An Introduction to Lattices, Lattice Reduction, and Lattice-Based Cryptography

Joseph H. Silverman

Brown University

PCMI Lecture Series

July 6–10, 2020

Lecture 3. Public Key Cryptography 101: A Brief Introduction

Cryptography in the (pre-1970s) Dark Ages

Fundamental Problem: Bob wants to send Alice a secret message that their adversary Eve cannot read.

- **Step 1**: Bob and Alice share a Secret Key that Eve does not know.
- **Step 2**: Bob uses the Secret Key to encrypt a Message.
- **Step 3**: Bob sends the Encrypted Message to Alice.
- **Step 4**: Alice uses the Secret Key to decrypt the Encrypted Message.
- **Step 5**: Eve intercepts the Encrypted Message, but since she does not know the Secret Key, she cannot recover the Message.

Problem: Bob and Alice cannot send messages until they exchange a Secret Key. What if they've never met?

Example: Alice is Amazon, and the message is Bob's credit card number.

Public Key Cryptography to the Rescue

- In the mid-1970s Diffie and Hellman proposed creating cryptosystems that use two keys, a Private Key that Alice keeps secret, and a Public Key that she publishes.
- Bob only needs to know the Public Key in order to encrypt a Message (aka a PlainText) and create the Encrypted Message (aka the CipherText).
- Alice, who knows the Private Key, is easily able to decrypt the CipherText and recover the PlainText.
- Eve, since she does not know the Private Key, is unable to decrypt.

This all sounds great, but Diffie and Hellman were not able to propose an explicit example of such a

Public Key Cryptosystem

A Mathematical Formulation of Public Key Cryptosystems

Encryption and decryption are really functions:

$$\{\text{Public Keys}\}\times\{\text{Plain Texts}\}\xrightarrow{\text{Encrypt}}\{\text{Cipher Texts}\},$$

$$\{\text{Private Keys}\}\times\{\text{Cipher Texts}\}\xrightarrow{\text{Decrypt}}\{\text{Plain Texts}\}.$$

If (PubKey,PrivKey) is a valid public/private key pair, then for all messages Msg we want

$$\text{Decrypt}\big(\text{PrivKey},\text{Encrypt}(\text{PubKey},\text{Msg})\big)=\text{Msg}.$$

Eve knows the Public Key, but she can neither decrypt messages nor deduce the Private Key.

A **Trap Door Function** is an invertible functions f such that:

(1) $f(x)$ is easy to compute.
(2) $f^{-1}(y)$ is hard to compute.
(3) Knowledge of some extra piece of information (the "trapdoor") makes $f^{-1}(y)$ easy to compute.

Building Trapdoor Functions

Trapdoor functions and Public Key Cryptosystems are built from hard mathematical problems. I'm going to describe four examples.

The Integer Factorization Problem (IFP) (The "Roots Modulo pq Problem")

Given two large prime numbers p and q and an exponent e, the exponentiation function

$$\mathbb{Z}/pq\mathbb{Z} \longrightarrow \mathbb{Z}/pq\mathbb{Z}, \quad x \longmapsto x^e \bmod pq,$$

is easy to compute, but hard(?) to invert unless you know p and q.

The IFP is used to build the

RSA Public Key Cryptosystem.

$$\text{Public Key} = (pq, e), \qquad \text{Private Key} = (p, q).$$

Building Trapdoor Functions (continued)

The Discrete Logarithm Problem (DLP)

Let p be a large prime number, and let $g \in \mathbb{F}_p^*$. The powering function

$$\mathbb{Z}/(p-1)\mathbb{Z} \longrightarrow \mathbb{F}_p^*, \quad k \longmapsto g^k \bmod p,$$

is easy to compute, but hard to invert.

The DLP is used build the

Elgamal Public Key Cryptosystem.

Public Key $= (p, g, g^k)$ Private Key $= k$.

Building Trapdoor Functions (continued)

The Elliptic Curve Discrete Logarithm Problem (ECDLP).

Similar to the DLP, but the multiplicative group $\mathbb{F}_p^*$ is replaced by the group of points $E(\mathbb{F}_p)$ on an elliptic curve.

Why use elliptic curves? Becasue the ECDLP is (ostensibly) harder than the IFP or DLP, so keys and ciphertexts are smaller.

It's all about the never-ending battle between contradictory goals:

- Be maximally efficient!
- Be maximally secure!

Building Trapdoor Functions (continued)

The Closest Vector Problem (CVP)

Let L be a lattice and let $\mathcal{B}^{\text{bad}} = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n\}$ be a bad basis for L. Then the function

$$\{0,1\}^n \longrightarrow \mathbb{R}^n,$$

$$(\epsilon_1, \ldots, \epsilon_n) \longmapsto \epsilon_1 \boldsymbol{w}_1 + \cdots + \epsilon_n \boldsymbol{w}_n + \begin{pmatrix}\text{small ran-}\\ \text{dom vector}\end{pmatrix}$$

is easy to compute, but hard to invert.

We will discuss cryptosystems built from the CVP in Lecture 4.

From Trapdoor Functions to Public Key Cryptosystems

From IFP to RSA

The **RSA Cryptosystem** invented by Rivest, Shamir, and Adelman works as follows:

- Private Key: (p,q). Public Key: (pq,e).
- Plaintext: A number M mod pq.
- Ciphertext: The number $C \equiv M^e \pmod{pq}$.
- Decryption: Compute

$$C^d \bmod pq \text{ where } de \equiv 1 \pmod{pq - p - q + 1}.$$

We've Cheated! Factoring pq will indeed break RSA, but all we really need to do is solve the

Taking Roots Modulo pq Problem.

Is that easier?

From Trapdoor Functions to Public Key Cryptosystems

From DLP to Elgamal

The **Elgamal DLP-Based Cryptosystem** introduces randomness, a topic to which we shall return.

- Private Key: k Public Key: $(p, g, g^k \in \mathbb{F}_p^*)$.
- Plaintext: A number M mod p.
- Ciphertext: Choose a a random R mod $p-1$. The ciphtertext is the pair of values

$$C_1 \equiv g^R \pmod{p} \quad \text{and} \quad C_2 \equiv M \cdot (g^k)^R \pmod{p}.$$

- Decryption: Compute $C_1^{k-1} \cdot C_2$ mod p.

Elliptic Curve Elgamal works similarly, using the group law on E.

Although again we've cheated. Solving DLP will break Elgamal, but all that's really needed is to solve the

Diffie–Hellman Problem: Given g, g^a, g^b, compute g^{ab}.

From Trapdoor Functions to Public Key Cryptosystems

From CVP to GGH and NTRU

It's relatively straightforward to create a public key cryptosystem from the CVP, but lattice reduction algorithms such as LLL-BKZ make it insecure unless the key sizes are very large.

The use of lattices having additional structure leads to more practical cryptosystems.

Lectures 4 and 5 will be devoted to a detailed discussion of

Lattice-Based Cryptography.

And as a trailer of coming attractions, next week Kristen Lauter will tell you all about

Isogeny-Based Elliptic Curve Cryptography.

Signatures in the Dark Ages

Bob's signature on a document affirms that he created the document or is willing to abide by its terms.

Bobolink Bank of Boston

Pay to **Alice Adams** $100.00

BOB ROBERTS

The bank can verify Bob's signature by comparing it to a copy that they have on record.

But suppose that Bob wants to sign a computer file and send it to Alice. How can she verify his digital signature on the file? **Digital Signatures** provide the answer. They are at least as important for internet security as are public key cryptosystems.

Example: Bob is Microsoft sending an update for Alice's computer. Should she install it? Only if she can verify that it came from Bob.

Digital Signatures

Similar to public key cryptosystems, a

Digital Signature Scheme

also uses two functions:

$$\{\text{Private Keys}\}\times\{\text{Digital Docs}\} \xrightarrow{\mathsf{Sign}} \{\text{Signatures}\},$$

$$\{\text{Public Keys}\}\times\{\text{Signatures}\}\times\{\text{Digital Docs}\} \xrightarrow{\mathsf{Verify}} \{\text{Yes, No}\}.$$

If (PubKey,PrivKey) is a valid public/private key pair, then for all documents Doc and potential signatures Sig we want

$$\mathsf{Verify}(\text{PubKey},\text{Sig},\text{Doc}) = \text{Yes} \iff \text{Sig} = \mathsf{Sign}(\text{PrivKey},\text{Doc}).$$

Examples of Digital Signature Schemes

RSA Signatures

- Public Key: (pq,e).
  Private Key: d satisfying $de \equiv 1 \pmod{pq-p-q+1}$.
- Document: A number D mod pq.
- Signing: The number $S \equiv D^d \pmod{pq}$.
- Verifying: Signature is valid if $S^e \equiv D \pmod{pq}$.

Elgamal Signatures

- Private Key: k Public Key: $(p,g,g^k \in \mathbb{F}_p^*)$.
- Document: A number D mod p.
- Signing: Choose a a random R mod $p-1$. The signature is the pair of values

  $$S_1 \equiv g^R \pmod{p}, \qquad S_2 \equiv (D-k\cdot S_1)\cdot R^{-1} \pmod{p-1}.$$

- Verifying: Valid if $(g^k)^{S_1} \cdot {S_1}^{S_2} \equiv g^D \pmod{p-1}$.

Lattice-Based Signatures

Opening Friday at a Lecture Hall near you.

Cryptographically Secure Hash Functions

If Bob's document Doc is large, he could break it into pieces $\text{Doc}_1, \text{Doc}_2, \ldots$ and sign each piece. But that's inefficient. In practice, Bob signs a hash of his document.

Intuition: A **Hash Function** takes an arbitrary length input and creates a fixed length, deterministic, but unpredictable and random looking, output:

$$\left\{\begin{array}{c}\text{arbitrary length}\\ \text{bit strings}\end{array}\right\} \xrightarrow{\mathsf{Hash}} \left\{\begin{array}{l}\text{bit strings}\\ \text{of length } b\end{array}\right\}$$

Some desirable (essential) properties for Hash.

- For $D \in \{0,1\}^*$, computing $\mathsf{Hash}(D)$ is **very** fast.
- Given $H \in \{0,1\}^b$, it is **very hard** to find any $D \in \{0,1\}^*$ satisfying $\mathsf{Hash}(D) = H$.
- It is **very hard** to find distinct $D_1, D_2 \in \{0,1\}^*$ satisfying $\mathsf{Hash}(D_1) = \mathsf{Hash}(D_2)$. This is called **Collision Resistance**.
- Altering one bit of D changes $\mathsf{Hash}(D)$ unpredictably.

Random Numbers in Cryptography

For RSA, Bob and Alice need to choose *random* prime numbers. Elgamal uses *random* numbers to encrypt and sign. And even a deterministic cryptosystem such as RSA tends to have security problems unless some randomness is introduced.

Semi-Realistic Example: If Bob wants to send the message M to Alice, rather than encrypting M directly, he chooses a random string R and instead sends Alice

$$M' := R \parallel (R \texttt{ xor } M).$$

That way even if Eve guesses part of the message, she cannot use that knowledge to help with decryption, since the bits of M have been scrambled by R. And Alice can recover the actual plaintext by first recovering M', then computing

$$(R \texttt{ xor } M) \texttt{ xor } R = M.$$

Random Numbers and Pseudo-Random Number Generators

In principle, there are sources that are(?) truely random:

- Quantum phenomena, such as radioactive decay.
- Micro-changes in temperature.

Such sources can be used, but they are relatively inefficient. What Bob and Alice need is an iterative function

$$\mathsf{Rand} : \{0,1\}^N \longrightarrow \{0,1\}^N.$$

Starting from a seed value σ_0, the sequence of values

$$\sigma_1 = \mathsf{Rand}(\sigma_0), \quad \sigma_2 = \mathsf{Rand}(\sigma_1), \quad \sigma_3 = \mathsf{Rand}(\sigma_2), \ldots$$

should be "indistinguishable" from a sequence of values chosen randomly and uniformly from $\{0,1\}^N$.

After generating the seed σ_0 from a true random source, they can iterate Rand to get a long "random" list .

But don't be fooled by the definitons. Creating cryptographically secure hash functions and pseudo-random number generators is **hard**!

How Hard are Hard Problems?

So just how hard are famous "hard problems" such as the IFP, DLP, ECDLP, and CVP?

Honest Answer: No one knows!!! In the sense that we don't have a proof that any of these problems are hard.

Practical Answer: How hard are they to solve using existing algorithms on existing computers?

Problem	Steps Required to Solve the Problem	Key/Ciphertext Size to be Secure
IFP	$\approx \exp(\sqrt[3]{\log pq})$ steps	2000 to 4000 bits
DLP	$\approx \exp(\sqrt[3]{\log p})$ steps	2000 to 4000 bits
ECDLP	$\approx \sqrt{p}$ steps	300 to 400 bits
CVP	$\approx C^{\dim L}$ steps	2000 to 4000 bits

That's all great. Even 4000 bits isn't much. But you probably noticed the caveat:

existing algorithms on **existing** computers.

Quantum Computers Make Their Entrance

A **Quantum Computer** is a machine in which computation on bits (0's and 1's) is replaced by computation on **qubits**.

In the popular literature, a qubit takes on every real value between 0 and 1.

Slightly more precisely, a qubit is described by a complex number representing a superposition of 0 and 1 states with certain probabilities. A quantum computer with n qubits can "perform" a simultaneous computation on 2^n states, achieving an exponential speedup.

So far, the largest quantum computers built have only a handful of qubits. But governments and businesses are investing huge sums of money in the endeavor.

Analogy(?): First airplane flight 1903 — flew 852 feet.
WW I 1914–18 — airplanes ubiquitous.
WW-II 1939–45 — jets flying 500+ MPH.

Quantum Computers and Cryptography

Here's the bombshell paper that started all the fuss:

> *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*, Peter W. Shor. Proc. 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, Nov. 20–22, 1994

Shor's quantum algorithms solve the IFP and DLP (and ECDLP) in more-or-less **quadratic** time.

This has sparked much current research on public key systems that cannot (as far as we know) be broken by a quantum computer, including lattice-based systems, isogeny-based systems, and systems based on coding theory:

Post-Quantum Cryptography

What, Me Worry?

If large-scale working quantum computers are decades off, why worry about them now?

- Infrastructure change is slow, time-consuming, expensive.
- Even if PQC is not used now, we should build it into systems so that we can start using it with a flip of an (electronic) switch.
- How long do you want to protect your secrets? Your legal documents? If for 10 years, then you should *probably* encrypt/sign using PQC. If 50 years, then you *definitely* should.

The US National Institute of Standards and Technology (NIST) estimates that by 2030 it will cost roughly $1 Billion to build a quantum computer that can break 2048-bit RSA. NIST is running a competition to select and standardize post-quantum cryptosystems.

Code Makers Versus Code Breakers

History is littered with the invention of "unbreakable" cryptosystems that got broken.

So before you start touting your own brilliant new cryptosystem, here are a few lessons that I've painfully learned over the years:

- Cryptanalysts, the people who break cryptosystems, are *very very* clever people.
- Cryptanalysts don't play by your rules, they set their own rules. They'll break your algorithm, they'll break your software implementation, they'll break the hardware that you're using!
- Cryptanalysts attack the weakest part of your system or its implementation, frequently via a method that you never even considered.
- If you modify a cryptosystem to make it more efficient, 99 times out of 100 you'll end up compromising its security.

An Introduction to Lattices, Lattice Reduction, and Lattice-Based Cryptography

Joseph H. Silverman

Brown University

PCMI Lecture Series

July 6–10, 2020