0368.4162: Introduction to Cryptography Ran Canetti

Lecture 1

03 November 2008 Fall 2008

Scribes: D. Shahaf, D. Sotnikov, N. Bitansky

Lectures and Office Hours

Lectures are on Mondays, between 12:10–15:00, in Orenstein 110. Office hours are by appointment.

Prerequisites

- Basic complexity (the classes P, NP, BPP).
- Basic probability.
- Some prior informal-level knowledge of cryptography (such as an undergraduate course or equivalent prior experience).

Course requirements

Each week a group of students will prepare class notes covering that week's class. Each student will participate in preparing at least one set of notes.

The notes will be typeset in English, using a LaTeX template available from the lecturer and posted on the course's Web site. First draft is due by Thursday of the same week. Revisions due in a timely manner. (The hope is to have the notes available to all within a week or two of the lecture.)

A sign-up sheet for volunteering will be posted.

There will be approximately 3 to 5 homework sets. There will also probably be a final exam.

The final grade will be computed using the following weights:

- Exam: 30%
- Homework: 40%
- Class notes: 30%

(If there is no final exam, the weights would be 70% for homework and 30% for class notes.)

Mailing lists

The course's mailing list is `0368-4162-01@listserv.tau.ac.il`. The lecturer and all registered students are automatically subscribed. To subscribe, send an email to `listserv@listserv.tau.ac.il` with the line
`subscribe 0368-4162-01 <Real Name>`.
The course's Web site is `http://www.cs.tau.ac.il/~canetti/f08.html`.

Course materials

There is no single textbook we'll be following. However:

- Most of the technical material in the course (at least the first two thirds of it) is covered in at least one of:

  [G] *Foundations of Cryptography* by Oded Goldreich Vol I and II, Cambridge U. Press.

  [KL] *Introduction to Modern Cryptography* by Katz and Lindell, CRC Press.

  [BG] *Lecture Notes on Cryptography* by Bellare and Goldwasser, available online.

  [CLRS] *Introduction to Algorithms* by Cormen, Leiserson, Rivest, and Stein, MIT Press (Chapter 31, on number theory).

- More in-depth reading on number theory:

  [Sh] *A Computational Introduction to Number Theory and Algebra* by Shoup, Cambridge U. Press, also available online.

  [BS] *Algorithmic Number Theory* by Bach and Shallit, MIT Press.

- Applied cryptography:

  [MQV] *Handbook of Applied Cryptography* by Menezes, van Oorschot and Vanstone, CRC Press.

  [Sch] *Applied Cryptography* by Schneier, Wiley & Sons.

Overview of cryptography

Main goal: Protecting information systems from unwanted use.

Examples from the ancient world

Ciphers to protect secrecy: Caesar cipher

Definition: the Caesar Cipher erases all whitespace and punctuation and maps each letter to the (cyclically) nth next letter in the alphabet. For example, with $n = 3$, the string "`ABCXYZ`" is enciphered as "`DEFABC`".

Properties:

- Only those who know the secret can encrypt/decrypt
- Anyone who knows the secret can encrypt/decrypt.

Caveat:

- What's the "secret" — the method or the key (rotation amount)?

Source and contents integrity verification: Seals

Properties:

- Public verifiability.
- Message integrity.
- Sender authentication.
- Non-repudiation.
- Also limited Secrecy (e.g., if the message is put in an envelope, one can tell whether it had been opened in transit).

Trusted randomness: Lottery

Lottery was used in ancient Athens (among other places) for selecting officials at random: Notes are put in a bin, one or more are marked. People pick notes in turn. Whoever picks a marked note wins the prize (office) written on it.

Properties:

- Unbiased result; no need to trust anyone (if done right) – guarantees fairness in the presence of opponents.

Voting: Ostraca ballots

In ancient Athens, people voted by writing the name of their candidate on a pottery shard (ostraca) and putting the latter in a bin. Later the votes were counted and (if there are enough votes were cast) then the "winner" got exiled.

Properties:

- Individual votes remain private.
- No one can vote twice (since the act of voting is public).
- Public verifiability of the winner.
- Public verifiability of the number of votes.
- Collusion-resistant (the pottery shards are mutually indistinguishable).

Modern cryptography

In modern information systems, the basic concerns and goals are the same (of course, translated to our modern perspectives): secrecy, integrity, non-repudiation, fairness, correctness of computation, etc. What has changed is the setting:

- Scale: Ability to process information is greater
- Capabilities of cryptographers are greater
- Capabilities of attackers are greater

- Translation of physical media to digital ones
- Systems (both legitimate users and adversaries) are becoming more complex
- More ways of attacking a system
- Require a way to provide a rigorous and quantitative guarantee of the security properties of the scheme.

Consequently, the techniques are very different.

> Note: Since the efficiency of an algorithm can be evaluated empirically (by running it on representative sample inputs), we can often determine whether a given algorithm is sufficiently efficient for our needs even without a rigorous proof of efficiency. Contrarily, the security of a cryptographic algorithm cannot be established experimentally; we cannot be convinced that a protocol is secure unless we have proved it so. Therefore, every cryptographic scheme must be accompanied by a proof of its security. The proof is an inseparable part of the scheme, and should be a necessary requirement for putting the scheme to use in practice.

So, the question is: which techniques to use to *construct* schemes and to *analyze* them?

First approach: Use techniques from information theory

Information theory is concerned with quantitative study of "information" in digital media, particularly in the context of communication. It provides ways to make statements such as "this random variable has this much information on this data". Its foundation is commonly attributed to Shannon (around 1948); Hamming (around 1952) also made significant contributions.

Advantages: information theory allows one to make precise and unconditional statements such as "a ciphertext c produced using process A contains I amount of information on the plaintext message m". (When $I = 0$, we say that A is a *perfect cipher*.)

Main disadvantage: Information theoretic techniques are limited and can be used only in few contexts.

On result of the use of information-theoretic techniques is a theorem due to Shannon, which states that "Any encryption scheme that achieves perfect (information theoretic) secrecy must have secret key that is at least as long as the encrypted message." (For a more formal statement and proof, see [KL].)

(Corollary: a system that achieves perfect secrecy must not reuse a key with more than one message — in other words, it must use a new key k' for each new message m'.)

A simple scheme that achieves Shannon's lower bound for key length is the *one-time pad*: given a message m and a key k of the same length, the ciphertext c is constructed as the bitwise-xor of m and k. (Decryption is done in the same way.) It can be proven that, using this construction, the ciphertext c has no information on the plaintext message m.

Still, it should be stressed that information-theoretic techniques and concepts are often very useful in cryptography.

Second (and main) approach: Use techniques that work only against resource-bounded attacks

In practice, we're only concerned with resource-bounded attackers; therefore, it doesn't matter if the ciphertext holds (even complete) information on the plaintext, as long as there is no feasible way to compute the plaintext from the ciphertext.

Such techniques involve a small probability of error; either in the ability of the intended recipient to extract the plaintext from the transmitted ciphertext, or in the inability of an attacker to crack the system herself.

Advantage:

- Allows doing many many things that are otherwise impossible.

  This is the "magic of cryptography" — (almost!) anything you might think of ends up being possible one way or another.

Disadvantages:

- Not perfect. Inherently allows some probability of error.
- Cumbersome: Requires quantifying success probability against computational resources of attacks, e.g.: time, space, bandwidth, etc.
- Given the pitiful state of knowledge on proving lower bounds on the complexity of computational problems, we cannot hope to obtain unconditional statements of security. Statements will typically be of the form:

  > If problem A is hard for algorithms that use R amount of resource T, then the scheme S satisfies a given property P against adversaries that use R' amount of resource T'.

  In fact, things are even more complex: property P will include some probability p of error, where p depends on R', R and other parameters of the scheme.

Consequently, things become much more delicate, as described in the following sections.

The layers of modern cryptography and information security.

Note: This partition is somewhat artificial and does not always apply. Still it is useful to keep in mind.

Layer 1: Hard computational problems.

- Problems come from many different areas and in different flavors:
  - Number theory
  - Algebra
  - Combinatorics
  - Experimental heuristics
- Different assumptions have quite different properties, e.g.: Additional structure (can be both good and bad), time to compute, extensibility properties.
- Different assumptions vary in the "level of confidence" in the hardness: amount of time spent on breaking, relations to other assumed-to-be-hard problems.

Layer 2: Cryptographic primitives (schemes and protocols).

These are algorithms that obtain a well-defined goal (e.g. secrecy, integrity, correct tally, unbiased random choice). Typically, these algorithms use some assumed-to-be-hard computational problem(s), and the security properties of the scheme are reduced to the hardness of the problem(s). Typically, the design and analysis of cryptographic protocols and schemes is in itself a layered process; that is, a scheme uses other schemes as subroutines and relies on the security properties of the subroutine schemes to obtain its own goals.

Layer 3: Building secure systems.

Deploying cryptographic schemes in actual systems to guarantee security for the user involves many issues:

- Verifying correctness of the implementation (naive software bugs can be disastrous).
- Dealing with integration within existing systems.
- Dealing with interoperability across various platforms and OS's.
- Dealing with additional security pitfalls:
  - Peculiarities and imperfections of the actual system (e.g., using the user's input without correctness checking).
  - Side channel attacks (e.g., measuring the power consumption of a device).
  - Human interface pitfalls (e.g., phishing, weak passwords, etc.).
- Extensibility and composability.

A final semi-philosophical remark: The importance of abstraction in cryptography.

Abstraction is a very powerful tool in mathematics in general, and in computer science in particular:

- Allows sieving out the "non important" details and concentrating on the salient.
- Allows building systems in a modular way, where one component relies only on certain salient properties of the other components.
- Enables a common language for multiple problems, techniques and applications.
- Helps thinking, conceptualizing, and developing better algorithms.

Abstraction is especially useful in cryptography. In fact, one may say that modern cryptography is all about abstractions. Both the assumptions and the security goals are stated in terms of appropriate abstractions. This approach is essential for:

- Comparing and analyzing different computational problems and hardness assumptions.
- Building schemes and protocols that can depend on more that a single assumption.
- Making security analysis of protocols a more feasible task:
  - Making comprehensible and meaningful security statements.

– Deducing security of complex systems from properties of building blocks.

On the other hand, abstraction is a dangerous tool. Potential pitfalls:

- Overly cumbersome formalism that hides the essence.
- "Abstracting out" feasible attacks of actual adversaries.

For instance, traditional computational models do not capture "side channel attacks" such as measuring the power consumption of a device.

In conclusion, making the "right abstraction" is often half-way to success in cryptography. It should be noted, though, that the importance of the abstraction in cryptography is sometimes downplayed, both by mathematicians and by practitioners. In fact, layer 2 in the above description is sometimes ignored in security textbooks and courses, and the line of reasoning goes directly from the hard problem to the computer program that uses it, with no intermediate steps or analysis.

This clash of philosophies and cultures is the source of an ongoing debate. For some (often amusing) highlights see e.g.:

http://eprint.iacr.org/2004/152

http://www.wisdom.weizmann.ac.il/~oded/on-pmc.html

as well as multiple blogs (e.g., Luca Trevisan's blog).

Class goal and overview

The course aims to provide a graduate-level introduction to Cryptography and prepare students to doing research in that area. The goal is to give students a taste of the main concepts, abstractions and algorithms, as well as the main tools and techniques. Some advanced topics will be mentioned along the way. We'll cover mostly layers 1 and 2, but will make forays into layer 3 (and will try to always keep it in mind).

The exact material covered will be determined as the semester proceeds, based on the abilities of the instructor and the response of the class. Let me spend some time to give a brief overview of the general material to be covered.

Basic primitives:

One way functions (OWFs):

A function is one-way if it is "easy to compute but hard to invert".

This is the most basic common abstraction for computationally hard problems.

A successful abstraction:

- Has many different instantiations.
- Underlies practically any cryptographic scheme.

Hard-core bits (hard-core predicates):

Loosely speaking, a hard-core predicate $hc : \{0,1\}^* \longrightarrow \{0,1\}$ of a function f has the following property: given $f(x)$, it is infeasible for any polynomial-time algorithm to correctly determine $hc(x)$ with probability significantly better than $1/2$. (It is always possible to compute $hc(x)$ correctly with probability exactly $1/2$ by random guessing.)

Pseudorandom generators (PRGs):

A function is a pseudorandom generator if, given a short random string, it outputs a long(er) string that "looks random" for computationally bounded observers.

We'll see constructions:

- Direct (closely related to "stream ciphers").
- From one-way functions.

Pseudorandom functions (PRFs):

A family of functions $\{f_k\}_{k \in K}$ is pseudorandom if having oracle access to a box that answers queries according to f_k for a random $k \in K$ "looks like" having access to a completely random function.

This is a significantly stronger requirement that PRGs: yields an unbounded amount of randomness...might even look impossible. Still, show constructions:

- Direct (closely related to "block ciphers", e.g. DES, AES).
- From PRGs.

As we'll see, PRFs are an extremely useful primitive/abstraction.

Collision resistant hash function (RCHFs):

A function f is collision resistant if its range is smaller than the domain, and still it is hard to find a collision x, y s.t. $x \neq y$ and $f(x) = f(y)$.

Such functions are extremely useful in cryptography.

We'll see constructions:

- Number-theoretic.
- Ad-hoc.
- It is not possible to build RCHFs from OWFs in a generic way.

Trapdoor functions and permutations (TDFs):

A function is a TDOWF if it is a OWF, and in addition there is a secret "key" that allows for efficient inversion.

Here we have fewer candidates: Only based on number theory or algebra.

TDFs are the basis for "public-key cryptography". (This is a bit of a cheat, will correct when we get there.)

Basic protocols 1: Secure communication and related tasks

Remark: On the informal difference between primitives and protocols

Cryptographic primitives can be thought of as "low-level" algorithms that are designed for a very specific task. Cryptographic protocols, on the other hand, usually deal with more complex tasks. They use cryptographic primitives as basic building blocks and describe how they should be used in order to attain specified security properties.

Encryption

Encryption deals with the task of transmitting information over public channels so that only "authorized" parties can obtain the information. In modern cryptography, encryption is captured via a triple of algorithms (Gen, Enc, Dec) with an associated message domain M. In order to transmit a message $m \in M$, the sender encrypts it using E_k, and the recipient decrypts it using D_k. Both algorithms use the same secret key, k, generated by G. The underlying correctness requirement is naturally that $D_k(E_k(m)) = m$.

Note: this simple formalism encompasses an important principle that underlies modern cryptography: Algorithms are always assumed to be public knowledge. Only keys can be secret. In addition to being crucial for analytical study. This principle is practically important for many reasons, such as wide deployment, easy change of keys and public scrutiny. In fact, it was proposed in the 19th century by the Prussian general, Kerckhoffs.

There are two main settings for encryption schemes:

- **Shared key ("symmetric encryption")** — The sender and receiver share a secret key. This setting is modeled by having the key generator G give the same key to both E and D (encrypter and decrypter).
- **Public key ("asymmetric encryption")** — Anyone can encrypt a message to a given recipient. This setting is modeled by having G generate two keys: e for encryption (to be made public) and d for decryption (to be used only by the receiver). The main goal is to simplify the problem of key management in a large community, namely the parties need not be concerned with the task of secure key exchange. Moreover, each party has a single secret key (for decryption) as opposed to many secret keys (one for each peer) in the symmetric encryption setting.

Dealing with encryption, there is a variety of security properties to consider:

- **Secrecy** — against a "passive eavesdropper", against an "active adversary" that can choose messages to be encrypted, and even ciphertexts to be decrypted.
- **Homomorphism** — allowing to "combine" encryptions to a new encryption of a related message. For example, we may want to have $E(m_1) \cdot E(m_2) = E(m_1 + m_2)$.
- **Non-malleability** — preventing an adversary from changing an encryption of a message m to an encryption of a related message m' (opposite in spirit to homomorphism).
- **Forward secrecy** — requiring that in case the security of our system is compromised, it would not compromise any encrypted messages that were sent in the past. This in particular means that any decryption key has to be updated periodically.
- **Conditional/Partial decryption** — allowing a certain party, P, to decrypt only a subset $M_P \subseteq M$ of all messages. For example, allowing Bob to decrypt only messages starting with his name.

Message Authentication

Two parties have a shared secret key and wish to be able to confirm that the messages they receive have really been generated by the peer and were not generated or modified by a third party (authenticity and integrity). In order to transmit a message $m \in M$, the sender first uses an authentication algorithm, A, to produce a "tag" $t = A_k(m)$ and then sends (m, t). Upon receiving the message the recipient checks that indeed $A_k(m) = t$.

Note: It can be shown that having a family of pseudorandom functions $\{f_k\}_{k \in K}$ allows us to construct an authentication algorithm. Informally, the key generation is done by selecting some random $k \in K$. The tag for a message m will be $t = f_k(m)$. Since the result $f_k(m)$ can not be distinguished from a random string, any adversary will not be able to find a pair (m', c) such that $f_k(m') = c$ without having one of the parties send m' (except for some negligible probability).

Digital Signatures

In a digital signing scheme, one entity — and it alone — can generate "signatures" for documents, which signatures are verifiable by everyone. The notion is captured by a triple of algorithms (G, S, V). G generates a pair (k_1, k_2), where k_1 is a private signature and k_2 is a public verification key. The sender (who possesses k_1) sends $(m, S_{k_1}(m))$. Any recipient can verify a message (m, sig) by checking that $V_{k_2}(m, sig) = 1$. Digital signatures can be thought of as the public-key variant of message authentication. However, as opposed to private key authentication, they also provide "public non-repudiation".

Secure Communication

The task of communicating in an authenticated and secret way over untrusted network. We will formalize the task and show solutions.

Key Exchange

In principle secure communication can be done directly from public-key encryption and signatures, but in practice it turns out to be more efficient to go via an initial stage for generating a shared secret key. This stage is usually called "key exchange" and is treated as a separate primitive (i.e., abstraction).

Basic protocols II: The case of mutually distrustful parties

Commitment

The main idea of commitment schemes is to allow one party to commit to a certain value, while keeping it secret. Later, the commitment is "opened", with the guarantee that "opening" will result in the value to which the first party originally committed (it can not be tempered with).

Coin Tossing

Allowing two (remote) parties to flip a coin without any bias. Note that, having a commitment scheme, C, allows us the construction of a coin tossing protocol. The first party sends a commitment $C(a)$ for a random coin flip $a \in \{0, 1\}$. The second party then flips the coin, and sends the result to the first. Finally, the first party "opens" the commitment and the result is revealed.

Figure 1: Coin Tossing using a commitment scheme

$$\begin{array}{ccc} A & \xrightarrow{C(a)} & B \\ A & \xleftarrow{b} & B \\ A & \xrightarrow{open(a)} & B \end{array}$$

Zero knowledge Proofs

How can one prove he has a "solution" for a certain problem, without revealing any details about the solution.

Example: consider the Graph Isomorphism problem. Given two graphs (G_1, G_2) decide whether they are isomorphic (this NP problem is not known to be in either NPC or P). Suppose that Alice and Bob are given two such graphs. Alice claims that the graphs are isomorphic (i.e., she has a permutation σ of the vertices that transforms G_2 into G_1). Can she convince Bob of her ability without revealing the actual permutation?

Protocol:

`Alice`: Selects a random permutation of vertices π, which transforms G_1 to G'. She then sends G' to Bob .

`Bob`: Selects a random $b \in \{0, 1\}$ and asks Alice to show an isomorphism between G' and G_b.

`Alice`: In case $b = 1$ she sends bob π and in case $b = 2$ she sends him $\pi \circ \sigma$.

`Bob`: Verifies her answer.

It is not to difficult to see that if Alice is not lying she will always be able to go through with protocol and if she's not there is a probability of $\frac{1}{2}$ that she'll succeed cheating (i.e., by repeating the protocol Bob can get as much convinced as he would like). Informally, the "zero knowledge" principle is fulfilled, since Alice gives away only random permutations (which Bob is capable of producing by himself).

Oblivious Transfer

In an oblivious transfer protocol, the sender has two messages m_0 and m_1, and the receiver has a bit b. The receiver wishes to receive m_b, without the sender learning b, while the sender wants to ensure that the receiver receive only one of the two messages.

General function evaluation

General protocols for parties to carry out any joint computation on their respective local data while preserving: secrecy of local data, correctness of outputs, and maybe other properties such as: lack of influence, fairness, timeliness. We will describe how to perform a secure joint computation of any function using the primitives mentioned above.

Advanced topics

General notions of security and protocol composition

Two questions come to mind when coming to design and analyze secure systems:

- Is there a general methodology for defining security requirements of cryptographic tasks?
- What happens to the security properties of protocols in a large system that combines multiple protocols and primitives? Do they remain intact? Do additional concerns come up?

We'll see that these questions can be addressed by using a general framework for specifying and analyzing security of cryptographic protocols. We'll give some introduction to this framework (called UC framework, for universally composable security).

Mechanized analysis

We'll see that analyzing security even of small, simple protocols can be very complex. How can we analyze security of large protocols? Here mechanized analysis seems to be a must. We'll give some intro to the state of the art in mechanized analysis.

Cryptography and Game Theory

Game theory is a discipline of economics that studies mechanisms for interaction among parties with conflicting interests. As such, it is very close to the discipline of cryptographic protocols, at least in the subject matter. However, game theory takes a very different approach from cryptography, trying to *predict* how "rational" participants will behave in a given interaction setting. Recently, some initial work has been done that bridges between the two approaches, towards a more comprehensive understanding of interaction between parties with conflicting interests. We'll touch upon this work.

Program Obfuscation

Informally, a program is "obfuscated" if it is hard to "understand" how it works. Having a mechanism that can turn any given program into a program that has the same functionality but is "completely obfuscated" sounds like a very useful prospect. Can that be done? What does it mean to "completely obfuscate" a program? These are interesting questions and not too much is known to answer them. We'll give a brief introduction to the topic.

New assumptions

In recent years a number of new hardness assumptions have been proposed, utilizing number theoretic and algebraic constructs such as bilinear forms and lattices. These assumptions give rise to new schemes with some fascinating properties. We'll give a brief introduction.