Introduction to cryptography

PCMI 2022 Undergraduate Summer School

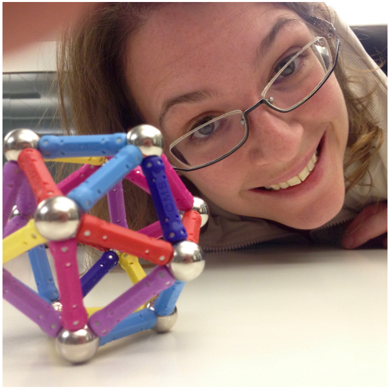
Lecture 1.5

Christelle Vincent

University of Vermont

July 18, 2022

Hi!

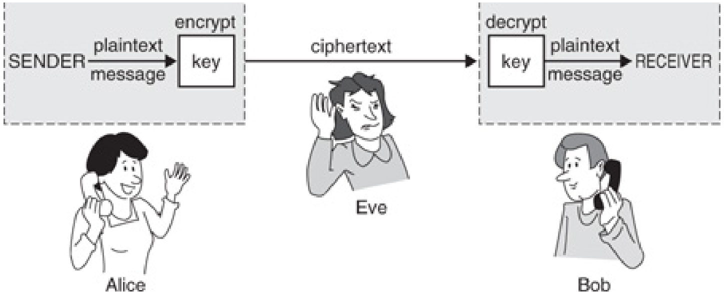


Quick overview of the course

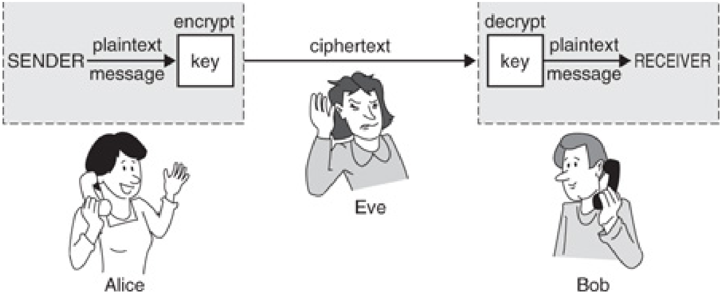
- **Today:** Introduction to cryptography
- **Week 1:** The discrete logarithm problem (DLP)
- **Monday July 25:** Quantum computers and cryptography
- **Week 2:** Lattice-based cryptography
- **Monday August 1:** Fully-homomorphic encryption (FHE)
- **Week 3:** Lattice-based cryptography, continued
- **Friday August 5:** Cryptography in the real world

What is cryptography?

Cryptography is the practice and study of techniques for **secure communication in the presence of adverse third parties.**



Some vocabulary



Encryption refers to the process of changing ordinary text (**plaintext**) into unintelligible text (**ciphertext**).

Decryption is the reverse.

Keys in cryptography

A **cipher** is an algorithm used to encrypt and decrypt.

The operation of a cipher can also be controlled by a **key**.

Example: Caesar cipher

The algorithm of the Caesar cipher is to replace each letter by the kth letter preceding it in the alphabet.

The specific choice of k in a given instance of this cipher is the key.

To decrypt, use the **same** key k, but choose the kth letter following a given letter in the alphabet.

Cryptography then: Secret key cryptography

Main idea: A and B share **privately** the cipher and key they will use to communicate.

Examples:

- Caesar cipher ($\sim$100BC)
- Enigma machine (WWII)

Cryptography then: Secret key cryptography

This is also often called “symmetric key” cryptography, since A and B use the **same** secret key to encrypt and decrypt the message.

Drawback: Requires a secure exchange before setting up the secure exchange...

Cryptography now: Public key cryptography

In June 1976, Diffie and Hellman proposed the notion of **public key** or asymmetric key cryptography.[1]

Idea: B generates **two** sets of keys, one public and one private.

[1] Actually this was suggested in 1970 by Ellis at Government Communications Headquarters, but that was classified until 1997.

Cryptography now: Public key cryptography

A uses B's public key to encrypt a message to B,

and then B uses their private key to decrypt the message.

The simplifying assumption

Modern public-key cryptography is expensive computationally.

Use it to encrypt a secret key, then use faster secret-key algorithms for communication.

Main idea

Public or asymmetric key cryptography relies on one thing:

There are some things in mathematics that are easy to do, but difficult to undo.

An example of a suitable problem

Which problem would you rather see appear on an exam:

1. Perform the multiplication 1489×701; or
2. Factor the number 1,043,789.

Rough idea

The primes 1489 and 701 are the private key;
their product 1,043,789 is the public key.

B can publish the public key for everyone to see without fear that people can recover the private key, since factoring is difficult.[2]

[2] At least we think it is, if we don't have a quantum computer.

One-slide detour: the RSA algorithm

In 1978, **R**ivest, **S**hamir and **A**dleman published the first public key cryptography system, which is now called the RSA algorithm.[3]

RSA's security relies on the fact that it is easy to multiply but hard to factor.

[3] Actually an equivalent system was developed by Cocks at GCHQ in 1973.

Our focus

This week we will focus on cryptosystems relying on a different difficult problem: the **discrete logarithm problem**.

The (discrete) logarithm problem

Let $G = \langle g \rangle$ be a finite cyclic group, written multiplicatively.

For example: $(\mathbb{Z}/p\mathbb{Z})^\times$ where p is prime.

We have

$$\begin{aligned}(\mathbb{Z}/13\mathbb{Z})^\times &= \langle 2 \rangle,\\ (\mathbb{Z}/37\mathbb{Z})^\times &= \langle 2 \rangle,\\ (\mathbb{Z}/101\mathbb{Z})^\times &= \langle 2 \rangle.\end{aligned}$$

The (discrete) logarithm problem

Let $G = \langle g \rangle$ be a finite cyclic group, written multiplicatively.

Then the **discrete logarithm problem** (DLP) in the group G is

Problem (The discrete log problem)

Given a generator g of G and $h \in G$, find $0 \leq x < \#G$ such that

$$h = g^x.$$

For fun: Find x such that $2^x \equiv 3 \pmod{101}$.

Why "discrete"?

In the real numbers, for $g > 0$, $g \neq 1$, we write

$$x = \log_g h$$

if (and only if)

$$h = g^x.$$

This logarithm is "easy" to compute, thanks to calculus!

The discrete logarithm

When G is discrete: no calculus so computing the log can be hard.

The easy problem is exponentiation, the hard problem is the log.

The hardness of DLP

Difficulty of computing the discrete log depends on the group G.

For ex. if $G = \mathbb{Z}/n\mathbb{Z}$ under addition with generator $g = 1$, then:

Problem (DLP for $G = (\mathbb{Z}/n\mathbb{Z}, +)$)

Given $h \in G$, find $0 \leq x < \#G$ such that

$$h \equiv x \cdot 1 \pmod{n}.$$

Using the DLP for profit

There is an encryption system, the **Elgamal encryption system**, which relies on the difficulty of DLP.

There is also a signature algorithm based on DLP, which we'll see again on August 5.

The famous Diffie-Hellman key exchange also relies on DLP.

Using the DLP for profit

Today: apply the difficulty of DLP to generating strings of “pseudorandom numbers.”

Random numbers

Most cryptographic protocols rely on the computer being able to generate random numbers.

For example both the Elgamal encryption system and the digital signature algorithm require a random number.

This is because a random number makes a good secret!

Pseudorandom numbers

Unfortunately generating truly random numbers is very hard!

So we settle for numbers that **appear** random, **pseudorandom numbers**.

Dual_EC_DRBG

This algorithm generates a sequence of pseudorandom numbers from one secret number.

The real Dual_EC_DRBG uses the DLP on an elliptic curve over a finite field, but we will explain how it would work if it were implemented with the DLP on $(\mathbb{Z}/p\mathbb{Z})^\times$.

We also vastly simplify the algorithm to focus on the part which is of interest to us.

How to generate a sequence of pseudorandom numbers

Start with a and b modulo p, and a **secret** number s_1.

When you need a **first** pseudorandom number, compute

$$a^{s_1} \pmod{p} \quad \text{and} \quad b^{s_1} \pmod{p}.$$

$r_1 = b^{s_1}$ is outputted as the first pseudorandom number.
$s_2 = a^{s_1}$ is kept secret.

When you need a **second** pseudorandom number, compute

$$a^{s_2} \pmod{p} \quad \text{and} \quad b^{s_2} \pmod{p}.$$

$r_2 = b^{s_2}$ is outputted as the second pseudorandom number.
$s_3 = a^{s_2}$ is kept secret.

And so, until no more numbers are needed.

How to generate a sequence of pseudorandom numbers

$$\begin{tikzcd}
\text{public} & & & & & & & & \\
r_1=b^{s_1} & & r_2=b^{s_2} & & r_3=b^{s_3} & & r_4=b^{s_4} & & \\
s_1 \arrow[u] \arrow[dr] & & s_2 \arrow[u] \arrow[dr] & & s_3 \arrow[u] \arrow[dr] & & s_4 \arrow[u] \arrow[dr] & & \cdots \\
& a^{s_1} \arrow[ur] & & a^{s_2} \arrow[ur] & & a^{s_3} \arrow[ur] & & \; & \\
\text{secret} & & & & & & & &
\end{tikzcd}$$

Inspired by an image due to Shumow and Ferguson and annotated by Matthew Green.

A bit more truthful picture

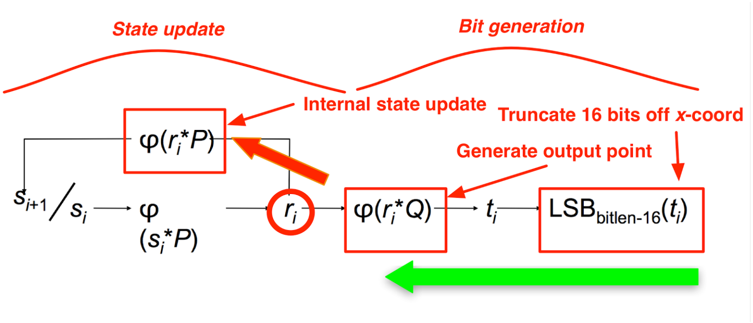


Image due to Shumow and Ferguson and annotated by Matthew Green.

A possible problem

Knowing b and $r_1 = b^{s_1}$ should not allow you to learn s_1. Therefore $s_2 = a^{s_1}$ is a secret, used to generate the next number.

However if $a \equiv b^k \pmod{p}$ for some **known** value of k, then you do not need s_1.

Indeed:

$$(b^{s_1})^k \equiv (b^k)^{s_1} \equiv a^{s_1} \equiv s_2 \pmod{p}.$$

This is a backdoor

The fact that a person installing Dual_EC_DRBG can choose to install it with parameters a and b of their choice, which could be related by an equation

$$a \equiv b^k \pmod{p},$$

is the phenomenon we call a cryptographic **backdoor**.

What is the big deal?

It might seem like using the knowledge of $s_2, s_3, \ldots$ to gain advantage is hard in practice.

Not so: witness the TLS/SSL protocols, which are used to secure basically all of our internet activities (HTTP(S), FTP, SMTP).

Randomness in TLS

TLS is basically a process your computer and the server go through to agree on a secret key:

1. Your computer sends a **(first) random** number to the server, as well as its supported cipher suites.
2. The server sends its own random number to you and authenticates itself using a certificate.

This is not encrypted.

Randomness in TLS

3. Your computer creates a **(second) random** pre-master secret and **encrypts** it with the server's public key from the certificate.
4. The server receives the pre-master secret, decrypts it, and uses it to generate the master secret.
5. Your computer generates the same master secret.
6. Both you and the server switch to a supported cipher suite and use your master secret to communicate.

Randomness in TLS

If someone can guess your random pre-master secret,
they can also generate the master secret,
and use it to listen to your communication.

Nevertheless Dual_EC_DRBG was used for communications with TLS for 8 years!

The timeline

- 1997: Young and Yung publish two papers presenting backdoors on the DLP.
- Early 2000s: the NSA begins pushing to include Dual_EC_DRBG in various standards. The possibility of the backdoor is brought up during discussions.

The timeline

- 2004: RSA Security makes Dual_EC_DRBG the default random number generator in their security suite. It was reported in 2013 that this was a result of a $10 million deal with the NSA.
- 2005: NIST includes Dual_EC_DRBG as part of its standard. In particular the choice of a and b made by the NSA is required for FIPS 140-2 validation.
- 2007: Shumow and Ferguson publicly announce the backdoor in Dual_EC_DRBG. There is a *Wired* article about it.

The timeline

- 2013: Snowden leaks NSA documents showing the existence of a program “to covertly introduce weaknesses into the encryption standards followed by hardware and software developers around the world.” RSA Security advises its customers to stop using Dual_EC_DRBG.
- 2014: NIST removes Dual_EC_DRBG from its standards.

It gets worse: Juniper backdoor

In 2015 it was discovered that in an implementation of Dual_EC_DRBG the value of b had been changed by unauthorized people.

It allowed a person to decrypt protected data passing through the VPN in Juniper NetScreen firewalls.

It is now believed (as of 2021) that the hack was performed by the Chinese group APT 5.

Thank you!