) if the forger is given pk but not sk. This is called *existentially unforgeable under a chosen message attack*. Can you specify some computational hardness assumptions?

Signature aggregation. Given triples (pk_i, m_i, σ_i) for $i = 1, \ldots, n$ (coming from n users), we can aggregate these n signatures $\sigma_1, \ldots, \sigma_n$ into a single signature by simply taking their sum in $\mathbb{G}_1$:

$$\sigma \leftarrow \sigma_1 + \cdots \sigma_n \in \mathbb{G}_1.$$

Given $(pk_1, m_1), \ldots, (pk_n, m_n)$ and the aggregate signature σ, show that we can verify that indeed all n users have signed their messages by checking that

$$e(g_0, \sigma) = e(pk_1, H(m_1)) + \cdots + e(pk_n, H(m_n)).$$

NOTE. The above signature aggregation scheme is not secure due to a *rogue public key attack*. One way to make the scheme secure is to ensure that all messages m_i are distinct, for example, by requiring that each m_i starts with the user's public key pk_i.

For more on BLS signature aggregation, see https://crypto.stanford.edu/~dabo/pubs/papers/BLSmultisig.html