

Signatures

Previously we looked at lattice-based key exchange (public key encryption)

- Now, we look at digital signatures

Proving Knowledge of a Secret

If we both hold an LWE instance (A, b) and I want to prove that I know a secret s and error e such that $b = As + e$:

- The direct way is to just reveal the pair (s, e)
- You can then verify the error vector is sufficiently small and that the computation holds

We want to do so in a zero-knowledge setting where I do not reveal what the pair is

- We want to prove that I know the pair (s, e) without giving it away
- We say that a prover **knows** something if with enough messages, we could extract the thing
- The "something" that we are trying to prove we know is typically called the **witness**

Zero-knowledge Proof of Knowledge

A three-move interactive protocol (P, V) is a zero-knowledge proof of knowledge that the prover P knows some x such that $f(x) = y$ for a given y is one that satisfies:

- **Completeness** (when both are honest):
 - The prover can always convince the verifier it knows an x such that $f(x) = y$
- **Knowledge soundness** (when the prover is cheating):
 - The only way a cheating prover can convince a verifier to accept is if the prover actually knows an x
 - There is some different algorithm Ext that when we run on a prover, attempts to figure out what x is
 - For knowledge soundness, we need the probability of being able to successfully extract a x from P is \geq the probability of getting verified in the first place, minus some ϵ that we call the knowledge gap

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

- **Zero knowledge** (when the verifier is cheating):
 - The verifier learns nothing about x during its interaction besides $f(x) = y$
 - It is possible for the verifier to by itself just write the entire transcript of the conversation in poly time
 - We prove this by showing there is some simulator that can produce a transcript indistinguishable from a real one in poly time
- One key is that the extractor may have to have the code of P available so you can run it / simulate it yourself:
 - The extractor may need to run P multiple times from the same starting state to get the knowledge out
 - If instead the prover is just over a network and you don't have this liberty, then you shouldn't be able to extract the key out

Proving Knowledge of an LWE Secret


We want to prove knowledge of a vector s and e such that $As + e = b$ for a public A and b and error bound B

In this scheme, we exchange three messages:

- **Commitment:** The prover makes a new LWE instance $v = Au + e'$ with the same dimensions and with an error bound of B'
 - The prover sends v to the verifier
- **Challenge:** the verifier chooses $\beta \in \{0, 1\}$ randomly and sends to the prover
- **Response:** the prover responds with u if $\beta = 0$ and $u + s$ if $\beta = 1$

The verifier accepts if $v + \beta b - Az$ is an error bounded by $B + B'$

We prove this satisfies the three properties:

- **Completeness:** the calculated quantity is $e' + \beta e$, so the sum is clearly $B + B'$ bounded
 - **Knowledge soundness:**
 - Suppose the extractor runs the scheme twice with the same v provudes by the prover
 - There is a probability with $1/2$ that the c generated are different
 - We can then extract the s, e by doing algebra with the returned values
 - This has a knowledge soundness error of $1/2$, meaning the prover can "cheat" with probability $1/2$ and not get "detected" by the verifier
 - We can reduce this knowledge soundness by doing the protocol t times in parallel which reduces the zero-knowledge aspect
 - It reduces to **honest verifier zero knowledge**
 - **Zero knowledge:**
 - The key is that with probability $1/2$, the verifier obtains $e' + e$ and with probability $1/2$ the verifier obtains e'
 - The idea is that e' should have its bound $B' \gg B$ so that it "hides" e
 - This requires the following lemma:
-  **Smudging / Noise Flooding Lemma**

Let $e \in \{-B, \dots, B\}$ be a fixed integer and 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'
- If we have $B' = 2^\lambda B$, then this says that $e + e'$ is effectively random
 - We can then construct a simulator of a verifier V as:
 - Make a guess $\hat{\beta}$ of the challenge
 - Choose a random z and error e'
 - This is the last message in the protocol
 - Compute $v = Az - \hat{\beta}b + e'$
 - Compute the message the prover would have needed to send for the verifier to accept
 - Run V on v and if the β it produces matches β' then we output this transcript
 - Otherwise, restart
 - We show this is indistinguishable from a real transcript:
 - The message v is $A(z - \beta s) - \beta e + e'$ which by the lemma is indistinguishable from $A(z - \beta s) + e'$
 - But $z - \beta s$ is uniformly distributed, so this is a possible message transcript

Making Proof Non-Interactive

We now use the Fiat-Shamir paradigm to convert this into a non-interactive proof using a hash function

- Instead of a real verifier providing a challenge, the prover computes it with a hash function:

$$\text{challenge} = H(\text{commitment}, \text{public data})$$

- The prover then sends out the commitment and the response to the corresponding challenge
- Any verifier can then check the proof by computing the challenge
 - The idea is that because a hash function was used, the challenge should be effectively randomized
 - There has to be some way of making sure the commitment is not deliberately chosen to make the challenge easy

This is down in our LWE scheme by choosing the challenge as:

$$(\beta_1, \dots, \beta_t) \leftarrow \text{Hash}(A, b, c)$$

Digital Signatures

We now take the zero knowledge proof and make a digital signature scheme

Signature Scheme

A signature scheme has a message space \mathcal{M} and three algorithms:

- $\text{Gen}(\lambda) \rightarrow (\text{sk}, \text{vk})$
- $\text{Sign}(\text{sk}, m) \rightarrow \sigma$
- $\text{Ver}(\text{vk}, m, \sigma)$

For this to be useful, a verifier must also accept messages from an honest signer

For this to be secure, even with polynomially many samples, an adversary cannot construct a new message signature pair that is accepted

We use LWE by making our public key an LWE instance and our secret key the solution

- To sign, the prover provides proof of knowledge of the secret key
 - We salt the hash function with the message we want to sign so the challenge is specifically tied to that message
- To verify, we just verify this zero-knowledge proof