Lattice-based signatures

Notes by Henry Corrigan-Gibbs

MIT - 6.5610
Lecture 10 (March 5, 2025)

> ***Warning:*** This document is a rough draft, so it may contain bugs. Please feel free to email me with corrections.

Outline

- Reminder: Learning-with-errors assumption
- "Proving knowledge" of an LWE secret in zero knowledge
- Digital signatures
  - Definition
  - Construction from ZK proof of knowledge
- Implementation issues

Plan for this lecture

One of the goals of this course is to teach you the fundamentals of lattice-based cryptography. As we have discussed, these tools are almost certain to displace classical cryptosystems (Diffie-Hellman, RSA, ElGamal, etc.) in the near future.

The two most widely used public-key primitives are key-exchange mechanisms and digital signatures. We have already seen lattice-based key exchange; today we will see lattice-based digital signatures.

For example, the NSA has asked its vendors to transition to using post-quantum algorithms by 2035.

The scheme I will present is not exactly the Dilithium scheme that NIST is standardizing, but it follows the same general strategy. If we have time at the end of the lecture, I will sketch the most important differences.

The general approach for constructing signatures today will be:

1. Construct an interactive protocol that "proves knowledge" of an LWE secret.
2. Use Fiat-Shamir to convert it into a signature scheme.

In practice, we typically construct lattice-based signatures schemes from the hardness of the related "short integer solutions" (SIS) problem. Since have we already seen LWE earlier in the semester, I thought I would stick with LWE.

Reminder: Learning with errors

The learning-with-errors assumption is parameterized by integers m, n, and q, and an error distribution χ over $\mathbb{Z}_q$. The (m, n, q, χ)-LWE assumption asserts that the following distributions are computationally indistinguishable:

$$\left\{ (\mathbf{A}, \mathbf{A} \cdot \mathbf{s} + \mathbf{e}) : \begin{array}{c} \mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{m \times n} \\ \mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n \\ \mathbf{e} \xleftarrow{R} \chi^m \end{array} \right\} \overset{c}{\approx} \left\{ (\mathbf{A}, \mathbf{u}) : \begin{array}{c} \mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{m \times n} \\ \mathbf{u} \xleftarrow{R} \mathbb{Z}_q^m \end{array} \right\}.$$

For the rest of this discussion, think of χ as being the uniform distribution over $\{-B, \dots, B\} \subseteq \mathbb{Z}_q$. It turns out (in a non-obvious way) that if the LWE assumption holds, then the function $f_{\mathbf{A}} \colon \mathbb{Z}_q^n \times \{-B, \dots, B\} \to \mathbb{Z}_q^m$, defined as:

$$f_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) := \mathbf{A}\mathbf{s} + \mathbf{e} \quad \in \mathbb{Z}_q^m, \tag{1}$$

is a one-way function. The restriction that the error vector $\mathbf{e}$ here is B-bounded is crucial. If we do not require the error vector to be "short," then the function $f_{\mathbf{A}}$ is no longer one-way. (Think about why this would be.)

In practice, we typically set χ to be a discrete Gaussian or binomial distribution. Using the uniform distribution over a small range also works, with some loss in efficiency, but it simplifies the discussion.

Recall that we say that a vector $\mathbf{v}$ is "B-bounded." if $\|\mathbf{v}\|_\infty \leq B$.

Proving knowledge of an LWE secret in zero knowledge

Say that you and I both hold an LWE instance $(\mathbf{A}, \mathbf{b})$. I would like to prove to you that I know a secret $\mathbf{s}$ and B-bounded error vector $\mathbf{e}$ such that $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \in \mathbb{Z}_q^m$. The direct way for me to prove this to you is to just send you the pair $(\mathbf{s}, \mathbf{e})$. You can confirm yourself that the error vector is short and that $f_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) = \mathbf{b}$.

A more interesting challenge, which ends up being very relevant for digital-signature schemes, is for me to prove to you that I "know" a pair $(\mathbf{s}, \mathbf{e})$, while revealing no information to you about these values.

Looking ahead, the pair $(\mathbf{A}, \mathbf{b})$ will be the public key for the signature scheme. The pair $(\mathbf{s}, \mathbf{e})$ will be the secret key. And a signature will prove knowledge of the secret key.

More generally, we can think of any poly-time computable function $f \colon \mathcal{X} \to \mathcal{Y}$. The verifier has $y \in \mathcal{Y}$ and wants to be convinced that the prover "knows" an $x \in \mathcal{X}$ such that $f(x) = y$. The proof should reveal nothing else about x. We can express our LWE task as a special case of this one, where we set $f := f_{\mathbf{A}}$.

Defining knowledge

We have already discussed interactive proofs at length. In that context, we talked about a prover convincing a verifier that some *fact is true*: that a graph G is 3-colorable, that a Boolean circuit C run on input x outputs 1, and so on.

Rather than talking about $y = f(x)$, the formal style is to think about an NP relation $\mathcal{R}$ of instance-witness pairs. The verifier takes as input a value v and proves knowledge of an NP witness w such that $(v, w) \in \mathcal{R}$. Since many of you may not have seen NP relations in much detail before, I prefer to stick with this simpler formalism.

Here, we are asking for something different: we want the the prover to convince the verifier that the prover *knows something*. It is not enough for the prover to convince the verifier that there exists some $x \in \mathcal{X}$ such that $y = f(x)$. The verifier should be convinced that the prover *knows* such a solution x.

I will sometimes call the value x here "the witness."

To even begin to construct such a proof system, we first have to answer a philosophical question: What does it mean to "know something?" More specifically, what does it mean for a Turing machine to "know something?" We need a definition of knowledge before we can hope to construct a "proof of knowledge."

Cryptographers have come up with a very clever and very natural definition for "knowledge" in this setting. It is one of those definitions that is obvious in hindsight but that is not at all obvious before you have seen it. The idea is to say that a prover "knows x" if it is possible to extract x from the prover under a vigorous enough interrogation.

A similar definition might be equally applicable to defining human knowledge.

In particular, we will say that an interactive proof has knowledge soundness if there is an efficient algorithm that "extracts" the witness x from any prover P^* that convinces the verifier with good probability.

For simplicity, we will restrict ourselves to three-move protocols in which the prover sends the first message. These are sometimes called "Sigma protocols."

In these three-move protocols, we can consider running P^* forward to get an accepting transcript (v, c, z), then rewinding P^* until the moment before the verifier sent it the challenge, and then running P^* again on a different challenge to obtain a second transcript (v, c', z'). We will say that the protocol satisfies knowledge soundness if it is possible to extract a witness from this pair of accepting transcripts.

Zero knowledge proof of knowledge

We say that a three-move interactive protocol (P, V) is a zero-knowledge proof of knowledge for preimages of $f\colon \mathcal{X} \to \mathcal{Y}$ if it satisfies these three properties:

I use $\langle P(x) \leftrightarrow V(f(x))\rangle$ to denote the output of the interaction between the prover on input x, $P(x)$, and the verifier on input $f(x)$, $V(f(x))$.

- **Completeness:** (*When prover and verifier are honest.*) If the prover "knows" x, it will always convince the verifier that it does. Formally, for all $x \in \mathcal{X}$,

$$\Pr[\langle P(x) \leftrightarrow V(f(x))\rangle = 1] = 1.$$

- **Knowledge soundness:** (*Protection for the verifier against a cheating prover.*) The only way that a cheating prover can convince a verifier $V(y)$ to accept, is if the prover "knows" an

$x \in \mathcal{X}$ such that $f(x) = y$. In particular, we can extract such an x from any prover P^* with roughly the same probability that that P^* convinces the verifier.

Formally, we say that a protocol has *knowledge soundness* with knowledge error ϵ if there exists a p.p.t. extractor Ext such that for all cheating provers P^* and all $y \in \mathcal{Y}$:

$$\Pr[\mathsf{Ext}^{P^*}(y) = x \text{ s.t. } f(x) = y] \geq \Pr[\langle P^*, V(y)\rangle = 1] - \epsilon.$$

- **Zero knowledge:** (*Protection for the prover against a cheating verifier.*) The verifier learns nothing about x during its interaction with the prover, apart from the fact that $y = f(x)$.

  Formally, for all (possibly malicious) verifiers V^*, there exists a p.p.t. algorithm Sim, called "the simulator," such that for all $x \in \mathcal{X}$,

$$\{\text{View of } V^* \text{ in } \langle P(x) \leftrightarrow V^*(f(x))\rangle\} \overset{c}{\approx} \{\mathsf{Sim}(f(x))\}.$$

We already spoke briefly about the intuition behind this definition of zero knowledge: The idea is that you have not learned anything from an interaction with P, if you could sit at home and write down a transcript of your interaction with P *even without ever talking to P*.

This zero-knowledge property seems at odds with knowledge soundness: how can we say that the verifier learns nothing from P when it's also possible for the extractor to somehow extract the preimage (witness) x from P?

The key is that running the extractor requires having the code of P in your hand: the extractor may need to run P forwards and backwards to extract the preimage. In contract, in a real interaction, the verifier communicates with P over a network and P will never rewind itself for the verifier's benefit. The knowledge-soundness definition effectively says that *if you could* obtain the code of the prover (rather than just talking to the prover over the network), you could extract the witness from it.

Protocol: Proving knowledge of an LWE secret

The proof is relative to LWE parameters (n, m, q, χ) and a public matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$. The proof makes use of a second error distribution χ' over $\mathbb{Z}_q$ that is B'-bounded. I will discuss this more after giving the protocol.

The interaction proves knowledge of a vector $\mathbf{s} \in \mathbb{Z}_q^n$ and B-bounded vector $\mathbf{e} \in \mathbb{Z}_q^m$ such that $\mathbf{A}\mathbf{s} + \mathbf{e} = \mathbf{b}$. The interaction is between the prover, holding $(\mathbf{s} \in \mathbb{Z}_q^n, \mathbf{e} \in \mathbb{Z}_q^m)$ and the verifier, holding $\mathbf{b} \in \mathbb{Z}_q^m$. The prover and verifier exchange three messages:

- **Step 1: Commitment.** The prover essentially cooks up a new fresh LWE instance and sends it to the verifier. That is, the prover computes

  $$\mathbf{v} \leftarrow \mathbf{A}\mathbf{u} + \mathbf{e}' \in \mathbb{Z}_q^m \qquad \text{for} \qquad \mathbf{u} \xleftarrow{R} \mathbb{Z}_q^n, \mathbf{e}' \xleftarrow{R} \chi'^m.$$

  The prover sends $\mathbf{v} = \mathbf{A}\mathbf{u} + \mathbf{e}' \in \mathbb{Z}_q^m$ to the verifier.
- **Step 2: Challenge.** The verifier chooses $\beta \xleftarrow{R} \{0, 1\}$ and sends β to the prover.
- **Step 3: Response.** The prover responds with:

  $$\mathbf{z} = \begin{cases} \mathbf{u} & \text{if } \beta = 0, \text{ and} \\ \mathbf{u} + \mathbf{s} & \text{if } \beta = 1. \end{cases}$$

The verifier accepts the proof if

$$(\mathbf{v} + \beta\mathbf{b} - \mathbf{A}\mathbf{z}) \quad \in \mathbb{Z}_q^m$$

is $(B + B')$-bounded.

Let us now make sure that the protocol satisfies our three properties.

Completeness. By construction:

$$\begin{aligned} \mathbf{v} + \beta\mathbf{b} - \mathbf{A}\mathbf{z} &= \mathbf{v} + \beta\mathbf{b} - \mathbf{A}(\mathbf{u} + \beta\mathbf{s}) \\ &= (\mathbf{A}\mathbf{u} + \mathbf{e}') + \beta(\mathbf{A}\mathbf{s} + \mathbf{e}) - \mathbf{A}(\mathbf{u} + \beta\mathbf{s}) \\ &= \mathbf{e}' + \beta\mathbf{e}. \end{aligned}$$

Since $\mathbf{e}$ is B-bounded and $\mathbf{e}'$ is B'-bounded, their sum is $(B + B')$-bounded and the verifier always accepts.

Knowledge soundness. We need to construct an extractor that extracts a witness x from a cheating prover P^*. We will only sketch the idea here.

The idea is that the extractor Ext runs the protocol with prover P^* to completion, getting a transcript $(\mathbf{v}, c, \mathbf{z})$. The extractor then *rewinds* the prover P^* until the moment after it send its commitment $\mathbf{v}$. The extractor then runs the protocol to completion with a fresh random challenge c' to get a transcript $(\mathbf{v}, c', \mathbf{z}')$.

Provided that $c \neq c'$, which happens with probability $1/2$, we can extract.

That is the vectors $\mathbf{e}_0$ and $\mathbf{e}_1$ are $(B + B')$-bounded:

$$\begin{aligned} \mathbf{e}_0 &= \mathbf{v} - \mathbf{A}\mathbf{z} & \in \mathbb{Z}_q^m \\ \mathbf{e}_1 &= \mathbf{v} - \mathbf{A}\mathbf{z}' + \mathbf{b}. \end{aligned}$$

The extractor then subtracts the first from the second to get:

$$\begin{aligned} \mathbf{e}_1 - \mathbf{e}_0 &= \mathbf{A}(\mathbf{z}' - \mathbf{z}) + \mathbf{b} \\ \mathbf{A}(\mathbf{z}' - \mathbf{z}) + (\mathbf{e}_1 - \mathbf{e}_0) &= \mathbf{b}. \end{aligned}$$

And now the extractor returns the pair $(\mathbf{s}, \mathbf{e})$, where $\mathbf{s} = \mathbf{z}' - \mathbf{z}$ and $\mathbf{e} = \mathbf{e}_1 - \mathbf{e}_0$.

Observe we have that (1) $\mathbf{As} + \mathbf{e} = \mathbf{b}$ and (2) $\mathbf{e}$ is $(B + B')$-bounded, so indeed the pair $(\mathbf{s}, \mathbf{e})$ is a valid LWE solution.

We will not prove it here, but the knowledge error of this protocol is $\approx 1/2$.

More generally, if a Sigma protocol like this has knowledge soundness, then the best attack we know runs in time $1/\left|\text{challenge space}\right|$.

More precisely, if there is a malicious prover that can cheat in the sigma protocol with probability ϵ, we can use it to invert f with probability $1/\left|\text{challenge space}\right| + \sqrt{\epsilon}$. See Theorem 19.14 in Boneh-Shoup for details.

Zero knowledge. Zero knowledge here is a bit more slippery. The key point to notice is that with probability $1/2$, whenever $\beta = 1$, the verifier obtains the vector $\mathbf{e} + \mathbf{e}'$, as we computed when arguing completeness.

Now, the $\mathbf{e}$ part is constant over each protocol run—that error term is part of the prover's secret input. In contract, the prover samples $\mathbf{e}'$ freshly at random from χ'^m with each protocol run.

If $\chi = \chi'$ and both are B-bounded for small-ish (e.g., polynomial) B, then the protocol is not actually zero knowledge and we have a problem. The issue is that, over many many protocol runs, a verifier can take the average of many many $\mathbf{e} + \mathbf{e}'$ terms, to get a good approximation of $\mathbf{e}$. Once the verifier has $\mathbf{e}$, it can recover the secret $\mathbf{s}$ and it has learned the prover's entire input.

The key to getting zero knowledge then, is for the prover to sample $\mathbf{e}'$ from a much wider distribution. That is, $B' \gg B$. Then we can argue that $\mathbf{e}'$ is big enough to hide $\mathbf{e}$ and the verifier learns nothing.

We need the following lemma, sometimes called the "smudging" or "noise-flooding" lemma:

Lemma 1. *Let $e \in \{-B, \dots, B\}$ be a fixed integer, let $e' \xleftarrow{R} \{-B', \dots, B'\}$ be chosen at random. Then the advantage of every algorithm at distinguishing e' from $e + e'$ is at most B/B'.*

The lemma implies that as long as $B' = 2^\lambda B$, on security parameter λ, the vector $\mathbf{e} + \mathbf{e}'$ in our protocol is as good as an independently sampled random vector with elements uniform in $\{-B, \dots, B\}$.

Having to choose B' so large is annoying since now we need the modulus $q \gg B' \gg B$, which implies that q must be exponential in the security parameter. So instead of $q \approx 2^{32}$, we might need $q \approx 2^{512}$ or something, which slows down all of the operations. Still, if we take the lattice dimension n large enough, we can satisfy all of these constraints while ensuring that LWE is still (apparently) hard.

Let's now try to construct the simulator for a malicious verifier V^*. The simulator works as follows:

$\mathsf{Sim}(\mathbf{b})$:

- Make a guess $\hat{\beta} \in \{0, 1\}$ of the challenge.

- Choose a random $\mathbf{z} \overset{R}{\leftarrow} \mathbb{Z}_q^m$ and $\mathbf{e}' \overset{R}{\leftarrow} \chi'^m$.
  (*Pick the last protocol message first.*)
- Compute $\mathbf{v} = \mathbf{A}\mathbf{z} - \hat{\beta}\mathbf{b} + \mathbf{e}' \in \mathbb{Z}_q^m$.
  (*Compute the first message that the prover needs to send for the verifier to accept after seeing the last message.*)
- Run the verifier V^* on first message $\mathbf{v}$.
- The verifier returns a bit β. If $\beta \neq \beta'$, restart.
- Output $(\mathbf{v}, \beta, \mathbf{z})$.

Now, we must argue that this transcript is indistinguishable from a real one. The verifier's first message is

$$\begin{aligned} \mathbf{v} &= \mathbf{A}\mathbf{z} - \beta(\mathbf{A}\mathbf{s} + \mathbf{e}) + \mathbf{e}' \\ &= \mathbf{A}(\mathbf{z} - \beta\mathbf{s}) - \beta\mathbf{e} + \mathbf{e}'. \end{aligned}$$

By Lemma 1, we can say that

$$\mathbf{v} \overset{s}{\approx} \mathbf{A}(\mathbf{z} - \beta\mathbf{s}) + \mathbf{e}',$$

where $\overset{s}{\approx}$ denotes statistical closeness.

But now the transcript is exactly as in the real protocol: the verifier sends $\mathbf{A}\mathbf{u} + \mathbf{e}'$ for some $\mathbf{u} \in \mathbb{Z}_q^n$ distributed uniformly over the space, and for $\mathbf{e}' \overset{R}{\leftarrow} \chi'^m$. (It just so happens that $\mathbf{u} = \mathbf{z} - \beta\mathbf{s}$.) Then the verifier sends $\mathbf{z} = \mathbf{u} + \beta\mathbf{s}$ as the last protocol message.

Reducing soundness error with parallel repetition

The protocol we have sketched has knowledge soundness error $1/2$. That means that the prover effectively can cheat with probability $1/2$. One way we discussed to drive the knowledge soundness down to 2^t is to run the protocol t times in parallel. This has the side-effect of weakening the zero-knowledge property to something called *honest-verifier zero knowledge*, but it turns out that that is good enough for our application.

Honest-verifier zero knowledge essentially say that an honest verifier learns nothing (in the zero-knowledge sense) from its interaction with the prover. Formally, we only require the simulator to output transcripts of an honest verifier's interaction with the prover.

Making it non-interactive

Finally, we can apply the Fiat-Shamir paradigm from the last lecture to make this entire protocol non-interactive. In our setting, that means that the prover will construct the challenge bits as

$$(\beta_1, \ldots, \beta_t) \leftarrow \mathsf{Hash}(\mathbf{A}, \mathbf{b}, \mathbf{v}), \quad \in \{0,1\}^t$$

where $(\mathbf{A}, \mathbf{b})$ is the LWE instance and $\mathbf{v}$ is the prover's first message.

We can now only prove security if we model the hash function $\mathsf{Hash}(\cdot)$ as a random oracle. But that is good enough in practice. In fact, we analyze most of the cryptosystems we use in practice in the random-oracle model.

Digital signatures

It turns out that it is a very small step from the Fiat-Shamir-based zero-knowledge proof of knowledge to an LWE-based digital-signature scheme.

Definition

We pulled the following directly from the 6.1600 lecture notes.

Definition 2 (Signature Scheme)**.** A signature scheme is associated with a message space $\mathcal{M}$ and three efficient algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$.

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{sk}, \mathsf{vk})$. The key-generation algorithm as input a security parameter $\lambda \in \mathbb{N}$ and outputs a secret signing key sk and public verification key vk. The algorithm Gen runs in time poly(λ).
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$. The signing algorithm takes as input a secret key sk and a message $m \in \mathcal{M}$, and outputs a signature σ.
- $\mathsf{Ver}(\mathsf{vk}, m, \sigma) \to \{0, 1\}$. The signature-verification algorithm takes as input a public verification key vk, a message $m \in \mathcal{M}$, and a signature σ, and outputs $\{0, 1\}$, indicating acceptance or rejection.

For a signature scheme to be useful, a correct verifier must always accept messages from an honest signer. Formally, we have:

Definition 3 (Digital signatures: Correctness)**.** A digital-signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is *correct* if, for all messages $m \in \mathcal{M}$:

$$\Pr\left[\mathsf{Ver}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1 \colon (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(\lambda)\right] = 1.$$

Definition 4 (Digital signatures: Security – existential unforgeability under chosen message attack)**.** A digital-signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is *secure* if all efficient adversaries win the following security game with only negligible probability:

- The challenger runs $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(\lambda)$ and sends vk to the adversary.
- For $i = 1, 2, \ldots$ (polynomially many times)
  - The adversary sends a message $m_i \in \mathcal{M}$ to the challenger.
  - The challenger replies with $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, m_i)$.
- The adversary outputs a message-signature pair (m^*, σ^*).
- The adversary wins if $\mathsf{Ver}(\mathsf{vk}, m^*, \sigma^*) = 1$ and $m^* \notin \{m_1, m_2, \ldots\}$.

Construction from LWE

The basic idea is that the public key is an LWE instance and the secret key is a solution to the LWE instance.

To sign, the prover uses the non-interactive zero-knowledge proof of knowledge that we sketched to prove knowledge of the LWE secret, which proves knowledge of the secret key. In the proof, we salt the hash function used in the Fiat-Shamir step with the message to be signed. In this way, each distinct message to be signed gives a distinct random challenge.

Why would we expect this signature scheme to be secure?

First, by the zero-knowledge property of the proof system, we can argue that the verifier learns nothing about the secret key after seeing many signatures, apart from what it learns via the signatures themselves.

Second, by the knowledge soundness property of the proof system, the verifier is convinced that the signer actually knows the secret key. It is not possible for someone without knowledge of the secret key to forge signatures.

The formal proof is in the Boneh-Shoup book (Chapter 19.2).

The scheme uses a hash function $\mathsf{Hash}\colon \{0,1\}^* \to \{0,1\}^t$, which we model as a random oracle.

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{sk}, \mathsf{vk})$.
    - Choose LWE parameters (n, m, q, χ), where all are a function of λ.
    - Compute

$$\begin{aligned} \mathbf{A} &\xleftarrow{R} \mathbb{Z}_q^{m \times n} \\ \mathbf{s} &\xleftarrow{R} \mathbb{Z}_q^n \\ \mathbf{e} &\xleftarrow{R} \mathbb{Z}_q^m \\ \mathbf{b} &\xleftarrow{R} \mathbf{A}\mathbf{s} + \mathbf{e} \in \mathbb{Z}_q^m. \end{aligned}$$

    - Set $\mathsf{sk} = (\mathbf{A}, \mathbf{b}, \mathbf{s}, \mathbf{e})$, $\mathsf{vk} = (\mathbf{A}, \mathbf{b})$.

- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$. Prove knowledge of a solution to the LWE instance $(\mathbf{A}, \mathbf{b})$ using the secret key. In the Fiat-Shamir step, use the salted the hash function $\mathsf{Hash}(m, \cdot)$. Output the proof as the signature.

- $\mathsf{Ver}(\mathsf{vk}, m, \sigma) \to \{0,1\}$. Run the verifier for the zero-knowledge proof using LWE instance $(\mathbf{A}, \mathbf{b})$ and the hash function $\mathsf{Hash}(m, \cdot)$. Accept iff the verifier accepts.

Practical considerations

There are a bunch of efficiency issues with the signature scheme I presented:

- The public key includes the matrix $\mathbf{A}$, can take megabytes or more to represent. Instead, we can represent the matrix $\mathbf{A}$ using a seed and expand it using a hash function (again modeled as a random oracle).

- Both signing and verification here require computing matrix-vector products with the large matrix $\mathbf{A}$. Schemes used in practice replace LWE with the "Ring-LWE" variant we discussed a while back. The high-order bit is that the vectors and matrices become polynomials. Then matrix-vector products become polynomial multiplications, which are much faster.

- Repeating the protocol $t = 128$ times to get 128-bit security blows up the running time and signature size by a factor of 128. Real implementations sample the challenge c from a larger space, which means that we can set the iteration count t smaller to get the same level of security.

- The modulus q must be $\gg 2^\lambda$ for the proof system to be zero-knowledge. In practice we would like to use a much smaller modulus $q \approx 2^{32}$ so that we have smaller keys and faster $\mathbb{Z}_q$ operations.

  The reason why we needed to take the bound B' on the norm of $\mathbf{e}'$ to be so large is that we need $\mathbf{e} + \mathbf{e}'$ to completely hide $\mathbf{e}$. Choosing $\mathbf{e}'$ to have gigantic entries ("noise flooding") as we did is one way to solve this problem.

  A more clever way to solve this problem is for the signer to run the protocol for the first two step and then to abort if the value $\mathbf{e} + \mathbf{e}'$ is too large. By choosing the parameters carefully, it is possible show that the resulting protocol gives a secure signature scheme.

This is called "Fiat-Shamir with Aborts" and is due to Lyubashevsky.

The NIST-standardized schemes use all of these optimizations and many more. But the core of the schemes follows the same framework we described here.

References