Course Introduction

Notes by Henry Corrigan-Gibbs

MIT - 6.5610
Lecture 1 (February 5, 2024)

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

Logistics

- Course website with video links
- Piazza and Gradescope
- Pset 1 out tomorrow (due 2/16)

Outline

- History of cryptography
  - If time: Merkle puzzles
- Course logistics
- **Stretch break.**
- Post-quantum cryptography
  - Crypto impact of hypothetical quantum computers
  - Grover search

A few remarks before we get started

This is a graduate course in applied cryptography. The goal of an introductory course, such as 6.1600, is to teach you how to *use* cryptographic tools. The goal of this course, in contrast, is to teach you how to *build* cryptographic tools.

The excellent thing about this type of course is that it is not a prerequisite for anything so we can teach whatever we want—whichever ideas are most beautiful, most powerful, and most fun to study. The sort of things we will look at are:

- how the most basic cryptographic primitives (e.g., AES) work,
- how to fetch record from a database without revealing what you fetched,
- how to do computation on encrypted data,
- how to verify a computation that someone else did without re-running the entire computation, and
- how to break cryptosystems: to factor big integers, and so on.

At the end of class, we will have two guest lectures with Ron Rivest and Jim Bidzos (the first CEO of the company that commercialized RSA).

New this year: Since this is a graduate course, we will assume some familiarity with basic cryptographic tools. At the same time, we will try to make the course as self-contained as possible. If you can complete the first problem set, you will be in good shape for the rest of this course.

The lecture notes for 6.1600 summarize much of the background material that you will need.

Note well: This lecture will be much less technical than all those following. But I hope you find it useful and entertaining.

Some history

WARNING: This history is grossly simplified.

The first chunk of this class will be dedicated to "post-quantum" cryptosystems—those that we believe can withstand attacks by a large-scale quantum computer. NIST is developing a new set of post-quantum-secure cryptosystems that all of our devices will soon be using to communicate with each other. The first draft specifications were just published in August 2023, so this is all very new and exciting.

One bonus of these new post-quantum cryptosystems is that they have all sorts of useful properties that factoring- and discrete-log-based cryptosystems do not have. We will discuss those over the next few weeks.

Since this is an *applied* cryptography course we wanted to begin with a brief history of how cryptography has been used over time. The history also puts these new post-quantum systems into context.

See Ron Rivest's excellent list of references, linked from the course website, for a number of books on the history of cryptography.

Most of the historical material in this section from David Kahn's *Codebreakers*.

In my comically simplified view of the world, I divide the history of cryptography into two eras: before 1976 and after 1976.

Before 1976

David Kahn's *Codebreakers* traces the history of cryptography back roughly 4,000 years, to an Egyptian scribe who substituted a few hieroglyphics for some others in a text. The idea of hiding secret messages appears more explicitly in the *Iliad* but the first cryptosystem that we would recognize as one today was used by the Spartan military around 475 BCE.

(Story of Bellerophon.)

The Spartan device was called a *Skytale* (pronounced like "*ski-tall-ee*"). To encode a message, the encryptor would take a strip of leather and wrap it around a stick of a certain fixed (secret!) diameter. Then, the encryptor would write the message down the length of the strip. After unrolling the leather strip, the resulting message looked like gibberish. The decryptor would decrypt by re-wrapping the leather using a stick of the same diameter. The scheme requires the encryptor and decryptor to agree on a *secret key*—the diameter of the stick—in advance.

Since the key space is extremely small—how many different stick sizes can there be? This is not a very secure cipher. Cryptosystems for thousands of years relied for security on the system itself also being secret. To decrypt, you need to know that you are looking for sticks: if you try to wrap the leather strip around a frog, for example, you will not be able to decrypt the message.

Cryptosystems got more and more sophisticated over the next 1,500 or so years, but the principle remained largely the same: the sender and recipient share a secret key and they run some algorithm to garble the message. Governments were, by and large, the most enthusiastic users of cryptosystems for most of this time. Kahn's book has scores of fascinating details on these cryptosystems and how they break over time.

1883: Kerckhoffs' principle

While the most dramatic conceptual developments in cryptography happened in the 20th century, there was one in 1883 worth noting. In that year, Kerckhoffs published a text on military cryptography that popularized the notion that *the only secret in a cryptosystem should be the key*. A designer should always assume that the attacker knows the methods of encryption and decryption.

In modern cryptography, we take this as a given, though most cryptosystems in history tended to implicitly assume that the adversary had some uncertainty about how the cryptosystem worked.

There are two reasons why this principle is a bedrock of cryptography:

1. *Key rotation:* If your algorithm must remain secret, then if an attacker learns the algorithm, you have to design an entirely new algorithm to use. Switching to use a new secret key is much easier

than changing an algorithm.

2. *Algorithm compromise:* A secret key can be smaller than and algorithm and is thus easier to hide. For example, the people building the cipher machines in your factory (or coding the algorithms) have to know the algorithm to do their work. But they don't need to know the secret keys.

1900-1970: Computers enter the picture One of challenges of cryptosystem design (even today!) is that for a scheme to be useful, it must not only be secure, it must also be *fast*. In the pre-computer era, when a human was doing the encipherment, this put some severe limits on how complicated the algorithms could be.

As calculating devices got more and more sophisticated, cryptosystems could get more and more sophisticated: from pen-and-paper to mechanical devices to electromechanical devices (e.g., Enigma), and finally to digital machines. People still design bad cryptosystems on fast computers, but the computing power we have now means that the space of feasible algorithms is much larger now than it was then.

The central role of computation in cryptography meant that early computer scientists (Turing, for example) were instrumental in breaking cryptosystems.

Information theorists, such as Shannon, were involved as well. After Merkle, Diffie, and Hellman, made explicit the connection between cryptography and complexity theory, the influence of information theory on cryptography has waned.

1970-1975: Commercial cryptography

The development of ATM machines in the banking industry drove the development of unclassified cryptosystems for commercial use. The Data Encryption Standard (DES) was the first major such scheme, and it became standard in the banking industry.

You can still find triple-DES in use in the financial data systems.

(Story of development of DES.)

AES is the successor to DES, and we will get into its design in detail on Wednesday.

This, along with Kahn's *Codebreakers*, published in 1967, caused many researchers outside of government to start to think about cryptographic problems.

How cryptosystems break

One thing that we can learn from this history is how cryptosystems fail in practice. The most common failure modes are:

- *Failure of key distribution:* An attacker is able to recover the secret keys, either because of key misuse, reuse, theft, etc.
- *Cryptanalysis:* Just by observing encrypted message traffic—sometimes encryptions of chosen messages, even—the at-

tacker can recover parts of the message.

- *Endpoint compromise:* An attacker is able to recover plaintext data *before* it is even encrypted. Paying off the right person can be a cheap way to get the information you need.
- *Traffic analysis:* Just observing the volume and timing of encrypted message traffic is enough to reveal secret information.

For an example of pre-computer endpoint compromise: Apparently some typewriters in the U.S. Embassy in Soviet Russia were "upgraded" to transmit key-presses via radio to a listening device. The details are here.

Modern cryptography has made serious progress on the first two of these attack vectors. Unfortunately, we have probably made negative progress on the latter two. :(

Developments of 1976 and after

As you may have learned in a prior cryptography course, Merkle proposed the idea of public key exchange in an undergraduate project in 1974. To refresh your memory: the idea is that Alice and Bob, communicating over a shared (but public) channel can agree on a shared secret key that an attacker watching the public channel cannot learn. In a world with billions of web clients and web servers, that all want to communicate with each other securely, key exchange is almost essential.

Diffie and Hellman (1976) then gave the first key-exchange protocol in which the honest parties run in polynomial time and an attacker—who is trying to recover the secret by observing the communication channel—runs in super-polynomial time.

Diffie and Hellman's paper launched academic cryptography as a field.

Merkle Puzzles

You may have seen Diffie and Hellman's simple and clever key-exchange protocol. You probably have not seen Merkle's, which is even simpler and which is too slick to not present. Merkle's scheme is ***NOT*** secure enough to be useful in practice—there is a polynomial-time attack on the scheme. At the same time, the key exchange it gives is non-trivial (the best attack is more costly than the honest parties' computation) and it's just a good construction to know.

The brilliance of Merkle is that he not only came up with the first key-exchange *protocol*, he also came up with the *concept* of key exchange, and of public-key cryptography more generally.

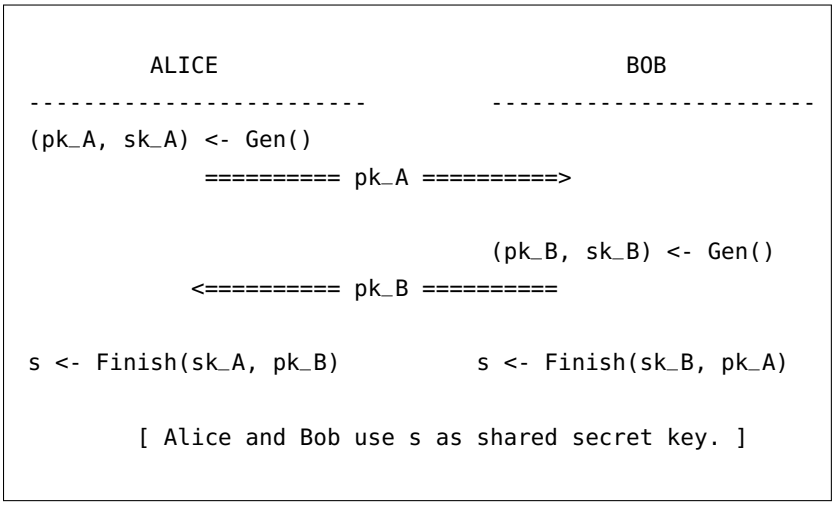
I should note that slightly earlier in the 1970s, Cocks, Ellis, and Williamson in Britain's GCHQ came up with many (though not all) of the same ideas that Merkle, Diffie, Hellman, Rivest, Shamir, and Adleman did. However, had the academics not published their work, it is not clear that we ever would have heard about these ideas for many many years. GCHQ only declassified the work of their researchers in the mid-1990s.

The goal. Before giving the protocol, we have to define the correctness and security goals. Yael will go in depth into cryptographic

formalism next Monday. Today, I will be less formal.

A key-exchange protocol over a shared-secret space $\mathcal{S}$ consists of two algorithms:

- $\mathsf{Gen}() \to (\mathsf{pk}, \mathsf{sk})$. Output a public key and a secret key.
- $\mathsf{Finish}(\mathsf{sk}, \mathsf{pk}) \to \mathcal{S}$. Take a secret key and a public key as input and generate a shared secret.



We want two things:

Correctness means that an honest Alice and honest Bob should agree on the same shared secret with good probability:

$$\Pr\left[\mathsf{Finish}(\mathsf{sk}_A, \mathsf{pk}_B) = \mathsf{Finish}(\mathsf{sk}_B, \mathsf{pk}_A) : \begin{matrix}(\mathsf{pk}_A, \mathsf{sk}_A) \xleftarrow{R} \mathsf{Gen}(1^\lambda) \\ (\mathsf{pk}_B, \mathsf{sk}_B) \xleftarrow{R} \mathsf{Gen}(1^\lambda)\end{matrix}\right] \geq \text{“large.”}$$

Security against time-T eavesdroppers means that an eavesdropper that runs in time T should not easily be able to recover the shared secret. That is, for all adversaries $\mathcal{A}$ running in time at most T:

$$\Pr\left[\mathcal{A}(\mathsf{pk}_A, \mathsf{pk}_B) = \mathsf{Finish}(\mathsf{sk}_A, \mathsf{pk}_B) : \begin{matrix}(\mathsf{pk}_A, \mathsf{sk}_A) \xleftarrow{R} \mathsf{Gen}(1^\lambda) \\ (\mathsf{pk}_B, \mathsf{sk}_B) \xleftarrow{R} \mathsf{Gen}(1^\lambda)\end{matrix}\right] \leq \text{“small.”}$$

Normally we want the even stronger security property that the shared secret is indistinguishable from random, but this is good enough for now.

Merkle's protocol. In Merkle's scheme, the shared secrets live in the set $\mathcal{S} = \{1, \ldots, n^2\}$. In his scheme, Alice and Bob share a public hash function $H\colon \{1, \ldots, n^2\} \to \mathcal{Y}$, where the space $\mathcal{Y}$ is just some exponentially large set of bitstrings. The important thing is that H has no collisions—distinct inputs give distinct outputs.

Then the scheme goes as follows:

1. $\mathsf{Gen}() \to (\mathsf{pk}, \mathsf{sk})$.
   - Sample n numbers at random $x_1, \ldots, x_n \xleftarrow{R} \{1, \ldots, n^2\}$.

- Set $\mathsf{sk} \leftarrow (x_1, \ldots, x_n)$.
- Set $\mathsf{pk} \leftarrow (H(x_1), \ldots, H(x_n))$.

2. $\mathsf{Finish}(\mathsf{sk}, \mathsf{pk}) \rightarrow \mathcal{S}$.
   - Parse $(x_1, \ldots, x_n) \leftarrow \mathsf{sk}$.
   - Parse $(y_1, \ldots, y_n) \leftarrow \mathsf{pk}$.
   - Find all pairs (i, j) such that $H(x_i) = y_j$. Find a canonical pair (e.g., the one with the smallest value y_j) and output x_i as the shared secret.

When put into play this looks like:

1. Alice chooses $a_1, \ldots, a_n \xleftarrow{R} \{1, \ldots, n^2\}$ and sends $H(a_1), \ldots, H(a_n)$ to Bob.
2. Bob chooses $b_1, \ldots, b_n \xleftarrow{R} \{1, \ldots, n^2\}$ and sends $H(b_1), \ldots, H(b_n)$ to Alice.
3. Alice and Bob both look for a "collision:" an (i, j) such that $H(a_i) = H(b_j)$.
4. Alice uses $a_i \in \{1, \ldots, n^2\}$ as her secret key.
5. Bob uses b_j as his secret key.

Correctness. Follows from the Birthday Paradox. If you choose two sets of n numbers at random from $\{1, \ldots, n^2\}$, you will get a collision with constant probability.

Security. Is the really clever part. Alice and Bob run in time n. But the attacker's task is more difficult: the attacker is given:

$$H(a_1), \ldots, H(a_n) \qquad H(b_1), \ldots, H(b_n).$$

In time n, the attacker can find the value $v^* = H(a_i) = H(b_j)$ whose preimage is the shared secret. But recovering this secret will take—at least if we model the hash function as a random function—time $\Omega(n^2)$.

We will not prove this here, but the basic idea is that in $o(n^2)$ guesses, the attacker is very unlikely to ever guess the preimage of v^*. In that case, the preimage of v^* could take on many possible equally likely values and the adversary has little chance of guessing it.

The foundation of public key cryptography In the years since the Diffie-Hellman paper, we have figured out how to build key-exchange protocols, public-key encryption, and digital signatures from a variety of cryptographic assumptions. But there are two assumptions are by far the most popular in practice:

- The discrete-log problem – either in $\mathbb{Z}_p^*$ or certain elliptic-curve groups
- The hardness of factoring integers (and derivate assumptions, such as RSA)

That's it. There are two. In fact, RSA is on the way to being deprecated, so there is really only one.

That is to say, the security of our *entire digital infrastructure*—the privacy of your web traffic and your Gmail messages, the integrity of software updates for your car, the security of nuclear power plant's control systems (I'm guessing), etc.—all rely on the hardness of basically one computational problem. This is a conspicuous single point of failure for our tech ecosystem.

And what if these problems turn out to be **easy**??? *[Cue suspensful music.]*

Okay, I know that you probably know the end of this story. But humor me and pretend you don't.

Course logistics

And now for something completely different...

- Course staff:
  - Yael Kalai, me (profs)
  - TAs/LAs: Katarina Cheng, Leo Wang, and Katherine Zhao
- Course website: `https://65610.csail.mit.edu/`
  - Piazza link is there
  - Gradescope code is on Piazza
  - Video links are there
  - We will try to post notes there before lecture
- Course policies: On the website, please take the time to read them. I will summarize the important points but please read the details.
- We have an exciting new policy on ChatGPT! (Preview: You can use it but you MUST quote/cite.)
- Communication: Please do by Piazza for almost everything.
  - Public notes for questions, please. (You can be anonymous!)
  - Exception: Sensitive personal situation that you want to share with me or Yael and do not want
  - Exception: Anonymous course feedback. We love it! Please send it this link, which is also on the website.

- Recitations: We will have them unless noted on the course calendar
- Assignments
  - 40% – Four problem sets (due on Fridays).
    - Since this is an applied crypto class, we will have some programming on almost every problem set.
    - **The first three you must do in teams assigned by the course staff.**
    - Will email assignments shortly.
    - We use Ron Rivest's very student-friendly system to weight the problem sets. See the course website.
  - 20% – One quiz on April 17.
  - 40% – The final project! Your chance to get a taste of security/cryptography research.
    - Done in teams of 3-4 students. (Teams may change; see details.)
    - The details are listed on the course website.
    - Many small check-in assignments to make sure that you are making good progress.
    - The report must explain who did what.
- Late policy: On the course policies page.
- Very important: Please use S^3! They are amazing and can help you whenever you get stuck. It's scary to ask for help. But

  Student mental health is also an excellent resource.

  You can always always come to me and/or Yael!

Effect of large-scale quantum computers

Before I get into this discussion, I should warn you: this is not a course on quantum cryptography and I am not an expert on quantum computing (though Yael is!). If you want to learn about quantum cryptography, you are in luck! You can take **6.S895** this semester T/Th 11am to go way into depth on quantum computing. Two of the world's experts will be teaching that class, so it should be good.

We left off musing on what a disaster it would be if someone cooked up a discrete-log algorithm...

Quantum computing in three minutes (SKIP)

Physicists and computer scientists had been thinking about using quantum mechanics for computation in the 1970s and 1980s.

The Stanford Encyclopedia of Philosophy article has more details on the history and pointers to the relevant papers.

At the crudest level, you can think of a quantum computer as a tweaked Boolean circuit. If you feed random inputs to a circuit, you will induce some probability distribution on the wires and output. We can describe the probability of a particular wire carrying a zero or a one as real number. For example, feeding random inputs to an AND gate gives you a 1 with probability 1/4 and a zero otherwise. In a quantum circuit, the wires instead carry *complex numbers*. The fact that you can add two non-zero complex numbers and get a zero (not true for probabilities) is the source of a quantum circuit's apparent power. The computational model is extremely simple and clean. At the same time, no one yet has been able to actually implement a large quantum circuit

The surprise

The gigantic surprise in early 1990s was the discovery by Shor (1994) that polynomial-size quantum circuits can factor integers and compute discrete logs—in $\mathbb{Z}_p^*$, elliptic curve groups, and actually any group we would ever use for cryptography.

More precisely, if you could implement a quantum circuit with some big-but-not-necessarily-infeasible number of gates (this paper estimates roughly 2^{40} gates) you could factor the 2048-bit integers that we use in the RSA cryptosystem.

There has been some exciting recent work on reducing the number of gates, but I don't think anyone has worked out the constants involved in these newer factoring algorithms yet.

The consequence of Shor's algorithm is that a large-scale quantum computer could theoretically break ***both*** of the popular assumptions underlying essentially all public-key cryptography in use today. In particular, a quantum computer will render our beloved Diffie-Hellman key-exchange protocol unusable.

On top of that, in 1996, Grover showed that quantum circuits can theoretically break block ciphers much faster than we would expect.

Important take-away messages

As an applied cryptographer, the two critical pieces of knowledge are that large-scale quantum computers give:

- **polynomial-size** circuits for factoring integers and computing discrete logs via Shor, so **our standard key-exchange protocols, public-key encryption systems, and digital-signature schemes are completely broken**, and
- **better exponential-size** circuits for breaking block ciphers and hash functions via Grover. The consequence is that **we**

One of the tragedies of quantum computing (from where I sit) is that even a gigantic quantum computer does not seem to help us solve everyday computational problems (those in NP, for example) much faster. My physicist friends tell me that they are potentially great at simulating quantum systems and in 6.S895 you can learn about other surprising things that they could potentially do.

need to double our key sizes (or output size, for hash functions*) to achieve the same level of security.

This explains why the post-quantum parts of this course will focus on public-key algorithms. Our symmetric-key cryptosystems are still basically fine to use, provided we bump up the key size.

Should we care? Whether or not you believe that anyone will build a quantum computer in the next 100 years, the world will soon start using a new generation of post-quantum-secure crypto primitives. So understanding these new tools is important no matter what.

Governments want to switch to new post-quantum cryptosystems now so that attackers in 30 years with a quantum computer can't decrypt today's traffic.

A small example: Grover search

Describing Shor's algorithm requires a little background; the API of Grover's algorithm is straightforward.

- **Given:** A function $f\colon \{0,1\}^n \to \{0,1\}$.
- **Find:** A value $x \in \{0,1\}^n$ such that $f(x) = 1$, if one exists.

With a classical computer, solving this type of "black-box" search problem is very hard. It will take every classical algorithm $\Omega(2^n)$ invocations of f to find a solution with constant probability.

In contrast, Grover gives a quantum circuit that has size $\text{poly}(n)\,|f| \cdot 2^{n/2}$, where $|f|$ denotes the size of the representation of f as a Boolean circuit. So if f is an efficient function, Grover search can solve the problem with a quantum circuit of size roughly $\text{poly}(n) \cdot 2^{n/2}$.

To compare: If $n = 128$, a classical computer takes an infeasible 2^{128} invocations of f; the quantum circuit uses a very plausible 2^{64} invocations of f. (The Bitcoin network is computing 2^{68} hashes *per second*.)

The power of Grover's search is that it's completely agnostic to the function f. It works for all efficient functions.

An application: Inverting a hash function. Say that you have a block cipher $E\colon \{0,1\}^n \to \{0,1\}^n \to \{0,1\}^n$. The following task captures a very standard goal in cryptanalysis:

- **Given:** Two encrypted messages

  $$c_1 \leftarrow E(k, \text{“}hello''\text{)} \qquad c_2 \leftarrow E(k, \text{“}world''\text{)},$$

  for $k \xleftarrow{R} \{0,1\}^n$.
- **Find:** The encryption key k.

We can apply Grover search to solve this problem:

$$f(x) := \begin{cases} 1 & \text{if } c_1 = E(x, \text{“}hello''\text{) and } c_2 = E(x, \text{“}world''\text{)} \\ 0 & \text{otherwise.} \end{cases}$$

So this cryptanalytic problem, which would have taken roughly 2^n time on a classical computer could theoretically take roughly $2^{n/2}$ time on a quantum computer. The upshot is that to protect against quantum attacks, you **need to use 256-bit symmetric encryption keys**.

Where next?

Tomorrow we will dive into the design of the AES block cipher (it's post-quantum secure!) and next week we will start in on lattice-based cryptography, the most promising replacement for factoring- and discrete-log-based public-key cryptosystems. As we will see, lattice-based cryptosystems have a number of other magical properties as well.

References